# Secure Public Auditing Schemes for Shared Data with User Revocation in Cloud Storage

**Submitted in partial fulfillment of the requirements**

**for the award of the degree of**

**DOCTOR OF PHILOSOPHY**

***Submitted by***

**Gudeme Jaya Rao**

**(Roll No. 716178)**

***Under the guidance of***

**Dr. P Syam Kumar**

**and**

**Dr. K Ramesh**


**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL**

**TELANGANA - 506004, INDIA**

**August 2021**

# DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
# NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL
# TELANGANA - 506004, INDIA

## THESIS APPROVAL FOR Ph.D.

**This is to certify that the thesis entitled, Secure Public Auditing Schemes for Shared Data with User Revocation in Cloud Storage, submitted by Mr. Gudeme Jaya Rao [Roll No. 716178] is approved for the degree of DOCTOR OF PHILOSOPHY at National Institute of Technology Warangal.**

**Examiner**

**Research Supervisor**
**Dr. P Syam Kumar**
**Center for Cloud Computing**
**Institute for Development &**
**Research in Banking Technology**
**India**

**Research Supervisor**
**Dr. K Ramesh**
**Dept. of Computer Science and Engg.**
**NIT Warangal**
**India**

**Chairman**
**Prof. P. Radha Krishna**
**Head, Dept. of Computer Science and Engg.**
**NIT Warangal**
**India**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**
**NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL**
**TELANGANA - 506004, INDIA**

# CERTIFICATE

**This is to certify that the thesis entitled, Secure Public Auditing for Shared Data with User Revocation in Cloud Storage, submitted in partial fulfillment of requirement for the award of degree of DOCTOR OF PHILOSOPHY to National Institute of Technology Warangal, is a bonafide research work done by Mr. Gudeme Jaya Rao [Roll No. 716178] under our supervision. The contents of the thesis have not been submitted elsewhere for the award of any degree.**

**Research Supervisor**
**Dr. P Syam Kumar**
**Center for Cloud Computing**
**Institute for Development &**
**Research in Banking Technology**
**India**

**Hyderabad**
**Date: 02-08-2021**

**Research Supervisor**
**Dr. K Ramesh**
**Dept. of Computer Science and Engg.**
**NIT Warangal**
**India**

**Warangal**
**Date: 02-08-2021**

# DECLARATION

This is to certify that the work presented in the thesis entitled "*Secure Public Auditing Schemes for Shared Data with User Revocation in Cloud Storage*" is a bonafide work done by me under the supervision of Dr. P Syam Kumar and Dr. K Ramesh. The work was not submitted elsewhere for the award of any degree.

I declare that this written submission represents my ideas in my own words and where others ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/date/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

**Gudeme Jaya Rao**
(Roll No. 716178)
Date: 02-08-2021

# ACKNOWLEDGMENTS

Every day during my Ph.D. has been a great opportunity for learning. This thesis work is the result whereby I have been supported by many people. It is a pleasant aspect that I have now the opportunity to express my gratitude for all of them.

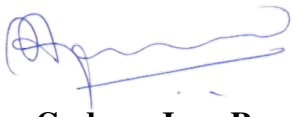
First and foremost, I would like to thank the Lord almighty for glorifying me with all the strength and health to carry out my research work. It is with great pleasure that I acknowledge my sincere thanks and deep sense of gratitude to my supervisors Dr. P Syam Kumar, Assistant Professor, Institute for Development and Research in Banking Technology (IDRBT), Hyderabad and Dr. K Ramesh, Associate Professor, Department of Computer Science and Engineering, National Institute of Technology (NIT) - Warangal for their valuable guidance throughout the course. Their technical perception, profound knowledge, sincere effort in guiding a student and devotion to work have greatly charmed and constantly motivated me to work towards the goal. They always gave me ample time for discussions, reviewing my work and suggesting requisite corrections.

I extend my gratitude to the Doctoral Scrutiny Committee (DSC) members comprising of Prof. S. G. Sanjeevi, Dr. S. Ravi Chandra, Prof. L. Anjaneyulu for their insightful comments and suggestions during oral presentations. I am also thankful to Prof. B. B. Amberker for his presence with valuable suggestions during the presentation of this research work. I am lucky to attend lectures by Prof. B. B. Amberker and Dr. Rashmi Ranjan Rout during my tenure. I am immensely thankful to Dr. Ch. Sudhakar, Prof. R. B. V. Subramanyam and Prof. P. Radha Krishna Heads of Dept. of CSE and chairmans of DSC, during my tenure for providing adequate facilities in the department to carry out the oral presentations.

I wish to express my sincere thanks to Prof. N.V. Ramana Rao, Director, NIT Warangal and Prof. D. Janakiram, Director, IDRBT, Hyderabad for providing the infrastructure and facilities to carry out the research. I am also very much grateful to the faculty members of Computer Science and Engineering Department, and IDRBT for their moral support throughout my research work.

On the personal level, I would also like to thank my scholar friends in IDRBT and NIT-Warangal for their valuable suggestions and for extending selfless cooperation. Lastly, my

gratitude to my family for their unconditional love, support and prayers for my success in achieving the goal.



**Gudeme Jaya Rao**

*Dedicated to My Beloved*

**Family, Teachers and Friends**

# ABSTRACT

Cloud storage is an important service that provides reliable and resilient storage infrastructure for users to store data remotely with the service provider based on pay-as-you-go pricing model. Today's most popular cloud-based storage services are Amazon S3, Google Drive, Microsoft Azure, Apple iCloud, Dropbox, etc. Cloud storage service brings significant benefits to data owners, say, (1) reducing capital and management costs (2) reducing cloud users' burden of storage management and equipment maintenance, (3) avoiding investing a large amount of hardware, (4) accessing data over an Internet connection from any location from any devices such as desktop computers, laptops, tablets, and smartphones which offers increased flexibility and accessibility. Because cloud storage offers scalable, pay-as-you-go, and location-independent storage services for cloud users, a growing number of organizations and individuals have been outsourcing their data to the cloud storage.

Despite these appealing benefits, cloud storage does trigger security challenges such as confidentiality, integrity, and availability. One of the significant concern is the integrity of outsourced data due to the following factors: i) Once the data is moved to the cloud, data owner loses physical control over the data. ii) Sometimes, for monetary benefits, cloud service providers (CSP) may delete rarely accessed data or hide the data loss incidents to have a good reputation. iii) Data stored in the cloud may be lost due to irresistible byzantine failures or human errors or intentional malicious activity, which can be a burden for the user and an embarrassment for the CSP. iv) CSP may maintain fewer replicas than what is paid for to save the storage space. Therefore, it is highly desirable for the data owner to check the integrity of the outsourced data in the cloud from time to time.

To check the integrity of the outsourced data, several schemes have been proposed in the literature without the local copy of data and without downloading complete data. However, most of these schemes only focus on checking the integrity of personal data, which are not valid under the situation of data shared in a group. In a shared data scenario, one of the users in a group uploads data to the cloud, and the rest of the group not only access but also modify the data. When data is shared among multiple users, some new challenges will arise that must be addressed such as user revocation, privacy preserving, identity pri-

vacy, data dynamics. To address these problems, many shared data auditing schemes have been proposed. However, some of the issues such as user revocation, privacy preserving, identity privacy, availability, data dynamics are not well solved in the existing schemes. Furthermore, most of the schemes suffering from complex certificate management and key escrow problem.

To address aforementioned issues, in this thesis we proposed five contributions, namely, i) An identity-based public auditing for shared data in cloud computing using identity-based signatures to achieve user revocation. This scheme also simplifies the certificate management. ii) An attribute-based public auditing for shared data in cloud storage to simplify complex key management in PKI and ID based schemes and to support user revocation. It also achieves user privacy. iii) Certificateless privacy preserving public auditing for dynamic shared data in cloud storage to achieve data privacy against verifier through random masking technique to blind the data proof during the process of auditing. Double linked list information table is used to support shared data dynamics such as insertion, modification and deletion. iv) Certificateless multi-replica public integrity auditing scheme for dynamic shared data in cloud storage to achieve the availability of data along with the integrity. v) Efficient pairing-free certificateless public auditing for shared big data in the cloud based on ECC to reduce the computation and communication cost substantially during auditing. Through the security analysis, we prove that all schemes are provably secure against various adversaries under the hardness assumption of the standard DL and CDH problems in the random oracle model (ROM). The performance and experimental evaluation show that our schemes are efficient and practical.

# Contents

# List of Figures

# List of Tables

# List of Notations

| | |
|---|---|
| $\lambda$ | Security parameter |
| $\wp$ | A bilinear map |
| $H_1, H_2$ | A collision-resistant hash functions |
| $G_1, G_T$ | Multiplicative groups with prime order $p$ |
| $g$ | Generators of group $G_1$ |
| $Z_p^*$ | Prime filed of order $p$ |
| $m_i$ | $i^{th}$ block of shared file |
| $n$ | Number of blocks |
| $C$ | Challenge |
| $\sigma$ | Signature set |
| $\Omega$ | User attribute set |
| $\Omega^*$ | Attributes associated with leaf nodes |
| $P$ | Proof message |
| $\tau$ | Access policy |
| $s$ | Denotes the sector number |
| $\mid G_1 \mid$ | Denotes the size of an element in $G_1$ |
| $\mid G_T \mid$ | Denotes the size of an element in $G_T$ |
| $\mid p \mid$ | Denotes the size of an element in $Z_p^*$ |
| $Pparams$ | Public parameters |
| ID | Identity of the user |
| $g$ | The generator of group $G_1$ |
| $C$ | Challenge message |
| $\sigma$ | Signature set |

# Glossary

| | |
|---|---|
| ABPIA | Attribute-Based Public Integrity Auditing |
| ABS | Attribute Based Signature |
| CA | Certificate Authority |
| CDHA | Computational Diffie-Hellman Algorithm |
| CLS | Certificateless Signature |
| CLMRPIA | Certificateless Multi-Replica Public Integrity Auditing |
| CLPPPA | Certificateless Privacy Preserving Public Auditing |
| CS | Cloud Server |
| CSP | Cloud Service Provider |
| DL | Discrete Logarithm |
| DPDP | Dynamic Provable Data Possession |
| ECC | Elliptic Curve Cryptography |
| GM | Group Manager |
| IBS | Identity Based Signature |
| IDPIA | Identity-based Public Integrity Auditing |
| IHT | Index Hash Table |
| KGC | Key Generation Center |
| MHT | Merkle Hash Tree |
| MR-PDP | Multi-Replica Provable Data Possession |
| PDP | Provable Data Possession |
| PKG | Private Key Generator |
| PKI | Public Key Infrastructure |

| | |
|---|---|
| RDIA | Remote Data Integrity Auditing |
| RL | Revocation List |
| RVT | Replica Version Table |
| TPA | Third Party Auditor |

# Chapter 1

# Introduction

In this chapter, we introduce the concept of cloud storage, benefits and challenges of the cloud storage. Thereafter, the motivation behind our research is discussed, followed by our research objectives and contributions. We give the thesis organization at the end of the chapter.

## 1.1 Cloud Storage

Cloud storage [1, 2] is an important service of cloud computing, which provides reliable and resilient storage infrastructure for users to store data remotely with the service provider based on pay-as-you-go pricing model. As an essential cloud computing service, cloud storage offers scalable, low-cost and location-independent platform for managing cloud users' data. Because of the unprecedented advantages, an increasing number of individuals and organizations are inclined to use the cloud storage service to save local storage space and enable them to manage resources on demand [3]. For example, a survey shows that 79% of organizations choose cloud storage outsourcing services due to low cost, and organizations can focus on their core business [4, 5, 6, 7]. Cloud storage also allows to form the groups and to share the data with other group users. All these features are not available with on-premises storage. The most popular cloud-based storage services available today are Amazon S3, Google Drive, Microsoft Azure, Apple iCloud, Google App Engine, Dropbox etc.

## 1.2 Cloud Storage-Benefits

A lot of research (for example NIST 2012; Armbrust et al. [3], 2010); focuses on various benefits of using cloud storage in general. The following are the benefits:

- Low or reduced cost. Nowadays, many data owners in academic and business environment are choosing cloud for storing their data in the cloud to save costs. The reason is because of its cost effectiveness, which is particularly true for small and medium-sized organizations. By outsourcing their data, data owners can avoid the capital expenditure, infrastructure setup, large equipment, and regular maintenance cost. [4, 5, 8].

- Accessibility. Cloud storage allows users to access their outsourced data at anytime, anywhere through any device such as desktop computers, laptops, tablets and smartphones.

- Sharing. With cloud storage, customers can create a group and easily share data in the group anywhere and anytime through the Internet. For example, employees in the same department of a company store and access data (reports, common file) as needed.

- Scalability. Traditional infrastructure cannot be scaled up on-demand basis because of its limitations, whereas a cloud infrastructure supports on-demand scalability (i.e., quickly scaling up and down virtual computing resources) with minimal management effort and service interruption or without impacting on the performance of the system. [3, 4, 9].

## 1.3 Motivation

While cloud storage benefits are clear, it also faces several security challenging issues such as confidentiality, integrity and availability. The primary issue is the integrity of outsourced data due to the following reasons : i) once the data is moved to the cloud, Data Owner (DO)

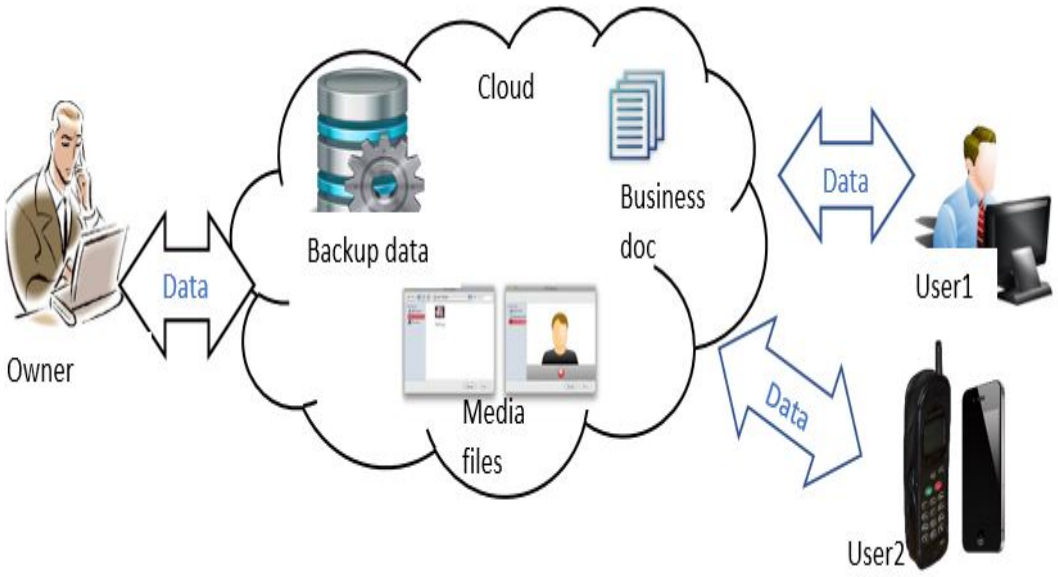



Fig. 1.1. Cloud storage service model

lose the physical control over the data. ii) some times, cloud providers may delete rarely accessed data or hide the data loss incidents to have a good reputation. iii) data stored in the cloud may be lost due to irresistible byzantine failures or human errors or intentional malicious activity, which can be a burden for the user and an embarrassment for the cloud service provider (CSP). iv) CSP may maintain less number of replicas than what is paid for to save the storage space. Therefore, it is highly desirable for the user to ensure the integrity of the data in the cloud at regular intervals.

Traditional methods to verify the integrity of the data, such as hash values or digital signatures for entire data, requires to retrieve the entire data from the cloud. Certainly, this simple approach can successfully verify the correctness of data. However, downloading the entire cloud data is not economical because it incurs unacceptable communication cost and may waste users' resources, particularly when data have been destroyed/corrupted in the cloud, which is not practical in a cloud scenario. In the cloud computing scenario, the user wants to check the integrity of data without downloading complete data.

### 1.3.1 Problem Statement

To check the integrity of the outsourced data, recently, lots of schemes have been proposed in the literature. In 2007, Ateniese et al.[1] introduced a concept of "Provable Data Possession (PDP) "to ensure the data integrity without downloading the complete data by utilizing RSA-based homomorphic verifiable tags and random sampling of blocks. The idea of PDP is that the DO pre-processes the file, constructs the metadata for the file blocks, stores the metadata locally, then uploads the data file along with the tags to CSP and removes the original file at local site. Later, the integrity of the data is validated through the "challenge-response"protocol. However it does not support data dynamics. To support dynamic operations such as insertion, modification or deletion [10] described a dynamic PDP scheme using skip lists. Later, some authors proposed dynamic PDP schemes by utilizing Merkle Hash Tree (MHT) [11, 12, 13, 14, 15] and using Indexed Hash Table (IHT) [15, 16, 17, 18, 19]. However, all these schemes does not support privacy of data against verifiers. To preserve the privacy, Wang et al. [20] proposed a privacy preserving auditing scheme by employing random masking technique. Later, some authors proposed privacy preserving PDP schemes [21, 22, 23, 24, 25, 26, 27, 28, 29, 30]. However, all of the schemes mentioned above deal only with the integrity verification of non shared data (personal data).

In shared data scenario, one of the user in a group uploads data to the cloud, and the rest of the group users not only access but also modify the data. Several schemes [1, 2, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44] have been proposed in the literature. However, most of these schemes [1, 2, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32] only focus on verifying the integrity for personal data, which are not suitable for shared data auditing. When data is shared among multiple group users, some new challenges such as user revocation will arise that must be addressed. Thus, addressing user revocations becomes a key research challenge for achieving practical cloud data auditing. Public integrity auditing of shared data with these existing schemes [33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43] will inevitably reveal confidential information—data privacy, identity

privacy—to TPA which affects the security of the system. To address these problems, lots of fruitful shared data auditing schemes [33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43] have been proposed. However, none of the these schemes [33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43] fully addressed the following challenges in the RDIC schemes for shared data.

- User Revocation: In shared data, different blocks may be signed by different users due to data modifications performed by different users, even if the block is the same. It brings complexity for checking the integrity of the data. Furthermore, the size of the group is dynamic i.e., any group member may leave or join the group at any time, so the user revocation is also an important problem that must be addressed. More specifically, once a user in the group is revoked, he should not be allowed to access or modify the data and all his pair of public and private keys are to be made invalid and all the signatures made by revoked user should be resigned by other existing valid user to ensure the integrity. To achieve user revocation, the traditional method is to download the blocks signed by revoked user from the CSP, calculate the new tags and upload the new tags to the cloud again. It will increase heavy computation and communication cost for the normal user. Therefore, this task should be performed by the CSP rather than by the normal user. How to design an efficient and secure method to outsource the task is a challenge issue[33, 38].

- Identity Privacy Preserving: During auditing, the TPA, who is usually considered to be trusted but curious, might collect the identity information of data generators to obtain significant privacy information to perform malicious activity. Thus, it is important to protect the identity privacy of user in the shared-data auditing[41].

- Data Privacy Preserving: Data privacy protection is an important issue for public auditing, which means that the TPA is not allowed to know any information about the data content while conducting credible auditing. In the public auditing, the core of this problem is how to preserve users' data privacy while introducing a TPA. Although exploiting data encryption prior to outsourcing is an approach to mitigate the privacy concern in cloud storage, it cannot prevent data leakage during the verification process[32]. Thus, it is important for the cloud auditing to include a privacy

preserving mechanism independent to data encryption.

- Data Dynamics: As it is well known that a cloud storage system is not just static in nature, the users often need to update the data dynamically for various application purposes. Therefore, it is significant for cloud storage auditing to support every group user to perform dynamic operations like insertion, deletion, and modification on outsourced data remotely without downloading the whole data [38].

- Data availability: Availability issues appear when the customer is unable to access his data. In cloud, customer's data may be lost due to natural disasters, adversaries' malicious activity or some CSPs may not provide back up. Data should always be available and retrievable in the cloud.

- Complex certificate management: Most of the aforementioned protocols are based on traditional public key infrastructure (PKI), which consists of a set of roles, policies and procedures that needed to issue, manage, distribute, store and revoke digital certificates. The most commonly adopted digital certificate in our daily life is X.509 certificates. However, there are three weaknesses when involving PKI based protocols. Firstly, the generation, management and revocation of digital certificates requires a highly complicated structure. Secondly, a PKI system is a tree structure and the authentication to the current CA relies on its parent CA. Thus, the root CA is a trusted center and self-signed, which is vulnerable since compromising root CA means all the related certificates should be reissued. Thirdly, the certificates issued by a CA may not secure enough to ensure the security of one's secret key[45].

- Key Escrow: The central problem with ID-based RDPC schemes is the inherent key escrow problem of a user's private key, that is to say, the private key generator (PKG) equipped with the knowledge of master secret key and generates all the private keys for the users. Consequently, a malicious PKG can forge a signature on any message on behalf of any user in the system without being detected. This is a serious security gap (issue) in existing ID-based [42, 43, 44] schemes.

## 1.4 Objectives

The core objective of this dissertation is to design public integrity auditing schemes for the shared data to to realize the following security and performance requirements.

- **Public auditing:** Anyone with public parameters should be able to verify the integrity of shared data in cloud storage.
- **Blockless verification:** The verifier can verify the integrity of shared cloud data without knowing the actual content and without retrieving all data blocks.
- **Auditing correctness:** The verifier should be able to verify the integrity of shared cloud data correctly.
- **Auditing soundness.** The malicious cloud server should not pass the TPA's verification if the data is replaced or modified.
- **Privacy preserving:** During auditing process, the TPA should obtain neither any content of shared data nor any identity details of group users.
- **User revocation:** User revocation should be achieved in a secure and efficient manner.
- **Data dynamics:** Dynamic data operations such as insertion, modification and deletion should be supported without downloading data back while the efficient public auditing is achieved.
- **Batch auditing:** The TPA should be able to perform several auditing tasks received from various group users in a fast and cost-effective manner.
- **Data availability:** In cloud, customer's data may be lost due to natural disasters, adversaries' malicious activity or some CSPs may not provide back up. Data should always be available and retrievable in the cloud.

## 1.5 Contributions

To meet the above objectives, in this thesis we proposed the following contributions for public integrity auditing techniques for shared data with user revocation in cloud storage.

- Proposed an Identity-Based (ID) Public Auditing for Shared Data in Cloud Computing using identity-based signatures. Whenever the user is revoked, our scheme enables the proxy server to resign the blocks to save existing group user's computation and communication costs. Meanwhile, a TPA always audits the integrity of shared data in the cloud through the challenge-response protocol. This scheme also simplifies the certificate management.

- Proposed an Attribute-Based public auditing for shared data in cloud storage to simplify complex key management in PKI and ID based schemes and to support user revocation. In this scheme, users sign the data blocks over attributes, and a unique public key used for integrity auditing, not individual public keys for each user in the group. Thus it simplifies the key management. It also achieves user privacy, i.e., signatures don't disclose identity information except that user attributes satisfy the defined access policy. Furthermore it also supports user revocation like the previous scheme.

- Proposed Certificateless Multi-Replica Public Integrity Auditing Scheme for Dynamic Shared Data in Cloud Storage to achieve the availability of data along with the integrity. It simplifies the problems of certificate management in PKI and eliminates the key escrow problem in IBC. We use a novel replica version table (RVT) to support shared data dynamic operations such as modification, insertion, and deletion. This scheme also supports secure user revocation.

- Proposed Certificateless Privacy Preserving Public Auditing for Dynamic Shared Data in Cloud Storage to achieve data privacy against verifier through random masking technique during the process of auditing. We use double linked list information table (DLIT) to support shared data dynamics such as insertion, modification and deletion. we also use the idea of proxy resignatures to support group user revocation.

i.e., whenever a user misbehaves or quits the group, the cloud server is able to carry out resigning process on behalf of group user.

- Proposed Efficient Pairing Free Certificateless Public Auditing for Shared Big Data in the Cloud based on ECC to reduce the computation and communication cost substantially during auditing. It eliminates certificate management and key escrow problems exist in the PKI-based and ID-based PDP schemes, respectively. It is further extended to support the batch auditing, where the TPA can handle multiple tasks concurrently. Since the cloud aggregates the multiple proofs and EPF-CLPA is pairing-free, the auditing performance is greatly improved. Additionally, our scheme also supports user revocation. During User revocation, the GM will not generate the time key for the revoked user. Without an updated time key, any user cannot generate valid signatures for data blocks.

## 1.6 Thesis Organization

The rest of the chapters of this thesis are organized as follows and given in Fig. 1.2.

- Chapter 2 describes the preliminaries such as digital signatures, identity based signatures, attribute based signatures, certificatelsess signatures, and some mathematical background including bilinear pairings, security assumptions, elliptic curve cryptography, threshold secret sharing etc.

- Chapter 3 reviews the remote data auditing protocols and describe about personal data auditing schemes and shared data auditing schemes.

- Chapter 4 describes Identity-based public integrity auditing mechanism for shared data using ID-Based Cryptography (IBC).

- Chapter 5 presents the Attribute-Based public auditing mechanism for shared data using Attribute based Cryptography (ABC).

- Chapter 6 details Multi-Replica Public Integrity Auditing Scheme for Dynamic Shared Data in Cloud Storage that relies on the use of Certificateless(CL) Cryptography.

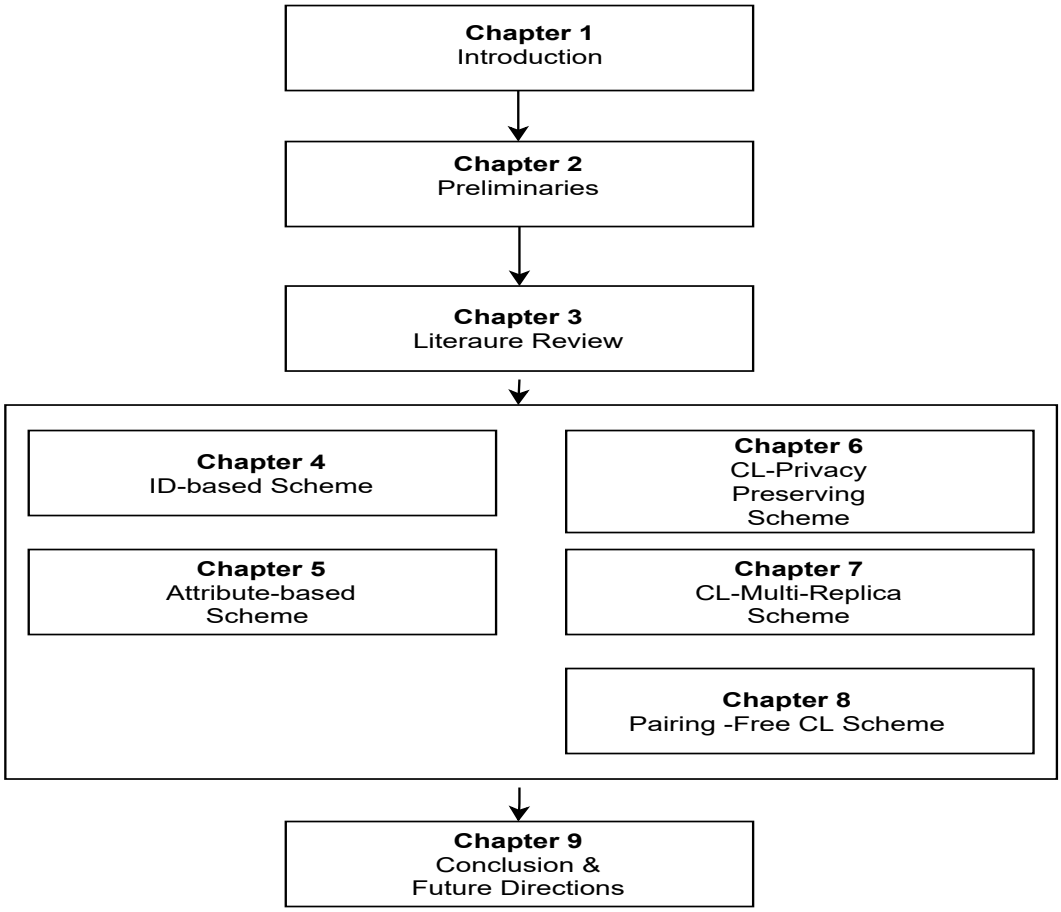



Fig. 1.2. Thesis organization

- Chapter 7 presents public auditing scheme for shared data to achieve the data privacy and support the data dynamics of shared data in Cloud Storage using Certificateless cryptography.
- Chapter 8 presents public auditing scheme for Shared Big Data in cloud storage designed based on ECC and without employing the bilinear pairings.
- The conclusions of the thesis and future directions are outlined in Chapter 9.

# Chapter 2

# Preliminaries

## 2.1 Introduction

This chapter discusses the background technologies used in our thesis.

## 2.2 Digital Signature

A digital signature scheme consists of the following three algorithms:

- KeyGen: It takes a security parameter $1^n$ as input and outputs outputs a private and public key pair $(pk, sk)$.
- SignGen: It takes private key $sk$ and a message $m$ as inputs and outputs a signature $\sigma$.
- Verify: It takes public key $pk$, a message $m$, and a signature $\sigma$ as inputs and returns $1$ for accept or $0$ for reject.

A digital signature provides the security services including message authentication, message integrity, and nonrepudiation. for message confidentiality we still need encryption/decryption.

In this this thesis, we use the following three types of signatures. 1. Identity-Based signatures, 2. Attribute Based signatures and 3. Certificateless signatures.

### 2.2.1 Identity-Based signatures(IBS)

An identity-based signature (IBS) [46] scheme includes the four algorithms namely Setup, KeyExtract, Sign and Verify.

- Setup: It generates key pair consists of master public key and secret key
- KeyExtract: It generates the secret key for the group user with identity id .
- Sign: It generates a signature for data.
- Verify: It verifies the signature for correctness.

### 2.2.2 Attribute-Based Signature (ABS)

An ABS [47] allows a user to sign the data over a given attribute set. It has four algorithms:

- Setup: It generates the master secret key ($msk$) and public key pair based on security parameter.
- Keygen: On input of msk and the attribute set, it generates a private key.
- Sign: It generates a signature for the given message using a private key.
- Verify: It takes the public key, data, access policy, and signature as input and returns 0 or 1.

### 2.2.3 Certificateless Signatures (CLS)

A CLS [48] allows a user to sign the given data. It has seven algorithms.

1. Setup: It generates system parameters $params$ and master secret key $msk$ using a security parameter.
2. Partial-Private-Key-Extract: It takes $params$, master secret key, and user's ID as inputs and returns a partial private key $sk_{ID}$ for the user.
3. Set-Secret-Value: It takes $params$ and user's ID as inputs and outputs the user's secret value $y_{ID}$.

4. Set-Private-Key: It takes $params$, user's ID, $sk_{ID}$, and secret value $y_{ID}$ as inputs and outputs private key $S_{ID}$.
5. Set-Public-Key: It takes $params$ and secret value $y_{ID}$ as inputs and outputs public key $PK_{ID}$.
6. Sign: It takes as inputs $params$, a message M to be signed and a private key $S_{ID}$. It outputs a signature $\sigma$.
7. Verify: It takes as inputs $params$, a message $M$, the ID and public key $PK_{ID}$ of user, and $\sigma$ as the signature to be verified. It returns 0 or 1.

## 2.3 Mathematical Background

### 2.3.1 Bilinear Pairing

"Assume $G_1$ and $G_T$ are the two cyclic groups of the same prime order $q$. A map $e : G_1 \times G_1 \rightarrow G_T$ is said to be a bilinear pairing if the following three conditions hold [49]:
**Bilinear:** $e(u^a, v^b) = e(u, v)^{ab}$, $\forall u, v \in G_1$ and $\forall\ a, b \in Z_p^*$.
**Computational:** $e(u, v)$ can be efficiently computable $\forall\ u, v \in G_1$.
**Non-degenerate:** $e(g, g) \neq 1$. "

### 2.3.2 Security assumptions

A. Computational Diffie-Hellman (CDH) Assumption [50]. "For any probabilistic polynomial time adversary $A_{CDH}$, the advantage of adversary $A_{CDH}$ on solving the CDH problem in $G_1$ is negligible, which is defined as

$$Pr[A_{CDH}(g, g^a, g^b) = (g^{ab}) : a, b \xleftarrow{R} Z_p^*] \leqslant \epsilon$$

where $\epsilon$ denotes a negligible value ".

B. Discrete Logarithm (DL) Assumption [51]. "For any probabilistic polynomial time adversary $A_{DL}$, the advantage of adversary $A_{DL}$ on solving the DL problem in $G_1$ is neg-

ligible, which is defined as

$$Pr[A_{DL}(g, g^a) = a : a \xleftarrow{R} Z_p^*] \leqslant \epsilon.$$

".

### 2.3.3 Elliptic Curve Cryptography

Let $E$ be any curve symmetric around x-axis that satisfies polynomial equation $y^2 = x^3 + ax + b$ over a finite field $F_p$ of prime integers such that $p \geq 3$ and $p$ is any set of large primes [52]. Where $a$, $b$ are any constants with $\Delta = 4a^3 + 27b^2 \neq 0$ along with infinite point $O$ for all set of point $P = (x, y)$. On performing point addition $R = P + Q$ (according chord-and-tangent rule) forms a cyclic group $G$ over the elliptic curve $E$. Therefore, we define the scalar multiplication as $t \cdot P = P + P + ... + P$ (t times).

### 2.3.4 Threshold Secret Sharing Scheme

"Threshold secret sharing scheme [51] enables to divide a secret $s$ into $\mathfrak{n}$ shares and distributes to $\mathfrak{n}$ players, say, $P_1, P_2...P_n$. With $k$ or more shares, one can reconstruct the secret; It utilizes a unique $k-1$ degree polynomial $f(x)$ where $f(x) \in Z_p$ and $f(0) = s$. When reconstructing the secret $s$, a set $\mathcal{S}$ of $k$ shares are chosen to recover $f(0)$ as follows $f(0) = \sum_{P_i \in \mathcal{S}} \Delta_{x_i,s}(x) s_i$ where $\Delta_{x_i,s}(x) = \prod_{P_i \in \mathcal{S}, j \neq i} \frac{x - x_j}{x_i - x_j}$ denotes the Lagrange coefficient."

### 2.3.5 Replica Version Table (RVT)

To support dynamic data operations at block level, many existing scheme [53, 54] used authenticated data structures such as IHT [53], MHT [53, 54] or skip lists [55]. However, it would create heavy computational costs and high communication cost during the updating and verification processes and does not support shared multi-replica data dynamics. In this paper, we propose RVT to support shared multi-replica data dynamics which is designed based on IHT [53] to store the different versions of the recently updated multi replica file

Table 2.1: RVT (for instance $n = 8$ and $j = 1(1 \leq j \leq c)$)

| SN | $BRN_{(1\leq i\leq n, 1\leq j\leq c)}$ | BVN | $U_{ID}$ |
|---|---|---|---|
| 1 | $b_{11}$ | 1 | 1 |
| 2 | $b_{21}$ | 1 | 1 |
| 3 | $b_{31}$ | 1 | 1 |
| 4 | $b_{41}$ | 1 | 1 |
| 5 | $b_{51}$ | 1 | 1 |
| 6 | $b_{61}$ | 1 | 1 |
| 7 | $b_{71}$ | 1 | 1 |
| 8 | $b_{81}$ | 1 | 1 |

blocks and stored at the verifier side. RVT is a table like authenticated data structure, which can be accessed or modified by the users of the group and maintained by the TPA. The structure of the RVT is shown in Table 2.1. It consists of four columns: Serial Number ($SN$), Block Replica Number ($BRN$), Block Version Number ($BVN$), and User Identity ($U_{ID}$). The $SN$ is used as a counter, which can be treated as the physical location of the block, and the Block $BRN$ is used to identify the specified block of a particular replica. The relation between $SN$ and $BRN$ can be viewed as a mapping between physical position and logical number of the replica file blocks. The $U_{ID}$ is used to identify the signer of the block. When a file of blocks is initially created the $BVN$ of all blocks is set to 1. If any of the block is updated, then its version number is incremented by 1. TPA keeps only one table irrespective of the number of replicated files that CSP stores.

# Chapter 3

# Review of Remote Data Integrity Auditing in Cloud

## 3.1 Introduction

In this chapter, an extensive review of remote data integrity auditing techniques in the cloud computing and a brief survey of the literature related to the contributions made in this thesis is given. Remote data integrity auditing (RDIA) is a technique to verify the integrity of outsourced data without retrieving and without having knowledge of the complete data. The remote data auditing involves the entities such as (1) DO, who has huge data to be stored in the cloud, can be an individual or an enterprise. (2) CSP, who provides the storage space for the user to store data and manage the cloud with large amount of resources and (3) TPA, who has expertise and capabilities that the user does not have, and audits on behalf of DO, as depicted in Fig 3.1. [1] In the process of remote data auditing, first, the DO pre-processes the data file before it is uploaded to the cloud. After uploading the file to cloud, the local copy of the file is deleted. Later, verifier (DO or verifier) sends a challenge specifying the blocks for which the CSP need to provide proof. Upon receiving the particular challenge, the CSP generates proof for the challenge and sends a proof as a response by executing the challenge-response protocol. Then, the verifier checks whether the data is intact or not in the cloud by comparing the local metadata with proof sent by the server. If both are matched, it indicates the integrity is maintained in cloud storage, otherwise, there is a data

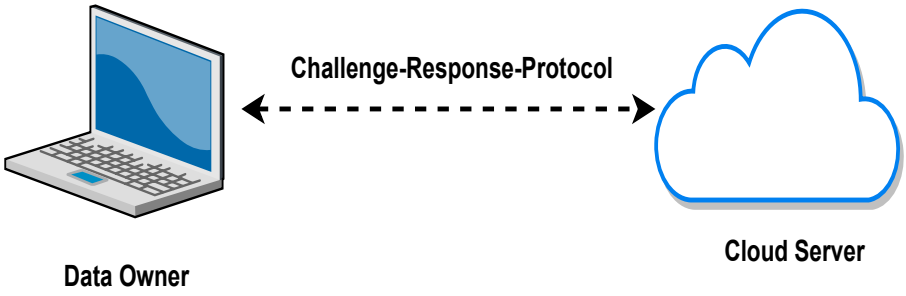



Fig. 3.1. Architecture of RDIA

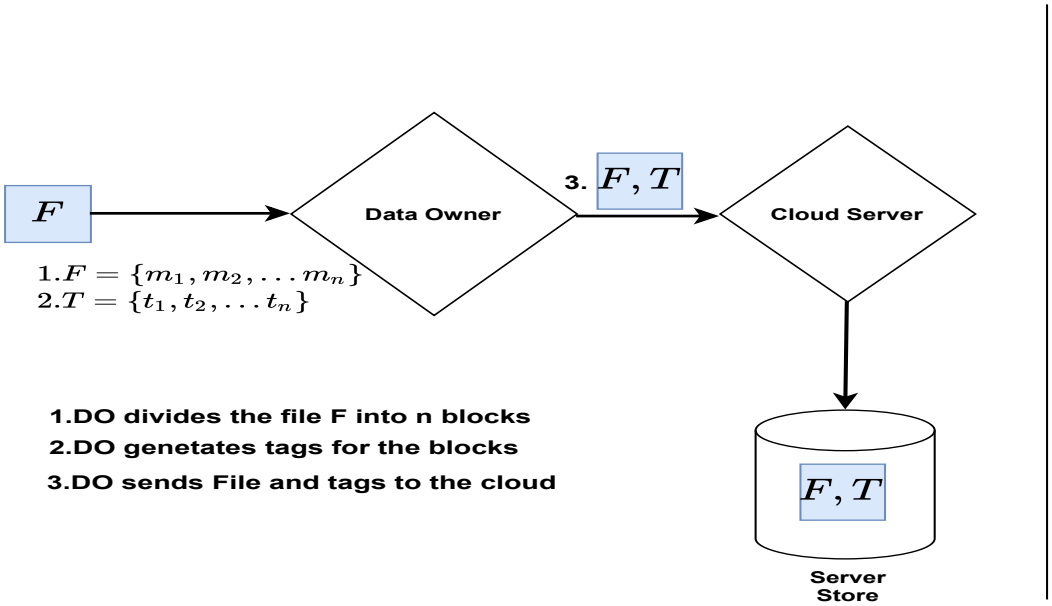



Fig. 3.2. PDP: setup phase

corruption or data alteration.

Based on PDP, in recent years, several RDIA schemes [1, 2, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 47, 53, 54, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66] have been proposed for catering to different needs of the integrity verification of cloud users' data, under different security and system models in the literature. We have broadly classified these schemes into two types based on their approach and nature of the data, such as **personal data auditing schemes and shared data auditing** schemes. Further, each of them are categorized into static and dynamic.

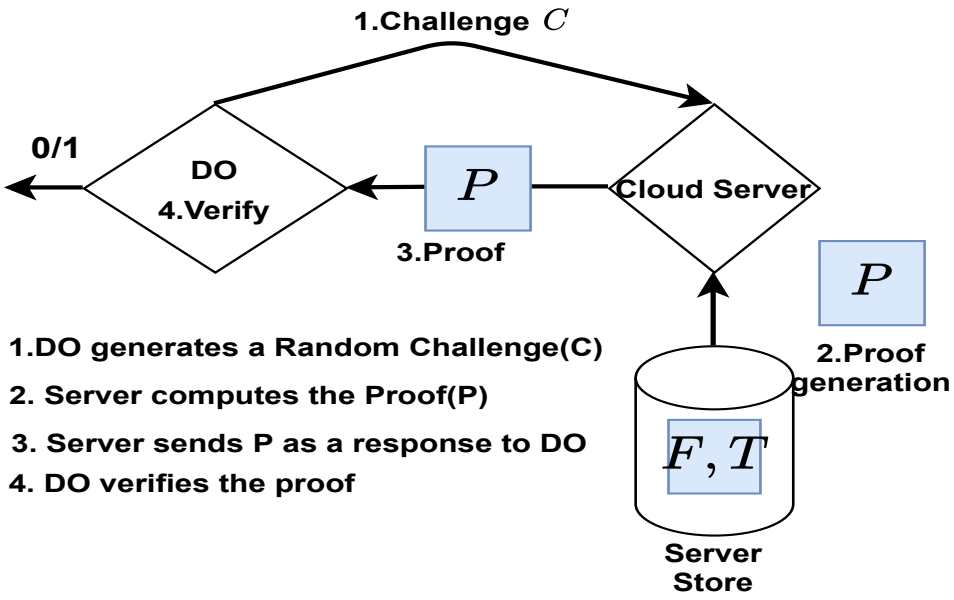



Fig. 3.3. PDP: verification phase

## 3.2 Personal data auditing schemes

To check the integrity of personal data in cloud storage, in 2007, Ateniese et al. [1] first proposed a PDP scheme. They utilized RSA-based homomorphic verifiable tags and sampling method to verify the integrity of data stored in the cloud. In this scheme, the DO divides the file $F$ into several blocks $F = \{f_1, f_2, \ldots . f_n\}$ and constructs metadata for every block before it is outsourced to untrusted CSP. After uploading the file $F$ and corresponding tags, the DO deletes the original file at their local store. Later, the integrity of the data is verified through challenge-response protocol. This technique gives a high probabilistic guarantee of data possession, where the auditor verifies a set of file blocks randomly with every challenge, as shown in the Fig.3.2 and Fig.3.3.

### 3.2.1 Static personal data auditing schemes

Based on PDP, many schemes [58, 59, 31] received research attention in the early days of cloud computing era. They focus on static data auditing, in which DO cannot modify the data online. These schemes mainly focus on checking the integrity of large repositories which does not change over time like libraries, archives, medical, and scientific data.

Curtmola et al. [58] proposed a multiple-replica PDP (MR-PDP) scheme to increase the reliability of data stored on remotely located servers, where the data owner can check whether a storage service provider stores single or multiple copies of a file. The additional benefit of MR-PDP is that it can generate additional copies on demand, when few of the current copies fail, at small expense. However, it suffers from huge tags, computation overhead on both the verifier and server side. Similarly, Wang et al. [59] presented a proxy PDP (PPDP) method by using the bilinear maps in which a remote data auditing performed by the proxy on behalf of the client according to a warrant. However, it does not support data dynamics; Similarly, Chen [31] used algebraic signatures to check data integrity to reduce computation overhead than using homomorphic cryptosystems. The main disadvantage is that it does not support dynamic data operations. All static PDP schemes [58, 59, 31] can be used well for verifying the integrity of the data with a high probabilistic guarantee, but they do not provide any support for remote data updates.

### 3.2.2 Dynamic personal data auditing schemes

DOs are subject to update their data at regular intervals for various application purposes. To support dynamic operations such as insertion, modification or deletion, Ateniese et al. [56] designed a scalable PDP (S-PDP) scheme, which is based on symmetric key cryptography. It improves the PDP [1] in terms of storage, computation and communication cost. S-PDP uses an authenticated data structure to support dynamic data operations, such as fragment changing, remove and append. However, this scheme fails to handle unlimited number of queries and insertion of blocks. Later, Erway et al. [10] described a dynamic PDP scheme, which extends the scheme [1] by introducing skip lists. This scheme can support full dynamic operations on outsourced data. This scheme is first in line to discover constructions for dynamic provable data possession. However, the performance of the scheme remains uncertain.

Wang et al. [32] combine Boneh–Lynn–Shacham (BLS) based block authentication signature [67] and the Merkle Hash Tree (MHT) to provide both public verifiability and

data dynamic operations along with integrity. In this method, they divide the given file $F$ into $n$ number of blocks and then constructed tags by applying hash function on each block. These hash values are stored at the leaf nodes of the MHT and in turn generates the root. However, it does not support privacy preserving of data against TPA. Similarly, Zhu et al. [16] proposed a dynamic auditing service for checking the integrity of remotely stored data in CSP servers. They utilized the fragment structure to reduce the storage of signatures, utilized index hash tables (IHT) to support dynamic operation. It supports batch auditing of many blocks from the same file but fails to provide privacy preserving against TPA. In [25], Zhu et al. proposed a cooperative PDP protocol in multi-cloud environments. Li et al. [68, 69] proposed a scheme in which a user can delegate TPA to execute high computing process to solve the user's bottleneck before the client outsources the data to CS. Li et al. [69] extended [68] to improve the tag generation. Liu et al. [12] thought that previous studies are not efficient with respect to data updates because it is a fixed-size block update. Therefore, they designed a scheme to support variable-size blocks to improve efficiency. Liu et al. [54] designed a scheme for data availability in the cloud. Consequently, the CS will store multiple replicas to enhance data availability. However, when the stored data is frequently updated, each dynamic update will affect every replica. Therefore, they proposed Later, some authors proposed dynamic PDP schemes by utilizing MHT [12, 13, 14, 15] and using Indexed Hash Table (IHT) [15, 16, 17, 18, 19]. With more data, it brings new challenges in data integrity.

However, all these schemes does not support privacy of data against verifiers. To preserve the privacy, Hao et al. [21] proposed a protocol for privacy-preserving based on RSA cryptosystem by making use of Seb´e et al. [60] protocol, to support public auditing of the data. However, it is infeasible if the size of the file is large and not achieved the anticipated goal of keeping data hidden from the third party. Similarly, [20] proposed a privacy preserving auditing scheme by employing random masking technique. They developed a method using HLA and random masking technology in such a way that TPA gets zero knowledge about the data stored in server during every auditing even though TPA may perform multiple auditing tasks simultaneously. However, it incurs heavy computation and communication overhead. Syam and Subramanian [19] described an integrity checking protocol

based on RSA, supports public auditing and data dynamics using Sobol Sequence. It gives the probabilistic guarantee of integrity. Its performance is better than the schemes using pseudorandom sequence [70]. Similarly, Barsoum and Hasan [71], presented a scheme to outsource the critical data to CSP and introduced indirect mutual trust between DO and CSP, i.e., trust for data storage in the cloud. Their solution not only provides secrecy, consistency, and authorisation for data but also maintains various versions of data for better security and integrity. Subsequently, they presented a map-based [53] provable multi-copy dynamic data possession (MB-PMDDP) to prevent CSP from cheating the client. However, it fails in finding the corrupted copy of the file. [17] proposed a protocol for storage auditing to protect data privacy against TPA using cryptosystems, bilinear pairing, and HVT. It supports dynamic data operations with less communication overhead between the server and the auditor and lowers the computing cost of the auditor. This protocol is further extended to support batch auditing for both multiple data owner's settings and multiple cloud settings with a proxy to significantly improve the verification performance, especially in distributed cloud storage systems. However, it incurs more storage space overhead on the server because of a large number of tags and the auditor because of increased size as well as increased count of files.

Liu et al. [12] introduced a technique for fine-grained updates and ensured fewer communication overheads for big data applications such as social media and business transactions. It works better for a single user and does not support shared data among users. Later in 2015, Liu et al. [54] presented a new external auditing method named MuR-DPA with a new data structure based on the MHT. It allows verification of several replica updates at the same time. However, it supports only small updates and works fine only with the single client.

Cloud storage service not only allows user to store data but also allows share information with other users in a group. However, all of the aforementioned schemes [1, 2, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 56, 57] deal only with the integrity verification of non shared data (personal data which is possessed by only a single user) which are not suitable for shared data auditing. That is these schemes do not consider the advantage of cloud storage where a user can share data with other users

Table 3.1: Summary of some personal data auditing schemes

| Schemes | Type of Guarantee | Integrity | Availability | PA | DD | PP | BA |
|---|---|---|---|---|---|---|---|
| Ateniese et al. [1] | Prob | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| Erway et al. [10] | Prob | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ |
| Wang et al. [11] | Prob | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ |
| Liu et al. [12] | Prob | ✓ | ✗ | ✓ | ✓ | ✗ | ✗ |
| Barsoum et al. [15] | Prob | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ |
| Zhu et al. [16] | Prob | ✓ | ✗ | ✓ | ✓ | ✓ | ✗ |
| Yang et al. [17] | Prob | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ |
| Syam et al. [19] | Prob | ✓ | ✗ | ✓ | ✓ | ✗ | ✗ |
| Hao et al. [21] | Prob | ✓ | ✗ | ✓ | ✓ | ✓ | ✗ |
| Chen et al. [31] | Prob/Det. | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| Wang et al. [35] | Prob | ✓ | ✗ | ✓ | ✓ | ✓ | ✗ |
| Jiang et al. [38] | Prob | ✓ | ✗ | ✓ | ✓ | ✓ | ✗ |
| Ateniese et al. [56] | Prob | ✓ | ✗ | ✗ | partial | ✗ | ✗ |
| Curtmola et al. [58] | Prob | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ |
| Wang et al. [72] | Prob | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ |

Note:PA: Public Auditing; DD:Data Dynamics; PP: Privacy preserving; BA:Batch Auditing; Prob: Probabilistic; Det: Deterministic;

on cloud.

## 3.3 Shared data auditing schemes

With the increasing demands of collaborative works in cloud, shared data auditing has become an important topic [35, 36, 38]. In shared data scenario, one of the users in a group creates and uploads data to the cloud, and the rest of the group members not only access but also modify the data. Shared data auditing schemes can be further classified into static and dynamic schemes.

### 3.3.1 Static shared data auditing schemes

To ensure outsourced shared data integrity, firstly, Wang et al. [34], proposed a scheme named Oruta with the help of ring signatures. This scheme increases complexity if the size of the group increases; Therefore, it is not suitable for big groups. To address this issue, in their subsequent work, they proposed Knox [33] using a group signature to support group dynamics. However, this scheme costs a huge amount of computational resources. Later,

they designed a scheme named Panda [35] for shared data supporting user revocation by using a proxy re-signatures technique. However, it has a security flaw since the resigning keys of users are stored with proxy in advance, which causes collusion attacks from revoked users or CSP. To address this problem, Yuan and Yu [36] has proposed a scheme by employing polynomial tags. However, it suffers from replay and replace attacks. To address these attacks, Luo et al. [37] designed a scheme for shared data to support user revocation by employing the concept of secret sharing. However, both [36, 37] schemes does not study privacy, which is an important property for public integrity auditing. Later, Jiang et al. [38] adopted the vector commitment and employed the group signatures to support user revocation. However, the scheme inefficient because of costly auditing operations. Fu et al. [39] proposed a scheme by combining a homomorphic verifiable group signature with a secure sharing technique to address identity privacy. Moving forward, Wu et al. [40] proposed a scheme by employing group signatures. It provides user identity privacy and data privacy by employing the random masking technique.

### 3.3.2 Dynamic shared data auditing schemes

To support dynamic operations along with integrity and user revocation, Wang et al. [35] designed a scheme named Panda [35] by utilizing a proxy re-signatures [73] technique which is a good choice a cut down the computation overhead for user during resigning.

PS fetches the signatures from the cloud and transforms Meanwhile, the PS cannot learn any private keys of the two users, which means it cannot sign any block on behalf of either revoked or non-revoked user.

When a user is revoked, the CS will transform revoked user's signature on a message $m$ into a non-revoked user's signature signature on $m$ whenever a signed user revoked from the group with a re-signing key. However, it has a security flaw since the resigning keys of users are stored with proxy in advance in order to generate resignation keys of new signatures, which causes collusion attacks from revoked users or CSP. Recently, Tian et al. [41] proposed a scheme to achieve data privacy and extended the DHT to support data dynamics.

Although several public auditing schemes for shared data have been proposed, none of them could realize all the security and performance requirements such as user revocation, privacy preserving, identity privacy, data dynamics. Moreover, most of these schemes [33, 34, 35, 36, 37, 38, 39, 40, 41] suffer from the complex certificate management because they rely on traditional PKI. PKI is widely used in several fields. However, the PKI-based scheme must deal with various complex certificate management activities including certificate generation, storage, delivery, renewing, and revocation. Furthermore, the security of PKI cannot be guaranteed completely, particularly, when the CA is intruded or controlled by a malicious hacker The most commonly adopted digital certificate in our daily life is X.509 certificates, an ITU-T standard for a PKI and privilege management infrastructure.

To simplify certificate management, identity based (ID-based) schemes for shared data [42, 43, 44] auditing have been presented using identity based cryptography [74]. In these schemes, signatures are generated based on the identity of the user. These protocols simplify certificate management by binding the user's identity with the secret key. Yang et al. [42] described a scheme using blind signatures to achieve identity traceability. Zhang et al. [43] presented remote auditing of big data with user revocation by employing ID-based signatures. In this scheme, they update the private keys of existing authorized users instead of updating the signatures of revoked users during revocation. However, the limitation of its application is that all users in the group have the same private key and the same public key. Similarly, [44] proposed a mechanism with sensitive information hiding by employing ID-based signatures. In this scheme, a special file called a disinfectant file is created to hide signature information of sensitive blocks of the file.

However, all these schemes [42, 43, 44] inevitably suffers from "key escrow problem "because the full private key of the user is generated by a third entity called the private key generator (PKG). As PKG knows each user's private keys, it can easily masquerade any client to sign the message by itself. This is a serious security gap (issue) existing in these ID-based [42, 43, 44] schemes.

To simplify the certificate management and mitigate key escrow problems existing in various shared data schemes [34, 35, 38, 39, 40, 41, 42, 43, 44, 75, 76] simultaneously, CL-PKC schemes [62, 63] have been proposed based on CL-PKC. Li et al. [62] initially,

designed a CLPDP scheme for shared data, which can support user revocation. But it does not consider the data privacy against verifier and data dynamics. That means, the user's data may be leaked to verifier during auditing. This weakness will affect the security of the data in cloud environment. To support privacy preserving, Yang et al. [63] designed a privacy-preserving CLPDP scheme by employing the technique of zero-knowledge proof and randomization method. However, these schemes [62, 63] are based on expensive bi-linear pairings, which incur heavy computation overhead and are not applicable for shared big data auditing.

Table 3.2: Summary of shared data data auditing schemes

| Schemes | Integrity | PA | DD | PP | BA | Availability | UR | CF | KE |
|---|---|---|---|---|---|---|---|---|---|
| Wang et al.[34] | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| Wang et al. [33] | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| Wang et al. [35] | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ |
| Luo et al.[37] | ✓ | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ |
| Jiang et al. [38] | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ |
| Fu et al. [39] | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ |
| Wu et al. [40] | ✓ | ✓ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ |
| Tian et al. [41] | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ |
| Yang et al. [42] | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ |
| Shen et al. [44] | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ |
| Li et al. [62] | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ |
| Yang et al. [63] | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ |
| Yuan & Yu. [76] | ✓ | ✓ | ✗ | ✗ | partial | ✗ | ✗ | ✗ | ✗ |
| Zhang et al. [77] | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ |

Note:PA: Public Auditing; DD:Data Dynamics; PP: Privacy preserving; BA:Batch Auditing;KE:Key Escrow Problem; CF:Certificate Freeness;UR:User Revocation;

## 3.4 Summary

In this chapter, we reviewed RDIA schemes in cloud storage and we classified RDIA schemes into personal data auditing schemes and shared data auditing schemes according to their approach and nature. From our review, we observed that the most of the existing schemes focused on auditing the integrity of personal data which are not suitable for shared data auditing. When data is shared among multiple group users in group, some new chal-

lenges will arise like secure and efficient user revocation, efficient data dynamics, privacy preserving, availability of shared data. To address these issues, efficient and secure RDIA schemes are designed in the subsequent chapters.

# Chapter 4

# Identity-based Public Integrity Auditing for Shared Data(IDPIA)

## 4.1 Introduction

In this chapter, we present the first contribution of the thesis: Identity-based (ID-based) public integrity auditing scheme to ensure the integrity of shared data with user revocation which is a significant issue in cloud storage. To ensure secure and efficient user revocation while avoiding complex certificate management in existing schemes [33, 34, 35, 36, 37, 38, 39, 40, 41], we propose an ID-based public integrity scheme for shared data with secure user revocation in cloud. The main contributions of this chapter are as follows:

- In this contribution, ID-based signatures are employed to generate signatures of file blocks that can simplify certificate management problem.
- Proxy re-signatures are used to support group user revocation. That means, whenever a user quits the group or misbehaves, proxy can carry out the resigning process.
- The security analysis proves the correctness, soundness of IDPIA based on DL assumptions in ROM.
- The performance analysis evaluates performance of IDPIA theoretically and experimentally in terms of computation overheads.

## 4.2 Problem Statement

Here, we the present problem statement, its description followed by the architecture, adversary model, design goals, and the security model of the scheme

To guarantee the outsourced data integrity in the cloud, a number of protocols have been proposed based on various methods [9-21]. In [9-17] schemes, data owner (DO) uploads the data file along with the signatures of file to the cloud and enables third-party verifier to validate the integrity of shared data through the execution of the challenge-response protocol. However, these schemes [9-17] merely focused on personal data auditing. The schemes [18-21] proposed to validate the integrity of the shared data. However, these schemes [18-21] increase the key management problems such as key storage, key exchange for resigning process and also cause collusion attacks in the system. Therefore, it is an open challenge to design a state of the art technique for shared data auditing to address above mentioned problems.

To ensure secure and efficient user revocation while avoiding complex certificate management in existing schemes [33, 34, 35, 36, 37, 38, 39, 40, 41], we propose an ID-based public integrity scheme for shared data with secure user revocation in cloud. In the proposed scheme, ID-based signatures are employed to generate signatures of file blocks that can simplify certificate management problem. Proxy re-signatures are used to support group user revocation. That means, whenever a user quits the group or misbehaves, proxy can carry out the resigning process.

### 4.2.1 Architecture

An Identity-based cloud storage architecture considered in this chapter consists of five entities as illustrated in Fig. 4.1.

1. PKG, who generates key pair for users. PKG calculates the private key by using user's identity, and forwards it to the user through a secure channel.

2. Users who store, share, trust and cooperate with each other in the group, which can be an enterprise or an individual.

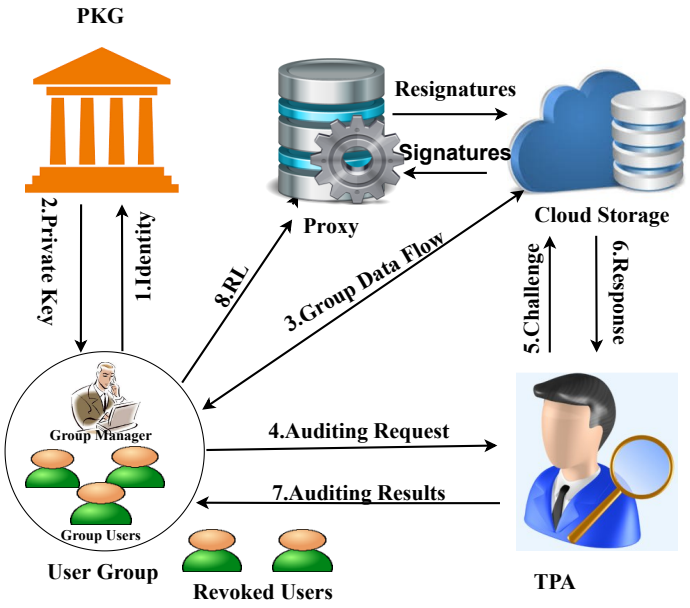



Fig. 4.1. System architecture of IDPIA scheme

3. Cloud server provides the storage space and manages the data in the cloud.

4. A proxy server is a semi-trusted cloud server, which performs resigning on revoked user blocks by utilizing proxy re-signatures resigning over the blocks which were signed by revoked user on behalf of the group.

5. The TPA audits the data upon receiving the auditing request from the user.

### 4.2.2 Overview of IDPIA

In IDPIA, initially a user submits a request to the GM to join the group. According to the request of user, GM generates a group key and securely sends to the user. Then the user request for the private key from PKG. The PKG authenticates the user, generates private key and secretly sends to the user. Upon receiving the private key, the user generate signatures for file data blocks using private key. After signing data blocks, uploads data blocks along with corresponding signatures to the cloud and deletes them from the local site to save the space. Later, to audit the shared data, TPA challenges the cloud by selecting blocks randomly. After receiving this challenge, the cloud returns the proof of shared data as a response to the TPA. After

receiving proof from the cloud, TPA verifies the correctness of data. Whenever a user in the group misbehaves or quits the group, GM updates the existing $RL$ and forwards to the Proxy. Upon receiving the updated $RL$ the proxy performs resigning on revoked user blocks by utilizing proxy re-signatures.

### 4.2.3 Adversary Model

In adversarial model, we consider the two types of adversaries: Internal adversaries, external adversaries.

**Internal adversaries:** Malicious insiders, who are cloud servers or users. They intentionally delete or modify the user's data in the cloud. Sometimes, they may try to translate signatures of one user into another user to cut down the operation costs or hide data loss to build their reputation.

**External Adversaries:** Who may try to avert the users from accessing the shared data by destroying/altering the data in the cloud.

### 4.2.4 Design Goals

We design our scheme to achieve following goals:

1. *Correctness:* The cloud passes the integrity check if every challenged block and its corresponding signatures are appropriately maintained in the cloud.

2. *Soundness:* A malicious cloud server server cannot generate a valid response if the data is replaced or modified.

3. *Public Auditing:* Anyone with public parameters can verify the integrity of group shared data having no knowledge of the data.

4. *Efficient User Revocation:* Proxy server re-signs the blocks efficiently during the revocation.

### 4.2.5 Security Model

We designed IDPIA scheme to withstand the the adversary, namely $\mathcal{A}_1$(represents malicious cloud). $\mathcal{A}_1$ tries to generate the forged integrity proof. We prove the security of IDPIA by considering the adversary $\mathcal{A}_1$ and we define the following Game 1 against $\mathcal{A}_1$.
*Game 1 (played by $\mathcal{A}_1$ and a challenger $\mathcal{B}$)*:

- Setup: $\mathcal{B}$ runs the Setup algorithm to generate the system parameters $PP$ and the master secret key $MK$. $\mathcal{B}$ sends $PP$ to $\mathcal{A}_1$ and keeps the $MK$ secret.
- SignGen_Query: $\mathcal{A}_1$ chooses the tuple $(ID, m)$ and forwards it to $\mathcal{B}$ for querying the signature. $\mathcal{B}$ generates and returns the signature of $m$ to $\mathcal{A}_1$ by running the SignGen algorithm.
- Challenge: $\mathcal{B}$ generates a challenge message Chal and sends it to $\mathcal{A}_1$ to get the corresponding proof P.
- Forge: Finally, for the Chal, $\mathcal{A}_1$ outputs a data integrity proof P and sends it to $\mathcal{B}$. $\mathcal{A}_1$ wins the game if P can pass the integrity check and the blocks in P is incorrect.

## 4.3 Algorithmic Framework

In this section, we give the syntax of the proposed identity-based RDIC scheme includes nine algorithms as follows:

- **Setup** $(1^\lambda) \rightarrow (PP, MK)$ : It takes security parameter $\lambda$ as input and outputs the public parameters PP and the master secret key MK.
- **KeyExtract** $(MK, ID) \rightarrow SK_{ID}$: It takes MK and a group user's identity $ID \in \{0,1\}^*$ as input and outputs the secret key $SK_{ID}$ that corresponds to the identity $ID$.
- **Sign** $(m, SK_{ID}) \rightarrow \sigma_i$ : It takes the data $m_i \in \{0,1\}^*$, and secret key $SK_{ID}$ as inputs and return signature $\sigma_i$ as output which will be stored in the cloud along with the file $F = \{m_1, m_2, ...m_n\}$.

- **Challenge** $(PP, F_n, ID) \rightarrow chal$ : It takes the public parameters $PP$, a unique file name $F_n$ and user's identity $ID$ as input, outputs a challenge message $chal$.

- **Proof** $(chal, \sigma, m) \rightarrow P$: It takes chal, signatures, and data as input and to generate a data proof P as output.

- **Verify** $(ID, chal, P) \rightarrow 0, 1$: It takes the pp, ID, chal, P as input and outputs 0 or 1.

- **Revoke** $(RL, id1, id2 \ldots idk) \rightarrow RL'$ : It takes RL, $\{id1, id2 \ldots idk\}$ as input and outputs $RL'$ .

- **Rekey** $(SK_{ID}, MK) \rightarrow SK_{rk}$: It takes $SK_{ID}, MK$ as input and generates a ReKey $SK_{rk}$ as output.

- **ReSign** $(\sigma, SK_{rk}) \rightarrow \sigma'$: It takes signatures, and re-key as input and produce re-signatures as output.

## 4.4 Detailed Construction

Now, we give the construction of proposed algorithms, which are defined in section 4.3 as follows:

- **Setup** In this algorithm, PKG generates master system secret key MK and public parameters PP as follows: Let $G_1$ and $G_2$ be two multiplicative cyclic groups of prime order $p$ and $g$ be a generators $G_2$. PKG randomly picks an integer $\alpha \xleftarrow{R} z_p^*$ as the master secret key and generates a public key $P_k = g^a$. Let PKG choose two cryptographic collision resistant hash functions as $H_1, H_2 : \{0, 1\}^* \rightarrow G$. System public parameters PP: $\{G_1, G_2, G_T, e, g, P, H_1, H_2\}$ published to all the users in the system. Here $MK = \alpha$, $Pk = g^\alpha$ . Additionally, KGC also generates an initial revocation list $RL = \phi$ .

- **KeyExtract** It takes $MK$ and $ID$ of a user as input, and generates the private key

$sk_i$ as output.

$$sk_i = (\, Q_i)^{\,\alpha}$$

where $Q^i = H_1(ID_i)$

- **Sign**: Given file $F =(\, m_1, m_2, ....m_n)$ named fid, the DO picks a random group element $u \xleftarrow{R} G_1$. Then computes the signature $\sigma_i$ for every block $m_i$ $(i = 1, 2, 3....n)$ as

$$\sigma_i = (\, sk_i.H_2(\, fid \| i)\, u^{m_i})^{\,\alpha} \tag{4.1}$$

- **Challenge**: To check the integrity of the file, TPA sends the challenge message $chal = \{i, v_i\}_{s_1 \le i \le s_c}$ to the cloud.

- **Proof**: After receiving the challenge $chal = \{i, v_i\}_{s_1 \le i \le s_c}$ from the TPA, the server computes $\mu = \sum_{i=s_1}^{s_c} v_i m_i$; $\sigma = \prod_{i=s_1}^{s_c} \sigma_i^{v_i} \in G_1$ where both signature blocks and data blocks are aggregated into a single block. Then the server sends proof $P = \{\mu, \sigma\}$ as a response.

- **Verify**: Once the proof $P$ is received from the server, the TPA verify

$$e(\sigma, g) \stackrel{?}{=} e(\prod_{i=S_1}^{S_c} s_{ki}^{v_i}, H_2(fid||i)) \cdot u^{\mu}, P_k) \tag{4.2}$$

  If this equation holds, data is intact otherwise corrupted

- **Revoke**: If a user $U_i$ in the group set to be revoked, KGC updates the revocation list and sends a copy to the proxy for resigning process. $RL' \leftarrow (RL \cup \{id_i\})$.

- **Rekey**: KGC generates a resigning key $SK_rk$ as follows. KGC picks a random number $rk_1 \xleftarrow{R} Z_p$ and sends it to nonrevoked user $U_i$.User $U_i$ calculates $rk_1/sk_1$ and sends back to the KGC. Then KGC takes the rekey as $SK_{rk} = rk_1/sk_i$ and forwards the rekey to the proxy.

- **ReSign**: Upon receiving the $RL'$ and rekey, proxy downloads the previous signature of the all the blocks from the cloud server and recomputes new signatures using rekey for all the data blocks as

$$\sigma_i = ( ( rk_i / sk_i \cdot H_2( fid \| i) \cdot u^{m_i} )^{\alpha} \tag{4.3}$$

on behalf of the existing user. Then proxy server uploads the new signatures back to the cloud.

## 4.5 Security Analysis

In this section, the security of IDPIA is analyzed in terms of correctness and soundness.

### 4.5.1 Correctness

**Theorem 1.** *In the proposed IDPIA, the cloud passes the auditing if all the selected data blocks of shared data and their corresponding signatures are intact in the cloud.*

**Proof:** To prove the correctness of our scheme is equivalent of proving Eq. 4.2 is correct. The proof lies in the following equations. The correctness of the protocol can be elaborated

as follows.

$$
\begin{aligned}
e(\sigma, g) &\stackrel{?}{=} e(\prod_{i=S_1}^{S_c} {s_{k_i}}^{v_i}, H_2(fid||i)) \cdot u^{\mu}, P_k) \\
e(\sigma, g) &= e(\prod_{i=S_1}^{S_c} \sigma_i^{v_i}, g) \\
&= e(\prod_{i=S_1}^{S_c} (s_{ki} \cdot H_2(fid||i) \cdot u^{m_i})^{\alpha})^{v_i}, g) \\
&= e(\prod_{i=S_1}^{S_c} (s_{ki} \cdot H_2(fid||i) \cdot u^{m_i})^{\alpha V_i}), g) \\
&= e(\prod_{i=S_1}^{S_c} (s_{ki} \cdot H_2(fid||i) \cdot u^{m_i})^{V_i}), g^{\alpha}) \\
&= e(\prod_{i=S_1}^{S_c} s_{ki} \cdot H_2(fid||i) \cdot u^{\Sigma m_i v_i}), g^{\alpha}) \\
&= e(\prod_{i=S_1}^{S_c} s_{ki} \cdot H_2(fid||i) \cdot u^{\mu}, P_k)
\end{aligned}
$$

In other words, if any selected block in the challenge message is damaged, the cloud cannot to generate a proof that can pass the verification. Hence, it is easy to detect the misbehavior of the server using our scheme.

### 4.5.2 Soundness

The soundness of the IDPIA scheme can be given based on the following theorem:

**Theorem 2 (Auditing soundness):** *It is computationally infeasible for an adversary or an untrusted cloud to generate a forgery of a proof that can pass the verification process if the DL problem in group $G_1$ is hard.*

**Proof.** This theorem is proved based on the security Game 1[34, 78] defined in section 4.2.5

First, TPA forwards a challenge message $(i, v_i)_{i \in C}$ to $\mathcal{A}_1$, and correct proof should be $(\mu, \sigma)$ which can pass the verification with Eq.4.3. Now, based on the corrupted data $\mathfrak{M}'$, the adversary $\mathcal{A}_1$ computes proof $(\mu', \sigma^*)$, where $\mathfrak{M}' \neq \mathfrak{M}$, and at least one element of

$\{\Delta m_i = m_i{}' - m_i\}$ for $i \in C$ is nonzero. The adversary $\mathcal{A}_1$ wins the Game 1 if proof (computed over incorrect data) still passes the verification performed by verifier. Otherwise, it fails. Suppose $\mathcal{A}_1$ wins the Game 1, then we get the following equation from verification Eq.4.3.

$$e(\sigma, g) \stackrel{?}{=} e(\prod_{i=S_1}^{S_c} s_{k_i}{}^{v_i}, H_2(fid||i)) \cdot u^{\mu'}, P_k)$$

Because $(\mu, \sigma)$ is correct data proof, we can get

$$e(\sigma, g) = e(\prod_{i=S_1}^{S_c} s_{k_i}{}^{v_i}, H_2(fid||i)) \cdot u^{\mu}, P_k)$$

From above two results, based on the property of bilinear map, we deduce that

$$\prod_{i=1}^{c} u^{\mu} = \prod_{i=1}^{c} u^{\mu'} and \prod_{i=1}^{c} u^{\Delta\mu} = 1$$

Let two random generators $f, h \in G_1$, and $h = f^x$ for some element $x \in Z_p$. For the given $f, h \in G_1$, a random value $u \in G_1$ can be written as $u = f^{\epsilon} \cdot h^{\xi} \in G_1$, where $\epsilon$ and $\xi \in Z_p$. Then, we have

$$\prod_{i=1}^{c} u^{\Delta\mu} = \prod_{i=1}^{c} (f^{\epsilon} \cdot h^{\xi})^{\Delta\mu} = f^{\sum_{i=1}^{c} \epsilon \cdot \Delta\mu} \cdot h^{\sum_{i=1}^{c} \xi \cdot \Delta\mu} = 1$$

Since $f = h^x$ , we can solve the DL problem by calculating $f = h^x = h^{-\frac{\sum_{i=1}^{c} \epsilon \cdot \Delta\mu}{\sum_{i=1}^{c} \xi \cdot \Delta\mu}}, x = -\frac{\sum_{i=1}^{c} \epsilon \cdot \Delta\mu}{\sum_{i=1}^{c} \xi \cdot \Delta\mu}$ only when the denominator becomes zero. However, according to the definition of Game 1, at least one element of $\Delta\mu$ is nonzero, and the denominator is zero with a probability of $1/p$. Therefore, we can find a solution to the DL problem with a probability of $1 - 1/p$, which is a non-negligible value since $p$ is very large prime. It contradicts to the assumption defined in section 2.3.

## 4.6 Performance Analysis

Now, we give a theoretical analysis of IDPIA regarding computation and communication costs. A brief comparison of the proposed scheme IDPIA and previous works is presented in Table 4.1 and 4.2 respectively. Finally, we present experimental results.

### 4.6.1 Computation cost

Here, we present the computation cost of the TPA during the auditing process and proxy server during the revocation process. For simplicity, we denote by $EX_{G1}$, $EX_{G2}$ the exponentiations in $G_1$ and $G_2$, $Mul_{G1}$, $Mul_{G2}$ the multiplication in $G_1$ and $G_2$, $Hs$ is the hash function, $P_a$ and $R$ the pairing operation and the number of blocks signed by revoked user respectively. The computation cost of the verifier involves in generating sampling blocks of challenge and verifying the validity of a corresponding proof. The cost of the verifier while checking the proof is $CEXG1 + CMul_{G1} + CP_a + CH_s$. The computation cost for resigning a block in the cloud is $2EX_{G1} + 3Mul_{G1} + Hs_{G1}$ as shown in Table 4.1.

### 4.6.2 Communication Cost

We consider communication cost of IDPIA during the verification process, which consists of challenge and proof as a response. The challenge $chal = \{i, v_i\}_{s_1 \leq i \leq s_c}$, its size is $\mid c \mid \cdot(\mid n \mid + \mid q \mid)$ bits where C, $\mid n \mid, \mid q \mid$ represents sampled blocks, the sizes of a block and an element of $Z_q$ respectively. The size of proof P ,V is the size of an element of $G_1$ or $Z_p, \mid id \mid$ is the size of a block identifier. Hence, the complete communication cost of verification is $2D \cdot \mid p \mid + \mid c \mid \cdot(\mid id \mid)$ bits, where D represents nonrevoked users in the shared group, $\mid p \mid$ is the element size in $G_1$. A comparison of selected previous schemes and our scheme in terms of communication cost is given in Table 4.2.

### 4.6.3 Experimental Results

We utilized Pairing Based Crypto Library [79] to implement cryptographic operations. All experiments conducted with an Intel i5-7200U CPU @ 2.50GHz and 8 GB Memory. In

Table 4.1: Comparison of computation costs

| Schemes | Proxy server | Auditor |
|---|---|---|
| Wang et al. [35] | $R \cdot EX_{G1}$ | $(\mid c \mid +D)EX_{G1} + (D+1)Pa + (\mid c \mid) + 2D)Mul_{G1} + D \cdot Mul_{G2}$ |
| IDPIA | $2EX_{G1} + 3Mul_{G1} + Hs_{G1}$ | $CEX_{G1} + CMul_{G1} + CPa + CHs$ |

Table 4.2: Comparison of communication costs

| Schemes | Challenge | Proof | Type |
|---|---|---|---|
| Wang et al. [35] | $\mid c \mid \cdot(\mid n \mid + \mid q \mid)$ | $2D \cdot \mid p \mid + \mid c \mid \cdot(\mid id \mid)$ | PKI |
| IDPIA | $\mid C \mid \cdot(\mid n \mid + \mid q \mid)$ | $2D \cdot \mid p \mid + \mid C \mid \cdot(\mid id \mid)$ | IBC |

the following experiments, the security parameter $\lambda$ is fixed to 160 bits and the shared data is set to 1 GB. All the experiment results are mean of 20 trials. In experimental results, we measured the computation cost of Sign generation, Proof generation, Proof verification and Resign operations. In Fig 4.2a. We present computation cost of signature generation and we can observe that computation time is increasing linearly as the size of the data is increasing. In Fig 4.2b. We present computation cost of Proof generation and verification and we can observe that Proof generation and verification time is increasing linearly as the size of data is increasing. In Fig 4.2c. We present computation cost of resigning task and we can observe that resigning time is increasing linearly as the number of blocks to be signed is increasing.

## 4.7 Summary

In this chapter, we presented a public integrity auditing scheme for shared data with efficient and secure user revocation based on identity-based signatures. With our scheme each user doesn't have to keep traditional public key as like in the PKI and verification information do not have include any certificate of public key. Meanwhile, the TPA can audit the integrity of shared data through challenge-response protocol execution even if some part of shared data has been re-signed by the proxy. The collusion attack is practically infeasible

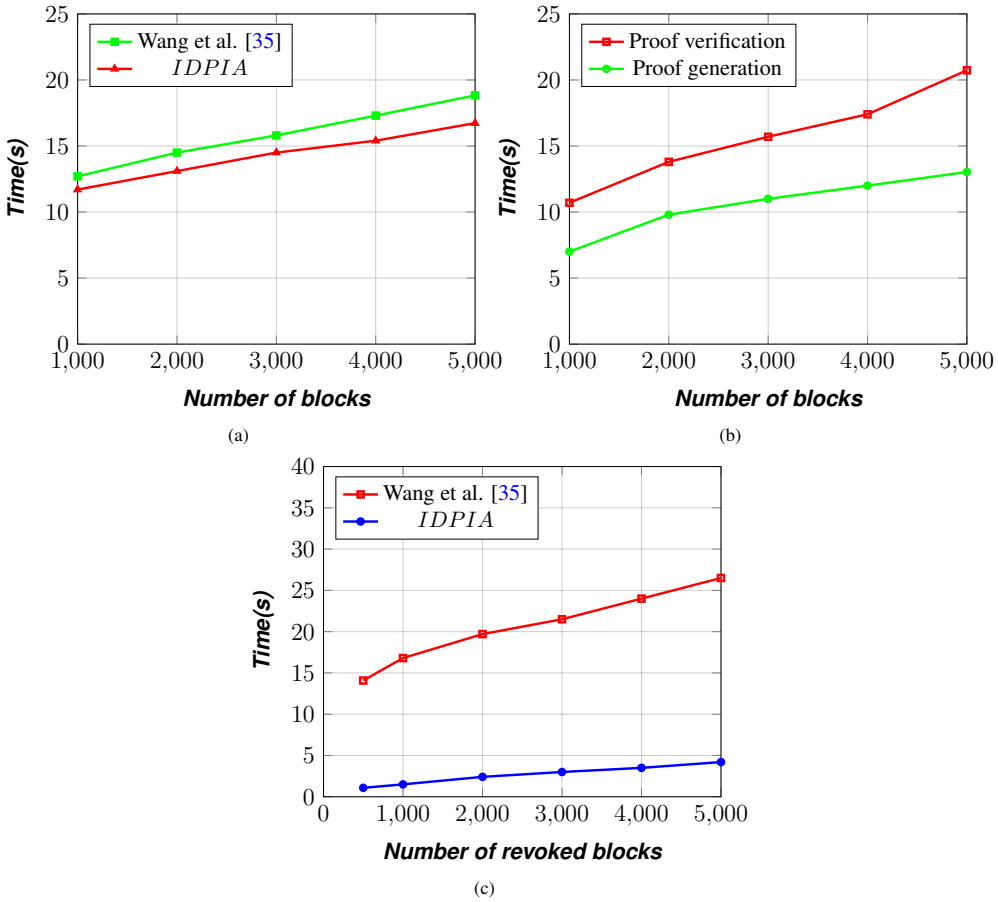



Fig. 4.2. **Computation cost: (a)** Signing time of different number of data blocks **(b)** Proof generation and verification time. **(c)** Time consumption of resigning

for the revoked user during the resigning process. The security analysis and performance analysis demonstrated that IDPIA is secure and efficient. The overhead of IDPIA is relatively small when compared to existing schemes. However, how to achieve user privacy is a significant challenge in shared data auditing. We address this issues in the next chapter.

# Chapter 5

# Attribute-based Public Integrity Auditing for Shared Data in Cloud Storage (ABPIA)

## 5.1 Introduction

In this chapter, we propose an attribute-based public integrity auditing for shared data to achieve identity privacy along with user revocation. In the scheme, the user private key is generated from the user attributes and user specify a designated auditor to check the integrity of the outsourced data. The main contributions are as follows:

- In ABPIA, users sign the data blocks over attributes, and a unique public key for the entire group is used for integrity auditing instead of using individual public keys for each user in the group. Thus it simplifies the key management.
- ABPIA achieves user privacy, i.e., signatures do not disclose identity information except that user attributes satisfy the defined access policy.
- ABPIA also supports user revocation through proxy re-signatures. That is, whenever a user quits the group, proxy can carry out the resigning process. That is proxy acts as a translator of signatures between two users, for example, Alice and Bob. More

specifically, the proxy is able to convert a signature of Alice into a signature of Bob on the same block. Meanwhile, the proxy is not able to learn any private keys of the two users, which means it cannot sign any block on behalf of either Alice or Bob.

- The comprehensive security analysis proves the correctness, unforgeability, and user privacy of ABPIA security in the random oracle model under the assumption that DL problem is hard in the bilinear group.

- The performance of ABPIA is evaluated through theoretical analysis and experimental results. The results demonstrate that ABPIA outperforms the previous schemes in terms of computational overhead.

## 5.2 Problem Statement

Here, we present the problem statement, its description followed by the architecture, design goals, Adversary model and the security model of the ABPIA scheme.

Although the existing PKI & ID-based [34, 33, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44] auditing schemes guarantee the integrity, lacks in providing the flexibility key management that means it suffer from the problems of key storage, key exchange, distribution, and verification.To address these issues, [45] proposed an attribute-based cloud data integrity auditing by employing threshold attribute-based signatures, which is different from Attribute Based Encryption (ABE) [80, 81]. In [45], the user private key generated from the attributes and data owner decides the verifier, who verifies data. However, this scheme deals only with the integrity of personal data and it is not suitable for shared data auditing because in shared data, users join or leave the group anytime dynamically. Therefore, it is required to design a public integrity auditing scheme for shared data with user revocation and flexible key management

In the proposed scheme ABPIA, users sign the data blocks over attributes, and a unique public key for the entire group is used for integrity auditing instead of using individual public keys for each user in the group. Thus it simplifies the key management. Also ABPIA achieves user privacy, i.e., signatures do not disclose identity information except

that user attributes satisfy the defined access policy.Moreover, ABPIA also supports user revocation through proxy re-signatures. That is, whenever a user quits the group, proxy can carry out the resigning process. That is proxy acts as a translator of signatures between two users, for example, Alice and Bob. More specifically, the proxy is able to convert a signature of Alice into a signature of Bob on the same block. Meanwhile, the proxy is not able to learn any private keys of the two users, which means it cannot sign any block on behalf of either Alice or Bob.

### 5.2.1 Architecture

An attribute-based cloud storage architecture considered in this paper consists of six entities as illustrated in Fig. 5.1.

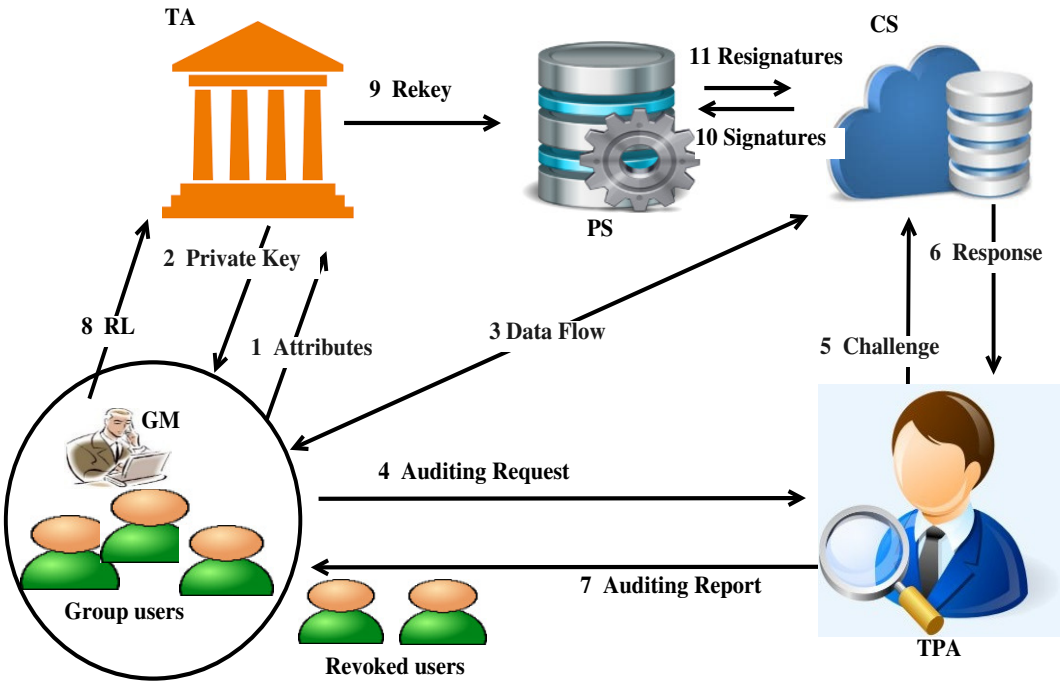



Fig. 5.1. Architecture of an attribute-based auditing scheme

1. Trusted Authority (TA) generates the master secret key and the public parameter. After receiving attributes, the TA authenticates the attributes of the user, and generates a private key for the valid group user.

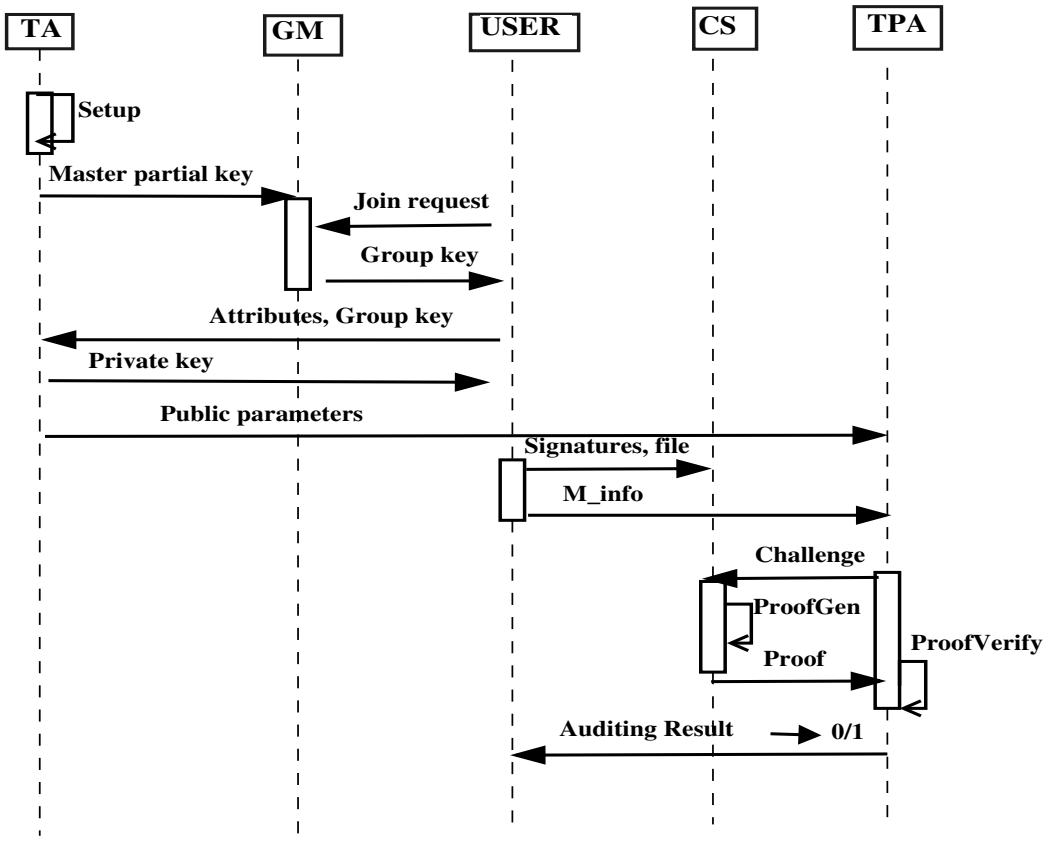



Fig. 5.2. Sequence diagram of the proposed ABPIA scheme

2. Group Manager (GM) is a trusted entity, plays the role of an administrator. It is responsible for successful user revocation and resists revoked user to perform collusion attack with cooperation of existing users.

3. The users store data and share among their group members through the cloud. New members can join and quit the group anytime. When a user wants to join the group, first, he/she sends a request to GM. After receiving the group joining key, the user submits his/her attributes along with joining the key to the TA for the private key generation. With the private key, the user generates signatures for data blocks in a file by satisfying the access policy and uploads data blocks and corresponding signatures to the cloud.

4. TPA is a trusted entity with sufficient resources and professional expertise to perform complete data auditing to ensure data accuracy. Upon receiving the auditing request from the user or according to service level agreement (SLA), TPA challenges the

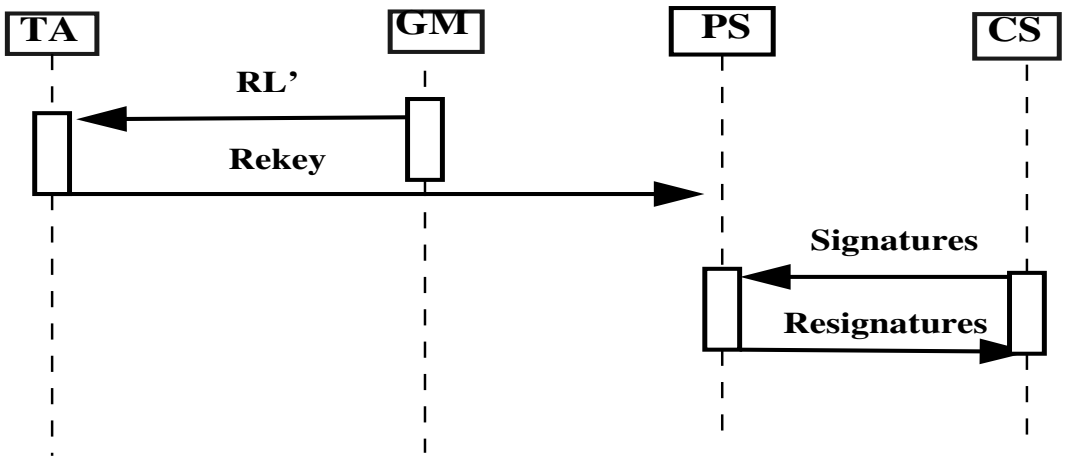



Fig. 5.3. Sequence diagram of the proposed ABPIA scheme for user revocation

cloud for randomly selected blocks to check the integrity. Upon receiving proof from the cloud, TPA verifies the correctness of data.

5. Proxy server (PS) is a semi-trusted entity, which means it is honest but curious. PS performs the given task correctly, but it is also curious to obtain the signer's identity on each block in shared data based on verification information (i.e., signatures). PS fetches the signatures from the cloud and transforms revoked user's signature on a message $m$ into a non-revoked user's signature on $m$ whenever a signed user revoked from the group. Meanwhile, the PS cannot learn any private keys of the two users, which means it cannot sign any block on behalf of either revoked or non-revoked user.

6. Cloud server (CS) is an untrusted entity that provides enormous storage to store and share the data. CS generates the proof and sends it as a response to TPA.

### 5.2.2 Overview of ABPIA

In ABPIA, initially a user submits a request to the GM to join the group. According to the request of user, GM generates a group key and securely sends to the user. Then the

user request for the private key from TA. The TA authenticates the user, generates private key based on the attributes of the user and secretly sends to the user. On receiving the private key from the TA, the user generate signatures for file data blocks using private key. After signing data blocks, uploads data blocks along with corresponding signatures to the cloud and deletes them from the local site. Later, to check the integrity of shared data, TPA challenges the cloud by selecting blocks randomly. After receiving this challenge, the cloud returns the proof of shared data as a response to the TPA. Upon receiving proof from the cloud, TPA verifies the correctness of data. Whenever a user in the group misbehaves or quits the group, GM updates the existing $RL$ and forwards to the TA. Upon receiving the updated rekey the CS performs resigning on revoked user blocks by utilizing proxy re-signatures. Our scheme also allows users to update the data dynamically that is user can modify the data in cloud without downloading the data. The detailed work flow of proposed ABPIA scheme and process of revocation are shown in Fig. 5.2 and Fig. 5.3 respectively.

### 5.2.3 Adversary Model

Here we consider the following attacks.

- **Internal adversaries:** Malicious insiders, who are cloud servers or users. They intentionally delete or modify the user's data in the cloud. Sometimes, they may try to translate signatures of one user into another user to cut down the operation costs or hide data loss to build their reputation.

- **External Adversaries:** Who may try to avert the users from accessing the shared data by destroying/altering the data in the cloud.

### 5.2.4 Design Goals

We design ABPIA scheme to meet the following goals:

1 Integrity. The designated verifier can check the integrity of shared data correctly by a validating the proof generated by the CSP by Verification algorithm with overwhelming probability.

2 Public verifiability. Anyone who knows the public key not just the client (data owner) and with sufficient resources can verify the correctness of data while keeping no private information.

3 Flexible key management. Each user in the group does not require a key pair (one private key for signing and one public key for verification.

4 User privacy. During integrity verification, the attributes of group members anonymous to the TPA. Thus TPA cannot learn anything about the attributes of the signer from the signatures.

5 Efficient user revocation. Revoked user blocks can be re-signed by existing group users efficiently with a re-signing key, while an existing user does not have to download those blocks, recompute signatures on those blocks and upload new signatures to the cloud. The resigning preformed by the proxy server improves the efficiency of user revocation and saves communication and computation resources for existing users.

### 5.2.5 Security Model

The security proof for ABPIA is performed by defining the two security games which involves two entities: an adversary $\mathcal{A}$ who plays the role of the cloud server and a challenger $\mathcal{B}$) who acts as a user. We designed ABPIA scheme to withstand the two types of adversaries, namely $\mathcal{A}_1$ (represents malicious cloud), $\mathcal{A}_2$ (represents malicious TPA). $\mathcal{A}_1$ tries to generate the forged integrity proof, and $\mathcal{A}_2$ tries to gain the identity of the signed user in the group. We prove the security of ABPIA by considering the adversary $\mathcal{A}_1$ and $\mathcal{A}_2$, and we define two interactive security games, Game 1, Game 2 against $\mathcal{A}_1$, $\mathcal{A}_2$ respectively.
The details of the games are as follows:
*Game 1 (played by $\mathcal{A}_1$ and a challenger $\mathcal{B}$)*:

- $Setup$ : $\mathcal{B}$ generates the $Pparams$, $msk$ by running the $Setup()$ algorithm. $\mathcal{B}$ forwards $Pparams$ to $\mathcal{A}_1$ while keeps the $msk$ secret.

- $pvt\_Keygen\_Query$ : $\mathcal{A}_1$ can query this oracle on the attribute set $\Omega$ satisfying access policy. For any such set $\Omega$ chosen by $\mathcal{A}_1$, challenger $\mathcal{B}$ runs the algorithm $pvt\_Keygen(Pparams, msk, \Omega)$ to produce a secret key $sk_\Omega$ corresponding to the attribute set $\Omega$, and then returns $sk_\Omega$ to $\mathcal{A}_1$ as an answer for the query.

- SignGen_Query: $\mathcal{A}_1$ chooses the file block $m_i$ and sends it to $\mathcal{B}$ for querying the signature. $\mathcal{B}$ generates and returns the signature $\sigma_i$ of $m_i$ to $\mathcal{A}_1$ by running $SignGen$ algorithm.

- Challenge: In order to check the integrity of the data file F, $\mathcal{B}$ generates challenge $chal$ and sends it to $\mathcal{A}_1$ and requests $\mathcal{A}_1$ to provide the corresponding proof P. After receiving the $chal$, the adversary $\mathcal{A}_1$ outputs a proof as response.

- Forge: For the chal, $\mathcal{A}_1$ generates P and sends to $\mathcal{B}$. $\mathcal{A}_1$ wins the game, if P can pass the integrity check and the data blocks in F are broken.

*Game 2 (played by $\mathcal{A}_2$ and a challenger $\mathcal{B}$)*: We say our ABPIA scheme achieves user privacy if for any polynomial time adversary $\mathcal{A}_2$ the advantage of $\mathcal{A}_2$ in the following game is negligible.

- Setup: The challenger $\mathcal{B}$ generates the $Pparams$, $msk$ by running the Setup() algorithm. $\mathcal{B}$ forwards both $Pparams$ and $msk$ to $\mathcal{A}_2$.

  and forwards param and mpk to the adversary, while keeps msk confidential.

- pvt_Keygen_Query: $\mathcal{A}_2$ can query this oracle on the sets of attributes. The adversary outputs attribute set $\{\Omega_i\}$ satisfying $\tau$ for each $i \in \{0, 1\}$ to the challenger $\mathcal{B}$. The challenger $\mathcal{B}$ picks a random bit b from {0,1}; then runs the algorithm $pvt\_Keygen(Pparams, msk, \Omega_b)$ to produce a secret key $sk_{\Omega_b}$ corresponding to the set $\Omega_b$ of attributes, and then sends $sk_{\Omega_b}$ as an answer for the query.

- SignGen_Query: The adversary ask the challenger to generate a signature on message $m^*$ with respect to $\Omega^*$ either from $\Omega_1$ or $\Omega_2$. $\mathcal{B}$ generates and returns the signature $\sigma^*$ of $m^*$ to $\mathcal{A}_2$ by the algorithm SignGen.

- Guess: The adversary wins the game if its output bit $b' = b$.

## 5.3 Algorithmic Framework

In this section, we define the proposed scheme algorithms.

- $Setup(1^\lambda) \rightarrow (Pparams, msk)$. It is a probabilistic algorithm run by the PKG. It takes a security parameter $\lambda$ as input and outputs public parameter $Pparams$ and $msk$.

- $Join(ID) \rightarrow \rho$. It is a probabilistic algorithm run by the PKG.It takes the identity (ID) of the user as input and outputs group key $\rho$ as output.

- $pvt\_Keygen(Pparams, msk, \Omega) \rightarrow SK_\Omega$. It is a probabilistic algorithm run by the PKG. It takes the $msk$, $Pparams$, user attributes $\Omega$ as input and outputs a private key $SK_\Omega$.

- $pub\_Keygen(Pparams, msk, \tau) \rightarrow gpk_\tau$. It is a randomized algorithm run by the PKG with identity ID for the user. It takes $\tau$, $msk$ , $Pparams$ as input and outputs a global public key $gpk_\tau$.

- $SignGen(Pparams, SK_\Omega, M, \tau) \rightarrow \sigma_i$. It is a randomized algorithm run by the data owner with identity ID. It takes $\tau$, $Pparams$, $SK_\Omega$, data blocks $\{m_i\}_{1 \leq i \leq n} \in M$ as input and outputs a set of block signatures $\{\sigma_i\}_{1 \leq i \leq n}$.

- $Challenge(M_{info}) \rightarrow C$. It is a randomized algorithm run by the TPA. It takes the abstract information of data as input and outputs the challenge $C$.

- $ProofGen(m_i, \{\sigma_i\}_{1 \leq i \leq n}, C) \rightarrow P$. It is a deterministic algorithm run by the cloud server. It takes the file blocks $m_i$, the block signatures $\{\sigma_i\}_{1 \leq i \leq n}$ and $C$ as input and outputs a proof $P$.

- $VerifyProof(Pparams, gpk_\tau, C, P) \rightarrow 1/0$. It is a deterministic algorithm run by the TPA. It takes the $Pparams$, $gpk_\tau$ of specific access policy $\tau$, $C$ and the proof $P$ as input and returns 1 or 0.

- $Revoke(RL, \{id_1, id_2, ...id_k\}) \rightarrow RL'$. It takes revocation list(RL) and user ID's as input and outputs the revised RL. $RL' \leftarrow (RL \cup \{id_1, id_2, ...id_k\})$.

- $Rekey(msk, \Omega_{new}) \rightarrow SK_{\Omega_{new}}$. It takes $msk$ and attribute set of the non revoked user from priority list as input and outputs a resigning key $SK_{\Omega_{new}}$ for resigning process.

- $Resign(\sigma_i, SK_{\Omega_{new}}) \rightarrow \sigma_i'$. It takes signature set of the revoked user(s) and rekey as input and outputs new signature set.

## 5.4 Detailed Construction

In this section, we present the concrete construction of ABPIA.

### 5.4.1 Construction of ABPIA

**Setup**

TA generates $msk$ and public parameter $Pparams$ as follows. Choose a random generator $g \in G_1$ and pick an integer element $\alpha \in Z_p^*$, and set $g_1 = g^\alpha$. Next, select a random value $g_2 \in G_1$ and compute

$$Y = \wp(g_1, g_2) \tag{5.1}$$

Also TA picks the master partial key $\gamma_0 \in Z_p^*$ randomly, which is sent to the GM for generating group joining key. Finally, two map-to-point cryptographic functions are chosen $H_1, H_2 : \{0,1\}^* \rightarrow G_1$, can map an arbitrary string $\{0,1\}^*$ into an element of $G_1$. The public parameter is published as $Pparams = (q, G_1, G_T, e, g_1, Y, H_1, H_2)$ and $msk$ is kept secret as $\alpha$.

**Join**

To join a group, user sends a request to the GM. After receiving the request GM generates

a group key as $\rho = g^{\gamma_0} \cdot H_1(ID)^{\gamma_0}$ and sends $\rho$ to the user in a secure way.

**pvt_Keygen**

Upon receiving the group key from GM, user sends attributes and group key to TA for private key generation. Then TA verifies

$$\wp(g, \rho) = \wp(g^{\gamma_0} \cdot H_1(ID), g^{\gamma_0}) \tag{5.2}$$

to know the validity of the user. If the result is false then outputs $\perp$. Else, TA generates the private key as follows. First, chooses $z \in Z_p$ randomly and calculates $d = g_2^{z+\alpha}$. Then chooses $\forall k \in \Omega$, $z_k \in Z_p$ and computes $d_{k0} = g_2^{z/\alpha} \cdot H_1(k)^{z_k}$, $d_{k1} = g^{z_k}$. Next, TA outputs

$$SK_\Omega = (d, \{d_{k0}, d_{k1}\}_{k \in \Omega}) \tag{5.3}$$

Finally, TA divides the private key into two parts and forwards part 1:$\{d_{k0}\}$ to GM and part 2:$\{d, d_{k1}\}$ to the user respectively through a secure channel.

**pub_Keygen**

In this step, TA computes a global public key $gpk_\tau$ for verifier to check the integrity of data as follows. Select a polynomial $t_x$ of degree $d_x = f_x - 1$, $\forall\ x$, where $f_x$ is the threshold value. This is done in a recursive manner from top to bottom. We start with the root $t_\gamma(0) = \alpha$ and $d_\gamma$ other points will be chosen randomly. The remaining nodes, we set as $t_x(0) = t_{parent(x)}(index(x))$ and choose $d_x$ other points randomly. After setting the polynomials, the $gpk_\tau$ for $\tau$ is $\{D_x = g^{t_x(0)}, h_k = H_1(k)^{t_x(0)}\}$, where $k = att(x)$ and $x$ is a leaf node.

**SignGen**

A user with attribute set $\Omega$ can sign the file $F = (m_i)_{1 \le i \le n}$ if the access policy (i.e.,$\tau(\Omega) = 1$) is satisfied. Signing is done in three phases.

- **Phase 1.** User chooses $r'_k \in Z_p$, randomly $\forall k \in \Omega^*$, and forwards $\{ID, \Omega^*, \{r'_k\}_{\forall k \in \Omega^*}\}$ to the GM to generate partial signature. Let $\Omega^*$ denote all the attributes associated with leaves in the access tree.

- **Phase 2.** Upon receiving the request, the GM checks the validity of the user with

RL. If user is invalid, then returns $\perp$ and does not perform any further computations. Otherwise, the GM computes partial signature as follows:

$$\{\sigma_{k0} = d_{k0} \cdot H_1(k)^{r'_k}\}_{(k \in \Omega \cap \Omega^*)}$$
$$\{\sigma_{k0} = H_1(k)^{r'_k}\}_{(k \in \Omega^* / \Omega \cap \Omega^*)}$$

and sends $\{\sigma_{k0}\}_{k \in \Omega^*}$ to the user.

- **Phase 3.** Upon receiving partial signature $\{\sigma_{k0}\}_{k \in \Omega^*}$ from the GM, user generates final signature as follows: chooses a random value $s_k$ from $Z_p$ for each $k \in \Omega$ and $u \in Z_p$ and compute

$$\sigma_{1i}^{(k)} = \left((H_2(m_i).u^{m_i})^{s_k} \cdot d\right)_{k \in \Omega} \tag{5.4}$$

$$\sigma_{2i}^{(k)} = \{g^{s_k}\}_{k \in \Omega}$$

$\forall k \in \Omega^*$ compute

$$\{\sigma_{k1} = d_{k1} \cdot g^{r'_k}\}_{(k \in \Omega \cap \Omega^*)}$$
$$\{\sigma_{k1} = g^{r'_k}\}_{(k \in \Omega^* / \Omega \cap \Omega^*)}$$

finally the user outputs the signature

$$\sigma = (\sigma_{1i}^{(k)}, \sigma_{2i}^{(k)}, \{\sigma_{k0}, \sigma_{k1}\}_{k \in \Omega^*}) \tag{5.5}$$

Then the user uploads the data and signature $\sigma$ and deletes them from local storage.

**Challenge**

After storing data in cloud, the user request the TPA to perform integrity verification. Then TPA selects a $c$-element subset $J$ of set [1, n], and selects $v_i \in Z_p$ for each $i \in J$ randomly. Let $C = \{(i, v_i)\}_{i \in J}$ be a challenge sends to the cloud.

**ProofGen**

After receiving the challenge $C = \{(i, v_i)\}_{i \in J}$, the server computes a proof, which consists of data proof and signature proof. To compute the data proof the cloud generates the linear

combination of sample blocks for the specified blocks in challenge.

$$\mu = \sum_{i \in C} v_i m_i \tag{5.6}$$

Next, the server also computes signature proof as

$$\sigma_1^{(k)} = \{\prod_{i \in C} \sigma_{1i}^{(k)^{v_i}}\}_{k \in \Omega} \tag{5.7}$$

$$\sigma_2^{(k)} = \{\sigma_{2i}^{(k)}\}_{i \in C, k \in \Omega} \tag{5.8}$$

Then the server returns $(\mu, \sigma_1^{(k)}, \sigma_2^{(k)}, \{\sigma_{k0}, \sigma_{k1}\}_{k \in \Omega^*})$ to the TPA as proof.

**VerifyProof**

TPA verifies the integrity of outsourced data blocks after receiving proof from the server as follows. We define a recursive algorithm $vfynode(\sigma, gpk, x)$ to return a point of $G_T$ or $\perp$. Let $k = attr(x)$. If $x$ is a leaf node, then

$$vfynode(\sigma, gpk, x) = \begin{cases} \frac{\wp(\sigma_{k0}, D_x)}{\wp(\sigma_{k1}, h_k)} & \text{if } \frac{\wp(\sigma_{k0}, D_x)}{\wp(\sigma_{k1}, h_k)} \neq 1 \\ \perp, & \text{otherwise.} \end{cases}$$

if $k \in \Omega \cap \Omega^*$,

$\wp(\sigma_{k0}, D_x)/\wp(\sigma_{k1}, h_k)$

$$\begin{aligned} &= \wp(d_{k0} \cdot H_1(k)^{r'_k}, g^{p_x(0)})/\wp(d_{k1} g^{r'_k}, H_1(k)^{p_x(0)}) \\ &= \wp(g_2^{z/\alpha} \cdot H_1(k)^{z_k + r'_k}, g^{p_x(0)})/\wp(g^{z_k + r'_k}, H_1(k)^{p_x(0)}) \\ &= \wp(g_2^{z/\alpha} \cdot H_1(k)^{z_k + r'_k}, g^{p_x(0)})/\wp(g^{p_x(0)}, H_1(k)^{z_k + r'_k}) \\ &= \wp(g, g_2)^{z/\alpha \cdot p_x(0)} \end{aligned}$$

if $k \in \Omega^*/\Omega \cap \Omega^*$,

$\wp(\sigma_{k0}, D_x)/\wp(\sigma_{k1}, h_k)$

$= \wp(H_1(k)^{r'_k}, g^{p_x(0)})/\wp(g^{r'_k}, H_1(k)^{p_x(0)})$

$= 1$

If $x$ is a non-leaf node, $vfynode(\sigma, gpk, x)$ proceeds as follows: $\forall z$ that are children of $x$, $vfynode(\sigma, gpk, x)$ is called and returns $F_z$. Let $S_x$ be the set of child nodes $z$ such that $Fz \neq \perp$. If no such set exists, then the function returns $\perp$. Otherwise, let

$$i = index(z), S'_x = index(z) : z \in S_x$$

and compute

$$\begin{aligned}
F_x &= \prod_{z \in S_x} F_z^{\Delta_{i,S'_x}(0)} \\
&= \prod_{z \in S_x} \left(\wp(g, g_2)^{z/\alpha p_z(0)}\right)^{\Delta_{i,S'_x}(0)} \\
&= \prod_{z \in S_x} \left(\wp(g, g_2)^{z/\alpha p_{parent(z)}(index(z))}\right)^{\Delta_{i,S'_x}(0)} \\
&= \prod_{z \in S_x} \left(\wp(g, g_2)^{z/\alpha p_x(i)\Delta_{i,S'_x}(0)}\right. \\
&= \wp(g, g_2)^{z/p_x(0)}
\end{aligned}$$

Now, calculate $F_\gamma$ and verify if

$$\frac{\wp(g, \sigma_1^{(k)})}{F_\gamma . \wp\left(\prod_{(i,v_i)_{i \in C}} H_2(m_i) u^\mu, \sigma_2^{(k)}\right)} \stackrel{?}{=} Y \tag{5.9}$$

If the equation holds, the blocks stored in the cloud are correctly maintained. Otherwise, the data is not intact. i.e., data is modified or corrupted.

**Revoke**

Whenever a user leaves or misbehaves, the GM revokes the user by updating the RL list and informs to TA. $RL' \leftarrow (RL \cup \{id_1, id_2, ...id_k\})$.

**Rekey**

To resign the blocks of revoked user, TA generates rekey as follows. It takes $msk$ and attribute set of the non revoked user from priority list and generates resigning key $SK_{\Omega_{new}}$. First, choose a random $z_n \in Z_p$ and compute $d = g_2^{z_n + \alpha}$. Then for each $k \in \Omega$, choose $z_k \in Z_p$ and compute $d_{k0} = g_2^{z_n/\alpha} \cdot H_1(k)^{z_k}, d_{k1} = g^{z_k}$. Finally, output the resigning key

$SK_{\Omega_{new}} = (d, \{d_{k0}, d_{k1}\}_{k\in\Omega_{new}})$ and forwards to the PS.

**Resign**

PS re-signs the revoked user(s) blocks on behalf of existing user. Choose a random value $s_r$ from $Z_p$ for each $k \in \Omega_{new}$ and $u \in Z_p$ and compute

$$\sigma_{1i}^{(k)} = ((H_2(m_i).u^{m_i})^{s_r} d)_{k\in\Omega_{new}}$$
$$\sigma_{2i}^{(k)} = \{g^{s_r}\}_{k\in\Omega_{new}}$$

$\forall k \in \Omega^*_{new}$, choose $r_k \in Z_p$ randomly and compute

$$\{\sigma_{k0} = d_{k0}H_1(k)^{r_k}, \sigma_{k1} = d_{k1}g^{r_k}\}_{(k\in\Omega_{new}\cap\Omega^*)}$$
$$\{\sigma_{k0} = H_1(k)^{r_k}, \sigma_{k1} = g^{r_k}\}_{(k\in\Omega^*/\Omega_{new}\cap\Omega^*)}$$

finally the PS sends a re-signature for cloud update.

$$\sigma' = (\sigma_{1i}^{(k)}, \sigma_{2i}^{(k)}, \{\sigma_{k0}, \sigma_{k1}\}_{k\in\Omega^*})$$

## 5.5 Security Analysis

In this section, we prove that the ABPIA is secure under the security model described in section 5.2.5 in terms of correctness, unforgeability, and user privacy. The security proof is based on the intractability of the well-known DL problem in group.

### 5.5.1 Correctness

**Theorem 1.** *In ABPIA, the verifier successfully audits data integrity if all the randomly selected file blocks and their corresponding signatures kept as it is in the cloud.*

**Proof.** ABPIA correctness can be proved by verifying the Eq. 5.9, based on properties of bilinear maps and access structure.

$$\frac{\wp(g, \sigma_1^{(k)})}{F_\gamma \cdot \wp\left(\prod_{(i,v_i)_{i\in C}} H_2(m_i)u^{\mu}, \sigma_2^{(k)}\right)} \stackrel{?}{=} Y$$

The LHS of Eq. 5.9 can be deduced as follows:

$$
\begin{aligned}
LHS &= \frac{\wp(g, \sigma_1^{(k)})}{F_\gamma \cdot \wp\Big(\prod\limits_{(i,v_i)_{i\in C}} H_2(m_i)u^\mu, \sigma_2^{(k)}\Big)} \\
&= \frac{\wp\Big(g, \Big(\prod\limits_{(i,v_i)_{i\in C}} \big(H_2(m_i)u^{m_i}\big)^{s_k} d\Big)^{v_i}\Big)_{k\in\Omega}}{F_\gamma \cdot \wp\Big(\prod\limits_{(i,v_i)_{i\in C}} H_2(m_i)u^\mu, (g^{s_k})\Big)_{k\in\Omega}} \\
&= \frac{\wp\Big(g, \Big(\prod\limits_{(i,v_i)_{i\in C}} \big(H_2(m_i)u^{m_i}\big)^{s_k} g_2^{z+\alpha}\Big)^{v_i}\Big)_{k\in\Omega}}{F_\gamma \cdot \wp\Big(\prod\limits_{(i,v_i)_{i\in C}} H_2(m_i)u^\mu, (g^{s_k})\Big)_{k\in\Omega}} \\
&= \frac{\wp(g, g_2^{z+\alpha})\wp\Big(g, \prod\limits_{(i,v_i)} \Big(H_2(m_i)u^{m_i s_k}\Big)^{v_i}\Big)_{k\in\Omega}}{F_\gamma \cdot \wp\Big(\prod\limits_{(i,v_i)_{i\in C}} H_2(m_i)u^\mu, (g^{s_k})\Big)_{k\in\Omega}} \\
&= \frac{\wp(g, g_2^{z+\alpha})\wp(g, \prod\limits_{(i,v_i)} \Big(\big(H_2(m_i)u^{v_i m_i}\big)^{s_k}\Big)_{k\in\Omega}}{F_\gamma . \wp\Big(\prod\limits_{(i,v_i)} H_2(m_i)u^\mu, g^{s_k}\Big)_{k\in\Omega}} \\
&= \frac{\wp(g, g_2^{z+\alpha})\wp\Big(g^{s_k}, \prod\limits_{(i,v_i)} \big(H_2(m_i).u^\mu\big)\Big)_{k\in\Omega}}{F_\gamma \cdot \wp\Big(\prod\limits_{(i,v_i)} H_2(m_i)u^\mu, g^{s_k}\Big)_{k\in\Omega}} \\
&= \frac{\wp(g, g_2)^{z+\alpha}}{\wp(g, g_2)^z} \\
&= \wp(g, g_2)^\alpha \\
&= \wp(g^\alpha, g_2) \\
&= \wp(g_1, g_2) \\
&= Y(RHS)
\end{aligned}
$$

From the above proof, we say that the data blocks indeed stored as it is and maintained properly.

### 5.5.2 Unforgeability

The unforgeability of the ABPIA scheme can be given based on the following theorem: **Theorem 2 (Auditing soundness):** *It is computationally infeasible for untrusted cloud or an adversary to generate a forgery of a proof that can pass the verification process if the DL problem in group $G_1$ is hard.*

**Proof.** We prove this theorem based on the security Game 1 defined in section 5.2.5

First, TPA sends a challenge message $(i, v_i)_{i\in C}$ to $\mathcal{A}_1$, and correct proof should be $(\mu, \sigma_1^{(k)}, \sigma_2^{(k)}, \{\sigma_{k0}, \sigma_{k1}\}_{k\in\Omega^*})$ which can pass the verification with equation (9). Now, based on the corrupted data $\mathfrak{M}'$, the adversary $\mathcal{A}_1$ computes proof $(\mu', \sigma_1^{(k)}, \sigma_2^{(k)}, \{\sigma_{k0}, \sigma_{k1}\}_{k\in\Omega^*})$, where $\mathfrak{M}' \neq \mathfrak{M}$, and at least one element of $\{\Delta m_i = m_i{}' - m_i\}$ for $i \in C$ is nonzero. The adversary $\mathcal{A}_1$ wins the Game 1 if proof (computed over incorrect data) still passes the verification performed by verifier. Otherwise, it fails. Suppose $\mathcal{A}_1$ wins the Game 1, then we get the following equation from verification Eq. 5.9,

$$\frac{\wp(g, \sigma_1^{(k)})}{F_\gamma \cdot \wp\left(\prod\limits_{(i,v_i)_{i\in C}} H_2(m_i) \cdot u^{\mu'}, \sigma_2^{(k)}\right)} = Y$$

Because $(\mu, \sigma_1^{(k)}, \sigma_2^{(k)}, \{\sigma_{k0}, \sigma_{k1}\}_{k\in\Omega^*})$ is correct data proof, we can get

$$\frac{\wp(g, \sigma_1^{(k)})}{F_\gamma \cdot \wp\left(\prod\limits_{(i,v_i)_{i\in C}} H_2(m_i)u^{\mu}, \sigma_2^{(k)}\right)} = Y$$

From above two results, based on the property of bilinear map, we deduce that

$$\prod_{i=1}^{c} u^{\mu} = \prod_{i=1}^{c} u^{\mu'} and \prod_{i=1}^{c} u^{\Delta\mu} = 1$$

Let two random generators $f, h \in G_1$, and $h = f^x$ for some element $x \in Z_p$. For the given $f, h \in G_1$, a random value $u \in G_1$ can be written as $u = f^{\epsilon} \cdot h^{\xi} \in G_1$, where $\epsilon$ and $\xi \in Zp$. Then, we have

$$\prod_{i=1}^{c} u^{\Delta\mu} = \prod_{i=1}^{c} (f^{\epsilon} \cdot h^{\xi})^{\Delta\mu} = f^{\sum_{i=1}^{c} \epsilon \cdot \Delta\mu} \cdot h^{\sum_{i=1}^{c} \xi \cdot \Delta\mu} = 1$$

Since $f = h^x$ , we can solve the DL problem by calculating $f = h^x = h^{-\frac{\sum_{i=1}^{c} \epsilon \cdot \Delta\mu}{\sum_{i=1}^{c} \xi \cdot \Delta\mu}}, x = -\frac{\sum_{i=1}^{c} \epsilon \cdot \Delta\mu}{\sum_{i=1}^{c} \xi \cdot \Delta\mu}$ only when the denominator is zero. However, according to the definition of Game 1, at least one element of $\Delta\mu$ is nonzero, and the denominator is zero with probability of $1/p$. Therefore, we can find a solution to the DL problem with a probability of $1 - 1/p$, which is non-negligible since $p$ is very large. It contradicts to the assumption defined in section 2.3.

### 5.5.3 User Privacy

**Theorem 3.** *Given shared data $\mathfrak{M}$ and its corresponding signature $\sigma$, it is computationally hard for TPA to learn the identity of the valid signer.*

**Proof**. We prove this theorem based on the security Game 2 defined in section 5.2.5

In our proposed ABPIA scheme, it is easy to see that $\mathcal{A}_2$ cannot learn anything about attribute information from signatures generated by two different sets of attributes for one file block as long as the two attribute sets satisfy the access policy. First, the challenger runs $Setup$ to get the public parameters $Pparams$ and the $msk$. It sends both to the adversary $\mathcal{A}_2$. After this interaction, $\mathcal{A}_2$ outputs two attribute sets $\Omega_1$ and $\Omega_2$ to the challenger $\mathcal{B}$. Next, $\mathcal{B}$ generates private keys as $sk_{\Omega_1} = (d^1, d^1_{i0}, d^1_{i1})$ and $sk_{\Omega_2} = (d^2, d^2_{i0}, d^2_{i1})$ for $\Omega_1$ and $\Omega_2$, respectively by running $pvt\_Keygen$ algorithm. The adversary $\mathcal{A}_2$ queries the challenger to generate a signature on message $m^*$ with respect to $\Omega^*$ from either $\Omega_1$ or $\Omega_2$. Next, the challenger chooses a random bit $b \in \{0, 1\}$ and outputs a signature $\sigma = (\sigma^{(k)}_{1i}, \sigma^{(k)}_{2i}, \{\sigma_{k0}, \sigma_{k1}\}_{k\in\Omega^*})$ by running $SignGen$ with the private key $sk_{\Omega_b} = (d^b, d^b_{i0}, d^b_{i1})_{i\in\Omega_b})$. Finally, $\mathcal{A}_2$ outputs a guess bit $b' \in \{0, 1\}$ and wins the game if $b' = b$.

On the basis of Lagrange interpolation for the tree $\tau$, it is clear that the signature could be generated from either $sk_{\Omega_1}$ or $sk_{\Omega_2}$. We also have proved it. From the above proof, we say that the probability of advantage for an adversary $\mathcal{A}_2$ to win the game is not better than

1/2, since the signature $\sigma$ is simply independent of everything except the message and the access policy. Hence, user privacy is achieved.

## 5.6 Performance Analysis

Here, we analyze the computation and communication overhead of ABPIA. Furthermore, we also analyze the performance of the Wang et al.[35] and Zhang et al.[43], and give a comparison for these two schemes in terms of computation overhead and communication overhead. The different notations used in performance analysis are listed in Table 5.1.

Table 5.1: Notations

| Notation | Description |
|---|---|
| $EX_{G1}, EX_{GT}$ | Exponentiations in $G_1, G_T$ |
| $Hs$ | Collision Resistant Hash function |
| $AddZ_p$ | Addition in $Z_p$ |
| $Pa$ | The pairing operation |
| $l$ | The size of $\Omega^*$ |
| $N_s$ | The set of least interior nodes satisfying $\tau$. |
| $Mul_{G1}, Mul_{GT}$ | The multiplication in $G_1$ and $G_T$ |
| $\mid p \mid$ | The size of an element in $G_1$ |
| $\mid q \mid$ | Size of an element of $Zp$ |
| $d$ | Nonrevoked users |
| $\mid \tilde{id} \mid$ | The size of a block identifier |

Table 5.2: Computation cost comparison

| | Computation Cost | | |
|---|---|---|---|
| Scheme | SignGen | VerifyProof | Resign |
| Wang et al. [35] | $2EX_{G1} + Mul_{G1} + Hs$ | $(C+d)EX_{G1}+(C+2d)Mul_{G1}+(d+1)Pa+dMul_{G1}+CHs$ | $2EX_{G1} + Mul_{G1} + 2Pa + Hs$ |
| Zhang et al. [43] | $n(2EX_{G1} + Mul_{G1} + Hs)$ | $CHs+2Hs+(C+3)Mul_{G1}+(C+3)EX_{G1}+2Pa$ | $Add_{Zp}$ |
| ABPIA | $2(l+1)EX_{G1} + (2l+1)Mul_{G1} + (l+1)H_s$ | $(2l+3)Pa + N_sEX_{G1}$ | $2(l+1)EX_{G1} + (2l+1)Mul_{G1} + (l+1)H_s$ |

Table 5.3: Communication cost comparison

| | Communication Cost | |
|---|---|---|
| Scheme | Challenge | Proof |
| Wang et al. [35] | $n(\mid p \mid + \mid q \mid)$ | $2d\cdot \mid p \mid + C \cdot \mid \tilde{id} \mid$ |
| Ahang et al. [43] | $n(\mid p \mid + \mid q \mid)$ | $(C+1)\cdot \mid q \mid \mid + \mid p \mid + C\cdot \mid \tilde{id} \mid$ |
| ABPIA | $n(\mid p \mid + \mid q \mid)$ | $2d \mid p \mid + C \cdot (\mid \tilde{id} \mid + \mid n \mid + \mid q \mid)$ |

### 5.6.1 Computation cost

Here, we give the computation cost of the signer during signing, verifier during verification and PS during the resigning. The computation cost of the signer is $2(l+1)EX_{G1} + (2l+1)Mul_{G1} + (l+1)H_s$. The computation cost of the verifier includes challenge generation and verification of proof i.e., $(2l+3)P_a + N_sEX_{G1}$. The computation cost of PS for resigning is $2(l+1)EX_{G1} + (2l+1)Mul_{G1} + (l+1)H_s$. Table 5.2 gives computation cost comparison with other auditing schemes like [35] and [43].

### 5.6.2 Communication Cost

Here, we consider only communication cost of challenge generation and proof generation because these are the primary parts of the auditing process. The size of challenge is $C \cdot (\mid p \mid + \mid q \mid)$ bits, the size of proof $(\mu, \sigma_1^{(k)}, \sigma_2^{(k)}, \{\sigma_{k0}, \sigma_{k1}\}_{k\in\Omega^*}, C)$ is $2d \mid p \mid + C \cdot (\mid \tilde{id} \mid)$ bits. Hence, the verifiers total communication cost is $2d \mid p \mid + C \cdot (\mid \tilde{id} \mid + \mid n \mid + \mid q \mid)$ bits. Table 5.3 gives communication cost comparison with other auditing schemes like [35] and [43].

### 5.6.3 Experimental analysis

We implemented ABPIA on a system with Intel i5-7200U CPU @ 2.50 GHz and 8 GB RAM and compared ABPIA with existing schemes [35] based on PKI settings and [43] based on ID-based cryptography in terms of computational and communication overheads. All experiments are carried out in python language using crypto-0.42 library [79]. The implementation uses a symmetric super singular curve where the base field size is 512-bit and the security parameter fixed to 160-bits. All results are a mean of 15 trials. The experimental results for private key generation, signature generation, verification, and revocation

are obtained and plotted as graphs from Fig. 5.4a to Fig. 5.4e.

Fig. 5.4a shows the time consumption of pvt_Keygen algorithm. From Fig.5.4a, we can notice that the time needed for the pvt_Keygen algorithm increases linearly as the number of attributes increases in the user attributes. It is reasonable because a user's private key computed from every attribute of the user. So the computation time of pvt_Keygen algorithms is dependent on the number of attributes an identity includes.

Fig. 5.4b shows the computation overhead for a user to generate signatures for the different number of data blocks with the same size. From Fig. 5.4b, we can notice that our proposed ABPIA scheme takes less time than existing [35], [43] because of utilization of expressive attribute-based signatures.

Fig. 5.4c and Fig. 5.4d shows the computation time of the TPA for proof verification with respect to the number of blocks and number of users, respectively. From Fig. 5.4c and Fig. 5.4d, we can learn that the proof verification time in [35], [43] is linear with the number of blocks in challenge message. In ABPIA scheme, it is relatively low since a unique public key is used during verification, whereas in [35], [43] different public keys used for different users. Fig. 5.4e shows the computation cost for resigning the different number of revoked user blocks. Here, we compare the ABPIA scheme with [35] and ignored [43] because [43] does not perform resigning on blocks whenever user revocation happens; instead, they update the private key of all users. From Fig. 5.4e, we can observe that the computation time for resigning in both schemes is linear with the number of revoked user blocks, but the ABPIA scheme performs better than [35].

## 5.7 Summary

In this chapter, we presented an attribute-based public auditing scheme for integrity checking in cloud storage. In ABPIA scheme, we used individual private keys of each user for signing and only one public key for integrity verification, which simplifies key management. In ABPIA scheme, the signature does not reveal any user identity; thus, signer privacy achieved. When a user revocation happens, ABPIA allows the proxy to re-sign blocks that were signed by the revoked user. The security analysis of ABPIA proved the

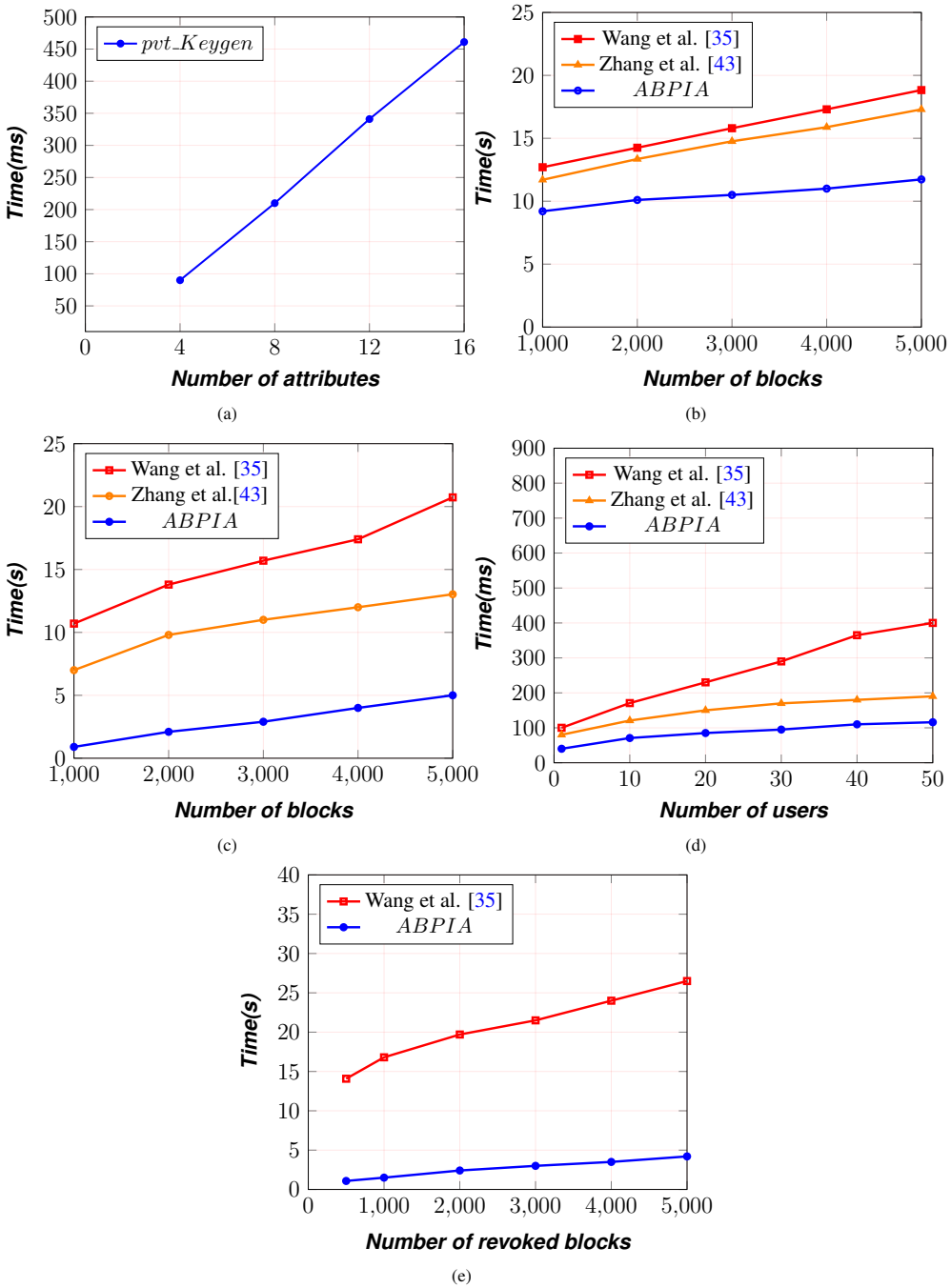



Fig. 5.4. **Computation cost: (a)** Computation Time of pvt_Keygen algorithm **(b)** Signing time of different number of data blocks **(c)** Proof verification time of different number of data blocks. **(d)** Time consumption of verification for various number of users **(e)** Time consumption of resigning

correctness, unforgeability against the untrusted server, and user privacy against TPA. The performance analysis and detailed experimental results show the practicality of our system.

In ABPIA, we observe that the TA generates and holds the private key. That is if TA is untrusted, the scheme is not secure either, which may results in impersonation attacks that leads to the forgery of the user's signatures to pass the verification successfully. This problem is known as key escrow problem. To address the key escrow problem while supporting data privacy and data dynamics along with the group user revocation, in the next chapter, a new remote data auditing scheme is proposed by utilizing the certificateless cryptography.

# Chapter 6

# Certificateless Privacy Preserving Public Auditing for Dynamic Shared Data in Cloud Storage(CLPPPA)

## 6.1 Introduction

In this chapter, we present certificateless public auditing scheme for shared data to achieve privacy preserving and data dynamics along with user revocation while reducing the complexity of certificate management in PKI schemes [33, 34, 35, 36, 37, 38, 39, 40, 41] and eliminating the inherent key escrow problem in ID-based schemes [42, 43, 44]. The main contributions are summarized as follows:

- In this scheme, we leverage certificateless signatures to generate signatures of file blocks, that can simplify certificate management and eliminates key escrow problem.
- CLPPPA achieves data privacy against verifier through random masking technique to blind the data proof during the process of auditing.
- We extend double linked list information table (DLIT) to support shared data dynamics such as insertion, modification and deletion.

- In CLPPPA, we also use the idea of proxy re-signatures to support group user revocation. i.e., whenever a user misbehaves or quits the group, the cloud server is able to carry out resigning process on behalf of group user.

- The security analysis proves the correctness, unforgeability and privacy of CLPPPA based on DL and CDH assumptions in ROM. We also provide the security comparison with some of the existing schemes.

- The performance analysis evaluates performance of CLPPPA theoretically and experimentally in terms of computation overheads.

## 6.2 Problem Statement

Here, we the present the problem statement, its description followed by the architecture, design goals, adversary model and the security model of the CLPPPA scheme.

To get rid of key escrow issue in ID-based schemes mentioned above, unlike [42, 43, 44], the key generation center (KGC) in the proposed scheme, chooses a random value $X \leftarrow Z_p^*$ as its private key and generates $Y = g^X \in G_1$ as its public key. Upon receiving identity (ID) from the user, the KGC generates the partial private key $D = H_1(ID)^X$ and sends it to the user through a secure channel. After receiving the partial private key $(D)$, user computes the full private key $(D, x)$, where $x$ ($x \in Z_p^*$) is a secret value chosen by herself/himself. This approach solves the key escrow problem by restricting the KGC to generate only the partial private key rather than the full private key. Therefore, the KGC cannot forge the user signature by any means. Based on this notion, Wang et al. [82] first proposed a certificateless provable data possession (CLPDP) scheme and a security model. However, He et al. [64], who pointed out that Wang et al.'s [82] scheme was not secure against the type I adversary and suggested a CLPDP scheme for cloud-assisted wireless body area networks to enhance security. Unfortunately, both the schemes [82, 64] cannot preserve privacy and focused on personal data auditing. Hence, not suitable for shared data auditing. Later, Li et al. [62] introduced a certificateless public auditing scheme for shared data to achieve the shared data integrity along with user revocation in the cloud. However,

this scheme does not support shared data dynamics and does not support security property such as privacy preserving, which are necessary demands for shared data auditing.

In CLPPPA, initially, a user submits a request to the GM to join the group. According to the user's request, GM generates a group key and securely sends it to the user. Then the user requests the partial private key from KGC. The KGC authenticates the user, generates a partial private key, and secretly sends it to the user. On receiving the partial private key from the KGC, the user generates his/her own private key and computes signatures for file data blocks using a private key. After signing data blocks, uploads data blocks along with corresponding signatures to the cloud and deletes them from the local site. Later, to check the integrity of shared data, TPA challenges the cloud by selecting blocks randomly. After receiving this challenge, the cloud returns the proof of shared data as a response to the TPA. Upon receiving proof from the cloud, TPA verifies the correctness of data. Whenever a user in the group misbehaves or quits the group, GM updates the existing $RL$ and forwards it to the CS. Upon receiving the updated $RL$, the CS performs resigning on revoked user blocks by utilizing proxy re-signatures. Our scheme also allows users to update the data dynamically. That is, the user can modify the data in the cloud without downloading the data.

### 6.2.1 Architecture

We consider certificateless cloud storage architecture with five entities as illustrated in Fig. 6.1.

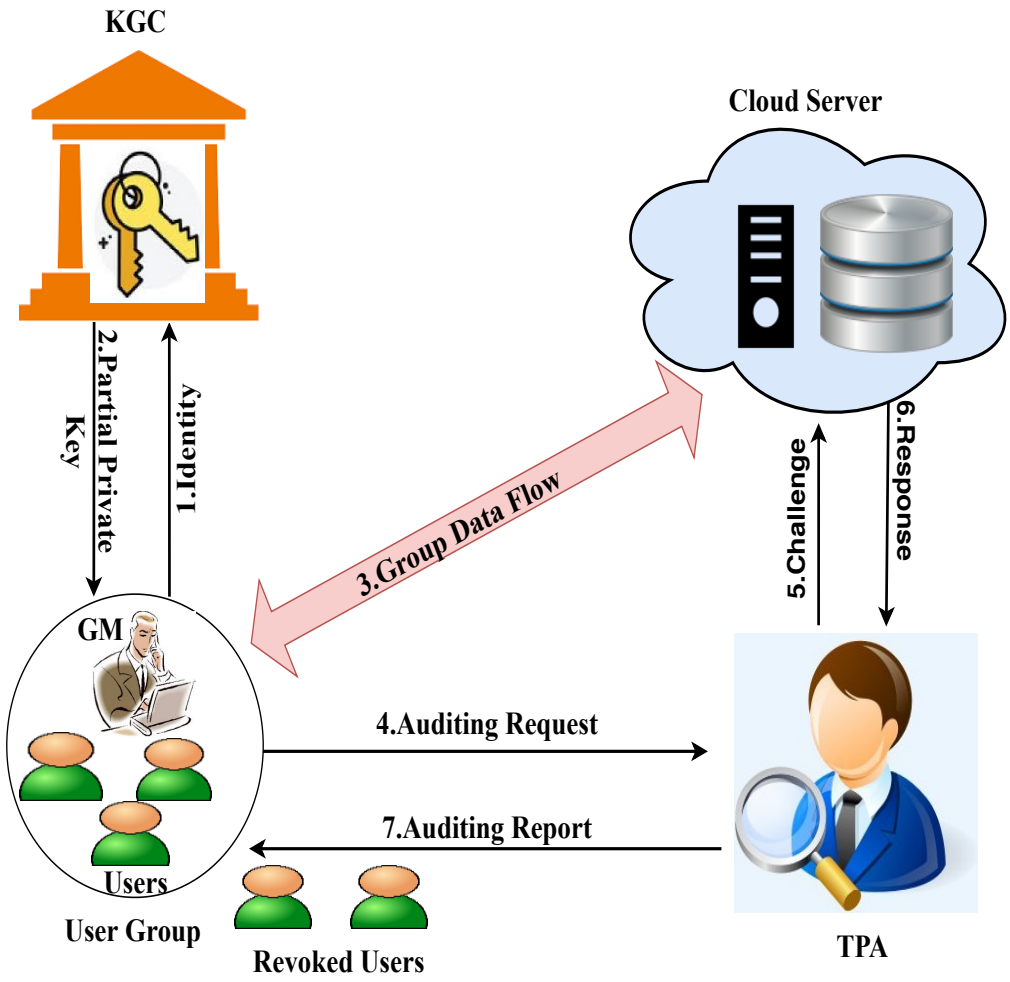



Fig. 6.1. Architecture of certificateless public auditing scheme

1. KGC is a trusted entity, which creates the parameters and the master private key. It also generates a partial private key for the user using master secret key ($msk$) and user unique identity.

2. Group manager (GM) is a trusted entity, plays the role of an administrator. It is responsible for creating the group. It also revoke users when a user in the group leaves or misbehaves.

3. Users store data in cloud, share with each other in the group and they can join and leave the group. Furthermore, users can update the data dynamically.

4. TPA is also called public verifier that has expertise and capabilities to perform auditing task on behalf of user regularly or upon request. Also convinces both cloud

server and users by providing unbiased auditing results.

5. Cloud server (CS) is a semi-trusted entity, which means it is honest, but curious. It provides the enormous storage space on its infrastructure to manage the file in the cloud. During user revocation, it also acts as proxy, which performs delegated re-sign task by utilizing proxy re-signatures on behalf of the group.

### 6.2.2 Overview of CLPPPA

In CLPPPA, initially a user submits a request to the GM to join the group. According to the request of user, GM generates a group key and securely sends to the user. Then the user request for the partial private key from KGC. The KGC authenticates the user, generates partial private key and secretly sends to the user. On receiving the partial private key from the KGC, the user generates his/her own private key and generate signatures for file data blocks using private key. After signing data blocks, uploads data blocks along with corresponding signatures to the cloud and deletes them from the local site. Later, TPA verifies the correctness of the data by selecting blocks randomly. After receiving this challenge, the cloud returns the proof of shared data as a response to the TPA. Upon receiving proof from the cloud, TPA verifies the correctness of data. Whenever a user in the group misbehaves or quits the group, GM updates the existing $RL$ and forwards to the CS. Upon receiving the updated $RL$ the CS performs resigning on revoked user blocks by utilizing proxy re-signatures. Our scheme also allows users to update the data dynamically that is user can modify the data in cloud without downloading the data. The detailed work flow of proposed CLPPPA scheme and process of revocation are shown in Fig. 6.2 and Fig 6.3 respectively.

### 6.2.3 Adversary model

We designed CLPPPA scheme to withstand the four types of adversaries namely $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$ and $\mathcal{A}_4$.

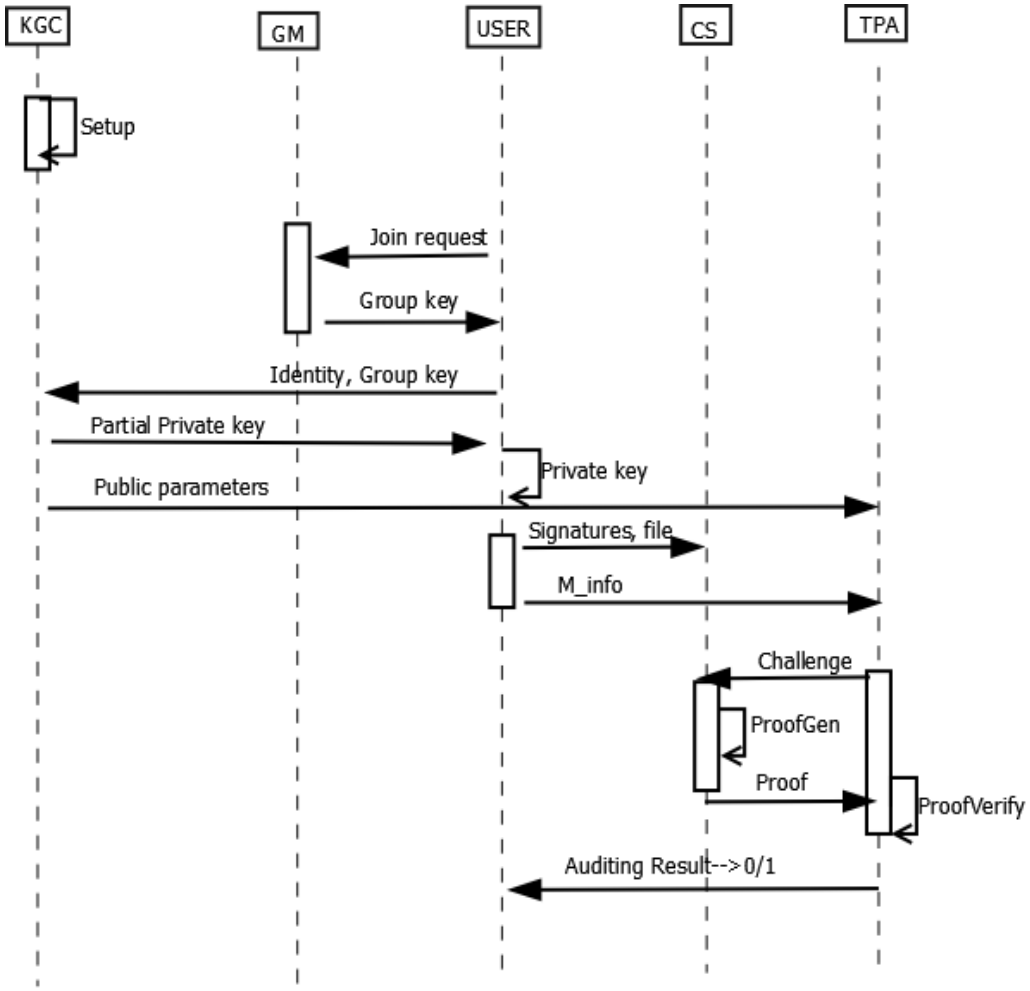



Fig. 6.2. Sequence diagram of the proposed CLPPPA scheme

- Type-I Adversary ($\mathcal{A}_1$): $\mathcal{A}_1$ (malicious outsider) tries to replace the user's public key with a false key even though he could not have access to KGC's master secret key ($msk$).
- Type-II Adversary ($\mathcal{A}_2$): $\mathcal{A}_2$ (malicious KGC) tries to mount an impersonation attack having access to the $msk$ of the KGC and it cannot replace the public key of the user. Even though the KGC is trusted entity, in a practical scenario, the KGC might engage in other adversarial activities such as eavesdropping on signatures and making signing queries, which is also known as Type II Adversary.
- Type-III Adversary ($\mathcal{A}_3$): $\mathcal{A}_3$ (malicious CSP) tries to compute a forged auditing proof that can pass the verification.

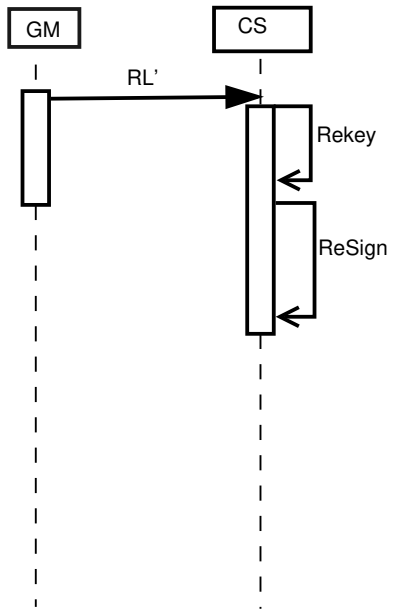



Fig. 6.3. The process of user revocation

- Type-IV Adversary ($\mathcal{A}_4$): $\mathcal{A}_4$ (malicious TPA) tries to gain access to private information of data during audit process.

### 6.2.4 Design goals

We design certificateless public integrity auditing scheme to achieve the following goals:

- **Public verifiability.** Any one-who knows public key can verify the integrity of data on behalf of user.
- **Correctness.** The public verifier is able to verify the integrity of shared data by challenging CS with randomness.
- **Soundness.** The cloud server cannot pass auditing process if the data is not intact.
- **Privacy preserving**. During integrity verification the TPA should not learn anything about data of the user.
- **Data dynamics.** Every group user is allowed to update the outsourced data remotely without downloading.

- **Group user revocation.** Cloud server transforms revoked user(s) blocks to designated existing group user blocks during the revocation. Revoked users should no longer update or sign the data in the cloud.

### 6.2.5 Security model

We designed CLPPPA scheme to withstand the four types of adversaries namely $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$ and $\mathcal{A}_4$ whose power has been defined in section 6.2.3. Among the four adversaries, both $\mathcal{A}_1$ and $\mathcal{A}_2$ try to forge the signature of blocks. $\mathcal{A}_3$ tries to generate the forged integrity proof and $\mathcal{A}_4$ tries to gain private data access during integrity auditing. The basic difference between $\mathcal{A}_1$ and $\mathcal{A}_2$ is that $\mathcal{A}_1$ cannot access the master key of the KGC, but can replace the public keys of any entity of his choice. $\mathcal{A}_2$ represents a malicious KGC who has the master key of the KGC, but cannot replace the public keys of users. Further these Type I and Type II can also be divided into into normal, strong, and super adversaries based on their attack power. Obviously, the super adversary has the better attack power than the other adversaries. Hence, we prove the security of CLPPPA, by considering the type I and type II adversaries and we define three interactive games Game 1, Game 2, Game 3 for $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$ respctively.

*Game 1 (played between $\mathcal{A}_1$ and a challenger $\mathcal{B}$)*:

Setup: Initially, $\mathcal{B}$ executes $Setup$ to obtain the $msk$ and $Pparams$. For super type I adversary $\mathcal{A}_1$, $\mathcal{B}$ just returns the public parameters $Pparams$; $\mathcal{B}$ keeps the $msk$ secret. $\mathcal{A}_1$ could access the following oracles controlled by challenger $\mathcal{B}$.

- Create_user_Oracle: On receiving a query with a different user's identity $ID$, $\mathcal{B}$ executes PartialPvtKeyGen, SetSecretValue, SetPublicKey to obtain partial private key, secret value and public key respectively. Finally, $\mathcal{B}$ then returns public key to $\mathcal{A}_1$.
- Partial_Private_Key_Oracle: On input of a query on the identity ID, $\mathcal{B}$ returns the partial private key to $\mathcal{A}_1$.

- Secret_Value_oracle: $\mathcal{B}$ returns the secret value to $\mathcal{A}_1$.
- Public_Key_Replacement_oracle: $\mathcal{B}$ replaces the user ID's original public key with a value of his choice.
- SignGen_oracle: $\mathcal{A}_1$ chooses the tuple (ID,m) and submits it $\mathcal{B}$. $\mathcal{B}$ executes SignGen algorithm to produce a signature $\sigma$ and sends it to $\mathcal{A}_1$
- Forge : Finally, adversary $\mathcal{A}_1$ outputs $\{\sigma^*, m^* \, ID^*\}$ as its forgery with identity $ID^*$. $\mathcal{A}_1$ is regarded to win this game if the following requirements are satisfied:
  - $1 \leftarrow ProofVerify(m, \sigma, Pparam, ID^*, PK^*_{ID})$
  - For $ID^*$ , the query Partial_Pvt_Key_oracle does not occur in the game before;
  - $\mathcal{A}_1$ has not submitted before the pair $(ID^*, m^*)$ to the $SignGen\_oracle$ with the public key $PK^*_{ID}$.

*Game 2 (played between $\mathcal{A}_2$ and a challenger $\mathcal{B}$)*:
Setup: Initially, $\mathcal{B}$ executes $Setup$ to obtain the $msk$ and $Pparams$. For super type II adversary $\mathcal{A}_2$, $\mathcal{B}$ returns both $msk$ and $Pparams$. $\mathcal{A}_2$ could access the following oracles controlled by challenger $\mathcal{B}$.

- Create_user_Oracle: On receiving from a different user's identity ($ID$) query , $\mathcal{B}$ executes SetSecretValue, SetPublicKey to obtain secret value and public key respectively. Finally, $\mathcal{B}$ returns public key to $\mathcal{A}_2$.
- Secret_Value_oracle: On input of a query on the identity ID, $\mathcal{B}$ returns the secret value to $\mathcal{A}_2$.
- Public_Key_oracle: $\mathcal{A}_2$ submits the query to $\mathcal{B}$. $\mathcal{B}$ executes the algorithm of SetPubKey to compute the public key of the ID and returns it to $\mathcal{A}_2$.
- SignGen_oracle: $\mathcal{A}_2$ adaptively chooses the tuple (ID, m) and submits it $\mathcal{B}$. $\mathcal{B}$ executes SignGen algorithm to produce a signature $\sigma$ for m and sends it to $\mathcal{A}_2$.
- Forge: Finally, adversary $\mathcal{A}_2$ outputs $\{\sigma^*, m^* \, ID^*\}$ as its forgery with the identity $ID^*$. If the following conditions are satisfied, $\mathcal{A}_2$ wins the game

- $1 \leftarrow ProofVerify(m, \sigma, Pparam, ID^*, PK^*_{ID})$
- For $ID^*$, the query Secret_Value_oracle does not occur in the game;
- $\mathcal{A}_2$ has never been submitted the pair $(ID^*, m^*)$ to the $SignGen\_oracle$.

*Game 3 (played by $\mathcal{A}_3$ and a challenger $\mathcal{B}$)*:

- Setup: $\mathcal{B}$ generates the $Pparams$, $msk$. $\mathcal{B}$ keeps the $msk$ secret, but sends $Pparams$ to $\mathcal{A}_3$.
- SignGen_Query: $\mathcal{A}_3$ selects the tuple $(ID, m)$ and sends it to $\mathcal{B}$ for querying the signature. $\mathcal{B}$ generates and returns the signature of m to $\mathcal{A}_3$ by the algorithm SignGen.
- Challenge: $\mathcal{B}$ generates Chal, sends it to $\mathcal{A}_3$ for getting $\mathcal{A}_3$ the appropriate proof P.
- Forge: For the Chal, $\mathcal{A}_3$ generates P and sends to $\mathcal{B}$. $\mathcal{A}_3$ wins the game, if P can pass the integrity check and the blocks in P is incorrect.

### 6.2.6 Algorithmic Framework

Here, we define proposed scheme algorithms.

- $Setup(1^\lambda) \rightarrow (Pparams, msk)$. It takes a security parameter $\lambda$ as input and outputs $msk$ and system public parameters $Pparams$.
- $Join(ID) \rightarrow \rho$. It takes the unique identity (ID) of user as input and outputs group joining key $\rho$ as output.
- $PartialPvtKeyGen(Pparams, msk, ID_i) \rightarrow D_i$. It takes the $Pparams$, $msk$, user identity $ID_i$ as input and outputs a partial private key $D_i$.
- $SetSecretValue(Pparams, ID_i) \rightarrow x_i$. It takes the $Pparams$, $msk$, user identity $ID_i$ as input and outputs a secret value $x_i$.
- $PvtKeyGen(D_i, x_i) \rightarrow S_i$. It takes $D_i$ and $x_i$ as input and outputs a private key $S_i$.

- $SetPubKey((Pparams, x_i) \rightarrow PK_i$. It takes $Pparams$, $x_i$ as input and outputs a public key $PK_i$.
- $SignGen(Pparams, S_i, M) \rightarrow \sigma_i$. It takes $Pparams$, $S_i$, and data blocks $\{m_i\}_{1 \leq i \leq n}$ as input and outputs a set of block signatures $\{\sigma_i\}_{1 \leq i \leq n}$ .
- $Challenge(M_{info}) \rightarrow C$. It takes the abstract information about the data as input and outputs the challenge $C$.
- $ProofGen(m_i, \{\sigma_i\}_{1 \leq i \leq n}, C) \rightarrow P$. It takes the file blocks $m_i$, the block signatures $\{\sigma_i\}_{1 \leq i \leq n}$ and $C$ as input and outputs a proof $P$.
- $ProofVerify(Pparams, PK_i, C, P) \rightarrow 0/1$. It takes the $Pparams, PK_i, C$ and the proof $P$ as input and returns 0 or 1.
- $Revoke(RL, \{id_1, id_2, ...id_k\}) \rightarrow RL'$. It takes current revocation list (RL) and user ID's as input and outputs the revised RL.
- $ReKey(S_i, S_j) \rightarrow S_{i \rightarrow j}$. It takes private key parts of revoked and non-revoked users and generates rekey $S_{i \rightarrow j}$ for resigning.
- $ReSignGen(\sigma_i, S_{i \rightarrow j}) \rightarrow \sigma_i'$. It takes signature $\sigma_i$ and rekey $S_{i \rightarrow j}$ as input and ouptputs the resignature $\sigma_i'$.
- $UpdateRequest(F_i', i, UO) \rightarrow UpdateReqInfo$. It takes new file block $F_i'$, the block position i and the update operation type UO as inputs, and outputs the update request information $UpdateReqInfo$. The UO may be insert, modify and delete.
- $ExecUpdate(UpdateReqInfo) \rightarrow \{1, 0\}$ . It returns 1 if the update operation is finished successfully, otherwise returns 0.

## 6.3 Detailed Construction

In this section, we present the detailed construction of CLPPPA scheme including data dynamics and revocation as follows:

### 6.3.1 Construction of CLPPPA

The detailed construction of CLPPPA is presented based on framework 6.2.6

**Setup** Given security parameter $\lambda$, $g$ is a generator of $G_1$. The KGC picks a random element $\alpha \in Z_p^*$, chooses a random and sets $g_0 = g^\alpha$. Also KGC picks $\gamma_0 \in Z_p^*$ randomly and sent to the GM for generating group joining key. Finally, two map-to-point cryptographic functions are chosen $H_1, H_2 : \{0,1\}^* \rightarrow G_1$, can map an arbitrary string $\{0,1\}^*$ into an element of $G_1$. Another hash function $h(.) : G_1 \rightarrow Z_p^*$ also chosen to map an element of $G_1$ to $Z_p^*$. The public parameter $Pparams = (p, G_1, G_T, \wp, g, g_0, H_1, H_2, h)$ is published i.e., made public to everyone and $\alpha$ is kept secret as master secret key .

**Join** Whenever a user wants to join a group, sends a request to the GM. Upon receiving the request, GM generates a group key as $\rho = g^{\gamma_0} \cdot H_1(ID)^{\gamma_0}$ and sends back $\rho$ to the authorized user in a secure way.

**PartialPvtKeyGen** Upon receiving the group key from GM, user $U_i$ sends his/her unique identity $ID_i$ and group key to KGC for partial private key generation. Then, KGC verifies the validity of the user by equation (1).

$$\wp(g, \rho) = \wp(g.H_1(ID), g^{\gamma_0}) \tag{6.1}$$

If the result is false then outputs $\perp$. Otherwise, KGC computes the partial private key ($D_i$) for the group user as follows:

(a) Compute $Q_i = H_1(ID_i) \in G_1$

(b) Compute $D_i = Q_i^\alpha$ and return to the user.

**SetSecretValue** After receiving $D_i$, user $U_i$ chooses $x_i \in Z_p^*$, $u \in G_1$ randomly and keeps $x_i$ as private secret value and makes $\beta \leftarrow u^{x_i}$ public.

**PvtKeyGen** After setting the secret value, the user $U_i$ combines $D_i$, and $x_i$ to generate actual private key $S_i = \{x_i, D_i\}$

**SetPublicKey** After generating $S_i$, the user $U_i$ computes public key as $PK_i = g^{x_i}$ with $Pparam$ and secret value $x_i \in Z_p^*$ .

**SignGen** After generating key pair $(S_i, PK_i)$, group user $U_i(1 \leq i \leq d)$ computes a signature for a block $m_j \in Z_p^*(1 \leq j \leq n)$ using private key $S_i$ as follows.

$$\sigma_j = H_2(W_j)^{x_i} \cdot (D_i \cdot u)^{m_j} \tag{6.2}$$

where $W_j = F_{id}||n||j$ and $F_{id}$ represents the file identity. Later, group user uploads blocks and corresponding signatures to the CSP.

**Challenge** After storing data into the cloud, the user request the TPA for data integrity verification. Upon receiving the request from user, TPA selects a subset $L = \{s_1, ..., s_c\}$ of c-elements from set $[1, n]$, and selects $v_i \in Z_p^*$ randomly for each $i \in L$. Let $C = \{(i, v_i)\}_{i \in L}$ be a challenge message generated for the cloud.

**ProofGen** Upon receiving the random challenge $C = \{(i, v_i)\}_{i \in L}$, the server computes a proof, which consists of data proof and signature proof as concretely.

(a) According to signature of each block, the CSP divides the challenged blocks in the set to $d$ disjoint subsets $\mathcal{L}_1$,$\mathcal{L}_2$...$\mathcal{L}_d$, where $\mathcal{L}_i$ is the subset of challenged blocks signed by group user $U_i$. Let $c_i$ is the count of elements in $\mathcal{L}_i$. So, the number of $c_i$ represents the no. of elements in subset $\mathcal{L}_i$. So, we have $c = \sum_{i=1}^{d} \mathcal{L}_i$, $\mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2 \cup \mathcal{L}_3 ... \cup \mathcal{L}_d$ and $\mathcal{L}_i \cap \mathcal{L}_j = \phi$ for $i \neq j$ signed by different users with different private keys.

(b) For each subset $\mathcal{L}_i$, CSP computes $(\bar{\sigma_i}, \bar{\mu_i})$ by

$$\bar{\mu_i} = \mu' + r \cdot h(R) \in Z_q^* \tag{6.3}$$

where

$$\mu' = \sum_{(i,v_i)_{i \in C}} v_i m_i \tag{6.4}$$

and $R = (\beta)^r \in G_1$ where $r \in Z_p^*$ is a random mask used to blind the data proof to preserve the data privacy against TPA. Meanwhile, the server also calculates

an aggregated signature for user $U_i$,

$$\bar{\sigma_i} = \prod_{(i,v_i)_{i \in C}} \sigma_i^{v_i} \in G_1 \tag{6.5}$$

(c) Then the server returns final proof $P = (\mu, \sigma, R)$ to the TPA as proof, where $\mu = (\bar{\mu_1}, \bar{\mu_2}...\bar{\mu_d}), \sigma = (\bar{\sigma_1}, ...\bar{\sigma_d})$.

**ProofVerify** The TPA verifies the integrity of outsourced data blocks after receiving proof for the challenge from the server by verifying the following equation.

$$\prod_{i=1}^{d} \wp\left(\bar{\sigma_i} \cdot R^{h(R)}, g\right) \stackrel{?}{=} \prod_{i=1}^{d} \wp\left(\prod_{(i,v_i) \in C} H_2(W_i)^{v_i}, PK_i\right) \wp\left(\prod_{i=1}^{d} H_1(ID_i)^{\bar{\mu_i}}, g_0\right) \tag{6.6}$$

If the equation holds, the blocks stored in cloud are properly maintained. Otherwise, the data is not intact.i.e., data is modified or deleted.

### 6.3.2 Support shared data dynamics

We describe dynamic operations such as block insertion ($B_{ins}$), block deletion ($B_{del}$) and block modification ($B_{mod}$) based on EDLIT for data dynamics.

**Block Insertion** : Assume the group user wants to insert block $m_x$ after the $i^{th}$ block $m_i$. At start, based on $m_x$ the group user computes the corresponding signature $\sigma_x$. Then, he generates an update request and sends new block and signature $B_{ins2C} = (ins, i, x, m_x, \sigma_x, V_x, S_i)$ to the server. After receiving $B_{ins2C}$ the insertion request, the server runs $ExecUpdate$ and inserts a corresponding file block $m_x$ after $m_i$ in the cloud; then the user sends the insertion instruction to the TPA $B_{ins2T} = (ins, i, x, V_x, S_i)$. Upon receiving the request, TPA updates the entries in EDLIT. The changes in the EDLIT can be found in Fig. 6.4.

**Block Modification**: Block modification refers to the replacement of specified block

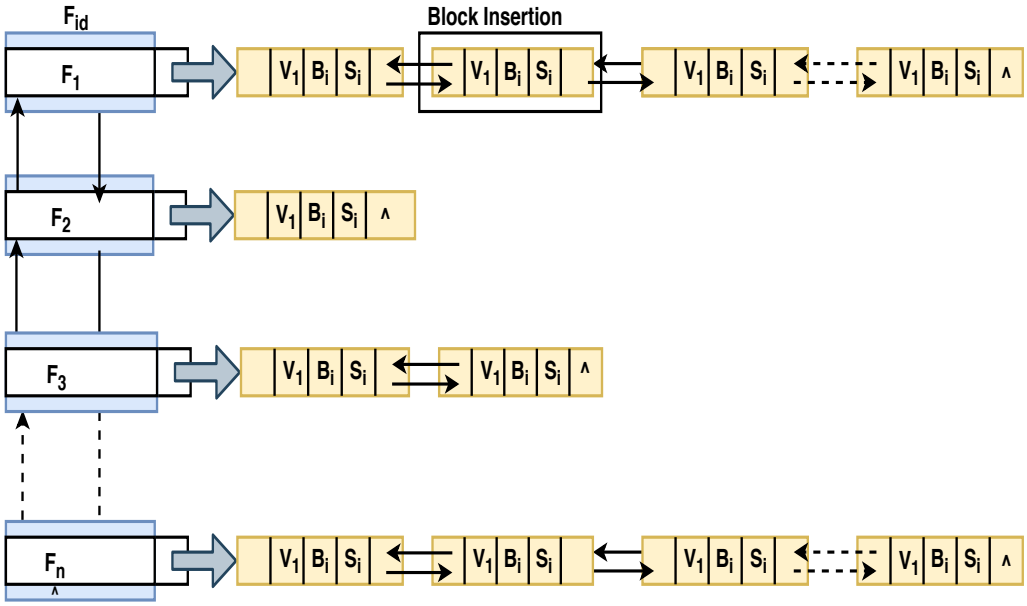



Fig. 6.4. Extended double linked list information table after block insertion

with a new one. To modify the $i^{th}$ block $m_i$ to $m_i'$, first, user generates new version number$V_i^*(V_i^* = V_i + 1)$ and signature $\sigma_i'$ for the new block $m_i'$. Then, user constructs an update request message $B_{mod2C} = (mod, i, m_i', \sigma_i', S_i)$ and sends to the server. After receiving$B_{mod2C}$ request, the server replaces the block $m_i$ with $m_i'$ and replaces the $\sigma_i$ with $\sigma_i'$. Then, the user sends $B_{mod2T} = (mod, i, V_i^*, S_i)$ to the TPA. TPA updates the EDLIT accordingly. For example, in Fig. 6.5, block 2 of file 1 is taken to show a block modification operation. It is clear that the version number of the data block is updated.

**Block Deletion**: Block deletion is just the opposite operation of block insertion. Suppose the server receives the update request $B_{del2C} = (del, i, S_i)$ for deleting block $m_i$ , it will execute the deletion instruction as shown in Fig. 6.6. Then user sends $B_{del2T} = (del, i, S_i)$ to the TPA. Upon receipt, the TPA would find $m_i$ and delete its information in the EDLIT. During deletion no new parameters are generated.

### 6.3.3 Secure group user revocation

In a group, it is common that users join and leave the group any time. Whenever a existing user is revoked from the group, the revoked users' pair of keys should be made

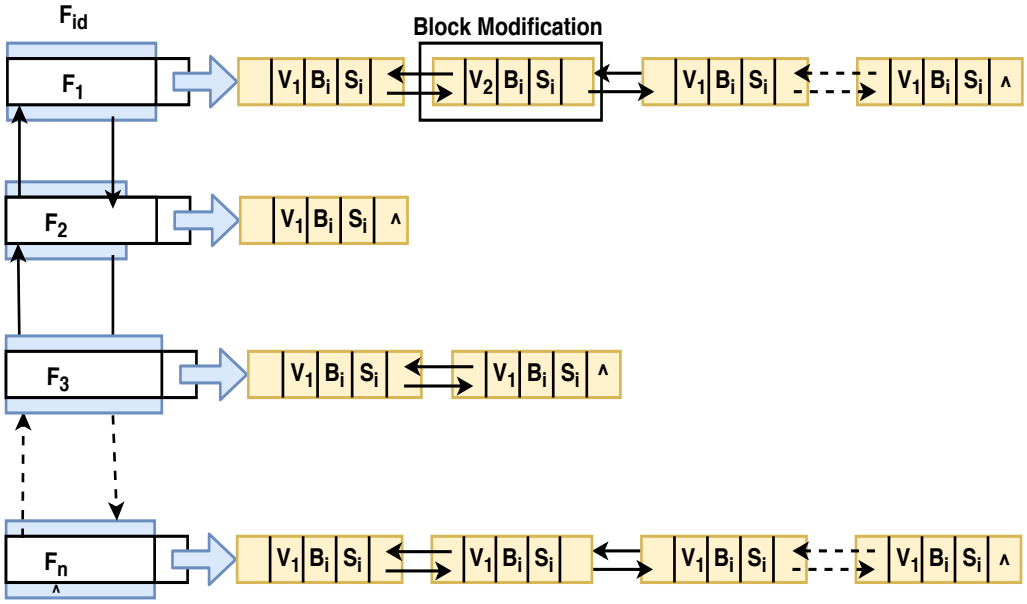



Fig. 6.5. Extended double linked list information table after block modification

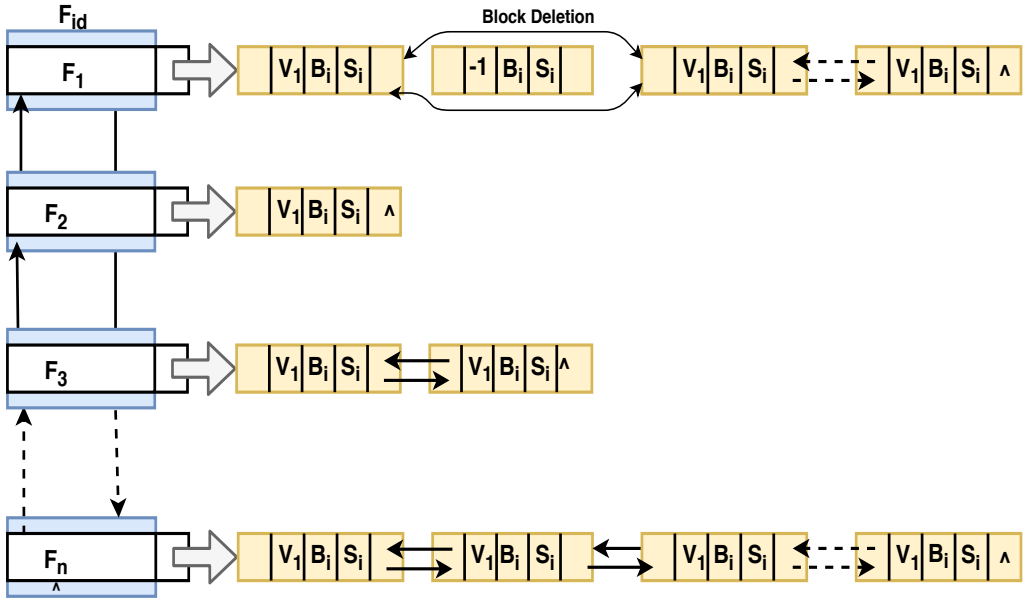



Fig. 6.6. Extended double linked list information table after block deletion

invalid to disable the access rights and signatures must be resigned by the existing user [83, 84]. The cloud server runs the algorithms $ReKey$, $ReSignGen$ to generate a rekey and update the revoked users' signatures during revocation upon receiving the the revised $RL'$ from GM. GM obtains $RL'$ by running $Revoke(RL, \{id_1, id_2, ...id_k\}) \rightarrow RL'$. The description of $ReKey$, $ReSignGen$ are given below, in which we treat $u_i$

and $u_k (1 \leq k \leq d, k \neq i)$ be the revoked user and a valid non-revoked user respectively in the group.

- **ReKey**: This algorithm involves some interactions among $u_i, u_k$ and cloud server. Besides, it is required that $u_i, u_k$ and cloud server are online simultaneously during the revocation process.

(a) cloud server chooses $\eta \in Z_p^*$ randomly and sends $\eta$ to $u_k$ in a secure way.

(b) $u_k$ computes and sends $(\omega_1 = (D_k)^{\frac{1}{x_k}}, \omega_2 = \eta \cdot x_k)$ to $u_i$.

(c) $u_i$ computes and sends $(R_1 = \dfrac{{\omega_1}^{x_i}}{D_i}, R_2 = \dfrac{\omega_2}{x_i})$ to cloud server.

(d) Upon receiving $(R_1, R_2)$, cloud server calculates $R_3 = \dfrac{R_2}{\eta} = \dfrac{x_k}{x_i}$ as rekey.

- **ReSignGen**: The cloud server transforms all signature-block pairs $[\sigma_{i'}, m_{i'}] (1 \leq i' \leq n)$ generated by $u_i$. That is the signature $\sigma_{i'}$ for the block $m_{i'}$ transformed as

$$\sigma_{i'}{}' = ({R_1}^{m_{i'}} \cdot \sigma_{i'})^{R_3}$$

The proof of correctness of above algorithm is as follows:

$$\begin{aligned}
\sigma_{i'}{}' &= \left({R_1}^{m_{i'}} \cdot \sigma_{i'}\right)^{R_3} \\
&= \left( \left( \frac{{D_k}^{x_i/x_k}}{D_i} \right)^{m_{i'}} H_2(W_{i'})^{x_i} (D_i \cdot u)^{m_{i'}} \right)^{\frac{x_k}{x_i}} \\
&= \left( {D_k}^{\frac{x_i \cdot m_{i'}}{x_k}} H_2(W_{i'})^{x_i} u^{m_{i'}} \right)^{\frac{x_k}{x_i}} \\
&= H_2(W_{i'})^{x_k} (D_k \cdot u)^{m_{i'}}
\end{aligned}$$

where $\sigma_{i'}{}'$ is the valid signature of $m_{i'}$ for the generating user $u_k$.

## 6.4 Security Analysis

The security of CLPPPA is proved in terms of completeness, soundness and comprehensive privacy preserving as described in section 6.2.5.

### 6.4.1 Correctness

**Theorem 1.** *In CLPPPA, the verifier successfully audit the integrity of data iff all the challenged file blocks and its corresponding signatures are intact in the cloud.*

**Proof**. The correctness of CLPPPA can be proved by verifying the Eq. 6.6, with the help of bilinear maps. The verification Eq. 6.6 can be elaborated as follows:

$\prod_{i=1}^{d} \wp\left(\bar{\sigma}_i \cdot R^{h(R)}, g\right)$

$$
\begin{aligned}
&= \prod_{i=1}^{d} \wp\left( \prod_{(i,v_i)\in C} \sigma_i^{v_i} \cdot (\beta)^{r\cdot h(R)}, g \right) \\
&= \prod_{i=1}^{d} \wp\left( \prod_{(i,v_i)\in C} \left( H_2(W_i)^{x_i} \cdot (D_i u)^{m_i} \right)^{v_i} \cdot (u^{x_i})^{r\cdot h(R)}, g \right) \\
&= \prod_{i=1}^{d} \wp\left( \prod_{(i,v_i)\in C} \left( H_2(W_i)^{x_i} \cdot \left( H_1(ID_i)^{\alpha} u \right)^{m_i} \right)^{v_i} \cdot (u)^{r\cdot h(R)}, g \right) \\
&= \prod_{i=1}^{d} \left( \wp\left( \prod_{i=1}^{s_c} H_2(W_i)^{v_i}, g^{x_i} \right) \wp\left( \prod_{i=1}^{s_c} H_1(ID_i) u^{m_i v_i} u^{rh(R)}, g^{\alpha} \right) \right) \\
&= \prod_{i=1}^{d} \left( \wp\left( \prod_{i=1}^{s_c} H_2(W_i)^{v_i}, g^{x_i} \right) \wp\left( H_1(ID_i) u^{\sum_{i=1}^{s_c} v_i m_i} u^{rh(R)}, g^{\alpha} \right) \right) \\
&= \prod_{i=1}^{d} \left( \wp\left( \prod_{i=1}^{s_c} H_2(W_i)^{v_i}, PK_i \right) \wp\left( H_1(ID_i) u^{\sum_{i=1}^{s_c} v_i m_i + rh(R)}, g_0 \right) \right) \\
&= \prod_{i=1}^{d} \wp\left( \prod_{i=1}^{s_c} H_2(W_i)^{v_i}, PK_i \right) \wp\left( \prod_{i=1}^{d} H_1(ID_i) u^{\bar{\mu_i}}, g_0 \right)
\end{aligned}
$$

From the above proof, we say cloud generates the valid proof for challenged blocks as the selected blocks were not corrupted. Thus cloud will not fail the auditing process launched by TPA.

### 6.4.2 Soundness

Here, we prove CLPPPA is unforgeable against $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$ as defined in Section 6.2.5

**Theorem 2:** *CLPPPA is secure against $\mathcal{A}_1$ if the CDH problem is hard in $G_1$.*

**Proof:** If $\mathcal{A}_1$ wins the Game 1 with a nonnegligible probability $\epsilon$ ; then, we could construct an algorithm that simulates a challenger $\mathcal{B}$ to solve the CDH problem with a nonnegligible probability. Initially, $\mathcal{B}$ contains two hash lists $L_{H_1}$ and $L_{H_2}$ and a public key list $L_{PK}$ which are empty initially. $\mathcal{A}_1$ and $\mathcal{B}$ interacts as follows.

- $H_1$-Query: If $\mathcal{A}_1$ makes an $H_1$-query with identity ID. $\mathcal{B}$ checks whether $L_{H_1}$ contains $(ID, PK_{ID})$. If it holds, $\mathcal{B}$ returns $H_1$; otherwise , $\mathcal{B}$ returns a random $H_1$ to $\mathcal{A}_1$ and then adds $(ID, X_{ID}, PK_{ID}, H_1)$ into $L_{H_1}$.
- $H_2$-Query: If $\mathcal{A}_1$ makes an $H_2$-query with identity ID. $\mathcal{B}$ checks whether $L_{H_2}$ contains $(ID, g_0, PK_{ID})$. If it holds, $\mathcal{B}$ returns $H_2$; otherwise , $\mathcal{B}$ returns a random $H_2$ to $\mathcal{A}_1$ and then adds $(ID, g_0, PK_{ID}, H_2)$ into $L_{H_2}$
- Setup: $\mathcal{B}$ produces the public parameters set including KGCs master public key to $\mathcal{A}_1$.
- PartialPvtKeyGen: Upon receiving a query with identity $ID$, $\mathcal{B}$ does the following.
  1.If $ID \neq ID^*$, $\mathcal{B}$ computes $H_1(ID, g_0) = v$ and then, store $(ID, g_0, PK_{ID}, v)$ into $L_{H_1}$.
  2) Return $D_{ID}$ .
- SecretValue: $\mathcal{B}$ looks up $L_{H_1}$ and returns $x_{ID}$.
- PublicKeyGen: $\mathcal{B}$ returns user's public key $PK_{ID} = (g^{x_{ID}})$ to $\mathcal{A}_1$.
- ReplacePublicKey: On receiving this query on $(ID, PK'_{ID})$, $\mathcal{B}$ returns $PK'_{ID}$ if it is already exists in $L_{PK}$; Otherwise, $\mathcal{B}$ replaces user's public key $PK_{ID}$ with $PK'_{ID}$ and then adds $(ID, PK'_{ID})$ into $L_{PK}$.
- SignGen : Upon receiving a query on $(ID, m)$, $\mathcal{B}$ finds $H_1$ and $H_2$ from $L_{H_1}$ and $L_{H_2}$ and computes the signature $\sigma$ for ID on m if $ID \stackrel{?}{=} ID^*$ and returns the result to $\mathcal{A}_1$. Otherwise , $\mathcal{B}$ aborts the game.

- Forge: Finally, $\mathcal{A}_1$ outputs a signature $\sigma'$ on a corresponding message $m\prime$. We then show the probability that $\mathcal{A}_1$ successfully wins the game as follows.

    1. $\mathcal{E}_1$: $\mathcal{B}$ does not abort Game 1 in query Partialpvtkeygen.
    2. $\mathcal{E}_2$:$\mathcal{A}_1$ outputs forgery of a signature $\sigma$ on m for ID.
    3. $\mathcal{E}_3$: After event $\mathcal{E}_2$ happens, the signature $\sigma$ satisfies ID = ID*.

From the above simulation, we have

$$Pr[\mathcal{E}_1] \geq (1 - \frac{pH_1}{p})^{pH_1}$$
$$Pr[\mathcal{E}_2|\mathcal{E}_1] \geq \epsilon$$
$$Pr[\mathcal{E}_3|\mathcal{E}_1 \wedge \mathcal{E}_2] \geq \frac{p_s}{p}$$

From these equations, the probability that $\mathcal{B}$ could solve the given CDH problem is

$$Pr[\mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \mathcal{E}_3]$$
$$= Pr[\mathcal{E}_1]Pr[\mathcal{E}_2|\mathcal{E}_1]Pr[\mathcal{E}_3|\mathcal{E}_1 \wedge \mathcal{E}_2]$$
$$\geq \frac{p_s}{p}(1 - \frac{pH_1}{p})^{pH_1}\epsilon$$

From above equation, we conclude that $\mathcal{B}$ cannot break the CDH problem since $\epsilon$ is non-negligible. Therefore, CLPPPA is secure against $\mathcal{A}_1$ in the random oracle model.

**Theorem 3**: *In the ROM, if $\mathcal{A}_2$ wins the Game 2 with a nonnegligible probability, then an algorithm $\mathcal{B}$ can find a solution to the CDH problem.*

**Proof:** If $\mathcal{A}_2$ wins the Game 2 with a nonnegligible probability $\epsilon$ ; then, we could constuct an algorithm that simulates a challenger $\mathcal{B}$ to solve the CDH problem with a nonnegligible probability. $\mathcal{B}$ generates a random number $\alpha \in Z_p^*$ as the master secret key, computes public key $g_0 = g^\alpha$, and returns public parameters $Pparams = \{p, G_1, G_T, \wp, g_0, g, H_1, H_2, h\}$ and the master secret key $\alpha$ to $\mathcal{A}_2$. $\mathcal{B}$ picks an identity ID as a challenge identity and answers the $H_1, H_2$ and $SignGen$ queries as it does in the proof of the previous theorem. $\mathcal{B}$ interact with $\mathcal{A}_2$ as follows.

- PartialPvtKeyGen: $\mathcal{B}$ computes $H_1(ID, g_0) = v$ and then store $(ID, g_0, v)$ into $L_{H_1}$ and then return $D_{ID}$ to $\mathcal{A}_2$.
- SecretValue: $\mathcal{B}$ looks up $L_{H_1}$ and returns $x_{ID}$ if $ID = ID^*$. Otherwise, $\mathcal{B}$ aborts the game.
- PublicKeyGen: Upon receiving this query, $\mathcal{B}$ returns the user's public key $PK_{ID} = (g^{x_{ID}})$ to $\mathcal{A}_2$.
- Forge: Eventually, $\mathcal{A}_2$ generates $(\sigma', m')$. We then show the probability that $\mathcal{A}_2$ successfully wins the Game 2 as follows.

    1. $\mathcal{E}_1$: $\mathcal{B}$ does not abort Game 2 in SecretValue query.
    2. $\mathcal{E}_2$: $\mathcal{A}_2$ outputs forgery of a signature $\sigma$ on m for ID.
    3. $\mathcal{E}_3$: After event $\mathcal{E}_2$ happens, the signature $\sigma$ satisfies ID = ID*.

From above process, we have

$$Pr[\mathcal{E}_1] \geq (1 - \frac{pH_1}{p})^{pH_1}$$
$$Pr[\mathcal{E}_2|\mathcal{E}_1] \geq \epsilon$$
$$Pr[\mathcal{E}_3|\mathcal{E}_1 \wedge \mathcal{E}_2] \geq \frac{p_s}{p}$$

From above equations, the probability that $\mathcal{B}$ could solve the given CDH problem is

$$Pr[\mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \mathcal{E}_3]$$
$$= Pr[\mathcal{E}_1]Pr[\mathcal{E}_2|\mathcal{E}_1]Pr[\mathcal{E}_3|\mathcal{E}_1 \wedge \mathcal{E}_2]$$
$$\geq \frac{p_s}{q}(1 - \frac{pH_1}{q})^{pH_1}\epsilon$$

we conclude that $\mathcal{B}$ cannot break the CDH problem since $\epsilon$ is nonnegligible. Thus $\mathcal{A}_2$ cannot win the Game 2. Therefore, CLPPPA is secure against $\mathcal{A}_2$ in the random oracle model.

**Theorem 4:** *Given shared data $\mathcal{M}$ and its signatures $\sigma$, it is hard for the cloud to forge the proof that can pass the verification under DL assumption.*

**Proof:** We prove this theorem 4 according to the security game defined in 6.2.5 as follows: **Game 3**: First, challenger $\mathcal{B}$ sends a challenge message $(i, v_i)_{i\in C}$ to $\mathcal{A}_3$, and the correct proof should be $P = (\mu, \sigma, R)$ return to $\mathcal{B}$ as proof, where $\mu = (\bar{\mu_1}, \bar{\mu_2}...\bar{\mu_d}), \sigma = (\bar{\sigma_1}, ...\bar{\sigma_d})$ which can pass the verification successfully with Eq. 6.6. Now, $\mathcal{A}_3$ computes an invalid auditing proof of $(\mu', \sigma, R)$ based on the incorrect data $\mathcal{M}'$, where $\mathcal{M}' \neq \mathcal{M}$ , and at least one element of $\Delta m_i = m_i{}' - m_i$ for $i \in C$ is nonzero. If $\mathcal{A}_3$ wins the Game 3, then, according to Eq. 6.6, we have

$$\prod_{i=1}^{d} \wp\left(\bar{\sigma}_i \cdot R^{h(R)}, g\right) \stackrel{?}{=} \prod_{i=1}^{d} \wp\left(\prod_{(i,v_i)\in C} H_2(W_i), PK_i\right) \wp\left(\prod_{i=1}^{d} H_1(ID_i) u^{\bar{\mu_i}'}, g_0\right)$$

Because $(\mu, \sigma, R)$ is a correct proof, we have

$$\prod_{i=1}^{d} \wp\left(\bar{\sigma}_i \cdot R^{h(R)}, g\right) \stackrel{?}{=} \prod_{i=1}^{d} \wp\left(\prod_{(i,v_i)_{i\in C}} H_2(W_i), PK_i\right) \wp\left(\prod_{i=1}^{d} H_1(ID_i) u^{\bar{\mu_i}}, g_0\right)$$

Based on bilinear maps

$$\prod_{i=1}^{c} u^{\bar{\mu_i}} = \prod_{i=1}^{c} u^{\bar{\mu_i}'} and \prod_{i=1}^{c} u^{\Delta\bar{\mu}} = 1$$

For any two random values $g, h \in G_1$, there exists $x \in Z_p^*$ such that $h = g^x$ because $G_1$ is a cyclic group. For the given $g, h \in G_1$, each $u$ can be randomly and correctly generated by computing $u = g^{\epsilon} \cdot h^{\xi} \in G_1$, where $g^{\epsilon}$ and $h^{\xi} \in Z_p^*$. Then, we have

$$1 = \prod_{i=1}^{c} u^{\Delta\mu} = \prod_{i=1}^{c} (g^{\epsilon} \cdot h^{\xi})^{\Delta\mu} = g^{\sum_{i=1}^{c} g^{\epsilon}\cdot\Delta\mu} \cdot h^{\sum_{i=1}^{c} g^{\xi}\cdot\Delta\mu}$$

Clearly, the solution for DL problem can be found. That is, if $\mathcal{A}_3$ wins Game 3, we can

find a solution to the DL problem with a probability of $1 - 1/p$, which is non-negligible. It contradicts the assumption in Section 2.3. Therefore, it is computationally infeasible, for $\mathcal{A}_3$ (malicious cloud) to output a forged auditing proof that can pass the verification.

### 6.4.3 Privacy preserving

It can be proven that $\mathcal{A}_4$ (TPA) could not learn user's data content during auditing process.

**Theorem 5:** *From the given cloud's auditing proof* $(\mu, \sigma, R)$ *, it is computationally infeasible for* $\mathcal{A}_4$ *to gain access to private information* $(\mu')$ *of shared data.*

**Proof:** Theorem 5 is proved in two steps. First, we show that no private data on $(\mu')$ can be derived from shared data $\mu$ it is masked by $r$ as $\bar{\mu_i} = \mu' + r \cdot h(R)$ where $\mu = (\bar{\mu_1}, \bar{\mu_2}...\bar{\mu_d})$ and $R = (\beta)^r$, where $r \in Z_p^*$ is randomly chosen by server and is hidden from auditor. Thus, privacy of $\mu'$ is protected from $\mathcal{A}_4$ even $\mu$ is given to $\mathcal{A}_4$ . Second, we prove that no private information on $\mu'$ can be obtained from $\sigma$, where $\sigma = (\bar{\sigma_1}, ...\bar{\sigma_d})$

$$
\begin{aligned}
\bar{\sigma_i} &= \prod_{(i,v_i)\in C} \sigma_i^{v_i} \in G_1 \\
&= \prod_{(i,v_i)\in C} \left( H_2(W_i)^{x_i} \cdot (D_i\ u)^{m_i} \right)^{v_i} \\
&= \prod_{(i,v_i)\in C} \left( H_2(W_i)^{v_i} \cdot (D_i\ u)^{m_i \cdot v_i} \right)^{x_i} \\
&= \prod_{(i,v_i)\in C} \left( H_2(W_i)^{v_i} \cdot (D_i)^{m_i \cdot v_i} (\ u)^{m_i \cdot v_i} \right)^{x_i} \\
&= \prod_{(i,v_i)\in C} \left( H_2(W_i)^{v_i} \cdot (D_i)^{m_i \cdot v_i} \cdot u^{\sum_{i=1}^{s_c} v_i m_i} \right)^{x_i} \\
&= \prod_{(i,v_i)\in C} \left( H_2(W_i)^{v_i} \cdot (D_i)^{m_i \cdot v_i} \right)^{x_i} \cdot (u^{\mu})^{x_i}
\end{aligned}
$$

Analysis: $(u^{\mu})^{x_i}$ is masked by $\prod_{(i,v_i)_{i\in C}} \left( H_2(W_i)^{v_i} (D_i)^{m_i v_i} \right)^{x_i}$.

However, to compute $\prod_{(i,v_i)_{i\in C}} \left( H_2(W_i)^{v_i} (D_i)^{m_i \cdot v_i} \right)^{x_i}$ from $\prod_{(i,v_i)_{i\in C}} (H_2(W_i)^{v_i} (D_i)^{m_i \cdot v_i})$ and $g^{x_i}$, which is the only information TPA can utilize, is a CDH problem. According to

CDH problem, it is infeasible for $\mathcal{A}_4$ (TPA) to derive private information.

### 6.4.4 Comparative summary

Here, we compare the security of CLPPPA with some of the existing PKI/ID-based CL-PKC based [82, 64, 62] schemes against $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$, $\mathcal{A}_4$ whose power has been defined in Section 6.2.5 and is presented in Table 6.1. As shown in Table 6.1, schemes [82, 64, 62] does not provide privacy protection. In addition, Wang et al.'s [82] scheme is vulnerable to Type I adversary attacks and He et al.'s [64] scheme does not provide formal proof against $\mathcal{A}_3$. Our CLPPPA, however, satisfies the requirements of public auditing scheme including privacy preserving and it is proven to be secure against all adversaries $\mathcal{A}_1$, $\mathcal{A}_2$ , $\mathcal{A}_3$, $\mathcal{A}_4$.

Table 6.1: Security comparison

| Schemes | $\mathcal{A}_1$ | $\mathcal{A}_2$ | $\mathcal{A}_3$ | $\mathcal{A}_4$ |
|---|---|---|---|---|
| Wang et al. [82] | No | Yes | Yes | No |
| He et al. [64] | Yes | Yes | No | No |
| Li et al. [62] | Yes | Yes | Yes | No |
| Proposed scheme | Yes | Yes | Yes | Yes |

*$\mathcal{A}_1$: super Type I Adversary, $\mathcal{A}_2$: super Type II Adversary , $\mathcal{A}_3$: Type III Adversary, $\mathcal{A}_4$: Type IV Adversary*

## 6.5 Performance Analysis

We evaluate the computation cost of CLPPPA theoretically, experimentally and compare it with the some of the state-of-the-art schemes [35, 43, 62].

### 6.5.1 Theoretical Analysis

First, we define some notations used in computation cost analysis:

1. $T_p$: one bilinear pairing operation
2. $T_{G1\_ex}$: one exponentiation operation on group $G_1$
3. $T_{G1\_mul}$: one multiplication operation on group $G_1$

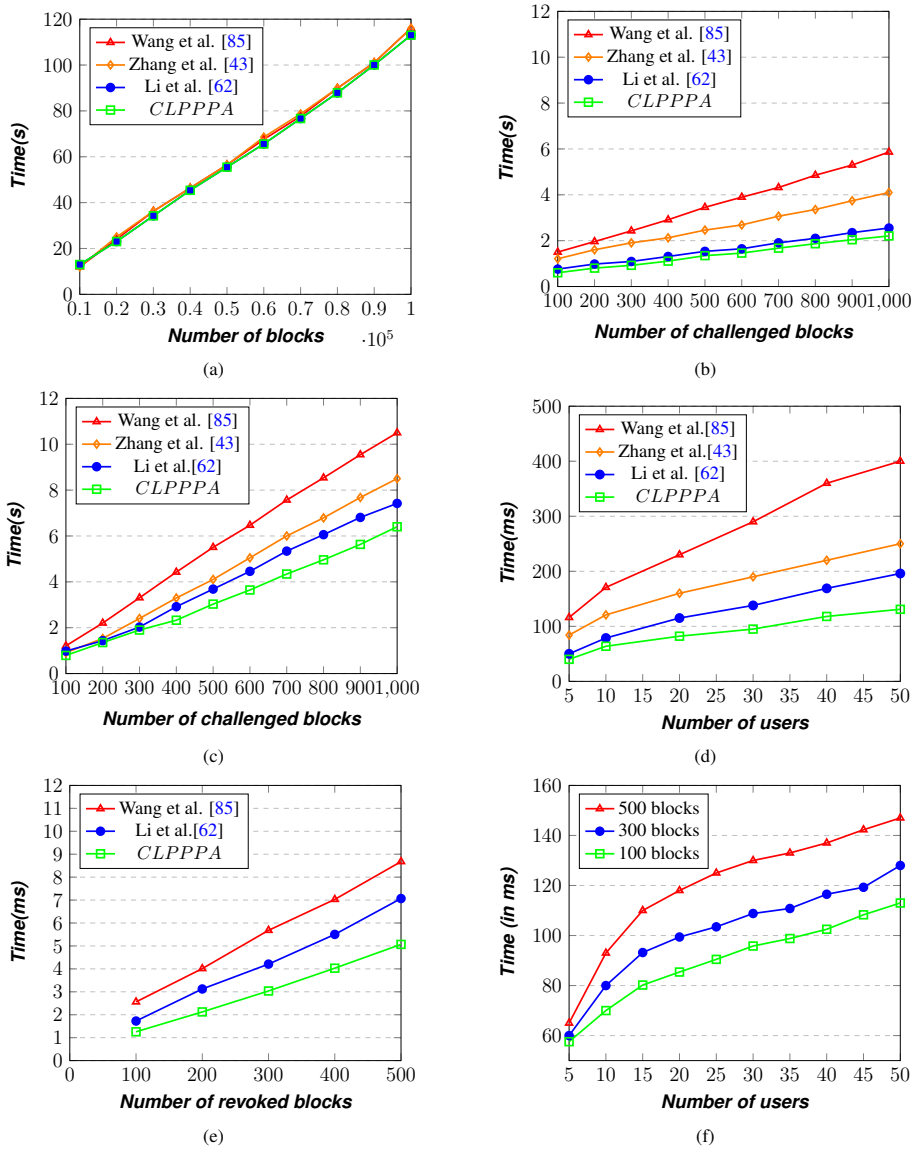



Fig. 6.7. **Computation cost: (a)** Computation Time of SignGen algorithm for different number of blocks **(b)** Time consumption of ProofGen algorithm **(c)** Time consumption of ProofVerify algorithm **(d)** Time consumption of verification for various number of users **(e)** Computation cost of ReSignGen algorithm for different number of revoked blocks **(f)** Computation cost of revocation process for different number of users

4. $T_{G2\_mul}$: one multiplication operation on group $G_2$.

5. c: the number of blocks in challenge.

6. d: user subsets for the challenge.

7. $H_s$: one hash operation in group $G_1$

8. $MulZ_p^*$: one multiplication operation in $Z_p^*$.

9. $AddZ_p^*$: one addition operation in $Z_p^*$.

Table 6.2: Computation cost comparison of SignGen, ProofGen, ProofVerify and ReSignGen

| Schemes | SignGen | ProofGen | ProofVerify | ResignGen | Type |
|---|---|---|---|---|---|
| Wang et al. [35] | $2T_{G1\_ex} + T_{G1\_mul} + Hs$ | $c \cdot T_{G1\_ex} + c \cdot T_{G1\_mul} + d(c-1)AddZ_p^*$ | $(c+d)T_{G1\_ex} + (c+2d)T_{G1\_mul} + (d+1)Tp + dT_{G1\_mul} + cHs$ | $2T_{G1\_ex} + T_{G1\_mul} + 2Tp + Hs$ | PKI |
| Zhang et al. [43] | $2T_{G1\_ex} + T_{G1\_mul} + Hs$ | $(c-1)T_{G1\_mul} + cT_{G1\_ex} + (c-1)Add_{Z_p^*} + cMul_{Z_p^*}$ | $cHs+(2c+2)T_{G1\_mul}+(2c+3)T_{G1\_ex}+2Tp+(c-1)Add_{Z_p^*} + cMul_{Z_p^*}$+ | ... | IBC |
| Li et al. [62] | $2(T_{G1\_ex} + T_{G1\_mul})$ | $c \cdot T_{G1\_ex} + c \cdot T_{G1\_mul}$ | $(d+2)T_p + (c+d)T_{G1\_ex} + (c+2d)T_{G1\_mul} + dT_{G2\_mul}$ | $R \cdot (2T_{G1\_ex} + T_{G1\_mul})$ | CL |
| Our scheme | $2(T_{G1\_ex} + T_{G1\_mul})$ | $c \cdot T_{G1\_ex} + (c-1) \cdot T_{G1\_mul}$ | $(d+2)T_p + dT_{G1\_ex} + (c+2d)T_{G1\_mul} + dT_{G2\_mul}$ | $R \cdot (T_{G1\_ex} + T_{G1\_mul})$ | CL |

We consider computation overhead mainly comes from bilinear pairings, exponentiation and multiplication on the group $G_1$ since our protocol CLPPPA is built from the bilinear pairings. To generate the signature for a block in CLPPPA, the user in the group needs to run $SignGen$ algorithm and whose computation cost is $2(T_{G1\_ex} + T_{G1\_mul})$. To generate the challenge message for the CSP, the verifier needs to run $challenge$ algorithm, which incur the negligible cost. Therefore, we ignore the the computation cost of $challenge$. The computation cost of CLPPPA is mainly generated by the proof generation phase and the proof verification phase and revocation phase. To generate the integrity proof P, server needs to execute the algorithm $ProofGen$, which requires $c \cdot T_{G1\_ex} + (c-1) \cdot T_{G1\_mul}$ computation cost. To check the data integrity, the TPA runs the algorithm $ProofVerify$ which requires $(d+2)T_p + dT_{G1\_ex} + (c+2d)T_{G1\_mul} + dT_{G2\_mul}$ computation cost. The revocation cost is $R \cdot (T_{G1\_ex} + T_{G1\_mul})$ and it depends on the number blocks signed by the revoked user in the file, where $R$ denotes the total count of signatures that needs to be updated. Moreover, we compare CLPPPA with the some of the existing schemes such as PKI-based [35], IBC-based [43] and CLPKC-based [62] and list the results in the Table

6.2. From Table 6.2, we can see that our proof generation, proof verification and resigning is efficient than [35, 43, 62] .

### 6.5.2 Experimental results

We implemented CLPPPA on a laptop with Intel i5-7200U CPU @ 2.50 GHz and 16 GB RAM. All experiments are carried out in python 2.7 language (PyCharm IDE) using crypto-0.42 library [79]. The implementation uses a symmetric super singular elliptic curve where the finite field size is 512-bit and security parameter fixed to 160-bits, that means, the length of the prime order $p$ in the experiments is 160 bits. The experimental results for signature generation ($SignGen$), proof generation ($ProofGen$), proof verification ($ProofVerify$) and revocation ($ResignGen$) are obtained and plotted as graphs from Fig. 6.7a to Fig. 6.7f.

#### 6.5.2.1 Computational costs for generating signatures

Fig. 6.7a shows the consumption cost of $SignGen$ algorithm. We set the group size to be 50 and the number of blocks ranges from 10,000 to 1,00,000. From Fig. 6.7a, in all four schemes, we observe that the time needed for $SignGen$ algorithm increases linearly as the number of blocks increases in the file. Both our scheme and [62] takes almost same time to generate signatures whereas [35, 43] requires slightly less time than [62] and our proposed scheme, since it has one less multiplication operation whose computation cost is much less than exponentiation operation's cost. Furthermore, sign generation is done once for the entire life-time of the scheme and brings little influence on the performance of integrity checking.

#### 6.5.2.2 Computational costs for proof generation

Fig. 6.7b shows computation cost of the $ProofGen$ algorithm against the number of challenged blocks in $Challenge$ message during verification. We increase the counter of challenged blocks from 100 to 1000 with an increment of 100 in each experiment. From Fig. 6.7b, we can learn that in all the four schemes, the time for proof generation is proportional to the block number; and for the same number of data blocks, CLPPPA spends relatively

less time than [35, 43, 62].

#### 6.5.2.3 Computational costs for proof verification

Fig. 6.7c and 8. 6.7d shows computation cost of the $ProofVerify$ algorithm against the number of challenged blocks and number of users in the group respectively. From Fig. 6.7c, in all four schemes, we observe that the time needed for $ProofVerify$ algorithm increases linearly as the number of blocks increases in the file. From Fig. 6.7c we can see that when the number of challenged blocks is 100, the time of proof verification takes about 1.1456s in all schemes.Likewise, if the count of number of challenged blocks is 1000 it needs nearly 10.3768901s, 8.2822234s, 7.832451s in [35], [43], [62] respectively, whereas in our scheme it takes only 6.402s. From this observation, we can say that our scheme takes relatively less time than [35, 43, 62]. Thus, our scheme is feasible for real-life applications. From Fig. 6.7d in all four schemes, we can learn that the verification time is proportional to the size of the group. Moreover, for the same number of users in the group, the verification time of our scheme is less than the half of that of [35] and relatively less than that of [43, 62].

#### 6.5.2.4 Computational costs for revocation

Fig. 6.7e depicts the computation cost of $ReSignGen$ algorithm for different number of blocks to be resigned by one of the existing valid user in the group. Here, we compare CLPPPA scheme with [35, 62] and ignored Zhang et al. [43] because they simply update the private key of all non-revoked users in the group, instead of resigning the blocks. From Fig. 6.7e, we can see that the cost of $ReSignGen$ in all schemes is linear with the number of revoked user blocks and CLPPPA scheme performs better than [35, 62]. Fig. 6.7f also depicts the computation overhead of revocation process with respect to different number of users. From Fig. 6.7f, we can see that the revocation cost is linear to the number of revoked users for different number of blocks. For example, for a group with 50 users consumes 130ms for resigning 300 blocks, and consumes 143ms for 500 blocks.

## 6.6 Summary

In this chapter, we presented a privacy preserving public auditing system for dynamic shared data storage in cloud computing by utilizing certificateless signatures. CLPPPA achieves privacy preserving against TPA by masking the data proof during auditing process while refrain from both certificate management and key escrow. Besides, CLPPPA also supports data dynamics through EDLIT and efficient user revocation. We formally proved the security of CLPPPA against super Type I, super Type II, Type III and Type IV adversaries under DL and CDH assumptions in ROM and it is proven that CLPPPA is more secure than existing schemes. The performance is evaluated by theoretical analysis and experimental results. The results shows that the CLPPPA is efficient and can be used in practice. Although CLPPPA achieves the privacy preserving and data dynamics along with integrity, it does not consider the data availability which is also an important issue to be considered in cloud storage. In the subsequent chapter, we propose a public integrity auditing scheme to ensure data availability.

# Chapter 7

# Certificateless Multi-Replica Public Integrity Auditing Scheme for Dynamic Shared Data in Cloud Storage (CLMRPIA)

## 7.1 Introduction

In this chapter, certificateless multi-replica public integrity auditing scheme is presented for dynamic shared data to achieve the data privacy, user revocation along with availability. The contributions are:

- In this scheme, we leverage certificateless signatures to generate signatures of multi-replica of shared data using the user's complete private key. This process simplifies the certificate management by allowing the verifier to check data integrity without managing certificates.
- To solve the key escrow problem, the user generates and uses the complete private key. The complete private key contains two components in which the first component (partial private key) is generated by KGC and the second one (secret value) is generated by the user itself. KGC only knows partial private key and does not know

secret value of the user. Therefore, the curious KGC cannot forge the user signature by any means.

- We propose a novel replica version table (RVT) to support shared data dynamic operations such as modification, insertion, and deletion.
- Our scheme also supports secure user revocation. i.e., whenever a user in the group misbehaves or leaves the group, the cloud can accomplish the resigning process.
- The security analysis proves the correctness, unforgeability of CLMRPIA against type I/II/III adversaries based on DL and CDH assumptions in ROM by simulating a game involving two players: a challenger and an adversary.
- The performance analysis evaluates the efficiency of CLMRPIA theoretically and experimentally in terms of computation and communication overheads.

## 7.2 Problem Statement

Here, we present problem statement, its description followed by the architecture, design goals, adversary model and the security model of the CLMRPIA scheme.

In this scheme, we focus on a multi-replica public integrity auditing scheme for shared data by leveraging CL-PKC [48] to eliminate the problems, namely, certificate management and key escrow. Consider a scenario in which a department manager who creates the data, user group and allows the other group users to share, access data on a given cloud server through the Internet. Later, every user can create multiple copies of the data. In a shared data pattern, users not only access but also modify the data for various purposes. Another essential characteristic to be considered in shared data is user revocation. That means, once a user in the group is revoked, all the signatures generated by a revoked user should be resigned by one of the existing non revoked users to ensure the correctness of data. Unfortunately, this problem has remained unexplored in previous researches. Hence, designing a multi-replica public integrity auditing scheme for shared data to support efficient data dynamics and user revocation while free from key escrow problem and complex

certificate management is a significant challenge.

In this scheme, first, the KGC generates the public parameters and $msk$ using security parameter ($\lambda$). Then publish the public parameters while keeps the $msk$ secret. Every user in the group submits his/her identity to the KGC to get the partial private key from KGC. According to the request of user, the KGC generates partial private key using master secret key and secretly sends it to the user. On receiving the partial private key from the KGC, the user generates his/her own complete private key by using combination of partial private key from KGC and randomly chosen secret key by himself/herself. Afterwards, GM creates a large raw data file, divides the file into $n$ raw data blocks, generates replica blocks for each original file data block. Next, computes signatures for all replica blocks using private key. After signing replica data blocks, GM uploads replica data blocks along with corresponding signatures to the cloud and deletes them from the local storage. Later, to check the integrity of multiple-replica shared data, TPA challenges the cloud server by selecting blocks randomly. After receiving the challenge, the cloud returns the proof of shared data as a response to the TPA. On receiving proof from the server, TPA verifies the correctness of data. Any group user may update outsourced data block without downloading the data at any time by forwarding an update request to a cloud. The CSP performs the operation and responds with an update proof to the user. Finally, whenever a user in the group misbehaves or quits the group, GM updates the existing $RL$ and forwards to the CSP. Upon receiving the updated $RL$ the CSP performs resigning on revoked user blocks.

### 7.2.1 Architecture

We consider certificateless cloud storage architecture with four entities, as illustrated in Fig.7.1.

- KGC is a third party entity which generates the $msk$ and the public parameters. After receiving ID of the group user, it generates the corresponding partial private key of the user using the master secret key ($msk$). KGC is assumed to be a semi-trusted, which means it is honest but curious. KGC honestly follows protocol, but it may try to replace the public key of the user.

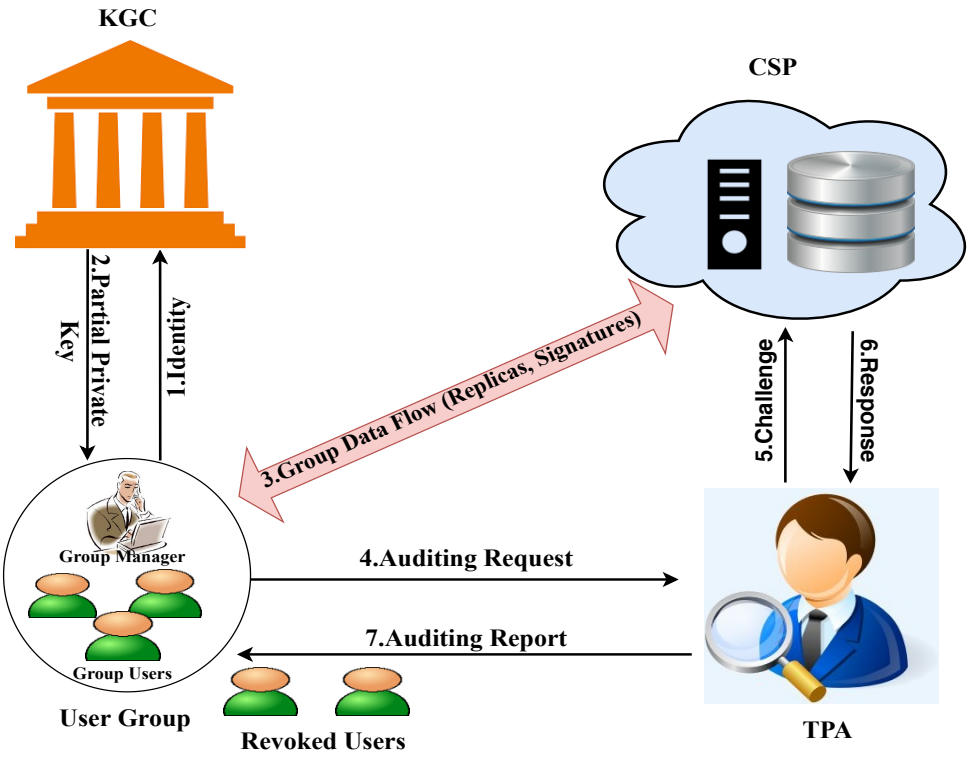



Fig. 7.1. The system model of certificateless multi-replica public auditing scheme

- TPA is assumed to be faithful by user and CSP. It is also called public verifier who has sufficient resources and professional capabilities to perform complete data auditing on behalf of the group users regularly or upon request. Upon receiving the auditing request from the user, TPA challenges the CSP for randomly selected blocks to check the integrity. After receiving proof from the CSP, TPA verifies the correctness of data.
- User Group. The user group includes group members and group manager (GM). GM is a trusted entity. The GM will divide the file into fixed sized data blocks and generate multiple replicas for the data blocks. We consider GM as the owner of the data. New members can join and quit the group anytime. There are multiple users in a group. We assume the legal group users are honest.
- CSP is an untrusted entity that has significant storage and computational resources, and it is responsible for maintaining user group data. CSP generates the proof and sends it to TPA as a response for integrity verification.

### 7.2.2 Adversary model

We designed CLPPPA scheme to withstand the four types of adversaries namely $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$ and $\mathcal{A}_4$.

- Type-I Adversary ($\mathcal{A}_1$): $\mathcal{A}_1$ (malicious outsider) tries to replace the user's public key with a false key even though he could not have access to KGC's master secret key ($msk$).
- Type-II Adversary ($\mathcal{A}_2$): $\mathcal{A}_2$ (malicious KGC) tries to mount an impersonation attack having access to the $msk$ of the KGC and it cannot replace the public key of the user. Even though the KGC is trusted entity, in a practical scenario, the KGC might engage in other adversarial activities such as eavesdropping on signatures and making signing queries, which is also known as Type II Adversary.
- Type-III Adversary ($\mathcal{A}_3$): $\mathcal{A}_3$ (malicious CSP) tries to compute a forged auditing proof that can pass the verification.
- Type-IV Adversary ($\mathcal{A}_4$): $\mathcal{A}_4$ (malicious TPA) tries to gain access to private information of data during audit process.

### 7.2.3 Design Goals

We propose CLMRPIA to achieve the following goals:

- **Correctness.** The public verifier can verify correctly the integrity of data by challenging CSP with randomness.
- **Data availability.** Data should always be available and retrievable in the cloud.
- **Public verifiability.** Any one-who knows public key and with sufficient resources can verify the integrity of data on behalf of user.
- **Soundness.** The cloud server never pass the auditor's auditing process if it does not possess the data intact.
- **Data dynamics.** Every group user is allowed to update the outsourced data remotely without downloading.

- **User revocation.** Whenever a user is revoked from the group, all signatures of the revoked user can be translated to the non revoked user signatures securely.

### 7.2.4 Security Model

We design CLMRPIA scheme to withstand the three types of adversaries namely $\mathcal{A}_1$ (represents malicious outsider), $\mathcal{A}_2$ (represents malicious KGC), $\mathcal{A}_3$ (represents malicious CSP). Both $\mathcal{A}_1$ and $\mathcal{A}_2$ try to forge the signature of blocks. $\mathcal{A}_3$ tries to generate the forged integrity proof. The basic difference between $\mathcal{A}_1$ and $\mathcal{A}_2$ is that $\mathcal{A}_1$ cannot access the master key of the KGC, but can replace the public keys of any entity of his choice. $\mathcal{A}_2$ represents a malicious KGC who has the master key of the KGC, but cannot modify the public keys of users. $\mathcal{A}_3$ tries to forge replicas and integrity proofs to cheat the verifier. We define games Game 1, Game 2, Game 3 formally between a challenger $\mathcal{B}$ and adversaries $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ respectively.

*Game 1 (for type I adversary $\mathcal{A}_1$ )*:

**Setup:** Initially, $\mathcal{B}$ executes $Setup$ to obtain the $msk$ and $params$. For type I adversary $\mathcal{A}_1$, $\mathcal{B}$ just returns the public parameters $params$ including the master public key; $\mathcal{B}$ keeps the $msk$ secret.

**Attack:** $\mathcal{A}_1$ could access the following oracles controlled by challenger $\mathcal{B}$. The adversary $\mathcal{A}_1$ can perform the following type of queries in an adaptive manner.

- Create_user_Oracle: This oracle takes ID as an input. Nothing will be returned by this oracle if identity ID has been created before. Otherwise, $\mathcal{B}$ executes PartialPvtKeyGen, SetSecretValue, SetPublicKey to obtain partial private key $D_i$, secret value $x_i$ and public key $PK_i$ for ID respectively. Finally, it adds $< ID, D_i, x_i, PK_i >$ to the $L_{PK}$ list and $\mathcal{B}$ returns public key $PK_i$ to $\mathcal{A}_1$.
- Partial_Private_Key_Oracle: On input of a query on the identity ID by adversary $\mathcal{A}_1$, $\mathcal{B}$ returns the partial private key $D_i$ to $\mathcal{A}_1$.
- Secret_Value_oracle: On input of a query on the identity ID by adversary $\mathcal{A}_1$, $\mathcal{B}$ returns the secret value $x_i$ to $\mathcal{A}_1$.
- Public_Key_Replacement_oracle: On receiving this query from $\mathcal{A}_1$, $\mathcal{B}$ replaces the

user ID's original public key $PK_i$ with a value of his choice $PK_i'$.

- ReplicaGen_oracle. $\mathcal{A}_1$ can query a selected file F to query $\mathcal{B}$ for the replica generation. $\mathcal{B}$ runs the ReplicaGen to produce a set of replicas. Subsequently sends it to $\mathcal{A}_1$.
- SignGen_oracle: $\mathcal{A}_1$ chooses the tuple $(ID, b_{i,j})$ and submits it to $\mathcal{B}$. $\mathcal{B}$ executes SignGen algorithm to produce a signature $\sigma_{i,j}$ and sends it to $\mathcal{A}_1$.

**Forgery**: Finally, adversary $\mathcal{A}_1$ outputs $\{\sigma_{i,j}^*, b_{i,j}^*\}$ as its forgery with identity $ID^*$. $\mathcal{A}_1$ is regarded to win this game if the following requirements are satisfied:

- $1 \leftarrow Verify(\sigma_{i,j}, b_{i,j}, param, ID^*, PK_{ID}^*)$
- For $ID^*$, the query Partial_Pvt_Key_oracle does not occur in the game before;
- $\mathcal{A}_1$ has not submitted never before the pair $(ID^*, b_{i,j}^*)$ to the $SignGen\_oracle$ with the public key $PK_{ID}^*$.

*Game 2 (for type II adversary)*:

**Setup:** Initially, $\mathcal{B}$ executes the Setup to obtain the $msk$ and $params$, and then returns both to type II adversary $\mathcal{A}_2$.

**Attack:** In this phase, $\mathcal{A}_2$ could access the following polynomially bounded number of oracles in an adaptive manner controlled by challenger $\mathcal{B}$.

- Create_user_Oracle: On receiving a query with a different user's $ID$, $\mathcal{B}$ executes SetSecretValue, SetPublicKey to obtain secret value and public key respectively. Finally, $\mathcal{B}$ returns public key to $\mathcal{A}_2$.
- Secret_Value_oracle: On input of a query on the identity ID, $\mathcal{B}$ returns the secret value to $\mathcal{A}_2$.
- Public_Key_Replacement_oracle: Upon receiving this query from $\mathcal{A}_2$, $\mathcal{B}$ replaces the user $ID_i$'s original public key $PK_i$ with a value of his choice $PK_i'$.
- ReplicaGen_oracle. $\mathcal{A}_2$ can query a selected file F to query $\mathcal{B}$ for the replicas. $\mathcal{B}$ runs the ReplicaGen algorithm to generate a set of replicas $\bar{F}$. Subsequently forwards it to $\mathcal{A}_2$.
- SignGen_oracle: $\mathcal{A}_2$ chooses the tuple $(ID, b_{i,j})$ and submits it to $\mathcal{B}$. $\mathcal{B}$ executes SignGen algorithm to produce a signature $\sigma_{i,j}$ and sends it to $\mathcal{A}_2$.

**Forgery**: Finally, adversary $\mathcal{A}_2$ outputs $\{\sigma^*_{i,j}, b^*_{i,j}\}$ as its forgery with the identity $ID^*$. $\mathcal{A}_2$ wins the game if this pair satisfies the following requirements:

- $1 \leftarrow Verify(\sigma_{i,j}, b_{i,j}, param, ID^*, PK^*_{ID})$
- For $ID^*$, the query Secret_Value_oracle does not occur in the game;
- $\mathcal{A}_2$ has never been submitted the pair $(ID^*, b^*_{i,j})$ to the $SignGen\_oracle$.
- $\mathcal{A}_2$ has not requested the $Public\_Key\_Replacement$ query on $ID^*$.

*Game 3 (for type III adversary)*:

- Setup: $\mathcal{B}$ executes the Setup algorithm to generate the parameters $params$ and the master secret key $msk$. $\mathcal{B}$ sends $params$ to $\mathcal{A}_3$ and keeps the $msk$ secret.
- SignGen_Query: $\mathcal{A}_3$ selects the tuple $(ID, m)$ and sends it to $\mathcal{B}$ for querying the signature. $\mathcal{B}$ generates and returns the signature of $m$ to $\mathcal{A}_3$ by running the SignGen algorithm.
- Challenge: $\mathcal{B}$ generates a challenging message Chal and sends it to $\mathcal{A}_3$ to get the corresponding proof P.
- Forge: Finally, for the Chal, $\mathcal{A}_3$ outputs a data integrity proof P and sends it to $\mathcal{B}$. $\mathcal{A}_3$ wins the game if P can pass the integrity check and the blocks in P is incorrect.

## 7.3 Algorithmic Framework

Here, we define proposed scheme algorithms.

- $Setup(1^\lambda) \rightarrow (params, msk)$. It takes $\lambda$ as input and outputs $msk$ and system public parameters $params$.
- $PartialPvtKeyGen(params, msk, ID_i) \rightarrow D_i$. It takes the $params$, $msk$, user identity $ID_i$ as input and outputs $D_i$ (partial private key).
- $SetSecretValue(params, ID_i) \rightarrow x_i$. It takes the $params$, user identity $ID_i$ as input and outputs a secret value $x_i$.
- $PvtKeyGen(D_i, x_i) \rightarrow S_i$. It takes $D_i$ and $x_i$ as input and outputs a private key $S_i$.
- $SetPubKey(params, x_i) \rightarrow PK_i$. It takes $params$, $x_i$ as input and outputs a public key $PK_i$.

- $ReplicaGen(F, S_i) \rightarrow F_{j_{(1\leq j\leq c)}}$. It takes original file F and copy number c as input, and generates c number of replicas. $F_j = (F_1, F_2, ..., F_c)$.
- $SignGen(params, S_i, F_j) \rightarrow \sigma_{i,j}$. It takes $params$, $S_i$, and data blocks $F_j = \{b_{i,j}\}_{1\leq i\leq n, 1\leq j\leq c}$ as input and outputs a set of block signatures $\{\sigma_{i,j}\}_{1\leq i\leq n}$ .
- $Challenge(M_{info}) \rightarrow C$. It takes the abstract information as input and outputs the challenge $C$.
- $ProofGen(b_{i,j}, \{\sigma_{i,j}\}_{1\leq i\leq n, 1\leq j\leq c}, C) \rightarrow P$. It takes the file blocks $b_{i,j}$, the block signatures $\{\sigma_{i,j}\}$ and $C$ as input and outputs a proof $P$.
- $ProofVerify(params, PK_i, C, P) \rightarrow 0/1$. It takes the $params, PK_i, C$ and the proof $P$ as input and returns 0 or 1.
- $UpdateRequest(F_i', i, UO) \rightarrow UpdateReqInfo$. It takes new file block $F_i'$, the block position $i$ and the update operation type UO as inputs, and outputs the update request information $UpdateReqInfo$. The UO may be insert, modify and delete.
- $ExecUpdate(UpdateReqInfo) \rightarrow \{1, 0\}$. It returns 1 if the update operation is completed successfully, otherwise returns 0.
- $Revoke(RL, \{ID_1, ID_2, ..., ID_m\}) \rightarrow RL'$. It takes the current revocation list ($RL$) and revoked user identities $\{ID_1, ID_2, ..., ID_m\}$ as input and returns updated RL ($RL'$). It is forwarded to CSP and TPA whenever a user is revoked from the group.

## 7.4 Detailed Construction

The details of the proposed CLMRPIA algorithms are as follows. The detailed process flow is illustrated in Fig.7.2.

**Setup**

Given security parameter $\lambda$. KGC randomly selects a big prime $q$ and two cyclic multiplicative groups $G_1$ and $G_T$ with order $q$. Let $g$ is a generator of $G_1$. $e$ is bilinear map of $G_1 \times G_1 \rightarrow G_T$. KGC gets a random element $\alpha \in Z_p^*$ and sets $g_0 = g^\alpha$. Finally, three map-to-point cryptographic functions are chosen $H_1, H_2, h(.) : \{0, 1\}^* \rightarrow G_1$, can map an arbitrary string $\{0, 1\}^*$ into an element of $G_1$. The security analysis views $H_1, H_2$ as the random oracle [86]. The public parameter $params = (p, q, G_1, G_T, e, g_0, g, H_1, H_2, h)$ is

published i.e., made public to everyone and $\alpha$ is kept secret as master secret key.

**PartialPvtKeyGen**

Authorized user $U_i$ sends its unique identity $ID_i$ to KGC for generating a partial private key ($D_i$). KGC computes ($D_i$) for the user as follows:

1. Compute $Q = H_1(ID_i) \in G_1$
2. Compute $D_i = Q^{\alpha}$. After computing ($D_i$), KGC forwards the $D_i$ to the user through a secure channel.

**SetSecretValue**

After receiving $D_i$, user $U_i$ selects $x_i \in Z_p^*$, $u \in G_1$ randomly and keeps $x_i$ as private secret value and makes $\beta \leftarrow u^{x_i}$ public. $\mathcal{S} = Z_p^*$ is the valid secret key value space.

**PvtKeyGen**

The user $U_i$ sets the combination of $D_i$ and $x_i$ as private key $S_i = (D_i, x_i)$.

**SetPublicKey**

The user $U_i$ computes public key as $PK_i = g^{x_i}$ with $param$ and secret value $x_i \in Z_p^*$. $\mathcal{PK} = G_1$ is the valid public key space.

**ReplicaGen**

The user divides the original data file $F$ into $n$ blocks, i.e., $F = (b_1, b_2, ..., b_n), b_{i\{1 \leq i \leq n\}} \in Z_p^*$. To ensure data availability, the user generates $c$ distinguishable replicas for the file $F$ as $F_j = (F_1, F_2, ..., F_c)$ with each $F_j = (b_{1,j}, b_{2,j}, ...b_{n,j})$. The replica block is generated as $b_{i,j} = b_i + \psi_k(i||j)$. Note that $\psi$ is a PRF with a key chosen randomly $k \in Z_p^*$ to differentiate replica of the files. Note that for any $F_j = (b_{1,j}, b_{2,j}, ...b_{n,j})$, the user can recover the original data file $F = (b_1, b_2, ..b_n)$ easily by computing $b_i = b_{i,j} - \psi_k(i||j)$. This allows the users seamlessly access the copy from the CSP. Finally, the user sends $k$ to the verifier and $k$ must be kept secret from the cloud.

**SignGen**

After generating replicas, user $U_i$ computes homomorphic certificateless signature $\sigma_{i,j}$ for each block $b_{i,j} \in Z_p^*(1 \leq i \leq n)$ using private key (combination of partial private key and

secret value) as follows.

$$\sigma_{i,j} = H_2(\omega_i)^{x_i} \cdot (D_i \cdot u)^{b_{i,j}} \tag{7.1}$$

where $\omega_i = (F_{id}||n||i)$ and $F_{id}$ denotes the file identity chosen from $Z_p$ by the user. It is to be noted that the $F_{id}$ is embedded into the block signature to prevent the CSP from cheating. Later, user uploads multi-replica file blocks $b_{i,j}$ and corresponding signatures $\sigma_{i,j}$ to the CSP. Finally, deletes $F$ from local storage. Also, user forwards revocation list ($RL = \{\emptyset\}$: Initially empty) to both the CSP and TPA to allow the user to update the replica blocks and to let the TPA to verify the updated replica blocks.

**Challenge**

After outsourcing data in cloud, the user request the TPA to check the integrity of the data. Upon receiving the request from user, TPA picks a nonempty subset $I \subseteq [1, n]$, and selects $v_i \in_R Z_p^*$ randomly, $\forall i \in I$. Suppose that the TPA wants to check $\bar{n} \subseteq n$ blocks with all $\bar{n}c$ replicas. The verifier chooses three temporary keys $\tau_1$, $\tau_2$, $\tau_3 \in Z_p^*$, sets the challenge token $C = (F_{id}, \bar{n}, \tau_1, \tau_2, \tau_3)$. Then the verifier forwards $C$ to the cloud.

**ProofGen**

After receiving the challenge token $C = (F_{id}, \bar{n}, \tau_1, \tau_2, \tau_3)$, server first computes $I = \{\pi_{\tau_1}(i)|i = 1, ..., \bar{n}\}$, $\{v_i\} = \{\psi_{\tau_2}(i)|i \in I\}$, $\{w_j\} = \{\psi_{\tau_3}(j)|j = 1, ..., c\}$, then it computes a proof which includes data proof and signature proof as follows:

1. CSP computes $\mu$ where

$$\mu = \sum_{j=1}^{c} \sum_{i \in I} w_j v_i b_{i,j} \tag{7.2}$$

Meanwhile, the server also calculates an aggregated signature for user $U$ for $c$ replicas,

$$\sigma = \Big( \prod_{i \in I} \sigma_{i,j}^{v_i} \Big)^{\sum_{j=1}^{c} w_j} \in G_1 \tag{7.3}$$

for every replica. Then the server returns final proof $P = (\mu, \sigma)$ to the TPA as a

response.

**ProofVerify**

After receiving the proof from the server, TPA verifies the integrity of outsourced data by verifying the following equation.

$$e\left(\sigma, g\right) \stackrel{?}{=} e\left(\prod_{i \in I} \prod_{j=1}^{c} w_j H_2(\omega_i)^{v_i}, PK_i\right) e\left(H_1(ID_i) u^{\mu}, g^0\right) \tag{7.4}$$

If the Eq.7.4 holds, the blocks stored in cloud are kept intact. Otherwise, the data is damaged or lost.

### 7.4.1 Dynamic Data Operations

In this phase, we perform dynamic operations such as modification, insertion and deletion using RVT, which is created by the user and maintained at the TPA side. To perform these operations user prepares and sends a request $UpdateRequest$ to the CSP in the general form $(F_{id}, Op, i, \{b_j^*\}_{1 \leq j \leq c}, \sigma_j^*, U_{ID})$ where $F_{id}$ is the file identifier, $Op$ denotes dynamic operation; that is 0 for update, 1 for insert, 2 for deletion, $i$ denotes the index of the block to be operated on, $\{b_j^*\}_{1 \leq j \leq c}$ is the new block value for all replicas and $\sigma_j^*$ is new signature for the new block, $U_{ID}$ is the signing user identity. Upon receiving the $UpdateRequest$, CSP verifies if the user is revoked or not with the help of $RL'$. If the user is revoked it returns $\perp$. Otherwise, CSP executes $ExecUpdate$ algorithm for data update operations as follows. Examples of different dynamic operations on multi replica shared data with our RVT are described in Table 7.1, 7.2 and 7.3.

#### 7.4.1.1 Modification

To update the file block $b_i$ with $b_i^*$ in a file $F = \{b_1, b_2, ...., b_n\}$, user specifies the index of the block for all the copies. The user prepares the update request by following the steps below:

1. Update the block version number value of $BVN_i = BVN_i + 1$ in RVT.

Table 7.1: Modifying block at position 3

| SN | $BRN_{(1\leq i\leq n, 1\leq j\leq c)}$ | BVN | $U_{ID}$ |
|---|---|---|---|
| 1 | $b_{11}$ | 1 | 1 |
| 2 | $b_{21}$ | 1 | 1 |
| **3** | $\mathbf{b_{31}}$ | **2** | **3** |
| 4 | $b_{41}$ | 1 | 1 |
| 5 | $b_{51}$ | 1 | 1 |
| 6 | $b_{61}$ | 1 | 1 |
| 7 | $b_{71}$ | 1 | 1 |
| 8 | $b_{81}$ | 1 | 1 |

Table 7.2: Insert block after position 5

| SN | $BRN_{(1\leq i\leq n, 1\leq j\leq c)}$ | BVN | $U_{ID}$ |
|---|---|---|---|
| 1 | $b_{11}$ | 1 | 1 |
| 2 | $b_{21}$ | 1 | 1 |
| 3 | $b_{31}$ | 2 | 3 |
| 4 | $b_{41}$ | 1 | 1 |
| 5 | $b_{51}$ | 1 | 1 |
| **6** | $\mathbf{b_{91}}$ | **1** | 3 |
| 7 | $b_{61}$ | 1 | 1 |
| 8 | $b_{71}$ | 1 | 1 |
| 9 | $b_{81}$ | 1 | 1 |

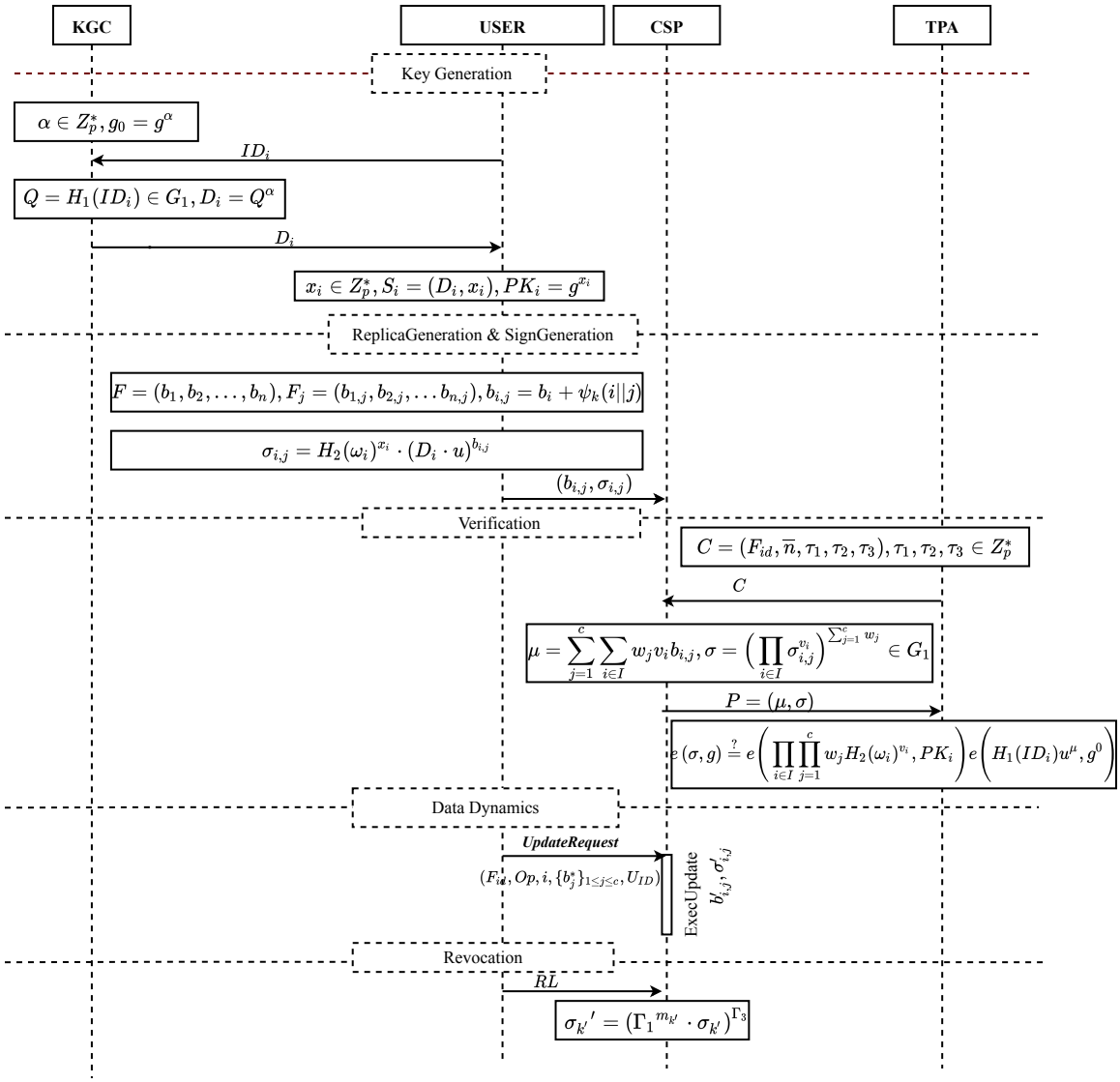



Fig. 7.2. Process flow of the proposed CLMRPIA scheme

2. Creates a new $c$ distinct file blocks $b'_{i,j}$, where $b'_{i,j} = b_i + \psi_k(i||j)$

3. Calculates a new signature $\sigma^*_{i,j}$ for the each new block $b'_{i,j}$ as $\sigma'_{i,j} = H_2(\omega_i)^{x_i} \cdot (D_i u)^{b'_{i,j}}$.

4. Sends a modify request $(F_{id}, 0, i, b'_{i,j\{1 \le j \le c\}}, \sigma'_i)$ to the CSP.

After receiving the update request from the user, CSP executes ExecUpdate to modify the data as follows:

1. It put back the block $b_{i,j}$ with $b'_{i,j} \forall j$ in the cloud storage.

Table 7.3: After deleting at position two

| SN | $BRN_{(1\leq i\leq n,1\leq j\leq c)}$ | BVN | $U_{ID}$ |
|---|---|---|---|
| 1 | $b_{11}$ | 1 | 1 |
| 2 | $b_{31}$ | 2 | 3 |
| 3 | $b_{41}$ | 1 | 1 |
| 4 | $b_{51}$ | 1 | 1 |
| 5 | $b_{91}$ | 1 | 3 |
| 6 | $b_{61}$ | 1 | 1 |
| 7 | $b_{71}$ | 1 | 1 |
| 8 | $b_{81}$ | 1 | 1 |

2. Replaces the $\sigma_i$ with $\sigma_i'$ in the signatures set $\sigma$, and updates the set.

#### 7.4.1.2 Insertion

Suppose the group user wants to insert a new block $b_i'$ after position $i$ in the file. Then signature for the new block can be constructed without recomputing the signatures for all the blocks because the serial number $SN$ of block is not included in the signature generation and the remaining blocks can be shifted to one position down. The procedure for inserting a new block after particular block is same as modification operation except that it will insert a new block after a particular block.

To insert of a new block $b_i'$ after position $i+1$ in all replicas of the file, the user prepares the update request, which does the following:

1. Creates new entry $(SN, BN, BVN) = (i+1, (Max\{BN_i\}_{1\leq i\leq n})+1, 1)$ and inserts this entry in the RVT after position $i$.
2. Creates $c$ number of new distinct blocks $\{b'_{i,j}\}_{1\leq j\leq c}$ where $b'_{i,j} = b_{i,j} + \psi_k(i||j)$
3. Calculates a new signature $\sigma_i^*$ for the each new block $b_i'$ as $\sigma'_{i,j} = H_2(\omega_i)^{x_i} \cdot (D_i u)^{b'_{i,j}}$.
4. Sends a insert request $(F_{id}, 1, i, b'_{i,j}\{1 \leq j \leq c\}, \sigma'_{i,j})$ to the CSP.

After receiving the insert request from the user, CSP inserts the data as follows:

1. Inserts $b'_{i,j}$ after $i^{th}$ position in all replicas.

2. Then add $\sigma'_{i,j}$ signature set $\sigma$ after $i^{th}$ position, and updates the signature set.

#### 7.4.1.3 Deletion

For deleting a file block, the group user sends a delete request to the CSP. To delete the requested file block, all subsequent blocks are moved one step forward. Suppose a group user wants to delete a block at position $i$, it sends a delete request $(F_{id}, 2, i, NULL, NULL)$ to CSP. Upon receiving a request, the CSP deletes corresponding block as follows:

1. Deletes the file block $\{b_{i,j}\}_{1 \leq j \leq c}$, and updates the replica files in cloud.
2. Similarly, deletes $\sigma_{i,j}$ from $\sigma$ and updates the signature list.

Table 7.1, 7.2 and 7.3 shows the dynamic operations such as insertion,modification, and deletion through the RVT. For example, if user 3 updates the third block, then its block version number $BVN$ is incremented by 1 which is shown in the Table 7.1. In the same way as shown in the Table 7.2, to insert a block after fifth position in $F$, a new record is inserted in the from $< 6, b_{91}, 1, 3 >$ where the serial number is 6, its block number is 91 and it is updated by user 3. All the subsequent entries after position 5 are shifted to one position down. The block number $BRN$ of newly inserted block is computed by adding 1 to the maximum block replica number value in the table. To delete a block entry at position 2, the serial numbers $(SN)$ of all the subsequent blocks after block 2 are decremented by 1 and all the entries are moved to one position up. $SN$ indicates the actual storage positions of data blocks of file F which is shown in Table 7.3.

### 7.4.2 User revocation

In shared data scenario, whenever a user is revoked from the group, the revoked user signatures must be resigned by one of existing authorized user to ensure the intactness of data. Let $u_k \, (1 \leq k \leq d, k \neq l)$ be the revoked user and $u_l$ valid non-revoked user respectively in the group. This procedure involves some interactions among $u_k, u_l$ and CSP. Besides, it is required that $u_k, u_l$ and CSP are online simultaneously during the revocation procedure.

Table 7.4: RVT after revocation

| SN | $BRN_{(1\leq i\leq n, 1\leq j\leq c)}$ | BVN | $U_{ID}$ |
|---|---|---|---|
| 1 | $b_{11}$ | 1 | 2 |
| 2 | $b_{31}$ | 2 | 3 |
| 3 | $b_{41}$ | 1 | 2 |
| 4 | $b_{51}$ | 1 | 2 |
| 5 | $b_{91}$ | 1 | 3 |
| 6 | $b_{61}$ | 1 | 2 |
| 7 | $b_{71}$ | 1 | 2 |
| 8 | $b_{81}$ | 1 | 2 |

1. CSP randomly chooses $\eta \in Z_p^*$ and sends $\eta$ to $u_l$ by a secure channel.
2. $u_l$ computes and sends $(\xi_1 = (D_l)^{\frac{1}{x_l}}, \xi_2 = \eta \cdot x_l)$ to $u_k$.
3. $u_k$ computes and sends $(\Gamma_1 = \frac{{\xi_1}^{x_k}}{D_k}, \Gamma_2 = \frac{\xi_2}{x_k})$ to CSP.
4. Upon receiving $(\Gamma_1, \Gamma_2)$, CSP calculates $\Gamma_3 = \frac{\Gamma_2}{\eta} = \frac{x_l}{x_k}$. Then the CSP transforms the tag $\sigma_{k'}$ for the block $m_{k'}$ (where $[\sigma_{k'}, m_{k'}](1 \leq k' \leq n)$ generated by $u_k$) as,

$$\sigma_{k'}{}' = ({\Gamma_1}^{m_{k'}} \cdot \sigma_{k'})^{\Gamma_3} \tag{7.5}$$

5. After resigning the blocks, CSP updates $RL$ to $RL'$.
6. TPA updates the existing $RL$ to $RL'$ and updates the RVT as shown in the Table 7.4.

Table 7.4 shows status of the table after revocation of the user 1 from the group.

## 7.5 Security Analysis

We perform the formal security analysis of CLMRPIA by considering type I/II/III adversaries. The security analysis of CLMRPIA depends on the hardness of DL and CDH problems in the ROM.

### 7.5.1 Correctness

As the data is not under the direct control of the user, every user wants to be assured that their are correctly maintained. The proposed scheme CLMRPIA provides integrity proof which identifies the data corruption at the CSP if some part of the data is modified or deleted by CSP.

**Theorem 1** *In the proposed scheme, if every entity performs honestly and correctly, the CSP passes the verification if all the challenged blocks are correctly stored at the server.*

**Proof**: We prove this by verifying the Eq. 7.4.

$$e\left(\sigma, g\right) \stackrel{?}{=} e\left(\prod_{i\in I}\prod_{j=1}^{c} w_j H_2(\omega_i)^{v_i}, PK_i\right) e\left(H_1(ID_i)u^{\mu}, g^0\right)$$

$$\begin{aligned}
LHS &= e\left(\sigma, g\right) \\
&= e\left(\left(\prod_{i\in I}\sigma_{i,j}^{v_i}\right)^{\sum_{j=1}^{c} w_j}, g\right) \\
&= e\left(\left(\left(\prod_{i\in I} H_2(\omega_i)^{x_i}\cdot(D_i u)^{b_{i,j}}\right)^{v_i}\right)^{\sum_{j=1}^{c} w_j}, g\right) \\
&= e\left(\left(\left(\prod_{i\in I} H_2(\omega_i)^{x_i.v_i}\cdot(D_i u)^{b_{i,j}\cdot v_i}\right)\right)^{\sum_{j=1}^{c} w_j}, g\right) \\
&= e\left(\left(\left(\prod_{i\in I} H_2(\omega_i)^{x_i.v_i}\cdot(D_i u)^{b_{i,j}\cdot v_i}\right)\right)^{\sum_{j=1}^{c} w_j}, g\right) \\
&= e\left(\prod_{i\in I} H_2(\omega_i)^{x_i.v_i\sum_{j=1}^{c} w_j}, g\right) e\left(\prod_{i\in I} H_1(ID_i)u^{\alpha b_{i,j}\cdot v_i\sum_{j=1}^{c} w_j}, g\right) \\
&= e\left(\prod_{i\in I}\prod_{j=1}^{c} w_j H_2(\omega_i)^{v_i}, g^{x_i}\right) e\left(H_1(ID_i)u^{\sum_{i=1}^{l} b_{i,j}\cdot v_i\sum_{j=1}^{c} w_j}, g^{\alpha}\right) \\
&= e\left(\prod_{i\in I}\prod_{j=1}^{c} w_j H_2(\omega_i)^{v_i}, PK_i\right) e\left(H_1(ID_i)u^{\sum_{i=1}^{l}\sum_{j=1}^{c} b_{i,j}v_i w_j}, g^{\alpha}\right) \\
&= e\left(\prod_{i\in I}\prod_{j=1}^{c} w_j H_2(\omega_i)^{v_i}, PK_i\right) e\left(H_1(ID_i)u^{\mu}, g^0\right) \\
&= RHS
\end{aligned}$$

Hence, data is stored correctly and maintaining with consistency.

### 7.5.2 User revocation correctness

The proof of correctness of Eq. 7.5 is as follows:

$$\begin{aligned}\sigma'_{k'} &= \left(\left(\frac{{D_l}^{x_k/x_l}}{D_k}\right)^{m_{k'}} H_2(\omega_k)^{x_k}(D_k \cdot u)^{m_{k'}}\right)^{\frac{x_l}{x_k}} \\ &= \left(D_l^{\frac{x_k \cdot m_{k'}}{x_l}} H_2(\omega_{k'})^{x_k} u^{m_{k'}}\right)^{\frac{x_l}{x_k}} \\ &= H_2(\omega_{k'})^{x_l}(D_l \cdot u)^{m_{k'}}\end{aligned}$$

where $\sigma'_{k'}$ is the valid signature of $m_{k'}$ by the non revoked user $u_l$.

### 7.5.3 Soundness

Here, we prove CLMRPIA is existentially unforgeable against $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$ as defined in Section 7.2.4

**Theorem 2:** *Our proposed scheme CLMRPIA achieves existentially unforgeability against adaptive chosen-message attack if for all probabilistic polynomial- time type I adversaries $\mathcal{A}_1$, the advantage of $\mathcal{A}_1$ winning the experiment is negligible in the ROM assuming CDH problem is hard in $G_1$.*

**Proof:** If $\mathcal{A}_1$ wins the Game 1 with a nonnegligible probability $\epsilon$ ; then, we could simulate a challenger $\mathcal{B}$ to solve the CDH problem with a non-negligible probability. Initially, $\mathcal{B}$ contains two hash lists $L_{H_1}$ and $L_{H_2}$ and a public key list $L_{PK}$ which are initially empty. $\mathcal{A}_1$ and $\mathcal{B}$ interacts as follows.

- $H_1$-Query: $\mathcal{A}_1$ can query an $H_1$ with identity ID. $\mathcal{B}$ verifies whether $L_{H_1}$ contains $(ID, PK_{ID})$. If it holds, $\mathcal{B}$ returns $H_1$; otherwise , $\mathcal{B}$ returns a random $H_1$ to $\mathcal{A}_1$ and then adds $(ID, X_{ID}, PK_{ID}, H_1)$ into $L_{H_1}$.
- $H_2$-Query: $\mathcal{A}_1$ can query an $H_2$ with identity ID. $\mathcal{B}$ verifies whether $L_{H_2}$ contains $(ID, g_0, PK_{ID})$. If it holds, $\mathcal{B}$ returns $H_2$; otherwise , $\mathcal{B}$ returns a random $H_2$ to $\mathcal{A}_1$ and then adds $(ID, g_0, PK_{ID}, H_2)$ into $L_{H_2}$.

- Setup: In this algorithm, $\mathcal{B}$ takes a secure parameter $\lambda$ and $\mathcal{B}$ produces the public parameters set including KGCs master public key and then sends it to $\mathcal{A}_1$.
- PartialPvtKeyGen: Upon receiving a PartialPvtKeyGen query from $\mathcal{A}_1$ with identity $ID_i$, $\mathcal{B}$ does the following.

  1. $\mathcal{B}$ returns $\perp$, if $ID_i$ has not been created.

  2.Else, if $ID_i$ has been created and $ID \neq ID^*$, $\mathcal{B}$ returns $D_i$ from $L_{PK}$. Otherwise, $\mathcal{B}$ returns failure and terminates.
- SecretValue: $\mathcal{A}_1$ can submit $ID_i$ to this oracle. $\mathcal{B}$ looks up $L_{PK}$ and returns $x_i$ if $ID_i$ has been created. Else $\mathcal{B}$ returns $\perp$.
- PublicKeyGen: On receiving such a query from $\mathcal{A}_1$ with identity $ID$, $\mathcal{B}$ returns user's public key $PK_{ID} = (g^{x_{ID}})$ to $\mathcal{A}_1$.
- ReplacePublicKey: $\mathcal{A}_1$ can submit $(ID_i, PK_i')$ to replace the public key with $ID_i$ to this oracle. If $ID_i$ has been created, $\mathcal{B}$ replaces user's original public key $PK_i$ with $PK_i'$ and then adds $(ID, PK_i')$ to $L_{PK}$. Otherwise, it outputs $\perp$.
- ReplicaGen : $\mathcal{A}_1$ selects a file F to query $\mathcal{B}$ for the replicas with the identity $ID_i$. $\mathcal{B}$ runs the $ReplicaGen$ algorithm to compute a set of replicas $\bar{F} = \{\bar{b_{i,j}}\}$ and returns it to $\mathcal{A}_1$.
- SignGen : $\mathcal{A}_1$ invokes a SignGen query for $b_{i,j}$. $\mathcal{B}$ ouputs $\perp$ if $ID_i$ has not been created. Otherwise, $\mathcal{B}$ uses the lists $L_{H_1}$, $L_{H_2}$ and $L_{PK}$ to compute the signature $\sigma_{i,j}$ for $ID_i$ on $b_{i,j}$.
- Forge: After all above queries, finally, $\mathcal{A}_1$ outputs a signature $\sigma_{i,j}^*$ on a block $b_{i,j}^*$. We then show the probability that $\mathcal{A}_1$ successfully wins the game as follows.

  1. $\mathbb{E}_1$: $\mathcal{B}$ does not abort Game 1 in query Partialpvtkeygen.
  2. $\mathbb{E}_2$: $\mathcal{A}_1$ outputs forgery of a signature $\sigma_{i,j}^*$ on $b_{i,j}^*$ for $ID_i$.
  3. $\mathbb{E}_3$: After event $\mathbb{E}_2$ happens, the signature $\sigma_{i,j}^*$ satisfies $ID_i = ID_i*$.

From the simulation, we have

$$Pr[\mathbb{E}_1] \geq (1 - \frac{pH_1}{p})^{pH_1}$$
$$Pr[\mathbb{E}_2|\mathbb{E}_1] \geq \epsilon$$
$$Pr[\mathbb{E}_3|\mathbb{E}_1 \wedge \mathbb{E}_2] \geq \frac{p_s}{p}$$

From these equations, the probability that $\mathcal{B}$ could solve the given CDH problem is

$$Pr[\mathbb{E}_1 \wedge \mathbb{E}_2 \wedge \mathbb{E}_3]$$
$$= Pr[\mathbb{E}_1]Pr[\mathbb{E}_2|\mathbb{E}_1]Pr[\mathbb{E}_3|\mathbb{E}_1 \wedge \mathbb{E}_2]$$
$$\geq \frac{p_s}{p}(1 - \frac{pH_1}{p})^{pH_1}\epsilon$$

From above equations, we conclude that $\mathcal{B}$ cannot break the CDH problem since $\epsilon$ is non-negligible. Hence, CLMRPIA is secure against adversary $\mathcal{A}_1$ in the ROM.

**Theorem 3**: *In the ROM, if type II adversary $\mathcal{A}_2$ wins the Game 2 in polynomial-time with a non-negligible probability, then there exists another algorithm $\mathcal{B}$ can resolve the CDH problem instance with a non-negligible probability.*

**Proof:** If $\mathcal{A}_2$ wins the Game 2 with a nonnegligible probability $\epsilon$; then, we could constuct an algorithm that simulates a challenger $\mathcal{B}$ to solve the CDH problem. $\mathcal{A}_2$ and $\mathcal{B}$ interacts as follows.

- Setup: Initially, $\mathcal{B}$ chooses a random value $\alpha \in Z_p^*$ as the $msk$, computes public key $g_0 = g^\alpha$, and returns public parameters $params = \{p, G_1, G_T, e, g_0, g, H_1, H_2, h\}$ and $\alpha$ to $\mathcal{A}_2$. $\mathcal{B}$ picks an identity ID as a challenge identity and answers the $H_1, H_2$ and $Replicagen$, $SignGen$ oracles as it does in the proof of the theorem 2. $\mathcal{B}$ interact with $\mathcal{A}_2$ as follows.
- SecretValue: $\mathcal{A}_2$ can submit $ID_i$ to this oracle. $\mathcal{B}$ looks up $L_{PK}$ and returns $x_i$ if $ID_i$ has been created. Else $\mathcal{B}$ returns $\perp$.
- ReplacePublicKey: $\mathcal{A}_2$ can submit $(ID_i, PK_i')$ to replace the public key with $ID_i$ to this oracle. If $ID_i$ has been created, $\mathcal{B}$ replaces user's original public key $PK_i$ with

$PK_i'$ and then adds $(ID, PK_i')$ to $L_{PK}$. Otherwise, it outputs $\perp$.

- Forge: Finally, $\mathcal{A}_2$ generates $\sigma_{i,j}^*$ for $b_{i,j}^*$. We then show the probability that $\mathcal{A}_2$ successfully wins the Game 2 as follows.

  1. $\mathbb{E}_1$: $\mathcal{B}$ does not abort Game 2 in SecretValue query.
  2. $\mathbb{E}_2$: $\mathcal{A}_2$ outputs forgery of a signature $\sigma_{i,j}^*$ on $b_{i,j}^*$) for ID.
  3. $\mathbb{E}_3$: After event $\mathbb{E}_2$ happens, the signature $\sigma_{i,j}^*$ satisfies ID = ID*.

From above process, we have

$$Pr[\mathbb{E}_1] \geq (1 - \frac{pH_1}{p})^{pH_1}$$
$$Pr[\mathbb{E}_2|\mathbb{E}_1] \geq \epsilon$$
$$Pr[\mathbb{E}_3|\mathbb{E}_1 \wedge \mathbb{E}_2] \geq \frac{p_s}{p}$$

From above equations, the probability that $\mathcal{B}$ could solve the given CDH problem is

$$Pr[\mathbb{E}_1 \wedge \mathbb{E}_2 \wedge \mathbb{E}_3]$$
$$= Pr[\mathbb{E}_1]Pr[\mathbb{E}_2|\mathbb{E}_1]Pr[\mathbb{E}_3|\mathbb{E}_1 \wedge \mathbb{E}_2]$$
$$\geq \frac{p_s}{q}(1 - \frac{pH_1}{q})^{pH_1}\epsilon$$

we conclude that $\mathcal{B}$ cannot break the CDH problem since $\epsilon$ is nonnegligible. Thus, $\mathcal{A}_2$ cannot win the Game 2. Therefore, proposed CLMRPIA is secure against $\mathcal{A}_2$ in the ROM.

**Theorem 4:** *In the CLMRPIA, it is computationally infeasible for malicious cloud or an adversary to forge a proof or signature for the given shared data $\mathcal{M}$ and its signatures $\sigma$, that can succeed in the verification process, if the CDH and DL problem is hard in bilinear group $G_1$.*

**Proof:** We prove this by knowledge proof method with the help of several games similar to the literature [78, 44]. If the CSP can pass the TPA's verification, then, it is possible to extract challenged data blocks by conducting repeated interactions. We can attain our proof by a series of games.

**Game a.** It simply represents the game defined for $\mathcal{A}_3$ in Section 7.2.4

**Game b.** It is similar to the Game a except one difference. Challenger $\mathcal{B}$ keeps a series of file tags he produces. If $\mathcal{A}_3$ submits one valid tag generated by SignGen() rather than by $\mathcal{B}$; $\mathcal{B}$ will abort the game.

Analysis: If $\mathcal{A}_3$ makes $\mathcal{B}$ termnate in Game b, it is easy to employ $\mathcal{A}_3$ to design an attacker to break the SignGen() algorithm. Therefore, we can conclude that all $F_{id}, m$ and $r$ that interact with $\mathcal{A}_3$ are generated by $\mathcal{B}$.

**Game c.** It is similar to Game b, with one distinction. Challenger $\mathcal{B}$ maintains a list of responses to the queries of adversary $\mathcal{A}_3$. If $\mathcal{A}_3$ wins the game, but with the aggregate signature $\sigma$ is not equal to $\sigma = \left(\prod_{i\in I} \sigma_{i,j}^{v_i}\right)^{\sum_{j=1}^{c} w_j} \in G_1$ generated by the challenger based on correct file, then challenger $\mathcal{A}_3$ will discontinue.

Analysis. Assume that $(\sigma, \mu)$ is a correct proof provided by the honest prover (honest cloud). From the correctness of the CLMRPIA, we know that following verification equation holds.

$$e\left(\sigma, g\right) = e\left(\prod_{i\in I}\prod_{j=1}^{c} w_j \cdot H_2(\omega_i)^{v_i}, PK_i\right) e\left(H_1(ID_i)u^{\mu}, g^0\right) \tag{7.6}$$

Assume that the adversary $\mathcal{A}_3$ provides a forged response $(\sigma', \mu')$. Because the forgery is successful, the following verification equation holds.

$$e\left(\sigma', g\right) = e\left(\prod_{i\in I}\prod_{j=1}^{c} w_j H_2(\omega_i)^{v_i}, PK_i\right) e\left(H_1(ID_i)u^{\mu'}, g^0\right) \tag{7.7}$$

Obviously, $\mu' \neq \mu$, otherwise $\sigma' = \sigma$, which contradicts our above assumption. Here, we define $\Delta\mu = \mu' - \mu(\Delta\mu \neq 0)$. We will build a simulator that finds a solution to the CDH problem if the adversary makes the challnger $\mathcal{B}$ abort with a non-negligible probability.

Now, dividing Eq.7.6 by Eq.7.7, we obtain $e(\frac{\sigma'}{\sigma}, g) = e(u^{\Delta\mu}, g^0) = e(g^a h^b)^{\Delta\mu}, g^0) = e(g^{a\Delta\mu}.h^{b\Delta\mu}, g^0)$ Thus, we can know that,

$$e(\sigma'.\sigma^{-1}.(g^{\alpha})^{-b\Delta\mu}, g) = e(h, g^{\alpha})a\Delta\mu \tag{7.8}$$

From the Eq.7.8, we know that $h^{\alpha} = (\sigma'.\sigma^{-1}(g^{\alpha})^{-b\Delta\mu})^{1/a\Delta\mu}$. The probability of $a\Delta\mu =$

$0\,mod\,p$ is $1/p$ which is negligible since $p$ is a very large prime. Therefore, we can solve the CDH problem with a probability of $1 - 1/p$, which contradicts the assumption that the CDH problem in $G_1$ is computationally infeasible.

**Game d**. It is the same as Game c, with one dissimilarity. As before, challenger $\mathcal{A}_3$ still keeps and observes CLMRPIA protocol instances. For one of these instances, If $\mathcal{A}_3$ wins the game and aggregate message $\mu'$ in the forged proof differs from the expected $\mu$ in the correct proof P, then the challenger $\mathcal{B}$ will discontinue.

Analysis. Assume that adversary $\mathcal{A}_3$ wins the Game d with non-negligible probability. We will construct a simulator that can use the adversary to solve the DL problem. The simulator is given $g, h \in G_1$, its goal is to find a value $\alpha$ satisfying $h = g^{\alpha}$. Furthermore, given $g, h \in G_1$, we can obtain $u = g^a h^b \in G_1$, where $a, b \in Z_p$. The simulator acts like $\mathcal{B}$ in Game c, but with the following differences:

Assume that $(\sigma, \mu)$ is a correct proof P provided by the honest server. From the correctness property of CLMRPIA, we know that the following equation holds.

$$e\left(\sigma, g\right) = e\left(\prod_{i \in I}\prod_{j=1}^{c} w_j H_2(\omega_i)^{v_i}, PK_i\right) e\left(H_1(ID_i)u^{\mu}, g^0\right) \tag{7.9}$$

Assume that $\mathcal{A}_3$ provides a forged proof $(\sigma', \mu')$, which is different from the honest one. Because the forgery is successful, the following equation holds.

$$e\left(\sigma', g\right) = e\left(\prod_{i \in I}\prod_{j=1}^{c} w_j H_2(\omega_i)^{v_i}, PK_i\right) e\left(H_1(ID_i)u^{\mu'}, g^0\right) \tag{7.10}$$

Based on the above two verification Eq. 7.9 and 7.10, we have $u^{\mu} = u^{\mu'}$, and can further imply that $1 = u^{\Delta\mu} = (g^a h^b)^{\Delta\mu} = g^{a\Delta\mu} \cdot h^{b\Delta\mu}$. Therefore, we can find the solution to the DL problem. More specifically, given $g, h^x \in G_1$, we can compute $h = g^x = g^{-\frac{a\Delta\mu}{b\Delta\mu}} = g^{-\frac{a}{b}}$. However, $b$ is zero only with the probability $1/p$, which is negligible because $p$ is a large prime. Then, we can get a solution to the DL problem with a probability of $1 - 1/p$ which contradicts the assumption that the DL problem in $G_1$ is computationally infeasible. Based on the above discussed games, we can conclude that the CLMRPIA achieves unforgeability under CDH and DL assumptions.

Finally, we construct a knowledge extractor to extract all challenged sets of $\overline{n}$ data blocks $b_{i,j}\,(i \in I, |i| = \overline{n})$ using $\overline{n}$ different sets of random coefficients $v_i\,(i \in I, |i| = \overline{n})$ and executing $\overline{n}$ times different challenges on the same data blocks $b_{i,j}\,(i \in I, |i| = \overline{n})$. The knowledge extractor can accumulate $\overline{n}$ different linear equations $\mu_1, ..., \mu_{\overline{n}}$. By solving these equations, the knowledge extractor can extract $b_{i,j}\,(i \in I, |i| = \overline{n})$. It means that if the CSP can pass the TPA's verification successfully, it must correctly maintain the user's data intact.

### 7.5.4 Comparative summary

We compare the security of CLMRPIA with some of the existing ID-based [65, 66, 87] schemes against $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$ whose power has been defined in Section 3.5 and is presented in Table 7.5. As shown in Table 7.5, schemes [65, 66, 87] does not free from key escrow problem. Our proposed scheme, however, satisfies the requirements of public auditing including key escrow and is proven to be secure against all adversaries $\mathcal{A}_1$, $\mathcal{A}_2$ , $\mathcal{A}_3$.

Table 7.5: Security comparison

| Schemes | $\mathcal{A}_1$ | $\mathcal{A}_2$ | $\mathcal{A}_3$ | Type |
|---|---|---|---|---|
| Peng et al. [65] | No | No | Yes | IBC |
| Peng et al. [66] | No | No | Yes | IBC |
| Yu et al. [87] | No | No | Yes | IBC |
| Proposed scheme | Yes | Yes | Yes | CL-PKC |

*$\mathcal{A}_1$: Type I Adversary, $\mathcal{A}_2$: Type II Adversary , $\mathcal{A}_3$: Type III Adversary*

## 7.6 Performance Analysis

In this section, we provide the performance evaluation and experiment results of proposed scheme.

### 7.6.1 Performance Evaluation

In this section, we evaluate the computation cost and communication cost of the proposed scheme theoretically as follows. For simplicity, we define some notations used for perfor-

mance assessment in Table 7.6.

Table 7.6: Notations

| Notation | Description |
|---|---|
| $n$ | Number of blocks |
| $c$ | Number of replicas |
| $C_{\psi}$ | Time cost of Pseudo Random Function |
| $C_{exp}$ | Time cost of single exponentiation on $G_1$ or $G_T$ |
| $C_{mul}$ | Time cost of single multiplication on $G_1$ or $G_T$ |
| $C_e$ | Time cost of single bilinear pairing |
| $\bar{n}$ | Number of challenged blocks |
| $C_{enc}$ | Time cost of encryption on raw data block |
| $C_H$ | Time cost of hash operation the on data blocks |
| $1Hash$ | Denotes the size of hash value in bits |
| $1Sig$ | Denotes the size of one digital signature in bits |
| $s$ | Denotes the sector number |
| $\mid G_1 \mid$ | Denotes the size of an element in $G_1$ |
| $\mid G_T \mid$ | Denotes the size of an element in $G_T$ |
| $\mid p \mid$ | Denotes the size of an element in $Z_p^*$ |

#### 7.6.1.1 Computation cost

We give the computation cost of the four algorithms, namely, $ReplicaGen$, $SignGen$, $ProofGen$, $ProofVerify$ which play the significant role in our proposed CLMRPIA scheme.

Assume that the data user in the group stores $c$ replicas on CSP. Each replica has $n$ blocks. To generate $c$ replicas, the algorithm $ReplicaGen$ needs to run and its computation cost is $ncC_{\psi}$. To generate the signatures for all replicas, the algorithm $SignGen$ costs $2ncC_{exp} + 2ncC_{mul}$. The CSP runs the $ProofGen$ to generate the integrity proof and its computation cost is $\bar{n}C_{exp} + \bar{n}cC_{mul}$. To check the integrity, TPA runs the algorithm $ProofVerify$ with a cost of $3Ce + 2\bar{n}(C_{exp} + cC_{mul})$. We summarize the computation

Table 7.7: Comparison of computation costs ( $n$ data blocks with $c$ replicas and $\bar{n}$ challenged blocks)

| Schemes | ReplicaGen | SignGen | ProofGen | ProofVerify |
|---|---|---|---|---|
| Barsoum et al. [53] | $cC_{enc}$ | $ncC_{exp} + n(c-1)C_{mul}$+$ncC_H$ | $\bar{n}C_{exp} + c(\bar{n}-1)C_{mul}$ | $2Ce + (\bar{n}c+1)C_{exp} + \bar{n}cC_{mul}$ |
| Liu et al. [54] | $ncC_{\psi}$ | $2ncC_{exp} + 2ncC_{mul}$+$ncC_H$ | $\bar{n}C_{exp} + c(\bar{n}-1)C_{mul}$ | $2cCe + (\bar{n}c+c)C_{exp} + \bar{n}cC_{mul}$ |
| Peng et al. [65] | $ncC_{\psi}$ | $(2n+1)C_{exp} + nC_{mul}$ | $1C_e + (\bar{n}+2)C_{exp} + \bar{n}cC_{mul}$ | $1Ce + (\bar{n}+1)C_{exp} + \bar{n}cC_{mul}$ |
| Yu et al. [87] | $cC_{enc}$ | $(2nc+1)C_{exp} + ncC_{mul}$ | $\bar{n}C_{exp} + c(\bar{n}+1)C_{mul}$ | $cCe + (\bar{n}+1)C_{exp} + (\bar{n}-1)cC_{mul}$ |
| Our scheme | $ncC_{\psi}$ | $2ncC_{exp} + 2ncC_{mul}$ | $\bar{n}C_{exp} + \bar{n}cC_{mul}$ | $3Ce + 2\bar{n}(C_{exp} + cC_{mul})$ |

overhead of our proposed scheme by comparing it with some of existing state of the art PKI based [53, 54] and ID-based [65, 87] multi-replica cloud auditing schemes from different aspects and the results are shown in the Table 7.7.

From Table 7.7, in our scheme, we can observe that $ReplicaGen$ algorithm is faster than [53, 87] and almost as fast as that of [54, 65] because pseudo-random functions are faster than symmetric encryptions. $SignGen$ and $ProofGen$ algorithms are faster than those of [53, 54, 65, 87]. Similarly, $ProofVerify$ algorithm is faster than those of [53, 54, 87] but slightly slower than [65] since it requires one more pairing operation, but our scheme does not suffer from key escrow problem.

#### 7.6.1.2 Communication Cost

Table 7.8 shows the communication cost of the proposed scheme. The communication cost refers to the costs used for transmitting a random challenge from TPA to CSP and the corresponding proof from CSP to TPA. It mainly comes from the integrity auditing phase includes three algorithms $Challenge, ProofGen, ProofVerify$ according to the description of Section 7.4. In the phase of integrity auditing, the TPA sends a challenge $C = (F_{id}, \overline{n}, \tau_1, \tau_2, \tau_3)$ to the cloud. The size of an auditing challenge is $(log_2 p + 3 \mid p \mid$

Table 7.8: Comparison of communication costs ( $n$ data blocks with $c$ replicas and $\bar{n}$ challenged blocks)

| Schemes | Challenge | Proof | Type |
|---|---|---|---|
| Barsoum et al. [53] | $2log_2p+ \mid p \mid$ | $1 \mid G_1 \mid +s\,c\,log_2\,p$ | PKI |
| Liu et al. [54] | $\bar{n}clog_2\,p+ \mid G_1 \mid$ | $\mid p \mid+\mid G_1 \mid +\bar{n}clog_2p+ 1Sig$ | PKI |
| Peng et al. [65] | $1 \mid G_1 \mid +1 \mid G_T \mid +3log_2p+ \mid F_{name} \mid$ | $1 \mid G_T \mid +1Hash + 1Sig$ | IBC |
| Yu et al. [87] | $\bar{n}log_2p + 3 \mid p \mid$ | $\mid p \mid +1Sig$ | IBC |
| Our scheme | $log_2p + 3 \mid p \mid + \mid F_{id} \mid$ | $\mid p \mid +1 \mid G_1 \mid$ | CLPKC |

$+ \mid F_{id} \mid)$ bits. On receiving the random challenge $C$ from the TPA, the CSP generates a corresponding proof $P = (\mu, \sigma)$ to reply the TPA. The size of the proof $P = (\mu, \sigma)$ is $\mid p \mid + \mid q \mid$ bits. Therefore, for one auditing task, the entire communication overhead is $(log_2\,p + 4 \mid p \mid + \mid F_{id} \mid + \mid q \mid)$ bits. We compare the communication overhead of our proposed scheme with some of existing PKI based [53, 54] and ID based [65, 87] multi-replica cloud auditing schemes. We summarize the result in Table 7.8.

From the Table 7.8, we can see that the communication cost for integrity proof in our scheme is smaller than all existing schemes [53, 54, 65, 87] because it returns only one element in $G_1$ and one element in $Z_p^*$. Similarly, the communication cost of the challenge message is lower than those of [54, 65, 87] and slightly higher than [53], but the overall communication cost of our scheme is smaller than all existing schemes [53, 54, 65, 87].

### 7.6.2 Experimental Results

We implemented CLMRPIA on a system with Intel i5-7200U CPU @ 2.50 GHz and 8 GB RAM. All experiments are carried out in python language using crypto-0.42 library [88]. The implementation uses a symmetric super singular curve where the base field size is 512-bit and security parameter ($\lambda$) fixed to 160-bits, which has the equivalent security level of

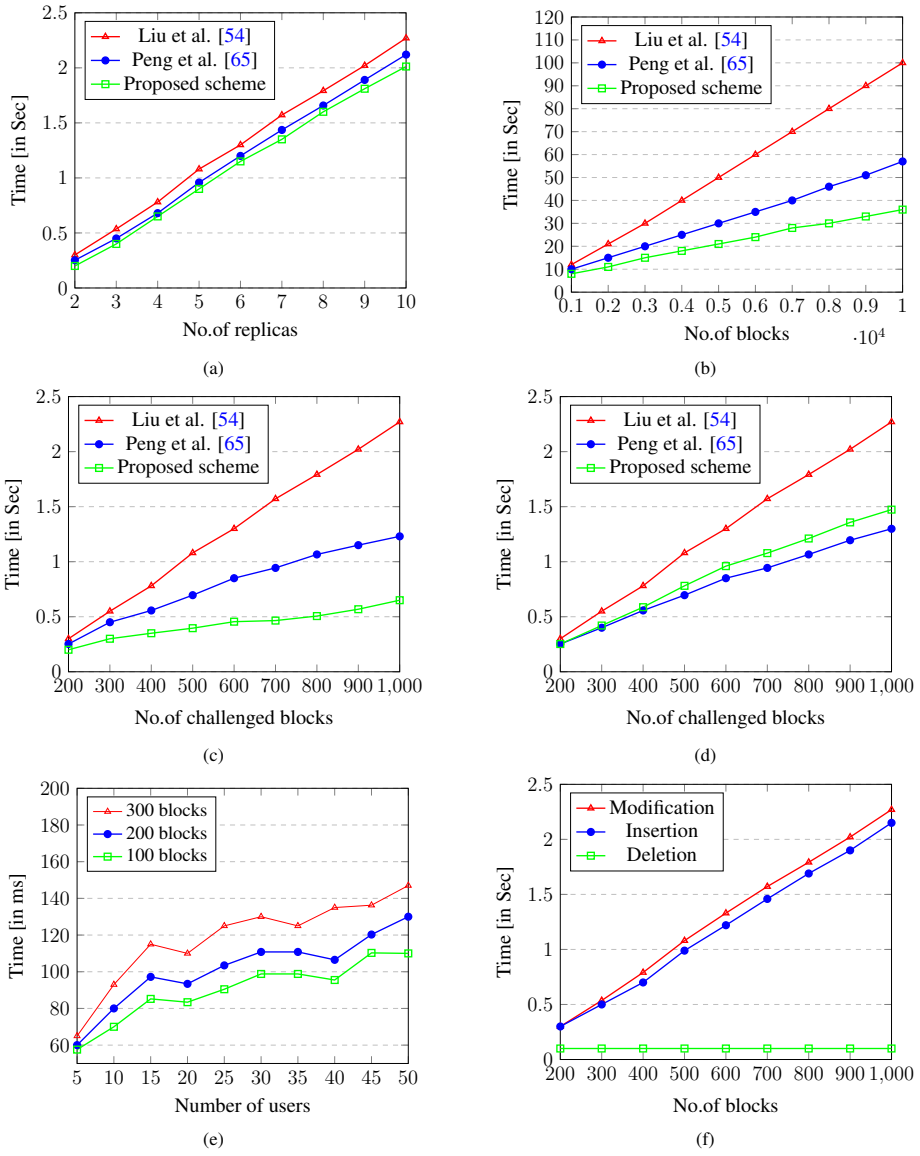



Fig. 7.3. **Computation cost: (a)** Computation Time of ReplicaGen algorithm **(b)** Time consumption of SignGen algorithm **(c)** Computation cost of ProofGen algorithm **(d)** Computation cost of ProofVerify algorithm **(e)** Computation cost of revocation process for different number of users **(f)** Computation cost of dynamic operations

1024-bit RSA. All results are mean of 20 trials. The experimental results for ReplicaGen, SignGen, ProofGen (Server-cost), ProofVerify (TPA-cost), revocation and data dynamics are obtained and plotted as graphs from Fig. 7.3a to Fig. 7.3f.

Fig. 7.3a shows the comparison of time consumption of ReplicaGen algorithm. From Fig. 7.3a, We observe that the time needed for ReplicaGen algorithm in all schemes increases linearly as the number of replicas increases in the system. The $ReplicaGen$ phase is almost as fast as that PKI-based schemes like MuR-DPA [54] and ID-based scheme such as [65], because computation of pseudo-random functions are faster than encryptions on the data file. Fig. 7.3b shows the comparison of time consumption of SignGen algorithm. From Fig. 7.3b, We notice that CLMRPIA is efficient than other existing schemes [54, 65] because of utilization of certificateless signatures. This is carried out only once during the life time of the system. Fig. 7.3c illustrates the comparison of computation costs for ProofGen. From Fig. 7.3c , it is obvious that our CLMRPIA has the better efficiency than existing PKI-based [54] and ID-based schemes [65]. Fig. 7.3d illustrates the comparison of computation costs of ProofVerify. From Fig. 7.3d, we can observe that verification time of CLMRPIA is lesser than PKI based scheme [54], since the CLMRPIA has less exponential and multiplication operations in group but consumes slightly more time than ID-based schemes [65]. Fig. 7.3e depicts the computation overhead of revocation process for different number of users against different number of blocks to be resigned. From Fig. 7.3e, we can see that the cost of revocation is linear to the number of users in the group. Fig. 7.3f. shows the computation cost of dynamic operations such as modification, insertion and deletion. From Fig. 7.3f, we can learn that the time of modification and insertion operations increases with the number of blocks and almost identical, while deletion takes negligible constant time because it requires no computations.

## 7.7 Summary

In this chapter, we have studied the problem of creating multiple copies of shared dynamic data file and verifying the integrity of the data on untrusted cloud servers.

In order to address the complex certificate management in existing PKI-based and key

escrow problems in ID-based data integrity auditing schemes, we proposed a novel certificateless multi-replica public integrity auditing scheme for dynamic shared data, where the user is capable of accessing and updating copies of blocks stored on the remote cloud servers. To the best of our knowledge, the proposed scheme is the first to simplify certificate management in PKI based schemes and eliminates key escrow problem in ID-based schemes simultaneously. A novel authenticated data structure, RVT was proposed to achieve efficient dynamic operations such as insertion, modification and deletion over data blocks in all replicas at once. It also supports secure user revocation when a user is revoked from the group. We proved the security of CLMRPIA against type I, type II and type III adversaries under the assumption that the DL and CDH problem are hard in ROM. The performance is evaluated by theoretical analysis, experimental results, and compared the results with the existing state of the art schemes. Extensive security and performance analysis proved that CLMRPIA is highly secure and efficient.

In future, the CLMRPIA can be further extended to support the feature of "error localization" and to support big data auditing.

# Chapter 8

# Efficient Pairing Free Certificateless Public Integrity Auditing for Shared Big Data in the Cloud (EPF-CLPA)

## 8.1 Introduction

In this chapter, we propose certificateless public integrity auditing for shared big data. The big data refers to the massive amount of data generated by digital devices (e.g., IoT, mobile devices), communication technologies (e.g., Internet, social networks), business applications, and many more [89]. To verify the integrity of shared big data, we propose a certificateless public auditing scheme for shared big data (EPF-CLPA) by leveraging elliptic curve cryptography (ECC) which does not require pairings. The contributions are:

- EPF-CLPA is designed based on ECC, which does not employ bilinear pairings. Hence, the computation and communication cost is substantially reduced.
- EPF-CLPA simplifies the certificate management and eliminates the key escrow problems exist in the PKI-based and ID-based PDP schemes, respectively.
- EPF-CLPA is further extended to support the batch auditing, where the TPA can handle multiple tasks concurrently. Since the cloud aggregates the multiple proofs and EPF-CLPA is pairing-free, the auditing performance is greatly improved.

- Additionally, our scheme also supports user revocation. Whenever a user is revoked, the GM will not generate the time key for the revoked user. Without an updated time key, any user cannot generate valid signatures for data blocks.
- We also performed security analysis that proves EPF-CLPA is secure against type I/II/III/IV adversaries based on the intractability of elliptic curve discrete logarithm problem (ECDLP).
- The theoretical analysis and related experimental results show that EPF-CLPA scheme is efficient than existing certificateless auditing schemes, and more suitable for shared big data auditing.

## 8.2 Problem Statement

Here, we present problem statement, its description followed by the architecture, design goals, adversary model and the security model of the EPF-CLPA scheme.

To avoid the costly certificates in PKI based schemes [34, 85, 76, 38, 75, 39, 40, 41] and to mitigate the key escrow problem in IBC [42, 43, 44] simultaneously, certificateless shared data auditing schemes [62, 63] have been proposed based on certificateless public key cryptography (CL-PKC)[48]. CL-PKC is a model for the use of public key infrastructure, which avoids the inherent escrow of identity-based cryptography and yet which does not require certificates to guarantee the authenticity of public keys. In such CLPKC schemes, the complete private key of the user consists of two parts, the first one is generated by the key generation center (KGC), and the user itself generates the second one (secret value). However, the schemes [62, 63] require pairing operations, causing a huge computation overhead because the pairing operation is computation intensive. In addition, they are very inefficient for the verifier to handle batch auditing. Hence, [62, 63] are not suitable for shared big data auditing as they incur a huge burden of computation. Moreover, in these schemes [62, 63] the revocation process involves resigning of all revoked data blocks, which further increases the computation overhead when applying for big data. Therefore, designing an efficient certificateless auditing scheme for shared big data without employing pairings is necessary.

In this scheme, initially, the KGC generates the public parameters $param$, master secret key ($msk$), and master time key using a system security parameter ($\lambda$). Then publish $param$ and keeps the $msk$ secret. To join the group, a user submits a request to the GM. Accordingly, the GM generates a group key and securely forwards it to the user. Then the user sends his/her identity and group key to the KGC in a secure way to get the partial private key $D_i$ from KGC. According to the request of the user, the KGC generates $D_i$ using $msk$, and $D_i$ will be sent to the corresponding user using a secure channel. On receiving $D_i$, the user generates his/her own complete private key by the combination of $D_i$ from KGC, time key from GM, and randomly chosen secret key by himself/herself. After creating the raw data file, the user divides and computes signatures using a private key for all blocks. Then user uploads data blocks along with corresponding signatures to the cloud and deletes them from the local hardware. Later, to check the integrity, TPA challenges the cloud by selecting blocks randomly. Upon receiving the random challenge, the cloud generates a corresponding proof. After receiving the proof, the TPA can check the correctness of data. Finally, when a user is revoked, GM updates the time-key of every non-revoked user and forwards existing $RL$ to the CSP.

### 8.2.1 Architecture

As shown Fig. 8.1, the system architecture of EPF-CLPA consists of four different entities.

1. User Group: A user group includes group members and a group manager (GM). GM is responsible for creating the group. We assume the GM and group users are honest. Each user can access outsourced shared data. The GM detects and refuses to send new time key for the revoked user to realize revocation.
2. CSP: It is an untrusted entity, which provides storage service and necessary computation resources.
3. TPA: It is assumed to be faithful by user and CSP. It has more expertise and capability than users to check the correctness of the data. Also convinces both cloud server and users by providing unbiased auditing results.

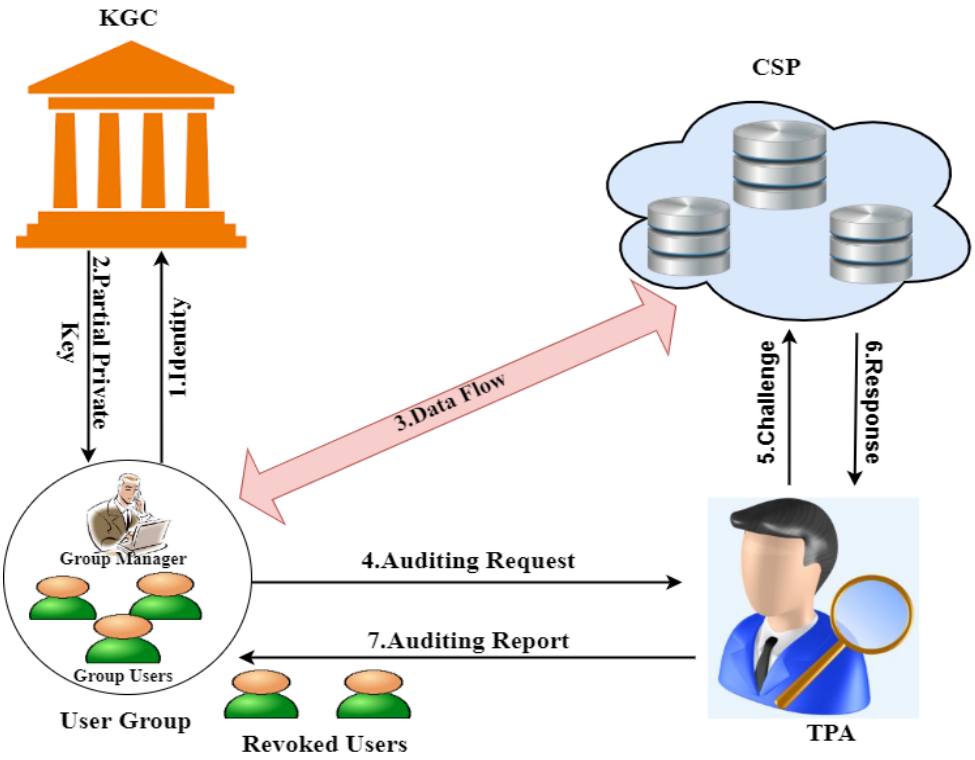



Fig. 8.1. System model of EPF-CLPA

4. KGC: An entity which generates the partial private key for the group user according to the unique identity of the user. Also generates master time key for the GM.

### 8.2.2 Design Goals

- **Public verifiability.** Anyone in the system who knows public key can verify the integrity of data in cloud.
- **Correctness.** The TPA can correctly verify the integrity of data by generating the challenge message for the CSP with randomness in the cloud.
- **Soundness.** The cloud server cannot pass auditing process if the data is altered.
- **User revocation.** Once a user is revoked from the group, all non revoked user's private key should be updated by sending the new time-key.
- **Batch auditing.**The verifier should be able to carry out several auditing tasks simultaneously from several users from the group.
- **Efficiency.** EPF-CLPA should reduce the computation overhead of auditing by em-

ploying the ECC.

### 8.2.3 Adversary Model

- Type-I Adversary ($\mathcal{A}_1$): $\mathcal{A}_1$ tries to replace the user's public key with a false key even though he could not have access to KGC's ($msk$) nor the user partial private key.
- Type-II Adversary ($\mathcal{A}_2$): $\mathcal{A}_2$ (malicious-but-passive KGC) tries to mount an impersonation attack having access to the $msk$ of the KGC.
- Type-III Adversary ($\mathcal{A}_3$): $\mathcal{A}_3$ (malicious revoked user) tries to generate a valid signature after time period $\Gamma_i$, but $\mathcal{A}_3$ cannot get the user time key after $\Gamma_i$.
- Type-IV Adversary ($\mathcal{A}_4$): $\mathcal{A}_4$ (malicious CSP) tries to generate a forged proof to pass the verification.

### 8.2.4 Security Model

Among the four adversaries, $\mathcal{A}_1$ and $\mathcal{A}_2$ and $\mathcal{A}_3$ try to forge the signature of blocks and $\mathcal{A}_4$ tries to forge the proof to pass the auditing. We define four games $Game$ 1, 2, 3 and 4, where $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$ and $\mathcal{A}_4$ interact with their corresponding challengers $C_1$, $C_2$, $C_3$ and $C_4$, respectively.

*Game 1 (for a type I adversary $\mathcal{A}_1$ )*

**Setup:** Initially, challenger $\mathcal{C}_1$ runs $Setup$ to get the $msk$ and $params$. Then, $\mathcal{C}_1$ gives $params$ to $\mathcal{A}_1$) And then, master time key $ts$ is forwarded to the GM by a secure channel and keeps the $msk$ to itself secretly.

**Queries:** $\mathcal{A}_1$ could access the following oracles adaptively controlled by challenger $\mathcal{C}_1$.

- $Create\_user\_Oracle$: It takes $ID_i \in \{0,1\}^*$ as an input. It returns nothing if ID has been created before. Otherwise, $\mathcal{C}_1$ executes PartialPvtKeyGen, SetSecretValue, SetPublicKey to obtain $D_i$, $x_i$ and $PK_i$ for $ID_i$. Finally, it adds $< ID_i, dD_i, x_i, PK_i >$ to the $L^{list}$ list and $\mathcal{C}_1$ returns $PK_A$ to $\mathcal{A}_1$.
- $Partial\_Private\_Key\_Oracle$: On receiving this query on $ID_i$ by adversary $\mathcal{A}_1$, $\mathcal{C}_1$ look up the list $L^{list}$ for the appropriate entry to the $ID_i$. If the tuple $< ID_i, dD_i, x_i, PK_i >$

is not found, $\perp$ is returned. Otherwise, the partial private key $dD_i$ is returned.

- $Secret\_Value\_oracle$: On input of a query on the identity ID by adversary $\mathcal{A}_1$, $\mathcal{C}$ returns the secret value $x_i$ to $\mathcal{A}_1$.
- $Time\_Key\_Update\_oracle$: $\mathcal{A}_1$ runs this oracle with $ID_i$ and $\Gamma_i$ as inputs, $\mathcal{C}_1$ runs the TimeKeyUpdate algorithm and returns a time key $d_{ID\Gamma_i}$ to $\mathcal{A}_1$.
- $Public\_Key\_Replacement\_oracle$: $\mathcal{A}_1$ can replace the user $ID_i$'s original $PK_i$ with a new value of his choice $PK_i^*$. $\mathcal{C}_1$ keeps record of this replacement.
- $SignGen\_oracle$: After receiving this request on $m_i$, a time period $\Gamma_i$, $\mathcal{C}_1$ executes SignGen algorithm to produce $\sigma_i$ for $m_i$ under identity $ID_i$ and returns it to $\mathcal{A}_1$.

**Forgery**: At last, $\mathcal{A}_1$ outputs $\{\sigma_i^*, m_i^*, \Gamma_i^*\}$ as its forgery with identity $ID_i^*$. $\mathcal{A}_1$ is regarded to win this game if the following conditions are satisfied:

- $1 \leftarrow Verify(\sigma_i, m_i, param, \Gamma_i^* ID^*, PK_i^*)$
- For $ID^*$, the query Partial_Pvt_Key_oracle has not been submitted;
- $\mathcal{A}_1$ has not submitted never before the tuple $(m_i^*, ID_i^*, \Gamma_i^*)$ to the $SignGen\_oracle$ with the public key $PK_{ID}^*$.

*Game 2 (for type II adversary $\mathcal{A}_2$)*:

**Setup:** Initially, challenger $\mathcal{C}$ obtain $msk$ and $params$ by running the Setup, and then, returns both to $\mathcal{A}_2$.

**Queries:** $\mathcal{A}_2$ could access the all oracles ( similar to $\mathcal{A}_1$) except Partial_Pvt_Key_Oracle as defined earlier in Game 1 because $\mathcal{A}_2$ knows $msk$.

**Forgery**: Eventually, adversary $\mathcal{A}_2$ outputs tuple $\{\sigma_i^*, m_i^*, \Gamma_i^*\}$ as its forgery with the identity $ID_i^*$. $\mathcal{A}_2$ is said to win the game if this tuple satisfies the following conditions:

- $1 \leftarrow Verify(\sigma_i, m_i, param, \Gamma_i^*, ID^*, PK_{ID}^*)$
- For $ID^*$, the query $Secret\_Value\_oracle$ does not occur in the game;
- The tuple $(m_i^*, ID^*, \Gamma_i^*)$ has never been sent to the $SignGen\_oracle$.
- $\mathcal{A}_2$ has never requested the $Public\_Key\_Replacement$ query on $ID^*$.

*Game 3 (for $\mathcal{A}_3$)*:

**Setup:** It is similar to Game 1.

**Queries:** In this phase, $\mathcal{A}_3$ could access all the oracles ( similar to that of $\mathcal{A}_1$) as defined in Game 1.

**Forgery**: Finally, adversary $\mathcal{A}_3$ outputs tuple $\{\sigma_i^*, m_i^*, \Gamma_i^*\}$ as its forgery with the identity $ID_i^*$ and wins the the game if the following requirements are satified:

- $1 \leftarrow Verify(\sigma_i, m_i, param, \Gamma_i^*, ID^*, PK_{ID}^*)$
- The tuple $(m_i^*, ID^*, \Gamma_i^*)$ has not been sent to the oracle $SignGen$.
- $(ID_i^*, \Gamma_i^*)$ has never been queried to the $Time\_Key\_Update$ oracle.

*Game 4 (for $\mathcal{A}_4$)* :

- Setup: $\mathcal{C}$ generates the $params$, $msk$. $msk$ is kept secret by $\mathcal{C}$, but $params$ are forwarded to the $\mathcal{A}_4$.
- SignGen_Query: $\mathcal{A}_3$ selects the tuple $(ID, m_i)$ and forwards it to $\mathcal{C}$ for the signature. $\mathcal{C}$ generates and returns the signature of $m_i$ to $\mathcal{A}_3$ by running the algorithm SignGen.
- Challenge: $\mathcal{C}$ generates a challenge message randomly $Chal$ and sends it to $\mathcal{A}_4$ and requests $\mathcal{A}_4$ to reply with the corresponding proof information $\mathbb{P}$.
- Forge: $\mathcal{A}_4$ generates $\mathbb{P}$ and sends it to $\mathcal{C}_4$, for the received challenge message $Chal$. $\mathcal{A}_4$ is said to win the game, if $\mathbb{P}$ passes the validation check and the blocks in $\mathbb{P}$ are incorrect.

## 8.3 Algorithmic Framework

The algorithms of our proposed scheme are defined as follows.

- $Setup(1^\lambda) \rightarrow (param, msk)$. It takes $\lambda$ as input and outputs $msk$ and $param$.
- $Join(ID) \rightarrow s_0$. It takes the identity (ID) of the user as input and outputs group key $s_0$ as output.
- $PartialPvtKeyGen(param, msk, ID_i) \rightarrow D_i$. It takes the $param$, $msk$, user identity $ID_i$ as input and outputs a partial private key $D_i$.
- $SetSecretValue(param, ID_i) \rightarrow x_i$. It takes the $param$, $msk$, $ID_i$ as input and outputs a secret value $x_i$.
- $SetPubKey(param, x_i) \rightarrow PK_i$. It takes $param$, $x_i$ as input and outputs a public key $PK_i$.
- $TimeKeyUpdate(param, ts, ID, T_i)$. It takes the $param$, time key $ts$, ID, time period $\Gamma_i$ as inputs and returns a new time key $dIDTi$ for the user.

- $SignGen(param, S_i, M) \rightarrow \sigma_i$. It takes $param$, $S_i$, and data blocks $\{m_i\}_{1 \leq i \leq n}$ as input and outputs a signature set $\{\sigma_i\}_{1 \leq i \leq n}$ .
- $Challenge(M_{info}) \rightarrow C$. It takes the abstract information $M_{info}$ about the file as input and outputs a random challenge $C$. $M_{info}$ include file name, number of data blocks, etc.
- $ProofGen(m_i, \{\sigma_i\}_{1 \leq i \leq n}, C) \rightarrow P$. It takes $m_i$, $\{\sigma_i\}_{1 \leq i \leq n}$ and $C$ as input and outputs a proof $P$.
- $ProofVerify(param, PK_i, C, P) \rightarrow 0/1$. It takes the $param, PK_i, C$ and $P$ as input and returns auditing result as 1 for success or 0 for failure.
- $Revoke(RL, \{id_1, id_2, ...id_k\}) \rightarrow RL'$. It takes current revocation list (RL) and different user ID's as input and outputs the new $RL'$.

## 8.4 Detailed Construction

**Setup**

KGC generates $params$, $msk$ and master partial secret key with the given security parameter $\lambda$ by conducting following steps:

1. KGC produces a group $\mathbb{G}$ with prime order $q$ from elliptic curve $E$ defined over $F_p$, where $P$ is a random generator of $\mathbb{G}$.
2. Choose $s, v \in Z_q^*$ randomly and computes $y_T = vP$, and keeps $s$ secretly, the $msk$. KGC sends $v$ to a GM for generating time key as the time master key by a secret channel.

   The KGC picks four secure cryptographic hash functions; $H_1 : \{0,1\}^* \rightarrow Z_q^*, H_2 : \{0,1\}^* \times \mathbb{G} \rightarrow Z_q^*$, $H_3 : \{0,1\}^* \rightarrow Z_q^*.H_4 : \{0,1\}^* \rightarrow Z_q^*$.
3. KGC computes the master public key $P_{pub} = sP$ and publishes $params = \{\mathbb{G}, p, q, P, H_1, H_2, H_3, H_4, P_{pub}, y_T\}$.
4. Further, KGC picks $\gamma_0 \in Z_p^*$ as master partial key randomly and forwards it to the GM to generate joining key for the group.

**Join**

To be a member of the group, initially, user sends a request to the GM. Then, GM generates

a group joining key $s_0$ as follows.

$$R_0 = \gamma_0 \cdot P \tag{8.1}$$

$$h_0 = H_1(R_0, P_{pub}) \tag{8.2}$$

$$s_0 = (\gamma_0 + h_0 \cdot s) \mod q. \tag{8.3}$$

and sends $s_0$ to the user in a secure way.

**PartialPvtKeyGen**

After receiving the group key $s_0$ from GM, a user sends both his/her identity $ID$ and $s_0$ to KGC for generating partial private key. On receiving $s_0$, the KGC verifies $s_0 \cdot P = R_0 + h_0 P_{pub}$ to know the validity of the user. If the user is invalid it outputs $\perp$. Else, KGC generates the partial private key for the user as follows. KGC chooses a random value $r \in Z_q^*$ and computes $w_{ID} = rP, h_1 = H_1(\, ID, w_{ID}, P, P_{pub}, y_T)$ , $d_{ID} = r + xh_1 \, mod \, q$ and returns $D_{ID} = (w_{ID}, d_{ID})$ as the user ID's partial private key.

**SetSecretValue**

A group user with an identity $ID$ selects a random number $x_{ID} \in Z_q^*$ and it is kept as secret value.

**SetPublicKey**

A Group user with an identity $ID$ computes public key $PK_{ID} = x_{ID}P$ using $params$ and $x_{ID}$.

**TimeKeyUpdate**

When GM receives a request from a group user with identity $ID$ and a time period $\Gamma_i$ for time key, the GM checks whether the user ID is present in RL or not. If not, the GM picks a random value $r_T \in Z_q^*$ and computes $w_{IDT} = r_T P, h_2 = H_2(ID, \Gamma_i, w_{IDT})$, $d_{IDT} = r_T + vh_2 \, modq$.

After that, the GM returns the time key $D_{IDT} = (w_{IDT}, d_{IDT})$ to the user by a secure channel. After the expiry of the period $\Gamma_i, D_{IDTi}$ is invalid. By this step we can ensure the user revocation.

**SignGen**

$F$ is a shared file to be outsourced to cloud and is split into n blocks, i.e., $F = m_{i(1 \leq i \leq n)}$.

User computes a signature for each block $m_i \in Z_q (1 \leq i \leq n)$ using a private key as follows.

1. For $i \in \{1, 2, ..., n\}$, the user, randomly choose $t_i \leftarrow Z_q^*$ and computes $U_i = t_i P$;
2. $h_3 = H_3(ID, PK_{ID}, w_{ID}, w_{IDT}, \Gamma_i, m_i, U_i, P_p ub, y_T)$,
   $l_i = H_4(\, ID, PK_{ID}, w_{ID}, w_{IDT}, \Gamma_i, m_i, U_i, h_3)$
3. Compute $s_i = m_i t_i + h_3 x_{ID} + l_i(d_{ID} + d_{IDT}) \mod q$. Finally, the signature on data block $m_i$ is $\sigma_i = (U_i, s_i, w_{ID_i}, w_{ID\Gamma_i})$.

Following this, the user outsources $F = \{m_i\}$ and signatures, $\sigma_{i(1 \leq i \leq n)}$ to the CSP. Later, the user removes the raw data files and signatures from local records to save space.

**Challenge**

After storing data in a cloud, an auditing challenge $C$ is generated by TPA to assess the integrity of file, which is demonstrated as follows:

- Pick a proper subset $I$ of $crandomly\,(c \leq n)$ elements from the set $[1, n]$, $\mid I \mid = c$, and TPA selects a random value $v_j \in Z_q^*$,$\forall j \in I$.
- Send the challenge $C = \{(j, v_j)\}_{j \in I}$ to the CSP.

**ProofGen**

On receiving $C = \{(j, v_j)\}_{j \in I}$, the server computes data integrity proof $\mathbb{P}$ which includes the data proof and signature proof.

1. Cloud computes $\Psi$ as the linear combinations of sampled blocks

$$\Psi = \sum_{j \in I} v_j s_j \cdot P \tag{8.4}$$

2. Meanwhile, the CS computes the aggregated signature.

$$\psi = \sum_{j \in I} v_j m_j \cdot U_j \tag{8.5}$$

   Then the server returns final proof $\mathbb{P} = (\Psi, \psi)$ to the TPA as a response.

**ProofVerify**

TPA checks the integrity of the challenged data blocks after receiving the corresponding proof $\mathbb{P} = (\Psi, \psi)$ for the challenge from the server by verifying the following equation.

$$\Psi \stackrel{?}{=} \sum_{j \in I} v_j \Big( U_j + h_3 PK_{ID} + l_j (w_{ID} + h_1 P_{pub} + w_{IDT} + h_2 y_T) \Big) \tag{8.6}$$

If the verification Eq.8.6 holds, the TPA concludes that the data blocks stored in cloud are properly maintained. Otherwise, TPA arrives at a conclusion that the outsourced file $F$ has been lost or corrupted.

### 8.4.1 User revocation

When a group user is revoked, all non-revoked group users will update their time by requesting the GM.

- If the user is non-revoked, the GM generates a new time key for the user by checking the $RL$, and forwards it through a secure channel.
- If the user is revoked, the GM rejects the users request. Therefore, revoked users cannot generate valid signatures, as they don't have the valid time key.

### 8.4.2 Batch Auditing

In cloud systems, it is tedious, and inefficient for the auditor to check the integrity of shared big data file by file in terms of both communication and computational overheads. Particularly, given a set of $N$ different files $F_t = \{m_{t,1}, ..., m_{t,n_t}\}, 1 \leq t \leq N, 1 \leq l \leq n_t$, it is desirable to reduces the cost by aggregating integrity checking operations of multiple files into one challenge. To achieve this, the TPA to carry out integrity verification of $N$ files at a time. In the batch auditing process, all algorithms are similar to as those of section 8.4 except $Challenge, ProofGen$ and $ProofVerify$ algorithms.

**Batch_Challenge.** The challenge message $C = \{(j, v_j), Msg\}$ forwarded to CSP for auditing data file of $N$ users, where $Msg$ includes the information about the users and the files to be verified.

**Batch_ProofGen.** Upon receiving challenge information $C$, the CSP computes

$$\Psi = \sum_{t=1}^{N} \sum_{j \in I} v_j \cdot s_{t,j} \cdot P \tag{8.7}$$

Meanwhile, the server computes the aggregated signature.

$$\psi = \sum_{t=1}^{N} \sum_{j \in I} v_j \cdot m_{t,j} \cdot R_{t,j} \tag{8.8}$$

Then the server returns final proof as a response $\mathbb{P} = (\Psi, \psi)$ to the TPA.

**Batch_ProofVerify.** TPA checks the integrity of outsourced data after receiving $\mathbb{P} = (\Psi, \psi)$ for the challenge from the server by verifying the following equation.

$$\psi = \sum_{t=1}^{N} \sum_{j \in I} v_j \Big( U_{t,j} + h_3 PK_t + l_{t,j}(w_{ID} + h_1 P_{pub} + w_{IDT} + h_2 y_T) \Big) \tag{8.9}$$

If the verification Eq. 8.9 holds, the TPA concludes that the data blocks stored in cloud are properly maintained. Else, TPA arrives at a conclusion that the outsourced files $F$ has been lost or corrupted.

## 8.5 Security Analysis

We perform a formal security analysis of EPF-CLPA in terms of correctness and unforgeability by considering type I/II/III/IV adversaries.

### 8.5.1 Correctness

**Theorem 1.** *In the proposed EPF-CLPA, the TPA can successfully verify the integrity of data if all the selected file blocks in the challenge* $C = \{(j, v_j)\}_{j \in I}$ *and their corresponding signatures are kept intact in the cloud.*

**Proof.** We prove the correctness of EPF-CLPA by verifying the Eq. 8.6 based on proper-

ties.

$$\begin{aligned}
\Psi &= \sum_{j \in I} v_j s_j \cdot P \\
&= \sum_{j \in I} v_j \Big( m_j t_j + h_3 x_{ID} + l_j (d_{ID} + d_{IDT}) \Big) \cdot P \\
&= \sum_{j \in I} v_j \Big( m_j U_j + h_3 \cdot PK_{ID} + l_j (d_{ID} + d_{IDT}) P \Big) \\
&= \sum_{j \in I} v_j m_j U_j + \sum_{j \in I} v_j \Big( h_3 PK_{ID} + l_j (r + x h_1 + r_T + v h_2) P \Big) \\
&= \psi + \sum_{j \in I} v_j \Big( h_3 PK_{ID} + l_j (w_{ID} + h_1 P_{pub} + w_{IDT} + h_2 y_T) \Big)
\end{aligned}$$

The correctness of the verification Eq. 8.9 is elaborated in the following.

$$\begin{aligned}
\Psi &= \sum_{t=1}^{N} \sum_{j \in I} v_j s_{t,j} \cdot P \\
&= \sum_{t=1}^{N} \sum_{j \in I} v_j \Big( t_{t,j} + h_3 x_{ID} + l_{t,j} (d_{ID} + d_{IDT}) \Big) \cdot P \\
&= \sum_{t=1}^{N} \sum_{j \in I} v_j \Big( U_{t,j} + h_3 PK_t + l_{t,j} (d_{ID} + d_{IDT}) P \Big) \\
&= \sum_{t=1}^{N} \sum_{j \in I} v_j \Big( U_{t,j} + h_3 PK_t + l_{t,j} (d_{ID} + d_{IDT}) P \Big) \\
&= \sum_{t=1}^{N} \sum_{j \in I} v_j \Big( U_{t,j} + h_3 PK_t + l_{t,j} (r + x h_1 + r_T + v h_2) P \Big) \\
&= \sum_{t=1}^{N} \sum_{j \in I} v_j \Big( U_{t,j} + h_3 PK_t + l_{t,j} (w_{ID} + h_1 P_{pub} + w_{IDT} + h_2 y_T) \Big)
\end{aligned}$$

### 8.5.2 Unforgeability

Here, we prove EPF-CLPA is existentially unforgeable against $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$ and $\mathcal{A}_4$ as defined in Section 8.2.4.

**Theorem 2:** *EPF-CLPA existentially unforgeable against $\mathcal{A}_1$, if the advantage of $\mathcal{A}_1$ winning the experiment is negligible in the ROM assuming ECDLP is hard in $G$.*

**Proof:** If $\mathcal{A}_1$ can succeed in the $Game\,1$ described in section 8.2.4 with a non-negligible

probability $\epsilon$; then, we could simulate a challenger $\mathcal{C}_1$ that uses $\mathcal{A}_1$ as a block box to find solution for the ECDLP with a probability

$$\epsilon' \geq \epsilon\Big((q_1 - 1)/q_1\Big)^{q_e}/q_1$$

.

Let $(P, xP) \in G$ be a random instance of the ECDLP. $\mathcal{C}_1$'s objective is to find the solution $x$ through the interactions with $A_1$. Initially, $\mathcal{C}_1$ maintains four lists $L^{list}$, $L^{list}_{T_K}$, $H^{list}_3$, $H^{list}_4$ which are initially empty. Lists $L^{list}$ and $L^{list}_{T_K}$ are used to record CreateUser queries and TimeKeyUpdate queries, respectively. Lists $H^{list}_3$, $H^{list}_4$ are used to record $H_3$ and $H_4$ queries, respectively. $\mathcal{A}_1$ and $\mathcal{C}_1$ interacts as follows.
$q_1$ denotes number of queries to the random oracle $H_1$ and $q_e$ denotes number of queries to the $PartialPvtKeyGen$ oracle.

- $H_1$-Query: When $\mathcal{A}_1$ makes an $H_1$-query with identity $ID_i$. $\mathcal{C}_1$ checks whether $L^{list}$ contains $(ID_i, w_{IDi}, d_{IDi}, h_{1i}, x_{IDi}, PK_{IDi})$. If it holds, $\mathcal{C}_1$ returns $h_{1i}$; otherwise, $\mathcal{C}_1$ picks a random $h_{1i} \in Z^*_q$ and returns $h_{1i}$ to $A_1$ then adds $(ID_i, w_{IDi}, d_{IDi}, h_{1i}, x_{IDi}, PK_{IDi})$ into $L^{list}$.
- $H_2$-Query: If $\mathcal{A}_1$ makes an $H_2$-query with identity $ID_i$. $\mathcal{C}_1$ checks whether $L^{list}_{T_K}$ contains $(ID_i, \Gamma_i, w_{IDiT}, d_{IDiT}, h_{2i})$. If it holds, $\mathcal{C}_1$ returns $h_{2i}$; Otherwise, $\mathcal{C}_1$ picks $h_{2i} \in Z^*_q$ and returns to $\mathcal{A}_1$ and then adds $(ID_i, \Gamma_i, w_{IDiT}, d_{IDiT}, h_{2i})$ into $L^{list}_{T_K}$
- $H_3$-Query: If $\mathcal{A}_1$ makes an $H_3$-query with identity $ID_i$. $\mathcal{C}_1$ checks whether $H^{list}_3$ contains $(ID_i, PK_{IDi}, w_{IDi}, w_{IDiT}, \Gamma_i, m_i, U_i, P_{pub}, y_T, h_{3i})$. If it holds, $\mathcal{C}_1$ returns $h_{3i}$; Else, $\mathcal{C}_1$ picks $h_{3i} \in Z^*_q$, returns to $\mathcal{A}_1$ then inserts $(ID_i, PK_{IDi}, w_{IDi}, w_{IDiT}, \Gamma_i, m_i, U_i, P_{pub}, y_T, h_{3i})$ into $H^{list}_3$.
- $H_4$-Query: If $\mathcal{A}_1$ makes an $H_4$-query with identity $ID_i$. $\mathcal{C}_1$ checks whether $H^{list}_4$ contains $(ID_i, PK_{IDi}, w_{IDi}, w_{IDiT}, \Gamma_i, m_i, U_i, h_{3i}, l_i)$. It returns $l_i$ if it exists; otherwise, $\mathcal{C}_1$ chooses $l_i \in Z^*_q$ randomly then returns to $\mathcal{A}_1$. Next, inserts the record $(ID_i, PK_{IDi}, w_{IDi}, w_{IDiT}, \Gamma_i, m_i, U_i, h_{3i}, l_i)$ into $H^{list}_4$.

- Setup: $C_1$ chooses $x, v \in Z^*_q$ randomly, computes $y = xP, y_T = vP$, and generates

$params : \{k, q, G, P, y, y_T, H_1, H_2, H_3, H_4\}$. $params$ forwarded to $\mathcal{A}_1$.

- Create-User (ID): $\mathcal{C}_1$ maintains a list $L^{list}$ of the form $(ID_i, w_{IDi}, d_{IDi}, h_{1i}, x_{IDi}, PK_{IDi})$. when $\mathcal{A}_1$ makes a $Create - User$ query with an identity $ID_i$. If $ID_i$ has already been in the list $L^{list}$, it simply returns $PK_{IDi}$. Otherwise, $\mathcal{C}_1$ computes partial private key $d_{IDi}$, secret key $x_{IDi}$ and public key $PK_{IDi}$ as follows:
  - $\mathcal{C}_1$ chooses a value $x_{IDi} \in Z_q^*$ randomly. and computes $PK_{IDi} = x_{IDi}P$.
  - $\mathcal{C}_1$ $r_i, h_{1i} \in Z_q^*$ randomly, computes $w_{IDi} = r_iP$, sets $H_1(ID_i, w_{IDi}, P, P_{pub}, y_T) = h_{1i}$ and computes $d_{IDi} = r_i + xh_{1i}$.

  The tuple $(w_{IDi}, d_{IDi})$ is the user $ID_i$'s partial private key and $x_{IDi}$ is secret value. Finally, $\mathcal{C}_1$ returns $PK_{IDi}$ to $A_1$, and adds the tuple $(ID_i, w_{IDi}, d_{IDi}, h_{1i}, x_{IDi}, PK_{IDi})$ into the list $L^{list}$ .
- PartialPvtKeyGen: After receiving a query with identity $ID_i$, $\mathcal{C}_1$ does the following.
  - If $ID_i \neq ID^*$, $\mathcal{C}_1$ lookup the list $L^{list}$ for the tuple $(ID_i, w_{IDi}, d_{IDi}, h_{1i}, x_{IDi}, PK_{IDi})$ and return $(w_{IDi}, d_{IDi})$ to A1.
  - Otherwise $\mathcal{C}$ stops the computation and output $\perp$.
- TimeKeyUpdate: When receiving a $TimeKeyUpdate$ query on $ID_i$ with a time period $\Gamma_i$, $\mathcal{A}_1$ $t_i \in Z_q^*$ and calculates $w_{IDiT} = t_iP$. Then picks $h_{2i} \in Z_q^*$ randomly, sets $H_2(ID_i, \Gamma_i, w_{IDiT}) = h_{2i}$, and calculates $d_{IDiT} = t_i + vh_{2i}$. $\mathcal{C}_1$ returns the time key $D_{IDiT} = (w_{IDiT}, d_{IDiT})$ to $\mathcal{A}_1$, and adds this record $(ID_i, \Gamma_i, w_{IDiT}, d_{IDiT}, h_{2i})$ to the list $L_{TK}^{list}$.
- SecretValue: $\mathcal{C}_1$ looks up $L^{list}$ and returns $x_{IDi}$.
- ReplacePublicKey: On receiving this query on $(ID_i, PK'_{IDi})$, $\mathcal{C}_1$ replaces original public key $PK_{IDi}$ with $PK'_{IDi}$ and then update the record $(IDi, PK'_{IDi})$ into $L^{list}$.
- SignGen: After receiving a query on inputs $m_i$, $\Gamma_i$ and $ID_i$ with the public key $PK_{IDi}$, $\mathcal{A}_1$ does the following:
  - If $ID_i \neq ID^*$, and the public key $PK_{IDi}$ has not been replaced, $\mathcal{A}_1$ executes $SignGen$ to produce a signature $\sigma_i$ and returns it to $\mathcal{A}_1$.
  - If $ID_i = ID^*$, $\mathcal{C}_1$ aborts the game.

- Forge: Finally, $\mathcal{A}_1$ produces a signature $\sigma_i^*$ on $m_i^*$ for an identity $ID_i$ with $PK_{IDi}^*$, $\Gamma_i$. If $ID_i \neq ID^*$, $\mathcal{A}_1$ aborts the game and outputs fail. Else, based on forking lemma [90], $C_1$ outputs another value to $H_4$. Then, $\mathcal{A}_1$ outputs a new forgery $\sigma_i'$ on the same message $m_i$ by replaying the same procedure but with a different choice of $H_4$. Then $C_1$ has the following two equations.

$$s_i = t_i + h_3 x_{ID} + l_i(d_{ID} + d_{IDT}) \mod q \tag{8.10}$$

$$s_i' = t_i + h_3 x_{ID} + l_i'(d_{ID} + d_{IDT}) \mod q \tag{8.11}$$

Here $l_i \neq l_i'$. From above two Eq.8.10,8.11's, $C_1$ can calculate $a = (s_i - s_i')/(l_i - l_i') - xh_{1i} - d_{IDiT}$. Hence, $C_1$ can determine solution $a$. $C_1$'s success probability can be analyzed as follows. $\mathcal{A}_1$ can win this game if event $E_3$ occur and $E_1$ and $E_2$ do not occur. We, then show the probability that $\mathcal{A}_1$ wins the game as follows.

1. $E_1$: $\mathcal{C}_1$ does not abort Game 1 in the $PartialPvtKeyGen$ query.
2. $E_2$: $\mathcal{A}_1$ computes forgery of a signature $\sigma_i$ on $m_i$ for $ID_i \neq ID^*$ in the the forgery phase.
3. $E_3$: The signature $\sigma_i$ is valid on $(m_i^*, ID_i)$.

Therefore, the success probability of solving ECDLP can be defined as

$$\epsilon' = P_r[\neg E_1 \wedge \neg E_2 \wedge E_3]$$
$$= P_r[\neg E_1] P_r[\neg E_2 | \neg E_1] P_r[E_3 | \neg E_1 \wedge \neg E_2]$$

From the simulation, we have

$$Pr[\neg E_1] \geq \left(1 - \frac{1}{q_1}\right)^{q_e}$$
$$P_r[\neg E_2 | \neg E_1] \geq \frac{1}{q_1}$$
$$P_r[E_3 | \neg E_1 \wedge \neg E_2] = \epsilon$$

From these equations, the probability that $\mathcal{C}_1$ could solve the given ECDL problem is

$\epsilon' \geq \epsilon\Big((q_1 - 1)/q_1\Big)^{q_e}/q_1$.

**Theorem 3:** *If an adversary $\mathcal{A}_2$ can break our EPF-CLPA with a non negligible probability $\epsilon$, then we can construct an algorithm $C_2$ by employing $\mathcal{A}_2$ to solve ECDLP with a probability.*

$$\epsilon' \geq \epsilon\Big((q_1 - 1)/q_1\Big)^{q_s}/q_1$$

**Proof:** Consider $(P, aP) \in G$ as an instance of the ECDLP. $\mathcal{C}_2$ determines the value of $a$ by interacting with $\mathcal{A}_2$.

- $H_1, H_2, H_3, H_4$ queries: Queries and answers of $H_1, H_2, H_3, H_4$ oracles are the same as those in Theorem 1.
- Setup: $C_2$ picks $x, v \in Z_q^*$, computes $y = xP, y_T = vP$, and generates $params$ : $\{k, q, G, P, y, y_T, H_1, H_2, H_3, H_4\}$
- Create-User (ID): $\mathcal{C}_2$ maintains a list $L^{list}$ of the form $(ID_i, w_{IDi}, d_{IDi}, h_{1i}, x_{IDi}, PK_{IDi})$. when $\mathcal{A}_2$ makes this query on $ID_i$. If $ID_i$ has already been created, it simply returns $PK_{IDi}$. Otherwise, $\mathcal{C}_1$ computes partial private key $d_{IDi}$, secret key $x_{IDi}$ and public key $PK_{IDi}$ as follows:
  - $\mathcal{C}_2$ chooses $x_{IDi} \in Z_q^*$ randomly and computes $PK_{IDi} = x_{IDi}P$.
  - $\mathcal{C}_2$ randomly chooses random values $r_i, h_{1i} \in Z_q^*$, computes $w_{IDi} = r_iP$, sets $H_1(ID_i, w_{IDi}, P, P_{pub}, y_T) = h_{1i}$ and calculates $d_{IDi} = r_i + xh_{1i}$.

  Finally, $\mathcal{C}_2$ returns $PK_{IDi}$ to $\mathcal{A}_2$, and adds the tuple $(ID_i, w_{IDi}, d_{IDi}, h_{1i}, x_{IDi}, PK_{IDi})$ into the list $L^{list}$ .
- TimeKeyUpdate: When receiving a TimeKeyUpdate query on $ID_i$ and $\Gamma_i$, $\mathcal{C}_2$ picks $t_i \in Z_q^*$ randomly and calculates $w_{IDiT} = t_iP$, and picks $h_{2i} \in Z_q^*$ randomly, sets $H_2(ID_i, \Gamma_i, w_{IDiT}) = h_{2i}$, and calculates $d_{IDiT} = t_i + vh_{2i}$. $\mathcal{C}_2$ returns the time key $D_{IDiT} = (w_{IDiT}, d_{IDiT})$ to $\mathcal{A}_2$, and adds this record $(ID_i, \Gamma_i, w_{IDiT}, d_{IDiT}, h_{2i})$ to the list $L_{TK}^{list}$.
- SecretValue: $\mathcal{C}_2$ looks up $L^{list}$ and returns $x_{IDi}$.
- SignGen: Upon receiving a query on inputs $m_i$, $\Gamma_i$ and $ID_i$ with $PK_{IDi}$, $\mathcal{C}_2$ acts as below:

- If $ID_i \neq ID^*$, and the public key $PK_{IDi}$, $\mathcal{C}_2$ normally executes the $SignGen$ algorithm to compute a signature $\sigma_i$ and returns it to $\mathcal{A}_2$.
- If $ID_i = ID^*$, $\mathcal{C}_2$ aborts the game.

- Forge: Finally, $\mathcal{A}_2$ produces a signature $\sigma_i^*$ on $m_i^*$ for $ID_i$ with $PK_{IDi}^*, \Gamma_i$. If $ID_i \neq ID^*$, $\mathcal{C}_2$ aborts the game. Else, based on forking lemma [90], $\mathcal{C}_2$ outputs another value to the $H_4$. Then, $\mathcal{A}_2$ returns a new forged signature $\sigma_i'$ on the same message $m_i$ by replaying the same procedure but with a different choice of $H_4$. Then $\mathcal{C}_2$ has the following two Eqs.

$$s_i = t_i + h_3 x_{ID} + l_i(d_{ID} + d_{IDT}) \mod q \tag{8.12}$$

$$s_i' = t_i + h_3 x_{ID} + l_i'(d_{ID} + d_{IDT}) \mod q \tag{8.13}$$

Here $l_i \neq l_i'$. From above two Eq.8.12,8.13's, $\mathcal{C}_2$ can calculate $a = (s_i - s_i')/(l_i - l_i') - x h_{1i} - d_{IDiT}$. Hence, $\mathcal{C}_2$ can determine the solution $a$ of the given ECDLP.

By following the same success probability analysis of Theorem 1 for $\mathcal{C}_1$, we can conclude that $\mathcal{C}_2$ can solve the ECDLP with a probability $\epsilon' \geq \epsilon\Big((q_1 - 1)/q_1\Big)^{q_s}/q_1$.

**Theorem 4:** *EPF-CLPA achieves existentially unforgeability against $\mathcal{A}_3$, if the advantage of $\mathcal{A}_3$ winning the experiment is negligible in the ROM assuming ECDLP is hard in $G_1$.*

**Proof:** If $\mathcal{A}_3$ wins the Game 3 with a non-negligible probability $\epsilon$; then, we could simulate a challenger $\mathcal{C}_3$ that uses $\mathcal{A}_3$ as a block box to find solution for the ECDL instance with a probability

$$\epsilon' \geq \epsilon/q_1$$

Let $(P, aP) \in G$ be an instance of the ECDLP. $\mathcal{C}_3$ finds the value $a$ by interacting with $A_3$.

- Setup: The Setup is similar to Theorem 1.
- Create-User (ID): $\mathcal{C}_3$ maintains a list $L^{list}$ of the form $(ID_i, w_{IDi}, d_{IDi}, h_{1i}, x_{IDi}, PK_{IDi})$. when A3 makes this query with an identity $ID_i$. If $ID_i$ has already been created, it simply returns $PK_{IDi}$. Otherwise, $\mathcal{C}_3$ computes partial private key $d_{IDi}$, secret key $x_{IDi}$ and public key $PK_{IDi}$ as follows:

- $\mathcal{C}_3$ chooses a $x_{IDi} \in Z_q^*$ randonmly and computes $PK_{IDi} = x_{IDi}P$.
- $\mathcal{C}_3$ chooses $r_i, h_{1i} \in Z_q^*$ randomly, computes $w_{IDi} = r_iP$, sets $H_1(ID_i, w_{IDi}, P, P_{pub}, y_T) = h_{1i}$ and calculates $d_{IDi} = r_i + xh_{1i}$.

The tuple $(w_{IDi}, d_{IDi})$ is the user $ID_i$'s partial private key and $x_{IDi}$ the secret value. Finally, $\mathcal{C}_3$ returns $PK_{IDi}$ to $\mathcal{A}_3$, and adds the tuple $(ID_i, w_{IDi}, d_{IDi}, h_{1i}, x_{IDi}, PK_{IDi})$ into the list $L^{list}$ .

- PartialPvtKeyGen: After receiving a query with identity $ID_i$, $\mathcal{C}_3$ does the following.
  - If $ID_i \neq ID^*$, $\mathcal{C}_1$ lookup the list $L^{list}$ for the tuple $(ID_i, w_{IDi}, d_{IDi}, h_{1i}, x_{IDi}, PK_{IDi})$ and return $(w_{IDi}, d_{IDi})$ to $\mathcal{A}_3$.
  - Otherwise $\mathcal{C}_3$ stops the computation and output $\perp$.
- TimeKeyUpdate: When receiving a TimeKeyUpdate query on $ID_i$ and $\Gamma_i$, $\mathcal{C}_3$ picks $t_i \in Z_q^*$ randomly and calculates $w_{IDiT} = t_iP$, and picks $h_{2i} \in Z_q^*$ randomly, sets $H_2(ID_i, \Gamma_i, w_{IDiT}) = h_{2i}$, and calculates $d_{IDiT} = t_i + vh_{2i}$. $\mathcal{C}_3$ returns the time key $D_{IDiT} = (w_{IDiT}, d_{IDiT})$ to $\mathcal{A}_2$, and adds this record $(ID_i, \Gamma_i, w_{IDiT}, d_{IDiT}, h_{2i})$ to the list $L_{TK}^{list}$.
- SecretValue: $\mathcal{C}_3$ looks up $L^{list}$ and returns $x_{IDi}$.
- ReplacePublicKey: On receiving $ReplacePublicKey$ query on $(ID_i, PK'_{IDi})$, $\mathcal{C}_1$ replaces original public key $PK_{IDi}$ with $PK'_{IDi}$ and then update the record $(IDi, PK'_{IDi})$ into $L^{list}$.
- $H_1, H_2, H_3, H_4$ queries: Queries and answers of $H_1, H_2, H_3, H_4$ oracles are the similar to those in Theorem 1.
- SignGen: Upon receiving a query on inputs $m_i$, $\Gamma_i$ and $ID_i$ with $PK_{IDi}$, $\mathcal{C}_3$ acts as below:
  - If $ID_i \neq ID^*$, and the public key $PK_{IDi}$ $\mathcal{C}_3$ executes $SignGen$ to produce a signature $\sigma_i$ and it is returned $\mathcal{A}_3$.
  - If $ID_i = ID^*$, $\mathcal{C}_3$ aborts the game.
- Forge: Finally, $\mathcal{A}_3$ produces a signature $\sigma_i^*$ on $m_i^*$ for $ID_i$ with the public key $PK_{IDi}^*$ and $\Gamma_i$. If $ID_i \neq ID^*$, $\mathcal{C}_3$ aborts the game and outputs fail. Else, based on forking

lemma [90], $\mathcal{C}_3$ outputs another value to $H_4$. Then, $\mathcal{A}_3$ returns a new forged signature $\sigma_i'$ on the same message $m_i$ by replaying the same procedure but a different choice of $H_4$. Then $\mathcal{C}_3$ will have the following two equations.

$$s_i = t_i + h_3 x_{ID} + l_i(d_{ID} + d_{IDT}) \mod q \tag{8.14}$$

$$s_i' = t_i + h_3 x_{ID} + l_i'(d_{ID} + d_{IDT}) \mod q \tag{8.15}$$

Here $l_i \neq l_i'$. From above Eq.8.14,8.15's, $\mathcal{C}_3$ can calculate $a = (s_i - s_i')/(l_i - l_i') - xh_{1i} - d_{IDiT}$.

It is observed that the probability to solve the given ECDLP by $\mathcal{C}_3$ is $\epsilon' \geq \epsilon/q_1$.

**Theorem 5**. *Assuming the hardness of the ECDLP in $\mathbb{G}$, it is hard for $\mathcal{A}_4$ to generate a forged proof to succeed in the verification process.*

**Proof:** We prove the theorem 5 by following the security game defined in section III for $\mathcal{A}_4$ as follows:

We assume $\mathcal{M}$ as the shared cloud data, and the challenger $\mathcal{C}_4$ (TPA) forwards a random challenge $(j, v_j)_{j\in I}$ to $\mathcal{A}_4$ (Cloud). The valid proof should be $(\Psi, \psi)$, where $\Psi = \sum_{j\in I} v_j s_j \cdot P, \psi = \sum_{j\in I} v_j m_j \cdot U_j$ return to $\mathcal{C}_4$ as a response that can pass the verification, while the cloud generates an invalid proof $(\Psi, \psi')$ based on the corrupt data $\mathcal{M}'$, where $\mathcal{M}' \neq \mathcal{M}$, and at least one element of $\Delta m_j = m_j{}' - m_j$, and $\Delta m_j \neq 0$ for $j \in I$ is nonzero. If the forged proof succeeds in the verification, then we say the $\mathcal{A}_4$ wins the Game 4 and we can find a solution to the ECDLP. Otherwise, $\mathcal{A}_4$ fails. If $\mathcal{A}_4$ wins, according to Eq. 8.6, we can have

$$\Psi = \psi^* + \sum_{j\in I} v_j \Big( h_3 PK_{ID} + l_j(w_{ID} + h_1 P_{pub} + w_{IDT} + h_2 y_T) \Big)$$

According to the correct data, $(\Psi, \psi)$ is a correct proof, we can get

$$\Psi = \psi + \sum_{j\in I} v_j \Big( h_3 PK_{ID} + l_j(w_{ID} + h_1 P_{pub} + w_{IDT} + h_2 y_T) \Big)$$

By observing the above two equations, we can say that $\psi^* = \psi$.

Hence,

$$\sum_{j \in I} v_j \cdot m_j^* \cdot U_j = \sum_{j \in I} v_j \cdot m_j \cdot U_j \tag{8.16}$$

Define $\Delta\mathfrak{d} = \sum_{j \in I} v_j \cdot (m_j^* - m_j) \cdot t_j$

i.e.,

$$\begin{aligned}
\psi^* - \psi &= \sum_{j \in I} v_j \cdot m_j^* \cdot U_j - \sum_{j \in I} v_j \cdot m_j \cdot U_j \\
&= \sum_{j \in I} v_j \cdot (m_j^* - m_j) \cdot U_j \\
&= \sum_{j \in I} v_j \cdot (m_j^* - m_j) \cdot t_j \cdot P \\
&= \Delta\mathfrak{d} \cdot P \\
&= 0
\end{aligned}$$

Assume $(Z, xZ)$ be an instance of the ECDLP.Then we can find the value of $x \in Z_q^*$. Let $P = r_1 xZ + r_2 Z$, in which $r_1, r_2 \in_R Z_q^*$. We can get the following equation.

$$\begin{aligned}
\Delta\mathfrak{d} \cdot P &= 0 \\
\Delta\mathfrak{d}(r_1 xZ + r_2 Z) &= 0 \\
r_1 \Delta\mathfrak{d} xZ + r_2 \Delta\mathfrak{d} Z &= 0 \\
xZ &= -\frac{r_2}{r_1} Z
\end{aligned}$$

Specifically, we can compute $x = -(r_2/r_1)$ when the $r_1$ is nonzero. The probability that $r_1$ become zero is $1/q$, where $q$ is a large prime. That means, it is computationally intractable for $\mathcal{A}_4$ to output a forged proof.

Table 8.1: Notations

| Notation | Description |
|---|---|
| $T_p$ | Bilinear pairing |
| $T_{G1\_ex}$ | Exponentiation on group $G_1$ |
| $T_{G1\_mul}$ | Multiplication on group $G_1$ |
| $T_{G2\_mul}$ | Multiplication on group $G_2$. |
| $c$ | Number of blocks in challenge. |
| $d$ | User subsets for the challenge. |
| $T_h$ | Hash operation in group $G_1$ |
| $MulZ_p^*$ | Multiplication in $Z_p^*$. |
| $AddZ_p^*$ | Addition in $Z_p^*$. |
| $T_{e.a}$ | Scalar addition in ECC |
| $T_{e.m}$ | Scalar multiplication in ECC |

## 8.6 Performance Analysis

In this section, we provide the performance evaluation and experimental results of EPF-CLPA. We define some notations used for performance assessment in Table 8.1.

### 8.6.1 Performance Evaluation

We evaluate the computation cost and communication cost of the proposed EPF-CLPA scheme theoretically as follows.

Table 8.2: Comparison of computation cost

| Schemes | SignGen | ProofGen | ProofVerify | revocation |
|---|---|---|---|---|
| Li et al. [62] | $2nT_{G1\_ex} + nT_{G1\_mul}$ | $cT_{G1\_ex} + (c - d)T_{G1\_mul}$ | $(d + 2)T_p + (c + d)T_{G1\_ex} + (c + 2d)T_{G1\_mul} + dT_{G2\_mul}$ | $n(T_{G1\_mul} + 2T_{G1\_ex})$ |
| Yang et al.[63] | $2nT_{G1\_ex} + nT_{G1\_mul}$ | $T_{G1\_mul} + cT_{G2\_mul} + cT_{G2\_ex} + cT_{G1\_ex} + T_p$ | $T_{G1\_mul} + cT_{G2\_mul} + cT_{G2\_exp} + cT_{G1\_ex} + P$ | $n(T_{G1\_mul} + 2T_{G1\_ex})$ |
| EPF-CLPA | $nT_{e.m} + 2nTh$ | $(c - 1)T_a + 2cT_m + (c - 1)T_{e.a} + (c + 1)T_{e.m}$ | $cT_{e.a} + 5cT_{e.m}$ | $Te.m + AddZ_q$ |

Table 8.3: Comparison of communication costs in auditing phase( $n$ data blocks with $c$ challenged blocks)

| Schemes | TPA to CSP (Challenge) | CSP to TPA (Proof) | Type |
|---|---|---|---|
| Li et al. [62] | $log_2 c + 2 \mid q \mid$ | $d \mid G_1 \mid + d \mid q \mid$ | Pairing-based |
| Yang et al. [63] | $2clog_2\lambda + (c+1)log_2 q$ | $d \mid G_1 \mid + d \mid q \mid$ | Pairing-based |
| EPF-CLPA | $c(\mid n \mid + \mid Z_q^* \mid)$ | $c \mid p \mid + \mid q \mid$ | Pairing-free |

#### 8.6.1.1 Computation cost

We calculate the computation cost of EPF-CLPA and compare with the some of the existing state of the art schemes [62, 63] and results are listed in Table 8.2.

We provide the computation cost of the four algorithms, namely $SignGen$, $ProofGen$, $ProofVerify$, $TimeKeyUpdate$ (for user revocation) which play the significant role in our proposed EPF-CLA scheme.

To generate the signatures for all blocks of the shared file, the algorithm $SignGen$ runs with a cost of $nT_{e.m} + 2nTh$. The CSP runs the $ProofGen$ with the running time

$(c-1)T_a + 2cT_m + (c-1)T_{e.a} + (c+1)T_{e.m}$. To check the integrity, TPA runs the algorithm $ProofVerify$ with a cost of $cT_{e.a} + 5cT_{e.m}$. In EPF-CLPA, users are revoked by executing the $TimeKeyUpdate$ algorithm. Therefore, the computation overhead of $TimeKeyUpdate$ is the computation cost of revocation which is $Te.m + AddZ_q$.

From Table 8.2, we can see that EPF-CLPA Sign generation, proof generation, proof verification and revocation is efficient than [62],[63] in all important aspects of cloud auditing scheme.

#### 8.6.1.2 Communication Cost

Table 8.3 summarizes the communication overhead of our proposed scheme by comparing it with some of existing [62, 63] cloud auditing schemes. The communication cost refers to the costs used for transmitting a randomly generated challenge from TPA to CSP and the corresponding proof from CSP to TPA. It mainly comes from the integrity auditing phase which includes three algorithms namely, $Challenge, ProofGen, ProofVerify$ according to the description of Section 8.3. The TPA sends a challenge $C = \{(j, v_j)\}_{j \in I}$ to the CSP, where $j \in n$ and $v_j \in Z_q^*$. The size of challenge is $c \cdot (\mid n \mid + \mid q \mid)$ bits. The CSP generates a corresponding proof $P = (\mu, \sigma)$ to send it to the TPA. The size of the proof is $\mid p \mid + \mid q \mid$ bits.

### 8.6.2 Experimental Results

We implemented EPF-CLPA on a system with Intel i5-7200U processor @ 2.50 GHz and 8 GB RAM running Windows 7 operating system. All experiments are conducted in python 2.7 language (Py-Charm IDE) using well known free crypto-0.42 library [79]. The implementation uses a non-singular elliptic curve $E(F_{q^k}) : y^2 = x^3 + ax + b \mod p$, where $a = -3$ and $p, q, b$ (random) are 160-bit prime numbers and $a, b \in Z_p^*$. The experimental results for $SignGen, ProofGen, ProofVerify$ and $Revocation$ are obtained and compared with the existing schemes and illustrated in Fig. 8.2a to Fig. 8.2e. Fig. 8.2a shows the comparison of computation time of $SignGen$ algorithm. From Fig. 8.2a, we notice

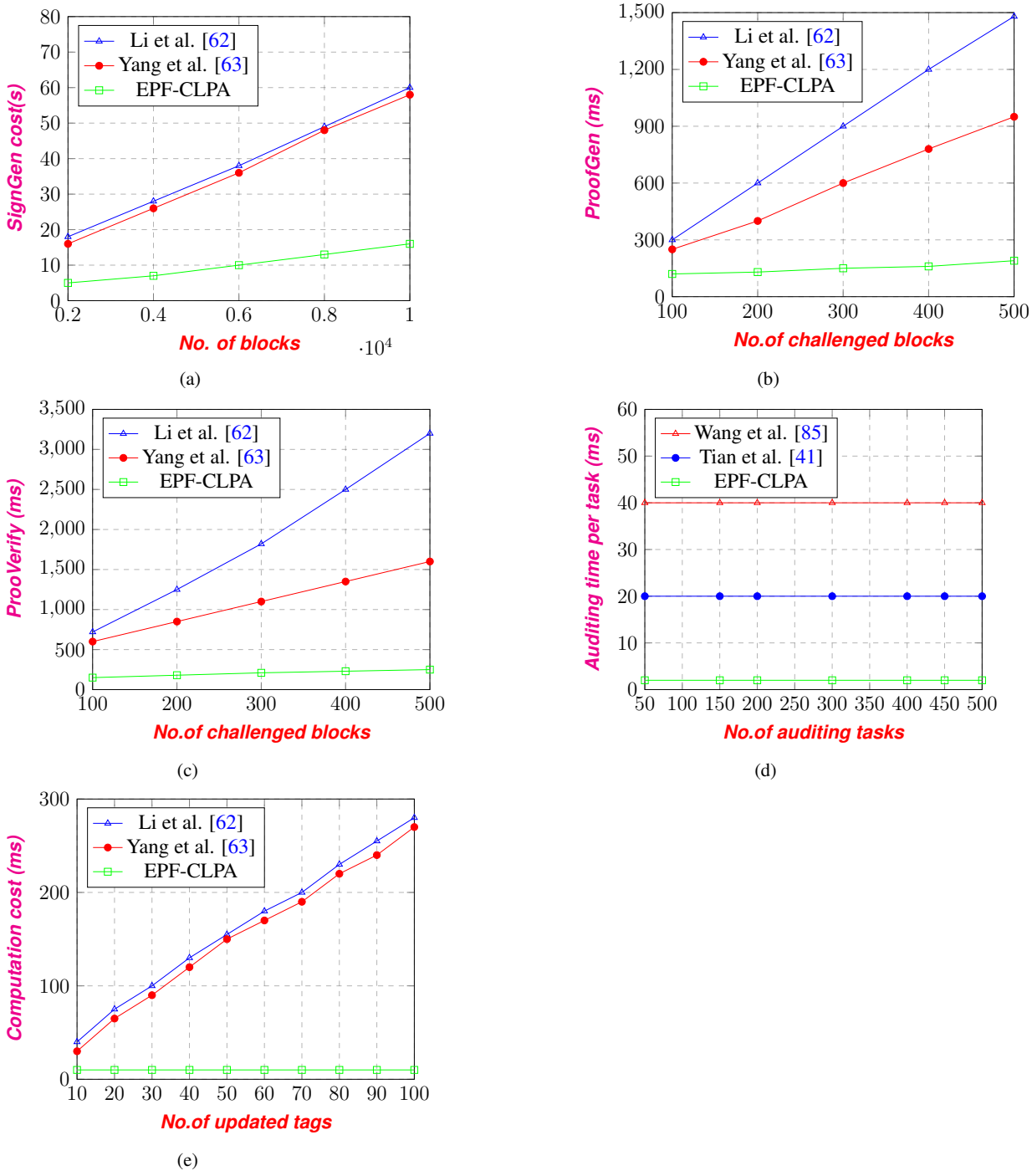



Fig. 8.2. (a) sign generation (b) proof generation (c) proof verification (d) Computation cost of batch auditing (e) Computation cost of user revocation

that EPF-CLPA is efficient than other existing schemes [62, 63] because of utilization of ECC. This is carried out only once during the life time of the system. Fig. 8.2b provides the comparison of computation costs for $ProofGen$. From Fig. 8.2b, it is obvious that our EPF-CLPA has the better efficiency than existing pairing-based [62, 63] the EPF-CLPA has less exponential and multiplication operations in group and. Fig. 8.2c illustrates the comparison of computation costs of $ProofVerify$. From Fig. 8.2c, we can observe that verification time of EPF-CLPA is lesser than all pairing based schemes [62, 63] since no involvement of bilinear pairing operations. Fig. 8.2d shows comparison cost of the batch auditing between our EPF-CLPA and the schemes in [85, 41]. From Fig. 8.2d we can observe that all the three schemes can handle different verification's from different users simultaneously, the average auditing time per task in EPF-CLPA is more efficient than that in [85, 41] because our EPF-CLPA is free from pairings, whereas [85, 41] uses pairings. Fig. 8.2e shows the computation cost of user revocation. From Fig. 8.2e we can see that our scheme is efficient than [62, 63] because in [62, 63], the cloud server performs resigning of revoked block with resigning key, which increases the computation overhead for big data, whereas in our scheme GM simply generates new time key for the non-revoked users, which only needs one scalar multiplication and one addition operation.

## 8.7 Summary

In this chapter, we presented an efficient PF-CLPA scheme for auditing shared big data in cloud by relying on the ECC which does not require pairings and reduces computation and communication cost drastically. Further, we leverage certificateless cryptography signature scheme to generate signatures for the blocks of a shared file, which simplifies certificate management and eliminates key escrow problem simultaneously. EPF-CLPA is extended to support batch auditing by handling multiple tasks simultaneously that improves the auditing performance and can applicable to shared big data storage systems. EPF-CLPA also supports secure user revocation by updating the time key of non-revoked users. The security analysis proved that proposed EPF-CLPA is secure against type I/II/III/IV adversaries in the ROM under a standard assumption. The performance analysis and experimental re-

sults demonstrate that the EPF-CLPA is efficient in terms of computation cost than existing schemes and more suitable for shared big data auditing in cloud.

# Chapter 9

# Conclusion and Future Directions

In this concluding chapter, we conclude the thesis, while we put forth certain open challenges in remote data integrity auditing scheme for future work.

## 9.1 Conclusion

In this thesis, we studied the importance of remote data integrity auditing in cloud storage, particularly public integrity auditing for shared data in cloud storage and proposed five RDIA schemes namely, i) An ID-Based public auditing for shared data in cloud computing using identity-based signatures to achieve user revocation. With this scheme we also simplified the certificate management. ii) In ABPIA scheme, we used individual private keys of each user for generating signatures and unique public key for integrity checking, which simplifies key management. In ABPIA scheme, the signature does not reveal any user identity; hence, user privacy achieved. iii) Certificateless privacy preserving public auditing system for dynamic shared data storage in cloud computing by utilizing certificate-less signatures. CLPPPA achieves privacy preserving against TPA by masking the data proof during auditing process while refraining from both complex certificate management and key escrow problem. Besides, CLPPPA also supports data dynamics such as insertion, modification and deletion through EDLIT and efficient user revocation utilizing proxy signatures. iv) Certificateless multi-replica public integrity auditing scheme for dynamic shared data in cloud storage to achieve the availability of data along with the integrity. v) Efficient Pair-

ing Free Certificateless Public Auditing for Shared Big Data in the Cloud based on ECC to reduce the computation and communication cost substantially during auditing. Through the security analysis, we prove that all schemes are provably secure against various adversaries under the hardness assumption of the standard DL and CDH problems in ROM. The performance and experimental evaluation show that our schemes are efficient and practical.

All these schemes achieve necessary functional and security features of shared data auditing such as correctness, soundness, public auditing, data dynamics, identity privacy, data privacy, user revocation, availability, and batch auditing. Furthermore, these schemes simplifies the certificate management and eliminating key escrow problem simultaneously with minimal computation and communication overhead during auditing process.

As the proposed shared data auditing schemes are secure, it would be more appropriate and suitable for real-time applications like financial, healthcare, scientific, and educational applications.

## 9.2 Future Directions

In all proposed schemes, we rely on trusted TPAs to execute auditing tasks, which reduces the burden on users. However, various risks associated with involving a TPA include potential privacy leaks, collusion, cheating, framing, and procrastination. Therefore, we cannot fully trust TPA in real-world scenarios. To avoid the trust problem of TPAs or to eliminate the need for TPA, the presented shared data schemes could be extended to use block-chain technologies to ensure integrity of the shared data.

# Author's Publications

**Journals:**

1. Gudeme JR, Pasupuleti SK, Kandukuri R (2019) Review of remote data integrity auditing schemes in cloud computing: taxonomy, analysis, and open issues. *Int J Cloud Comput*, 8(1):20–49
2. Gudeme, J.R., Pasupuleti, S.K. Kandukuri, R. (2020) Attribute-based public integrity auditing for shared data with efficient user revocation in cloud storage. *J Ambient Intelligence and Human Computing,* Springer, https://doi.org/10.1007/s12652-020-02302-6.
3. Gudeme, J.R., Pasupuleti, S.K. Kandukuri, R. (2021) Certificateless multi-replica public integrity auditing scheme for dynamic shared data in cloud storage. *Computers & Security*. Elsevier, 103, https://doi.org/10.1016/j.cose.2020.102176
4. Gudeme, J.R., Pasupuleti, S.K. Kandukuri, R. (2021) Certificateless privacy preserving public auditing for dynamic shared data with group user revocation in cloud storage *Journal of Parallel and Distributed Computing*. Elsevier, 156, pp: 163-175 https://doi.org/10.1016/j.jpdc.2021.06.001
5. Gudeme, J.R., Pasupuleti, S.K. Kandukuri, R. Efficient Pairing Free Certificateless Public Integrity Auditing for Shared Big Data on Cloud. *Journal of Super Computing.* Springer, (Communicated)

**Conferences:**

1. Gudeme, Jaya Rao, Syam Kumar Pasupuleti, and Ramesh Kandukuri."Public Integrity Auditing for Shared Data with Efficient and Secure User Revocation in Cloud Computing." *International Conference On Advances in Communication and Computing Technology (ICACCT)*. IEEE,pp.588-593, 2018.

# Bibliography


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

[2] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Osama Khan, Lea Kissner, Zachary Peterson, and Dawn Song. Remote data checking using provable data possession. *ACM Transactions on Information and System Security (TISSEC)*, 14(1):1–34, 2011.

[3] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[4] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616, 2009.

[5] Kui Ren, Cong Wang, and Qian Wang. Security challenges for the public cloud. *IEEE Internet Computing*, 16(1):69–73, 2012.

[6] Md Tanzim Khorshed, ABM Shawkat Ali, and Saleh A Wasimi. A survey on gaps, threat remediation challenges and some thoughts for proactive attack detection in cloud computing. *Future Generation computer systems*, 28(6):833–851, 2012.

[7] Mehdi Sookhak, Hamid Talebian, Ejaz Ahmed, Abdullah Gani, and Muhammad Khurram Khan. A review on remote data auditing in single cloud server: Taxonomy and open issues. *Journal of Network and Computer Applications*, 43:121–141, 2014.

[8] Ayad F Barsoum and M Anwar Hasan. On verifying dynamic multiple data copies over cloud servers. *IACR Cryptology ePrint Archive*, 2011:447, 2011.

[9] Jaya Rao Gudeme, Syam Kumar Pasupuleti, and Ramesh Kandukuri. Review of remote data integrity auditing schemes in cloud computing: taxonomy, analysis, and open issues. *International Journal of Cloud Computing*, 8(1):20–49, 2019.

[10] C Chris Erway, Alptekin Küpçü, Charalampos Papamanthou, and Roberto Tamassia. Dynamic provable data possession. *ACM Transactions on Information and System Security (TISSEC)*, 17(4):1–29, 2015.

[11] Qian Wang, Cong Wang, Jin Li, Kui Ren, and Wenjing Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. In *European symposium on research in computer security*, pages 355–370. Springer, 2009.

[12] Chang Liu, Jinjun Chen, Laurence T Yang, Xuyun Zhang, Chi Yang, Rajiv Ranjan, and Ramamohanarao Kotagiri. Authorized public auditing of dynamic big data storage on cloud with efficient verifiable fine-grained updates. *IEEE Transactions on Parallel and Distributed Systems*, 25(9):2234–2244, 2013.

[13] S Hariharasitaraman and SP Balakannan. A dynamic data security mechanism based on position aware merkle tree for health rehabilitation services over cloud. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–15, 2019.

[14] Hua Zhang, Tengfei Tu, et al. Dynamic outsourced auditing services for cloud storage based on batch-leaves-authenticated merkle hash tree. *IEEE Transactions on Services Computing*, 2017.

[15] Ayad Barsoum and Anwar Hasan. Enabling dynamic data and indirect mutual trust for cloud computing storage systems. *IEEE Transactions on Parallel and Distributed Systems*, 24(12):2375–2385, 2013.

[16] Yan Zhu, Gail-Joon Ahn, Hongxin Hu, Stephen S Yau, Ho G An, and Chang-Jun Hu. Dynamic audit services for outsourced storages in clouds. *IEEE Transactions on Services Computing*, 6(2):227–238, 2011.

[17] Kan Yang and Xiaohua Jia. An efficient and secure dynamic auditing protocol for data storage in cloud computing. *IEEE transactions on parallel and distributed systems*, 24(9):1717–1726, 2012.

[18] Jian Shen, Dengzhi Liu, Md Zakirul Alam Bhuiyan, Jun Shen, Xingming Sun, and Aniello Castiglione. Secure verifiable database supporting efficient dynamic operations in cloud computing. *IEEE Transactions on Emerging Topics in Computing*, 2017.

[19] P Syam Kumar and R Subramanian. Rsa-based dynamic public audit service for integrity verification of data storage in cloud computing using sobol sequence. *International Journal of Cloud Computing*, 1(2-3):167–200, 2012.

[20] Cong Wang, Sherman SM Chow, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for secure cloud storage. *IEEE transactions on computers*, 62(2):362–375, 2011.

[21] 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*, 23(9):1432–1437, 2011.

[22] Yong Yu, Man Ho Au, Yi Mu, Shaohua Tang, Jian Ren, Willy Susilo, and Liju Dong. Enhanced privacy of a remote data integrity-checking protocol for secure cloud storage. *International Journal of Information Security*, 14(4):307–318, 2015.

[23] Lifei Wei, Haojin Zhu, Zhenfu Cao, Xiaolei Dong, Weiwei Jia, Yunlu Chen, and Athanasios V Vasilakos. Security and privacy for storage and computation in cloud computing. *Information Sciences*, 258:371–386, 2014.

[24] Solomon Guadie Worku, Chunxiang Xu, Jining Zhao, and Xiaohu He. Secure and efficient privacy-preserving public auditing scheme for cloud storage. *Computers & Electrical Engineering*, 40(5):1703–1713, 2014.

[25] Yan Zhu, Hongxin Hu, Gail-Joon Ahn, and Mengyang Yu. Cooperative provable data possession for integrity verification in multicloud storage. *IEEE transactions on parallel and distributed systems*, 23(12):2231–2244, 2012.

[26] Chuan Yao, Li Xu, Xinyi Huang, and Joseph K Liu. A secure remote data integrity checking cloud storage system from threshold encryption. *Journal of Ambient Intelligence and Humanized Computing*, 5(6):857–865, 2014.

[27] Changsheng Wan, Juan Zhang, Bei Pei, and Changsong Chen. Efficient privacy-preserving third-party auditing for ambient intelligence systems. *Journal of Ambient Intelligence and Humanized Computing*, 7(1):21–27, 2016.

[28] Jiangtao Li, Lei Zhang, Joseph K Liu, Haifeng Qian, and Zheming Dong. Privacy-preserving public auditing protocol for low-performance end devices in cloud. *IEEE Transactions on Information Forensics and Security*, 11(11):2572–2583, 2016.

[29] Wenting Shen, Jia Yu, Hui Xia, Hanlin Zhang, Xiuqing Lu, and Rong Hao. Lightweight and privacy-preserving secure cloud auditing scheme for group users via the third party medium. *Journal of Network and Computer Applications*, 82:56–64, 2017.

[30] Yannan Li, Yong Yu, Bo Yang, Geyong Min, and Huai Wu. Privacy preserving cloud data auditing with efficient key update. *Future Generation Computer Systems*, 78:789–798, 2018.

[31] Lanxiang Chen. Using algebraic signatures to check data possession in cloud storage. *Future Generation Computer Systems*, 29(7):1709–1715, 2013.

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

[33] Boyang Wang, Baochun Li, and Hui Li. Knox: privacy-preserving auditing for shared data with large groups in the cloud. In *International conference on applied cryptography and network security*, pages 507–525. Springer, 2012.

[34] Boyang Wang, Baochun Li, and Hui Li. Oruta: Privacy-preserving public auditing for shared data in the cloud. *IEEE transactions on cloud computing*, 2(1):43–56, 2014.

[35] Boyang Wang, Baochun Li, and Hui Li. Panda: Public auditing for shared data with efficient user revocation in the cloud. *IEEE Transactions on Services Computing*, 1(8):92–106, 2015.

[36] Jiawei Yuan and Shucheng Yu. Efficient public integrity checking for cloud data sharing with multi-user modification. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 2121–2129. IEEE, 2014.

[37] Yuchuan Luo, Ming Xu, Shaojing Fu, Dongsheng Wang, and Junquan Deng. Efficient integrity auditing for shared data in the cloud with secure user revocation. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 434–442. IEEE, 2015.

[38] Tao Jiang, Xiaofeng Chen, and Jianfeng Ma. Public integrity auditing for shared dynamic cloud data with group user revocation. *IEEE Transactions on Computers*, 65(8):2363–2373, 2015.

[39] Anmin Fu, Shui Yu, Yuqing Zhang, Huaqun Wang, and Chanying Huang. Npp: a new privacy-aware public auditing scheme for cloud data sharing with group users. *IEEE Transactions on Big Data*, 2017.

[40] Libing Wu, Jing Wang, Sherali Zeadally, and Debiao He. Privacy-preserving auditing scheme for shared data in public clouds. *The Journal of Supercomputing*, 74(11):6156–6183, 2018.

[41] Hui Tian, Fulin Nan, Hong Jiang, Chin-Chen Chang, Jianting Ning, and Yongfeng Huang. Public auditing for shared cloud data with efficient and secure group management. *Information Sciences*, 472:107–125, 2019.

[42] Guangyang Yang, Jia Yu, Wenting Shen, Qianqian Su, Zhangjie Fu, and Rong Hao. Enabling public auditing for shared data in cloud storage supporting identity privacy and traceability. *Journal of Systems and Software*, 113:130–139, 2016.

[43] Yue Zhang, Jia Yu, Rong Hao, Cong Wang, and Kui Ren. Enabling efficient user revocation in identity-based cloud storage auditing for shared big data. *IEEE Transactions on Dependable and Secure Computing*, 2018.

[44] Wenting Shen, Jing Qin, Jia Yu, Rong Hao, and Jiankun Hu. Enabling identity-based integrity auditing and data sharing with sensitive information hiding for secure cloud storage. *IEEE Transactions on Information Forensics and Security*, 14(2):331–346, 2018.

[45] Yong Yu, Yannan Li, Bo Yang, Willy Susilo, Guomin Yang, and Jian Bai. Attribute-based cloud data integrity auditing for secure outsourced storage. *IEEE Transactions on Emerging Topics in Computing*, 2017.

[46] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Annual international cryptology conference*, pages 213–229. Springer, 2001.

[47] Jinshu Su, Dan Cao, Baokang Zhao, Xiaofeng Wang, and Ilsun You. epass: An expressive attribute-based signature scheme with privacy and an unforgeability guarantee for the internet of things. *Future Generation Computer Systems*, 33:11–18, 2014.

[48] Sattam S Al-Riyami and Kenneth G Paterson. Certificateless public key cryptography. In *International conference on the theory and application of cryptology and information security*, pages 452–473. Springer, 2003.

[49] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 514–532. Springer, 2001.

[50] Jonathan Katz. *Digital signatures*. Springer Science & Business Media, 2010.

[51] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[52] Arijit Karati, SK Hafizul Islam, and GP Biswas. A pairing-free and provably secure certificateless signature scheme. *Information Sciences*, 450:378–391, 2018.

[53] Ayad F Barsoum and M Anwar Hasan. Provable multicopy dynamic data possession in cloud computing systems. *IEEE Transactions on Information Forensics and Security*, 10(3):485, 2015.

[54] Chang Liu, Rajiv Ranjan, Chi Yang, Xuyun Zhang, Lizhe Wang, and Jinjun Chen. Mur-dpa: Top-down levelled multi-replica merkle hash tree based secure public auditing for dynamic big data storage on cloud. *IEEE Transactions on Computers*, 64(9):2609–2622, 2014.

[55] Mohammad Etemad and Alptekin Küpçü. Transparent, distributed, and replicated dynamic provable data possession. In *International Conference on Applied Cryptography and Network Security*, pages 1–18. Springer, 2013.

[56] Giuseppe Ateniese, Roberto Di Pietro, Luigi V Mancini, and Gene Tsudik. Scalable and efficient provable data possession. In *Proceedings of the 4th international conference on Security and privacy in communication netowrks*, pages 1–10, 2008.

[57] P Syam, M Ashok, and R Subramanian. A publicly verifiable dynamic secret sharing protocol for secure and dependable data storage in cloud computing [j]. *International Journal of Cloud Applications and Computing*, 2(3):1–25, 2012.

[58] Reza Curtmola, Osama Khan, Randal Burns, and Giuseppe Ateniese. Mr-pdp: Multiple-replica provable data possession. In *2008 the 28th international conference on distributed computing systems*, pages 411–420. IEEE, 2008.

[59] Huaqun Wang. Proxy provable data possession in public clouds. *IEEE Transactions on Services Computing*, 6(4):551–559, 2012.

[60] 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):1034–1038, 2008.

[61] Huaqun Wang, Debiao He, and Shaohua Tang. Identity-based proxy-oriented data uploading and remote data integrity checking in public cloud. *IEEE Transactions on Information Forensics and Security, DOI*, 10.

[62] Jiguo Li, Hao Yan, and Yichen Zhang. Certificateless public integrity checking of group shared data on cloud storage. *IEEE Transactions on Services Computing*, 2018.

[63] Hongbin Yang, Shuxiong Jiang, Wenfeng Shen, and Zhou Lei. Certificateless provable group shared data possession with comprehensive privacy preservation for cloud storage. *Future Internet*, 10(6):49, 2018.

[64] Debiao He, Sherali Zeadally, and Libing Wu. Certificateless public auditing scheme for cloud-assisted wireless body area networks. *IEEE Systems Journal*, 12(1):64–73, 2015.

[65] Su Peng, Fucai Zhou, Qiang Wang, Zifeng Xu, and Jian Xu. Identity-based public multi-replica provable data possession. *IEEE Access*, 5:26990–27001, 2017.

[66] Su Peng, Fucai Zhou, Jin Li, Qiang Wang, and Zifeng Xu. Efficient, dynamic and identity-based remote data integrity checking for multiple replicas. *Journal of Network and Computer Applications*, 134:72–88, 2019.

[67] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *Journal of cryptology*, 17(4):297–319, 2004.

[68] Jin Li, Xiao Tan, Xiaofeng Chen, and Duncan S Wong. An efficient proof of retrievability with public auditing in cloud computing. In *2013 5th International Conference on Intelligent Networking and Collaborative Systems*, pages 93–98. IEEE, 2013.

[69] Jin Li, Xiao Tan, Xiaofeng Chen, Duncan S Wong, and Fatos Xhafa. Opor: Enabling proof of retrievability in cloud computing with resource-constrained devices. *IEEE Transactions on cloud computing*, 3(2):195–205, 2014.

[70] Cong Wang, Sherman S-M Chow, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for secure cloud storage. *IEEE TRANSACTIONS ON CLOUD COMPUTING*, 2013.

[71] Ayad F Barsoum and M Anwar Hasan. Provable multicopy dynamic data possession in cloud computing systems. *IEEE Transactions on Information Forensics and Security*, 10(3):485–497, 2014.

[72] Yujue Wang, Qianhong Wu, Bo Qin, Wenchang Shi, Robert H Deng, and Jiankun Hu. Identity-based data outsourcing with comprehensive auditing in clouds. *IEEE transactions on information forensics and security*, 12(4):940–952, 2016.

[73] YANG Xiaodong, LI Yutong, WANG Jinli, MA Tingchun, and WANG Caifen. Revocable identity-based proxy re-signature scheme in the standard model. *Journal on Communications*, 40(5):153.

[74] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Workshop on the theory and application of cryptographic techniques*, pages 47–53. Springer, 1984.

[75] Yuchuan Luo, Ming Xu, Kai Huang, Dongsheng Wang, and Shaojing Fu. Efficient auditing for shared data in the cloud with secure user revocation and computations outsourcing. *Computers & Security*, 73:492–506, 2018.

[76] Jiawei Yuan and Shucheng Yu. Public integrity auditing for dynamic data sharing with multiuser modification. *IEEE Transactions on Information Forensics and Security*, 10(8):1717–1726, 2015.

[77] Jianhong Zhang and Qiaocui Dong. Efficient id-based public auditing for the outsourced data in cloud storage. *Information Sciences*, 343:1–14, 2016.

[78] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 90–107. Springer, 2008.

[79] Joseph A Akinyele, Christina Garman, Ian Miers, Matthew W Pagano, Michael Rushanan, Matthew Green, and Aviel D Rubin. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering*, 3(2):111–128, 2013.

[80] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 457–473. Springer, 2005.

[81] Praveen Kumar Premkamal, Syam Kumar Pasupuleti, and PJA Alphonse. A new verifiable outsourced ciphertext-policy attribute based encryption for big data privacy and access control in cloud. *Journal of Ambient Intelligence and Humanized Computing*, 10(7):2693–2707, 2019.

[82] Boyang Wang, Baochun Li, Hui Li, and Fenghua Li. Certificateless public auditing for data integrity in the cloud. In *2013 IEEE conference on communications and network security (CNS)*, pages 136–144. IEEE, 2013.

[83] Qi Li, Jianfeng Ma, Rui Li, Ximeng Liu, Jinbo Xiong, and Danwei Chen. Secure, efficient and revocable multi-authority access control system in cloud storage. *Computers & Security*, 59:45–59, 2016.

[84] Jiangang Shu, Kan Yang, Xiaohua Jia, Ximeng Liu, Cong Wang, and Robert Deng. Proxy-free privacy-preserving task matching with efficient revocation in crowdsourcing. *IEEE Transactions on Dependable and Secure Computing*, 2018.

[85] Boyang Wang, Baochun Li, and Hui Li. Panda: Public auditing for shared data with efficient user revocation in the cloud. *IEEE Transactions on services computing*, 8(1):92–106, 2013.

[86] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures-how to sign with rsa and rabin. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 399–416. Springer, 1996.

[87] Haiyang Yu, Yongquan Cai, Richard O Sinnott, and Zhen Yang. Id-based dynamic replicated data auditing for the cloud. *Concurrency and Computation: Practice and Experience*, 31(11):e5051, 2019.

[88] Joseph A Akinyele, Christina Garman, Ian Miers, Matthew W Pagano, Michael Rushanan, Matthew Green, and Aviel D Rubin. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering*, 3(2):111–128, 2013.

[89] John Gantz and David Reinsel. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. pages 1–16, 2012.

[90] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of cryptology*, 13(3):361–396, 2000.