

Wireless Networks

Yuan Zhang
Chunxiang Xu
Xuemin Sherman Shen

Data Security in Cloud Storage

 Springer

Wireless Networks

Series Editor

Xuemin Sherman Shen
University of Waterloo
Waterloo, ON, Canada

The purpose of Springer's new Wireless Networks book series is to establish the state of the art and set the course for future research and development in wireless communication networks. The scope of this series includes not only all aspects of wireless networks (including cellular networks, WiFi, sensor networks, and vehicular networks), but related areas such as cloud computing and big data. The series serves as a central source of references for wireless networks research and development. It aims to publish thorough and cohesive overviews on specific topics in wireless networks, as well as works that are larger in scope than survey articles and that contain more detailed background information. The series also provides coverage of advanced and timely topics worthy of monographs, contributed volumes, textbooks and handbooks.

More information about this series at <http://www.springer.com/series/14180>


Yuan Zhang • Chunxiang Xu •
Xuemin Sherman Shen

Data Security in Cloud Storage

 Springer

Yuan Zhang 
School of Computer Science & Engineering
University of Electronic Science
and Technology of China
Chengdu, Sichuan, China

Chunxiang Xu 
School of Computer Science & Engineering
University of Electronic Science
and Technology of China
Chengdu, Sichuan, China

Xuemin Sherman Shen 
Department of Electrical and Computer
Engineering
University of Waterloo
Waterloo, ON, Canada

ISSN 2366-1186

ISSN 2366-1445 (electronic)

Wireless Networks

ISBN 978-981-15-4373-9

ISBN 978-981-15-4374-6 (eBook)

<https://doi.org/10.1007/978-981-15-4374-6>

© Springer Nature Singapore Pte Ltd. 2020

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd. The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Preface

Cloud storage is a service that lets users store data by transferring it over the Internet or another network to an offsite storage system maintained by a third party. It is increasingly demanded along the users' data exploded and it provides users an efficient and convenient way to manage their data. Despite the appealing advantages of data outsourcing, cloud storage services are confronted with various threats from many aspects. Compared with traditional data storage systems (where users store their data locally), cloud storage provides users with a completely different paradigm to manage their data, which also introduces new and challenging threats towards data security. Specifically, in the cloud storage service, users do not physically own their data once outsourcing the data to a cloud server (which is subject to a cloud service provider), and these data are fully controlled by the cloud server. As such, once the cloud server (including insiders working at the cloud service provider) misbehaves, the outsourced data would suffer from threats, such as corruption, modification, removal, and privacy violation. In addition, since the data are transmitted over public and insecure networks, external adversaries (e.g., hackers) might eavesdrop on the communication channel between the user and the cloud server, tamper with the interaction messages between them, and extract the data contents from the cloud server for financial or political reasons.

This monograph gives a comprehensive overview of data security in cloud storage, which includes cloud storage reliability, cloud storage confidentiality, and data investigations in cloud storage. With these security issues, five research topics are introduced and studied, i.e., secure verification of data integrity, secure deduplication, secure keyword search, secure data provenance, and secure data time-stamping. This monograph not only presents basic paradigms and principles of the aforementioned research topics and the corresponding techniques that secure cloud storage but also provides a comprehensive survey on each of the research topics. In addition, this monograph also analyzes the relationship among these research topics.

As emerging techniques, such as indistinguishability obfuscation, blockchains, and trusted execution environments (TEEs), have been developed in the last decade, it has shown great potentials in enhancing data security. This monograph also introduces the latest advances in enhancing cloud storage reliability, confidentiality,

and investigations and analyzes their pros and cons. Finally, open research issues and future work on the related topics are also discussed.

We would like to thank Prof. Nan Cheng (Xidian University), Prof. Hongwei Li (University of Electronic Science and Technology of China), Prof. Xiaohui Liang (University of Massachusetts at Boston), Prof. Xiaodong Lin (University of Guelph), Prof. Jianbing Ni (Queen's University), Prof. Haomiao Yang (University of Electronic Science and Technology of China), Prof. Kan Yang (The University of Memphis), Prof. Shui Yu (University of Technology Sydney), Prof. Xiaojun Zhang (Southwest Petroleum University), and Prof. Jianying Zhou (Singapore University of Technology and Design) for their contributions in the presented research works. We would also like to thank Shanshan Li and Dongxiao Liu for reviewing parts of this monograph and all the members of BBCR group for the valuable discussions and their insightful suggestions, ideas, and comments. Special thanks also go to the staff at Springer Science+Business Media: Celine Chang, Susan Lagerstrom-Fife, Jane Li, and Suraj Kumar, for their help throughout the publication process.

Chengdu, China
Chengdu, China
Waterloo, Canada

Yuan Zhang
Chunxiang Xu
Xuemin Sherman Shen

Contents

1	Introduction	1
1.1	An Overview of Cloud Storage	2
1.1.1	Cloud Storage Architecture	2
1.1.2	Cloud Storage Applications	3
1.2	Data Security in Cloud Storage	5
1.3	Organization of the Monograph	7
	References	9
2	Basic Techniques for Data Security	11
2.1	Data Authentication	11
2.1.1	Message Authentication Code	12
2.1.2	Hash Function	13
2.1.3	Digital Signature	14
2.2	Data Confidentiality	16
2.2.1	Symmetric-Key Encryption	16
2.2.2	Public-Key Encryption	17
2.3	Threshold Cryptography	18
2.4	Public-Key Cryptosystems	18
2.4.1	PKI-Based Public-Key Cryptosystems	19
2.4.2	Identity-Based Public-Key Cryptosystems	19
2.4.3	Certificateless Public-Key Cryptosystems	20
2.5	Blockchain	20
2.6	Trusted Execution Environments	24
2.7	Summary and Further Reading	25
	References	25
3	Cloud Storage Reliability	29
3.1	Data Integrity	29
3.2	Proofs of Storage: Definition and Criteria	30
3.2.1	Threat Models	31
3.2.2	Security Criteria	33
3.3	Proofs of Storage for Cloud Storage Systems	34

3.3.1	Proofs of Storage for Dynamic Data	38
3.3.2	Enhancement of Security	40
3.3.3	Constructing Public Verification on Different Cryptosystems	42
3.3.4	Other Works	43
3.4	Latest Advances in Proofs of Storage	44
3.4.1	Proofs of Storage Based on Indistinguishability Obfuscation	44
3.4.2	Proofs of Storage Based on Blockchain	47
3.5	Summary and Further Reading	51
	References	52
4	Secure Deduplication	55
4.1	Deduplication Classification	55
4.2	Secure Deduplication: Threats and Countermeasures	57
4.2.1	Proofs of Ownership	58
4.2.2	Randomized Deduplication	60
4.3	Message-Locked Encryption	60
4.3.1	Overview	61
4.3.2	Threat Models of Encrypted Deduplication Storage Systems	64
4.3.3	Security Definition	65
4.4	Encrypted Deduplication Systems	65
4.4.1	Enhancement of Security	66
4.4.2	Practical Concern	73
4.4.3	Other Works	76
4.5	When Secure Deduplication Meets eHealth: A Case Study	76
4.5.1	Cloud-Based eHealth Systems	77
4.5.2	Adversary Model and Security Goals	78
4.5.3	Analysis of EMRs in Actual eHealth Systems	79
4.5.4	Study of HealthDep	81
4.6	Summary and Further Reading	84
	References	84
5	Secure Keyword Search	87
5.1	Keyword Search Over Encrypted Data	87
5.2	Symmetric-Key Searchable Encryption	89
5.2.1	System and Threat Models	89
5.2.2	Survey on Symmetric-Key Searchable Encryption	89
5.3	Public-Key Searchable Encryption	98
5.3.1	System model	99
5.3.2	Threat Model and Security Definition	100
5.3.3	Survey on Public-Key Searchable Encryption	100
5.4	Latest Advances in Public-Key Searchable Encryption	104
5.4.1	Public-Key Searchable Encryption Against Keyword Guessing Attacks	104

- 5.4.2 Remark and Further Discussion 112
- 5.5 Summary and Further Reading 113
- References 114
- 6 Secure Data Provenance** 119
 - 6.1 Introduction to Secure Data Provenance 119
 - 6.1.1 Data Provenance vs. Secure Data Provenance 120
 - 6.1.2 System and Threat Models 123
 - 6.2 Survey on Secure Data Provenance 125
 - 6.3 Blockchain: A Panacea for Secure Data Provenance 127
 - 6.3.1 Blockchain-Based Secure Data Provenance 128
 - 6.3.2 Implementation Based on Ethereum 135
 - 6.3.3 Data Provenance and Beyond: Further Discussion 137
 - 6.4 Summary and Further Reading 139
 - References 140
- 7 Secure Data Time-Stamping** 143
 - 7.1 Introduction to Secure Data Time-Stamping 143
 - 7.1.1 What Kinds of Data Would Benefit from Secure Time-Stamping? 144
 - 7.1.2 System and Threat Models 145
 - 7.2 Survey on Secure Time-Stamping 146
 - 7.3 Secure Time-Stamping and Blockchain 149
 - 7.3.1 Distributed Cryptocurrencies from Secure Time-Stamping 150
 - 7.3.2 Secure Time-Stamping from Blockchain 151
 - 7.4 Summary and Further Reading 164
 - References 164
- 8 Summary and Future Research Directions** 167
 - 8.1 Summary 167
 - 8.2 Future Work 169
 - 8.2.1 Secure Data Integrity Verification from Smart Contract 169
 - 8.2.2 Combination of Encrypted Deduplication and Symmetric-key Searchable Encryption 170
 - 8.2.3 Secure Provenance Under Complex Models 171
 - 8.2.4 Securely Time-stamping Operations in the Digital World 171

Acronyms

CA	Certificate authority
CDN	Content distribution network
CE	Convergent encryption
eHealth	Electronic healthcare
EHRs	Electronic health records
EMRs	Electronic medical records
EPD	Essential provenance data
FE	Functional encryption
FHE	Fully homomorphic encryption
HIPAA	Health Insurance Portability and Accountability Act
HVTs	Homomorphic verifiable tags
IdP	Identity provider
IMEI	International Mobile Equipment Identity
<i>iO</i>	Indistinguishability obfuscation
IoT	Internet of things
IRS	Index Repository Service
KGA	Keyword guessing attack
KGC	Key generation center
MHT	Merkle hash tree
MLE	Message-locked encryption
NPD	Nonessential provenance data
OPRF	Oblivious pseudorandom function
ORAM	Oblivious random access machine
PDP	Provable data possession
PIR	Private information retrieval
PKI	Public-key infrastructure
PKG	Private key generator
PoR	Proofs of retrievability
PoS	Proof of stake
PoW	Proof of work
POW	Proof of ownership

PSE	Public-key searchable encryption
SE	Searchable encryption
SSE	Symmetric-key searchable encryption
SIM	Subscriber identity module
TEEs	Trusted execution environments
TPA	Third-party auditor
TSP	Time-stamping service provider
WoL	Window of latching
WoT	Window of time-stamping

Chapter 1

Introduction



Currently, digital data are explosively generated and lots of data-intensive applications are emerging, which is pushing us towards the era of big data and we have to change the data management paradigm [1]. Recent reports from International Data Corporation (IDC, <https://www.idc.com/>) indicate that the digital data we create and copy are doubling in size every 2 years, have researched 18 ZB in 2018. IDC also predicts that the digital data in the digital world will reach 44 ZB by 2020 and will grow to 175 ZB in 2025 (<https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf>). We have enjoyed great advancements in our knowledge, services, and productivity brought by available big data sets. This can be reflected in several areas, such as the healthcare industry, education industry, and electronic commerce, and has deeply impacted on human society.

On the other hand, due to the large volume of data we create and the new paradigm of utilizing data in emerging applications, we have to deploy and maintain local storage devices and services to access and utilize the data, which causes considerable costs to us. With cloud storage services, both individuals and commercial users (e.g., enterprises) are able to outsource their data to a cloud server and access the data remotely via the Internet. Such services provide users with an efficient and flexible way to manage their data without bearing heavy costs to maintain the data locally. Some recent reports [2, 3] point out that more than 79% of organizations attempt to utilize data outsourcing services, and such an increasing demand for the cloud storage service leads to the growing number of cloud storage providers.

1.1 An Overview of Cloud Storage

In a sense, the cloud storage service serves as a fundamental component for most of the services provided by cloud service providers. In this section, we introduce cloud storage and its applications.

1.1.1 Cloud Storage Architecture

Generally, there are two entities in a cloud storage system: users and cloud service provider, as shown in Fig. 1.1.

Users are the data owners. They have a large number of data files to be outsourced to a remote server and want to access the outsourced data flexibly and efficiently. Such a data outsourcing service is provided by a cloud service provider who has significant storage space and computation resources. The storage space and computation capability are provided by a large number of powerful machines and devices that are deployed/employed and maintained by the cloud service provider.

Roughly speaking, the cloud service provider employs a three-layer framework to provide the cloud storage service [2]. On the top of the framework, the cloud service provider interacts with all users to receive their service requests including data outsourcing, data access, and other operations on their data. After receiving a service request from a user, the cloud service provider handles it via some prescribed algorithms and returns the corresponding result to the user as the response. The cloud service provider deploys or employs a large number of machines and devices

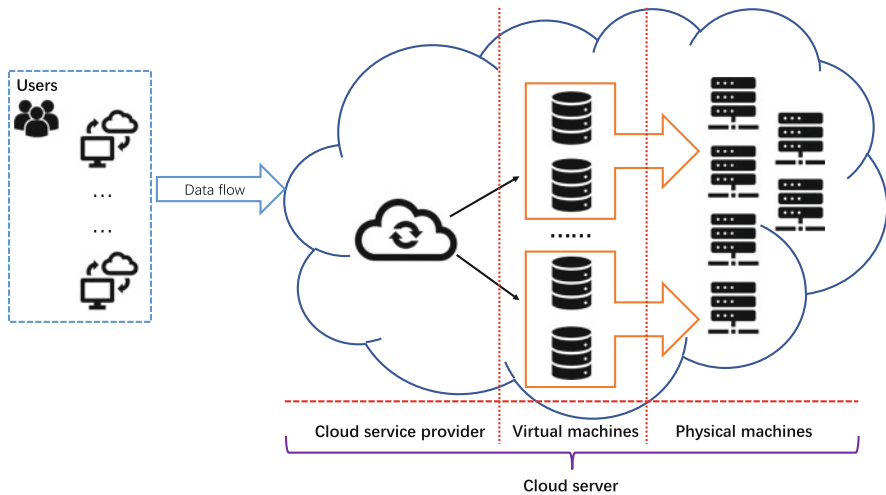


Fig. 1.1 Cloud storage architecture

to provide the cloud storage service. These machines and devices might be located in different places in the physical world to ensure the high quality of service, reduce operating costs, and/or enhance the reliability of the system. However, it also introduces additional costs to utilize these machines and devices for the cloud service provider. To address this problem, an intermediate layer between the cloud service provider and the physical machines and devices, called virtual machines, is introduced. It enables all machines and devices to form a group and work together, such that the cloud service provider is able to utilize them easily and efficiently.

In the cloud storage system, all the operations performed by the cloud service provider are transparent to its users. Users only need to focus on how to utilize their data, rather than how the data are stored. As such, from the perspective of users, the cloud service provider and its machines and devices can be considered as a whole entity, which is well known as the cloud server. Furthermore, such a data management paradigm enables users to outsource their data using a device (e.g., computers) and to subsequently access the outsourced data using different devices (e.g., smartphones and laptops). This is very useful in some data-intensive applications that will be elaborated in the next section.

1.1.2 Cloud Storage Applications

Cloud storage is the most prominent manifestation of cloud computing. It inherits the unprecedented advantages of cloud computing, such as on-demand self-service, broad network access, location independent resource pooling, rapid elasticity, usage-based pricing, and transference of risk [4]. In addition, it also has inherent characteristics, such as relief of the burden for storage management, universal data access with location independence, and avoidance of capital expenditure on hardware, software, and personnel maintenance.

Therefore, the cloud storage service has become a fundamental component in some applications and has deeply impacted on people's daily life. We briefly introduce several cloud-based applications that can be enhanced and improved significantly by utilizing cloud storage services, namely electronic healthcare, data sharing, and the Internet of things (IoT).

1.1.2.1 Electronic Healthcare

Compared with traditional paper-based systems, electronic healthcare (eHealth) systems provide a more efficient, less error-prone, and more flexible service for both doctors and patients [5, 6]. As modern eHealth systems are data-intensive, both the patients and medical institutions have to bear heavy costs to maintain electronic health records (short for EHRs).

Actually, the wide deployment of cloud storage services has already shown great benefits in managing EHRs, which not only allows patients and medical institutions

to outsource EHRs to the cloud server [7, 8], but also makes a great contribution to the judgment and dispute resolution in medical malpractices [9]. In addition, practical eHealth systems always need to support data sharing among different entities. Typical scenarios include group consultations, where a patient is treated by a group of doctors and EHRs are generated by these doctors one by one, and each doctor generates EHRs according to the ones generated by the previous doctor.

Traditionally, this is achieved by requiring the patient to transfer the EHRs generated by the previous doctor to the next one, which is very cumbersome and inefficient. With the employment of cloud storage services, the patient's EHRs are outsourced to the cloud server, she/he would never need to transfer the EHRs by herself/himself. Instead, she/he just needs to delegate all authorized doctors to access her/his EHRs from the cloud server, and thereby improving efficiency significantly.

1.1.2.2 Data Sharing

Data sharing is a fundamental requirement for many applications [10]. In the previous section, we have introduced the significance of data sharing in eHealth systems. In this section, we provide more applications and discuss the potentials of utilizing cloud storage services.

Particularly, in an open-source software development system, the system requirements are released by a project manager, and multiple developers would work together as a group on the project to complete the source code. In such an application, the source code completed by one developer might be revised by another one. Traditionally, the project manager has to keep online to assist all developers in completing the source code to ensure the consistency of the source code on each developer. Each developer not only requests the latest version of the source code from the manager, but also needs to know how the source code was revised and updated, which is error-prone and cumbersome for the manager. The same problems also exist in the collaboration systems wherein multiple users jointly edit a document.

With the employment of cloud storage services, the open-source software development system can be directly implemented, and it is easy to share data among developers via the cloud server [11]. Currently, several systems, such as Codeanywhere [12] and Google Doc [13], have been implemented.

1.1.2.3 IoT

In most IoT applications, IoT devices are lightweight and have limited capabilities in terms of storage and computation [14]. Therefore, these IoT devices cannot have capabilities of processing the data collected by them and can only send raw data to a target entity. In most cases, the entity not only needs to store all raw data for

data archiving, but also has to process the data before using it, which introduces considerable costs.

However, with the employment of cloud storage services, the IoT devices in these applications are enhanced significantly in terms of storage and computation capability. After collecting the data, the IoT device can send it to the cloud server which not only provides the data backup service, but also assists target entities in processing the data in an efficient and flexible way [15].

1.2 Data Security in Cloud Storage

While people enjoy the desirable benefits from the cloud storage service, critical security concerns in data outsourcing have been raised seriously [16–18]. Compared with traditional data storage systems (users store their data locally), the cloud storage service provides users with a completely different paradigm to manage their data, which introduces new and challenging threats towards data security. Specifically, in the cloud storage service, users do not physically own their data once outsourcing the data to the cloud server, and these data are fully controlled by the cloud server. As such, once the cloud server (including insiders working at the cloud service provider) misbehaves, the outsourced data would be confronted with threats, such as corruption, modification, removal, and privacy violation. In addition, since the data are transmitted over public and insecure networks, external adversaries (e.g., hackers) might eavesdrop on the communication channel between the user and the cloud server, tamper with the interaction messages between them, and extract the data contents from the cloud server for financial or political reasons [19, 20].

However, ensuring the security of cloud storage services is challenging in reality, which is reflected in two aspects. On the one hand, although the infrastructures under the cloud service provider are much powerful and reliable than personal machines and devices, they still suffer from internal faults for the data security. These internal faults include network failures, system malfunctions, misoperations, and software bugs. They would occur in practice, no matter what high degree of reliable countermeasures the cloud service provider would employ. On the other hand, both the cloud server and the external adversary have a strong motivation to compromise the data security if this could increase their profits in the system significantly. Therefore, it is critical and challenging to ensure data security in cloud storage [21, 22].

From the perspective of users, data security in cloud storage mainly includes three parts: data reliability, data confidentiality, and secure data investigations.

Data reliability of cloud storage service serves as the fundamental assurance of its security [23, 24]. Recall that users would not physically own their data once having outsourced the data to the cloud server. They are always worried about the data integrity, i.e., whether the outsourced data is well maintained on the cloud server. However, the integrity of outsourced data is being put at risk in practice.

For example, the cloud server may always conceal incidents of data corruption for a good reputation, or may delete a part of data that is never accessed to reduce storage costs. Moreover, an external adversary motivated by financial or political reward may attempt to tamper with the outsourced data but convince the users that their data are still retained intact.

Data confidentiality is an essential part of cloud storage security. In reality, some of the outsourced data include the privacy information about their owners, and thereby are very sensitive. Without proper protection, adversaries (including the internal and external ones) could easily extract the data contents from the outsourced data, which allows the adversaries to violate the users' privacy. To protect data contents against adversaries, users always encrypt their data before outsourcing. This can be achieved by utilizing conventional encryption, e.g., AES [25], but it makes efficient data outsourcing and fine-grained access to outsourced data impossible, which introduces considerable costs to users. This problem can be further divided into two parts and will be discussed below.

On the one hand, in reality, different users would outsource the same data to one cloud server, which enables the cloud server to perform data deduplication across its users to reduce storage costs. However, due to the randomness of conventional encryption algorithms (i.e., different users would produce different ciphertexts for the same data), deduplication over encrypted data is impeded [26].

On the other hand, in some cases, after a user outsources an entire (encrypted) data set to the cloud server, she/he would only need to retrieve some subset of the outsourced data set from the cloud server. In this case, if the data is encrypted by using conventional encryption algorithms, the user has to download the entire data set, decrypt it, and retrieve the target data from the entire data set. This introduces prohibitive costs in terms of communication and computation on the user side [27, 28].

Therefore, in actual cloud storage systems, although data confidentiality is a basic requirement for users, more interfaces that support efficient data outsourcing and data retrieval are also required. However, new security challenges and threats towards data confidentiality are also introduced.

Secure data investigations guarantee the trustworthiness of outsourced data and the cloud storage service. As discussed before, cloud storage has served as a fundamental component in several data-intensive applications, e.g., eHealth, to make the data management easy and reliable. On the other hand, the data outsourced to the cloud server also serves as the key evidence in post investigations. For example, in an eHealth system, when a medical malpractice occurs, an investigator who is subject to an authority needs to reconcile the dispute among the medical institution and the patient as well as her/his families. The most important evidence to reconcile the dispute is the corresponding EHRs outsourced to the cloud server. However, it is usual that the defendant challenges the authenticity of a digital evidence during the trial, since EHRs are fully controlled by the cloud server and the defendant may question the EHRs that investigators are working on and presented in the courtroom is not the same ones originated from the medical institution.

To interference with the judge's judgment, an adversary might launch the following two types of attacks.

First, the adversary would tamper with the outsourced data and the corresponding provenance information which keeps track of what happens to the data throughout the lifecycle of the data. If the adversary succeeds, the outsourced data cannot serve as the evidence since they are invalid [29].

Second, the adversary would also tamper with the time when the data is created to interference with the judgment. In some cases, the creation time of a data file is very critical in post investigations [30–32]. If the adversary successfully back-dates/forward-dates the data, the trustworthiness and authenticity of the judgment based on the data cannot be guaranteed.

1.3 Organization of the Monograph

In this monograph, we investigate the data security issues in cloud storage systems. The aim of this monograph is threefold.

First, we elaborate on the threats described above and analyze how they can be utilized by adversaries and why these vulnerabilities are critical. This presents basic paradigms and principles of secure cloud storage to a general audience with a basic computer, communication, or cryptography background. This also serves as a general introduction suitable for beginning researchers in related areas.

Second, this monograph includes five research topics in secure cloud storage: secure data integrity verification, secure deduplication, secure keyword search, secure data provenance, and secure data time-stamping. We make a comprehensive survey of each research topic and analyze the pros and cons of schemes and techniques in each research topic. We stress that these research topics are closely related to each other, and these research works jointly ensure data security in cloud storage. Specifically, the relationship among these topics is shown in Fig. 1.2. From the perspective of a user, after she/he outsources data to a cloud server, she/he first cares about the reliability of the storage service, i.e., whether the data remains intact on the cloud server. This is related to the data integrity verification technique. Once the reliability is ensured, the user would focus on their privacy, i.e., whether someone, who is not authorized by the user, is able to extract the contents of her/his data from the cloud server. This assurance corresponds to data confidentiality and can be provided by the employment of encryption algorithms. However, from the point of view of pragmatism, the user would wish the costs of utilizing cloud storage service as low as possible, but the privacy preservation retains. The technique that satisfies this requirement is secure deduplication. Furthermore, the user also needs to retrieve target data from the cloud server without downloading the entire ciphertext set, which requires the cloud storage system to guarantee the confidentiality of outsourced data without sacrificing its availability. The technique that accomplishes this goal is secure keyword search. In addition, a secure cloud storage system also needs to support data investigations, where the provenance

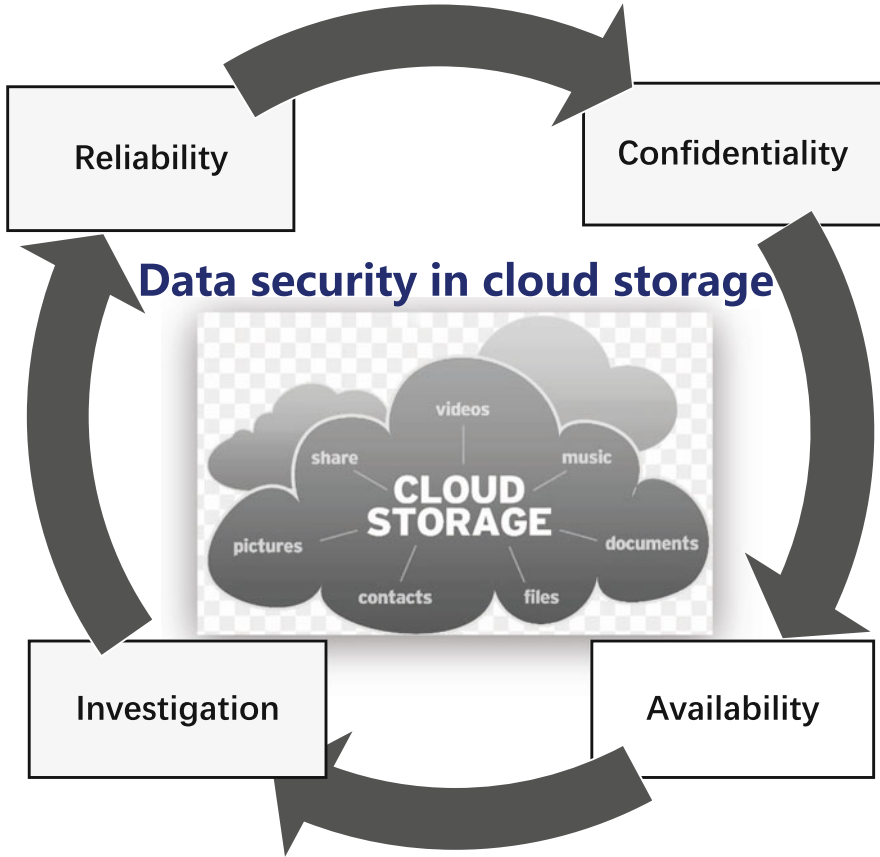


Fig. 1.2 Relationship among research topics

information about outsourced data should be securely maintained. Consequently, secure data provenance is of critical importance in cloud storage systems. Some cloud-based applications also need to certify the time when the data is created, since the creation time of the data also serves as a key evidence in data investigations for these applications. As such, secure time-stamping is also an important technique of data investigations in cloud storage systems. All these techniques have been deeply investigated in these research topics and can be utilized to construct secure cloud storage systems to provide users with reliable, economical, secure, and efficient storage services.

Third, we also provide a literature review and study on the latest advances in each research topic and outlook several potential research directions. This serves as a reference for experts in related areas.

The remainder of the monograph is organized as follows: In Chap. 2, we introduce basic theorems, cryptographic primitives, and techniques for the protection of

data security, which serves as preliminary knowledge for subsequent chapters. In Chap. 3, we introduce secure data integrity verification, which is the most important technique to ensure the reliability of cloud storage. In Chap. 4, we introduce secure data deduplication, which can be utilized to ensure data confidentiality while saving storage space. In Chap. 5, we introduce secure keyword search, which achieves search over ciphertexts by keywords. In Chap. 6, we introduce secure data provenance, which is a fundamental technique in secure data investigations. In Chap. 7, we introduce secure time-stamping, which is the key cryptographic primitive to certify the time when data is created and is critical to support secure data investigations. Finally, in Chap. 8, we summarize the monograph and outlook some future research directions in secure cloud storage systems.

References

1. McAfee A, Brynjolfsson E, Davenport TH, Patil D, Barton D (2012) Big data: the management revolution. *Harv Bus Rev* 90(10):60–68
2. Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I (2009) Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility. *Future Gener Comput Syst* 25(6):599–616
3. Sookhak M, Gani A, Talebian H, Akhuzada A, Khan SU, Buyya R, Zomaya AY (2015) Remote data auditing in cloud computing environments: a survey, taxonomy, and open issues. *ACM Comput Surv* 47(4):1–34
4. Mell P, Grance T (2011) The NIST definition of cloud computing. <http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf>
5. Zhang Y, Xu C, Li H, Yang K, Zhou J, Lin X (2018) HealthDep: an efficient and secure deduplication scheme for cloud-assisted eHealth systems. *IEEE Trans Ind Inf* 14(9):4101–4112
6. Zhang K, Shen X (2015) Security and privacy for mobile healthcare networks. Springer, Berlin
7. Liang J, Qin Z, Xiao S, Ou L, Lin X (2019) Efficient and secure decision tree classification for cloud-assisted online diagnosis services. *IEEE Trans Dependable Secure Comput*. <https://doi.org/10.1109/TDSC.2019.2922958>
8. Liang J, Qin Z, Xiao S, Zhang J, Yin H, Li K (2020) Privacy-preserving range query over multi-source electronic health records in public clouds. *J Parallel Distrib Comput* 135:127–139
9. Cao S, Zhang G, Liu P, Zhang X, Neri F (2019) Cloud-assisted secure eHealth systems for tamper-proofing EHR via blockchain. *Inf Sci* 485:427–440
10. Zhang Y, Xu C, Zhao J, Zhang X, Wen J (2015) Cryptanalysis of an integrity checking scheme for cloud data sharing. *J Inf Secur Appl* 23:68–73
11. Wang H, Zhang Y, Chen K, Sui G, Zhao Y, Huang X (2019) Functional broadcast encryption with applications to data sharing for cloud storage. *Inf Sci* 502:109–124
12. Codeanywhere. <https://codeanywhere.com>
13. Google doc. http://www.google.cn/intl/zh-cn_all/docs/about/
14. Zhang Y, Xu C, Li H, Yang K, Cheng N, Shen X (2020) PROTECT: efficient password-based threshold single-sign-on authentication for mobile users against perpetual leakage. *IEEE Trans Mob Comput*. <https://doi.org/10.1109/TMC.2020.2975792>
15. Lin X, Ni J, Shen X (2018) Privacy-enhancing fog computing and its applications. Springer, Heidelberg
16. Wang C, Wang Q, Ren K, Lou W (2010) Privacy-preserving public auditing for data storage security in cloud computing. In: *IEEE international conference on computer communications*, pp 1–9

17. Wang C, Chow SS, Wang Q, Ren K, Lou W (2013) Privacy-preserving public auditing for secure cloud storage. *IEEE Trans Comput* 62(2):362–375
18. Zhang Y, Xu C, Liang X, Li H, Mu Y, Zhang X (2017) Efficient public verification of data integrity for cloud storage systems from indistinguishability obfuscation. *IEEE Trans Inf Forensics Secur* 12(3):676–688
19. Kamara S, Lauter K (2010) Cryptographic cloud storage. In: *International conference on financial cryptography and data security*, pp 136–149
20. Zhang Y, Xu C, Yu S, Li H, Zhang X (2015) SCLPV: secure certificateless public verification for cloud-based cyber-physical-social systems against malicious auditors. *IEEE Trans Comput Social Syst* 2(4):159–170
21. Yang K, Jia X (2013) An efficient and secure dynamic auditing protocol for data storage in cloud computing. *IEEE Trans Parallel Distrib Syst* 24(9):1717–1726
22. Zhang Y, Xu C, Lin X, Shen X (2019) Blockchain-based public integrity verification for cloud storage against procrastinating auditors. *IEEE Trans Cloud Comput*. <https://doi.org/10.1109/TCC.2019.2908400>
23. Armknecht F, Barman L, Bohli J, Karame GO (2016) Mirror: enabling proofs of data replication and retrievability in the cloud. In: *{USENIX} security symposium*, pp 1051–1068
24. Zhang Y, Xu C, Li H, Liang X (2016) Cryptographic public verification of data integrity for cloud storage systems. *IEEE Cloud Comput* 3(5):44–52
25. Daemen J, Rijmen V (2013) *The design of Rijndael: AES—the advanced encryption standard*. Springer, Berlin
26. Bellare M, Keelveedhi S, Ristenpart T (2013) Message-locked encryption and secure deduplication. In: *Annual international conference on the theory and applications of cryptographic techniques*, pp 296–312
27. Curtmola R, Garay J, Kamara S, Ostrovsky R (2011) Searchable symmetric encryption: improved definitions and efficient constructions. *J Comput Secur* 19(5):895–934
28. Zhang Y, Xu C, Ni J, Li H, Shen X (2019) Blockchain-assisted public-key encryption with keyword search against keyword guessing attacks for cloud storage. *IEEE Trans Cloud Comput*. <https://doi.org/10.1109/TCC.2019.2923222>
29. Lu R, Lin X, Liang X, Shen X (2010) Secure provenance: the essential of bread and butter of data forensics in cloud computing. In: *ACM symposium on information, computer and communications security*, pp 282–292
30. Haber S, Stornetta WS (1990) How to time-stamp a digital document. In: *Annual cryptology conference*, pp 437–455
31. Zhang Y, Xu C, Li H, Yang H, She X (2019) Chronos: secure and accurate time-stamping scheme for digital files via blockchain. In: *IEEE international conference on communications*, pp 1–6
32. Zhang Y, Xu C, Cheng N, Li H, Yang H, Shen X (2019) Chronos+: an accurate blockchain-based time-stamping scheme for cloud storage. *IEEE Trans Serv Comput* 13(2):216–229. <https://doi.org/10.1109/TSC.2019.2947476>

Chapter 2

Basic Techniques for Data Security



In this chapter, we introduce basic theorems, cryptographic primitives, and techniques for protection of data security. These theorems, cryptographic primitives, and techniques serve as fundamental building blocks in subsequent chapters and have widely used in different application scenarios to ensure data security.

2.1 Data Authentication

We first present notations and conventions used in this monograph. We follow the usage of asymptotic notation in [1].

Given two integers $i, j \in N(i \leq j)$, where N is the natural number set, we denote by $[i, j]$ the set $\{i, i + 1, i + 2, \dots, j\}$. Given a finite set \mathbf{T} , $|\mathbf{T}|$ denotes the number of components in \mathbf{T} . Given two bit-strings x and y , $x||y$ denotes their concatenation. For a function F whose inputs include two fields, assuming its inputs are a and b , its output is y , the computation of F is denoted by $y = F(a, b)$.

Common notations and their description are given in the following. We denote by \oplus the exclusive-or (XOR) operator, by $\{0, 1\}^*$ the set of all finite bit-strings, by $\{0, 1\}^n$ the set of all bit-strings of length n , by p a prime number, by Z_p the additive group of integers modulo p and the set $\{0, 1, \dots, p - 1\}$, by Z_p^* the multiplicative group of invertible integers modulo p , by G an additive group, by G_T a multiplicative group, and by e a bilinear pairing described below.

Bilinear Maps We assume that G is an additive group whose order is a prime p and G_T is a multiplicative group with the same order. $e : G \times G \rightarrow G_T$ is a bilinear map if it has three properties:

1. Bilinearity: $e(xP, yQ) = e(P, Q)^{xy}$, $\forall P, Q \in G$ and $a, b \in Z_p^*$;
2. Non-degeneracy: $\forall P, Q \in G$ and $P \neq Q$, $e(P, Q) \neq 1$;
3. There is an efficient algorithm to compute e .

We introduce several data authentication techniques, including message authentication code, hash function, and digital signatures.

We stress that all these data authentication techniques only target at guarantee message integrity, rather than message confidentiality. Note that these techniques have the same goal (i.e., ensuring the integrity of the target message), but they are applied in completely different scenarios, which will be elaborated later.

2.1.1 Message Authentication Code

The objective of a message authentication code is to protect an adversary against tampering with a message sent by one entity to another, or against injecting a new message, without detecting by the receiver.

MAC requires the sender (say Alice) and the receiver (say Bob) to share a secret that is unknown to the adversary, it is a cryptographic primitive within the symmetric-key setting, i.e., Alice and Bob share the same secret key k . When Alice wishes to send a message M to Bob in an authenticated way, she computes a MAC tag τ based on M and k , and sends M and τ to Bob. Upon receiving M and τ , Bob verifies whether τ is valid by using k and the received M . Alice computes τ using a tag-generation algorithm MAC . Bob verifies the validity of τ using a verification algorithm Vrfy .

Formally, a MAC consists of three algorithms, i.e., KeyGen , MAC , and Vrfy , such that

- KeyGen is a key-generation algorithm, it takes as input the security parameter and outputs a key k ;
- MAC is a tag-generation algorithm, it takes as input k and a message $M \in \{0, 1\}^*$, and outputs a MAC tag τ .
- Vrfy is a verification algorithm, it takes as input k , M , and τ , outputs a bit of $\{0, 1\}$, where if the output is 1, it means that τ is valid under k and M ; if the output is 0, it means that τ is invalid under k and M .

The security of MAC requires that an adversary cannot generate a valid tag on any “new” message that its MAC tag was not previously generated. This security is captured by a security notion of “existentially unforgeable under an adaptive chosen-message attack.” Here, “existential unforgeability” refers to the fact that the adversary must not be able to forge a valid tag on any message; “adaptive chosen-message attack” refers to the fact that the adversary can get MAC tags on arbitrary messages that are chosen adaptively during its attack, i.e., he can request a MAC tag on any message from a so-called MAC oracle as needed without any limitation on his choice [2].

2.1.2 Hash Function

At a fundamental level, a hash function maps a long input string to a shorter output string [3]. In this monograph, all hash functions we will use have the fixed-size output, which means that the size of output string of the hash function is fixed. The primary security requirement of hash functions is to achieve collision resistance [4].

Essentially, hash functions are two-input function that takes as input a key and a string, and outputs a short, fixed-size string. However, the key is generally not kept secret and the security requirement should be achieved even when the adversary knows the key. In other words, in most cases, the key of the hash function is not explicitly appeared when we describe the function and is inherently embedded into the function as a public parameter.

Formally, a hash function h consists of two algorithms, i.e., **KeyGen** and h , such that the following conditions hold.

- **KeyGen** is a key-generation algorithm, it takes as input a security parameter ℓ and outputs a key s .
- h takes as input s and a string $x \in \{0, 1\}^*$ and output a string $h^s(x) \in \{0, 1\}^{f(\ell)}$, where f is a function that maps ℓ bits to some bits depending on the setting, and $h^s(x)$ refers to the fact that s is inherently embedded into the function.

There are three security notions to capture the security of hash functions: collision resistance, second-preimage/target-collision resistance, and preimage resistance. The first one is the strongest one, and the last two are weaker than the first one.

Informally, a hash function h is collision resistant if it is computationally infeasible for any adversary to find a collision in h ; h is the second-preimage resistant of given s and a uniform x , it is computationally infeasible for any adversary to find $x' \neq x$ such that $h^s(x') = h^s(x)$; h is preimage resistant if given s and a uniform y , it is computationally infeasible for any adversary to find an input x such that $h^s(x) = y$, which essentially means that h^s is a one-way function.

As introduced before, both the MAC and the hash function are cryptographic primitives to guarantee message integrity within symmetric-key cryptosystems. However, there is a key difference between them, i.e., they are applied in completely different scenarios [5].

Specifically, MAC is designed for such a scenario: Alice and Bob share a secret key and want to communicate with each other in an authenticated way; Both Alice and Bob do not hold the message and the MAC tag in advance; After receiving a message as well as the corresponding MAC tag from Alice/Bob, Bob/Alice can detect whether there is an adversary that has tampered with the message.

Whereas, the hash function is designed for such a scenario: Alice outsources a message to a remote server (e.g., a cloud server) and wants to share the message with Bob by allowing Bob to download the message from the server; The hash value of the message is stored in a public but secure space such that Bob is able to obtain it and confirms that the hash value is the correct one; Bob downloads the message from the server and checks whether the downloaded message matches the hash value.

2.1.3 Digital Signature

Digital signatures can be considered as the public-key counterpart of message authentication codes (MACs), and their syntax and security assurances are analogous [1]. In a signature scheme, a signer first generates a pair of keys (sk , pk), where sk is called a private key (or secret key) that should be kept secret and is used to generate the signature and pk is called a verification key (or public key) that is published publicly and is used to verify the validity of the signature. Given sk , it is easy to compute pk , but given pk , it is infeasible to compute sk . This is based on some mathematical hard problems. The signature scheme allows the signer to generate a signature on a message using sk such that anyone who knows pk is able to verify the validity of the signature and whether the message originated from the signer was modified in transit. Compared with MAC, the signature scheme frees from the heavy costs caused by sharing the MAC key among two entities. This is economical and favorable for the scenario that a sender wants to send data to multiple receivers. Signature schemes also provide the property of non-repudiation, which is very useful and desirable in some application scenarios, but MAC cannot provide. Non-repudiation refers to the fact that once the signer signs a message, he cannot deny having done so at any later point in time. Furthermore, the signatures are publicly verifiable, this allows that a signature on a message generated by the signer could be shown to other entities if necessary.

Formally, a digital signature scheme consists of three algorithms: **KeyGen**, **Sig**, and **Vrfy**, such that:

- **KeyGen** is a key-generation algorithm, it takes as input a security parameter ℓ and output a pair of keys (sk , pk) as described before.
- **Sig** is a signature-generation algorithm, it takes as input sk and a message M , outputs a signature σ . In this monograph, this process is denoted by $\sigma = \text{Sig}(sk, M)$.
- **Vrfy** is a verification algorithm, it takes as input pk , M , and σ , and outputs a bit of $\{0, 1\}$, where if the output is 1, it means that σ is valid under sk and M ; if the output is 0, it means that σ is invalid under sk and M .

The security of signature schemes requires that given a fixed pk generated by the signer, it is computationally infeasible to forge a valid signature σ^* on a message M^* , where M was not previously signed by the singer. This is also captured by a security notion of existentially unforgeable under an adaptive chosen-message attack, which is the direct analogue of the security notion for MAC.

In reality, to enable a sufficiently long message can be signed in one task, we always utilize the hash-and-sign paradigm, where given a message $M \in \{0, 1\}^*$, the signer first computes its hash value $h(M)$ as the digest, then signs the digest. Note that due to the collision resistance of h , such a paradigm would not impact the security.

Currently, some variants of the digital signature scheme are proposed with different features for different application scenarios. Now, we introduce three of them that would be used in subsequent chapters of this monograph.

Group Signature It is a specific type of signature algorithms [6, 7], where a group of signers and a group manager are involved in the system. It has the following properties:

- Only signers in the group can sign a message;
- Anyone can verify the validity of a group signature but cannot learn which singer in the group signs it;
- If necessary, the signature can be “opened” with the aid of the group manager such that the real identity of the signer in the group who generates the signature can be retrieved.

The group signature is typically used in such a scenario: in a laboratory, a specially crafted device (e.g., a dedicated printer) can only be used by a specific group of users (e.g., the faculties in the laboratory). Due to resource limitations, the number of service requests made by each user in the group submitted to the device should be securely recorded. This can be achieved by employing a signature scheme, where each request made by a user is related to a signature signed by the user. However, to protect the users’ privacy, i.e., protecting the information about who accesses the device and how many times a user requests service from the device from other users, the signature should not reveal any information about the signer’s identity. In addition, there is also a need to learn the information about the service requests for each user if necessary. Note that traditional signature schemes cannot satisfy the above requirements, but the group signature can be employed to address all these problems.

Ring Signature Different from the group signature, a ring signature scheme only involves a group of users without the group manager. As a consequence, the identity of a user who generates a signature would not be revealed. The ring signature is typically used in electronic auction systems and electronic voting systems. In an electronic auction system, with the employment of the ring signature, an auctioneer is able to determine the winning bid without leaking losing bids, which preserves the privacy of bidders against leakage. Similarly, in an electronic voting system, by adopting the ring signature, the “dealer” can calculate the voting results without knowing the choice of each voter.

Blind Signature A blind signature scheme enables a receiver to obtain a signature on a message from a signer, where both the message and the resulting signature remain unknown to the signer. The blind signature is typically used in digital cryptocurrencies. With the adoption of the blind signature, the bank is able to authorize a transaction conducted by a user but does not know anything about the transaction.

2.2 Data Confidentiality

Data confidentiality is another important goal of modern cryptography, which ensures that the data contents cannot be learned by any entity who is not authorized. Data confidentiality is achieved by utilizing encryption schemes that keep the plaintext generated by one entity hidden from an eavesdropper who can intercept the communication channel and observe the ciphertext.

Generally, encryption schemes can be classified into two types: symmetric-key encryption and public-key encryption, which will be elaborated in the following sections.

2.2.1 Symmetric-Key Encryption

In the setting of symmetric-key encryption, two entities (e.g., Alice and Bob) share a key and leverage this key when they want to communicate with each other secretly. Specifically, Alice and Bob share a key k for subsequent communications. Alice first generates a message M and encrypts M using k to obtain a ciphertext C . Then Alice sends C to Bob. Upon receiving C , Bob decrypts it using k and recovers the original message M .

In such an application, both Alice and Bob use the same key to convert the message (i.e., plaintext) into a ciphertext and back. In other words, both entities hold the same key that is used for encryption and decryption.

Formally, a symmetric-key encryption scheme consists of three algorithms: **KeyGen**, **E**, and **D**, such that:

- **KeyGen** is a key-generation algorithm, it outputs a key k selected according to some distribution.
- **E** is an encryption algorithm, it takes as input k and a message M and outputs a ciphertext C . In this monograph, the encryption of M using k is denoted by $E(k, M)$.
- **D** is a decryption algorithm, it takes as input k and C , and outputs M or \perp , where \perp means that the decryption fails. In this monograph, the decryption of C using k is denoted by $D(k, C)$.

The most basic security notion for symmetric-key encryption is the security against a ciphertext-only attack, where an adversary only gets a single ciphertext. This notion is motivated by the primary goal of encryption schemes that keeps the plaintext generated by one entity hidden from an eavesdropper who can intercept the communication channel. Ideally, we always desire that a symmetric-key encryption is able to ensure that the adversary cannot learn any partial information about the plaintext from the ciphertext. This requirement is captured by the notion of semantic security and can be defined by such an indistinguishability game: given two plaintext M_0 and M_1 with the same size, and one of their ciphertext C_b , ($b \in \{0, 1\}$), the adversary cannot know the underlying plaintext of C_b .

However, the above security notion is somewhat weak, since it only considers the case that the adversary passively eavesdrops on a single ciphertext sent between two honest entities. As such, two stronger security notions are proposed. The first one is chosen-plaintext attacks (short for CPA) and the second one is the chosen-ciphertext attacks (short for CCA).

In reality, it is common that Alice sends multiple ciphertexts that are generated using the same key to Bob. In this case, the security against a ciphertext-only attack cannot be suitable, since the adversary might be able to eavesdrop all these ciphertexts with additional information to break the confidentiality of ciphertexts. The security against such an adversary is captured by the notion of CPA. Intuitively, CPA security refers to the fact that it is secure against the adversary who can know the ciphertexts of some messages that are chosen by himself.

CCA is the strongest security notion, in which the adversary has the ability not only to get the ciphertexts of messages that are chosen by himself, but also to get the messages (i.e., plaintexts) of ciphertexts that are chosen by himself.

2.2.2 Public-Key Encryption

We notice that in a symmetric-key encryption, Alice and Bob have to agree on a secret key before they communicate. Public-key encryption does not require this key agreement process, i.e., it addresses the key-distribution problem that exists in the symmetric-key encryption. Specifically, in a public-key encryption, there are also two entities: a sender (say Bob) and a receiver (say Alice). Alice generates a pair of keys (sk, pk) , where sk is called a private key (or secret key) and pk is called a public key. Similar to the digital signature, given sk , it is easy to compute pk , but given pk , it is infeasible to compute sk . Alice publishes pk and lets everyone who wants to send messages know pk . When Bob wants to send a message M to Alice, he encrypts M using pk to obtain the ciphertext C , and sends C to Alice. Upon receiving C , Alice is able to decrypt it using sk to obtain M .

Formally, a public-key encryption consists of three algorithms: **KeyGen**, **Enc**, and **Dec**, such that:

- **KeyGen** is a key-generation algorithm, it takes as input the security parameter ℓ and outputs a pair of keys (sk, pk) as described above.
- **Enc** is an encryption algorithm, it takes as input pk and a message M and outputs a ciphertext C . In this monograph, the encryption of M using pk is denoted by $Enc(pk, M)$.
- **Dec** is a decryption algorithm, it takes as input sk and C , and outputs M or \perp , where \perp means that the decryption fails. In this monograph, the decryption of C using sk is denoted by $Dec(sk, C)$.

The security notions of public-key encryption follow those of symmetric-key encryption, including CPA security and CCA security. The basic ideas of these notions are the same with a slight difference, due to the different cryptosystems.

2.3 Threshold Cryptography

A threshold cryptosystem enables multiple entities to perform certain cryptographic operations (e.g., signing a message, decrypting a ciphertext, and computing a shared secret) in a threshold way, such that at least the threshold number of them can efficiently accomplish the operations but less than the threshold number of them cannot.

The first threshold cryptosystem is the Shamir's secret sharing scheme [8]. It considers such a problem: a dealer has a secret s and wants to share s among some set of n users $\mathcal{U}_1, \mathcal{U}_2, \dots, \mathcal{U}_n$ by providing each one of them with a secret share; It also wishes that any t of them can pool their secret shares and reconstruct s , but coalition of less than t users would obtain nothing about s from their collective shares. A secret sharing scheme that satisfies the above requirement is called a (t, n) -threshold secret sharing scheme.

Based on the Shamir's secret sharing scheme, several cryptographic schemes can be constructed to achieve cryptographic operations in a threshold way. For example, we can construct a threshold signature scheme to generate a signature if and only if more than the threshold number of signers agrees on it [9]. This can be utilized in an electronic voting system and group communications.

However, the Shamir's secret sharing scheme also suffers from two problems. First, its security relies on the security and reliability of the dealer. The dealer generates the secret and distributes it to all users. If the dealer is compromised, the secret would be leaked. At this point, the dealer becomes the single point of failure in the system. Second, once the secret is shared among all users, the secret share of each user would not be changed, which is vulnerable to a sophisticated adversary (or a mobile adversary) who can perpetually attempt to break into these users and corrupt their secret shares one by one. It is feasible and practical for such the adversary to recover the secret given enough time. Therefore, for long-lived secrets, protection provided by the Shamir's secret sharing could be insufficient.

To address the first problem described above, a verifiable secret sharing scheme is proposed [10]. To address the second problem, a proactive secret sharing scheme is proposed [11, 12]. We will elaborate on these schemes in the following chapters.

2.4 Public-Key Cryptosystems

In previous sections, we have introduced several public-key cryptographic primitives, such as public-key encryption and digital signature. We notice that a primary requirement of them is that the public keys should be securely distributed. However, this is non-trivial to be achieved, especially in large-scale networks. The key problem to ensure the trustworthiness of public keys is to authenticate public keys. From the perspective of methods that authenticate public keys, public-key cryptosystems can mainly be classified into three categories: the ones are based

on the public key infrastructure (PKI), the ones are based on users' identities, and the ones are based on certificateless cryptography, which will be elaborated in the following.

2.4.1 PKI-Based Public-Key Cryptosystems

The key technique of PKI-based public-key cryptosystem is the digital certificate. Essentially, a certificate is a signature binding an entity to some public key. To ensure the trustworthiness of the certificate, in such a cryptosystem, a trusted Certificate Authority (short for CA) is employed to issue certificates. At a high level, in a PKI-based public-key cryptosystem, the public-key distribution can be described as the following example.

We assume that there are three entities: Alice, Bob, and CA, where Alice wants to publish her public key pk_{Alice} to others including Bob. CA has generated a key-pair (sk_{CA}, pk_{CA}) for issuing certificates. After generating (sk_{Alice}, pk_{Alice}) (sk_{Alice} is the secret/private key of Alice), Alice requests a certificate from CA by sending pk_{Alice} to CA. Upon receiving pk_{Alice} , CA computes the signature $\sigma_{CA \rightarrow Alice}$ whose underlying message is $Ca_{Alice} = \text{"Alice's public key is } pk_{Alice}\text{"}$. We call $Cert_{Alice} = \{\sigma_{CA \rightarrow Alice}, Ca_{Alice}\}$ a certificate for Alice's key issued by CA.

Anytime Bob holds pk_{Alice} and needs to verify its trustworthiness, he can request $Cert_{Alice}$ from Alice, and verifies the validity of $Cert_{Alice}$. The process that CA verifies the validity of pk_{Alice} and Alice chooses a trusted CA relies on a PKI which enables the widespread distribution of public keys. In reality, a certificate includes more data fields than that in the above example and different PKI models are proposed for different application scenarios. Here we would not introduce them in detail, but readers can refer to the references mentioned in Sect. 2.7.

Despite the advantages of the PKI-based public-key cryptosystem, it also suffers from critical problems in terms of security and efficiency. Specifically, in such the cryptosystem, Bob needs to manage the Alice's certificate for subsequent communications, this causes the certificate management problem to Bob, especially when Bob needs to communicate with a large number of entities. The certificate management problem includes certificate revocation, certificate storage, certificate distribution, and certificate verification, which is inefficient and cumbersome in practice and also faces security problems.

2.4.2 Identity-Based Public-Key Cryptosystems

To avoid the certificate management problem, an identity-based cryptosystem (short for IBC) is proposed [13, 14], where an entity (say Bob) does not need to maintain and verify the certificate of a target entity (say Alice).

In IBC, the public key of Alice is her identity (e.g., her name, email address, and identity number) which is publicly known and verifiable, and might be human-memorisable. Therefore, it is easy to verify the validity of Alice's public key for Bob without the assistance of CA.

To achieve the above requirement, a trusted entity, called private key generator (Short for PKG), is employed to generate the private key for each user in the system. PKG holds a master secret key msk and publish the corresponding public key and generates sk_{Alice} using msk and the Alice's identity ID_{Alice} .

However, IBC has an inherent disadvantage, i.e., the key escrow problem. Note that in IBC, each user's private key is generated by PKG, obviously, this private key is not only known to the user herself/himself, but also known to PKG. As a consequence, IBC cannot ensure true non-repudiation in the way that PKI can, and is also vulnerable to compromised PKG.

2.4.3 Certificateless Public-Key Cryptosystems

To address the certificate management problem existing in PKI-based cryptosystems and the key escrow problem existing in IBC, a paradigm of certificateless cryptosystem is proposed [15]. In such a cryptosystem, an independent entity, called key-generation center (short for KGC), is employed to free from the use of certificates and the key escrow problem.

The certificateless cryptosystem is intermediate between the PKI-based cryptosystem and IBC. In a certificateless cryptosystem, a user's private key consists of two parts. The first part is computed by KGC using the user's identity and a master secret key, and the second part is generated by the user herself/himself.

Compared with the PKI-based cryptosystems and IBC, generally, in certificateless cryptosystems, the cryptographic algorithms are always less efficient. Therefore, we cannot say that which one of these three types of cryptosystems is "best," it depends on the target application scenarios.

2.5 Blockchain

From the perspective of technique, a blockchain is a linear collection of data elements, where each data element is called a block. All blocks are linked to form a chain which is secured using a cryptographic hash function and is maintained by a group of participants. Each block typically contains a hash pointer as a link to a previous block, a timestamp, and transaction data. Only if a transaction's validity is verified by a majority of the participants, it can be recorded into the block. Generally, blockchains can be classified into two types: private blockchain and public blockchain. For a private blockchain (including the consortium blockchain), the participants who perform the verification are authorized by the blockchain

managers or are the managers themselves. For a public blockchain, anyone in the network can become the participant to perform the verification without any limitation.

Public blockchains serve as a key component in decentralized cryptocurrencies, e.g., Bitcoin [16] and Ethereum [17]. Essentially, in these cryptocurrencies, the blockchain is leveraged to record a public ledger to keep track of the ownership of each underlying value token. A transaction can be considered as a function that changes the ownership of specific tokens and updates the ledger. The participants who maintain the blockchain and add new blocks containing transactions are called miners. The security of blockchains ensures that only valid transactions can be recorded. Consensus algorithms play a key role in blockchain systems. Currently, public blockchain systems can be based on multiple consensus algorithms, e.g., proof of work (PoW) [16], proof-of-stake (PoS) [18], proof-of-space [19], etc. In the remainder of this monograph, we only use public blockchains (especially Ethereum [17]) to enhance data integrity. Generally, the security level a public blockchain can achieve is related to lots of factors, but it can be reflected by the market capitalization. The higher market capitalization a public blockchain has, the higher costs that an adversary breaks the security are.

A simplified Ethereum blockchain is illustrated in Fig. 2.1. A block consists of two parts of data. The first one is called the block header which is used to compute the hash value of the block. It includes the following data fields:

- Hash value of the last block. It serves as a pointer to the previous block, such that all blocks form a chain.
- Nonce. It is a solution of the PoW puzzles. The miner, who is the first one that finds a valid nonce, can determine and publish this block.
- Timestamp. It is a physical time that indicates when the corresponding block was created.
- Merkle root. It is the root value of a Merkle hash tree computed from all transactions in the current block.

The second one is called the transaction data, which includes all transactions in the current block. A graphical transaction in Ethereum is depicted in “**Transaction**” of Fig. 2.1.

The ledger of Ethereum can be thought of as a state transition system, where there is a “state” consisting of the ownership status of all existing Ethers (which are the value token of the Ethereum blockchain) and a “state transition function” that takes a state and a transaction as input, and outputs a new state which is the result. When a new block is added into the chain, all transactions recorded in the block should be verified first, and then miners compute a valid nonce such that the hash value of the block is less than or equal to a value provided by the Ethereum system. This process is a proof of work and is well known as “Mining.” Here, we only introduce the PoW-based Ethereum, other versions of Ethereum that are based on PoS or other consensus algorithms would not be used in this monograph and would not be introduced here. The first miner who finds the nonce broadcasts the block of transactions together with this nonce. Other miners can verify that the nonce is a

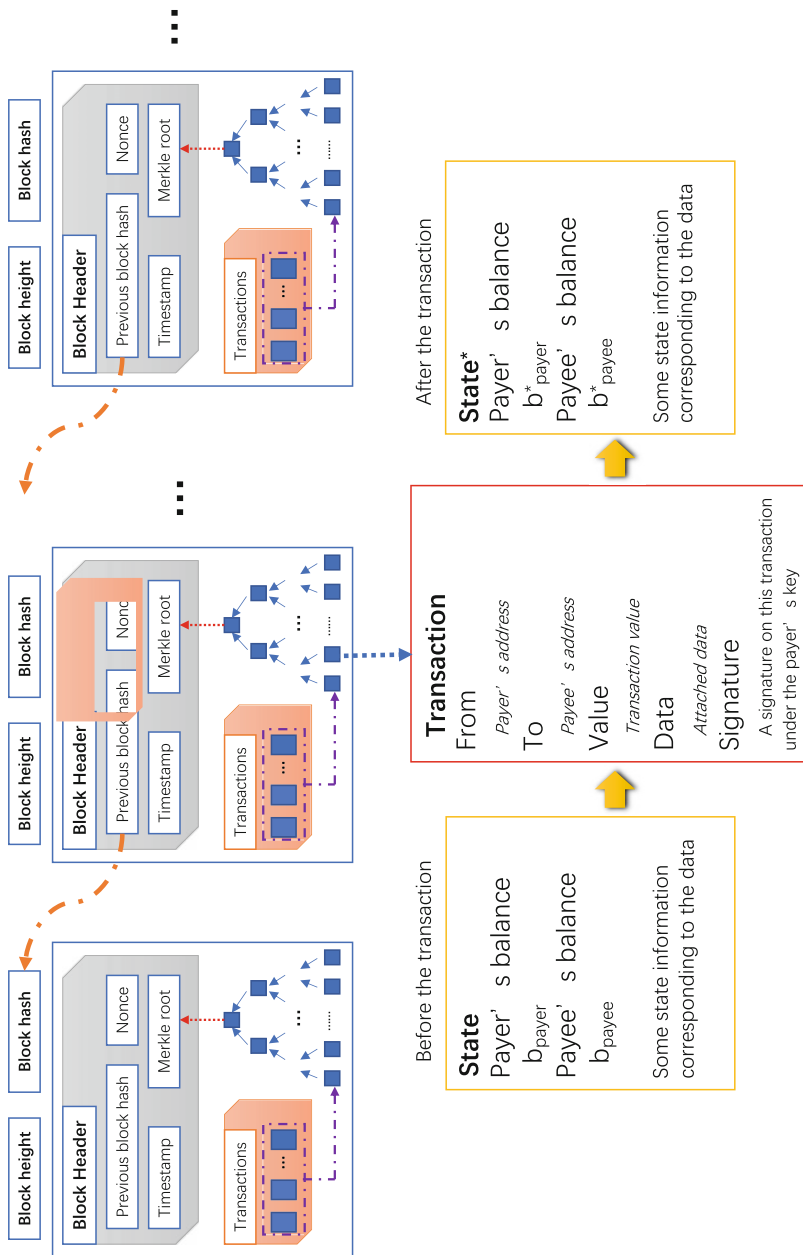


Fig. 2.1 Simplified Ethereum blockchain

valid solution, and hence add the new block to their blockchain. Once the block is added to the chain, all the corresponding state information has been updated.

In Ethereum, the state is made up of objects called “account.” In general, there are two types of accounts in Ethereum: externally owned accounts and contract accounts.

Externally owned accounts are controlled by secret keys and can conduct a transaction.

Contract accounts are controlled by their contract code, and each smart contract corresponds to a contract account. After the contract is deployed and contained by the Ethereum blockchain, one can send message to the contract account (i.e., transferring Ethers from an account to the contract account) to trigger the execution of the smart contract, where the data included in the “data field” can be set to be the input of the contract.

For the transaction between two external owned accounts, i.e., the payer transfers Ethers from her/his account to the payee’s account, if the transaction is recorded into the blockchain, the balances of these two accounts are updated. We notice that the transactions between two external accounts in Ethereum also include a “data” field. The payer can set the data field to be any binary data she/he selects when she/he generates the transaction.

There are three fundamental properties in the Ethereum blockchain as well as existing secure blockchain systems [18, 20–22]:

- φ -chain consistency. Blockchains of any two honest miners at any point in time during the mining execution can differ only in the last φ blocks, this is shown in Fig. 2.2.
- (ι, φ) -chain quality. For an honest miner’s blockchain, the fraction of blocks mined by honest miners in any sequence of φ or more successive blocks is at least ι . In other words, the probability that any φ successive blocks in a secure PoW-based blockchain are generated by an adversarial miner whose hashrate is less than 51% of the network’s mining hashrate can be negligible, as shown in Fig. 2.2. In Ethereum, $\varphi \geq 12$.

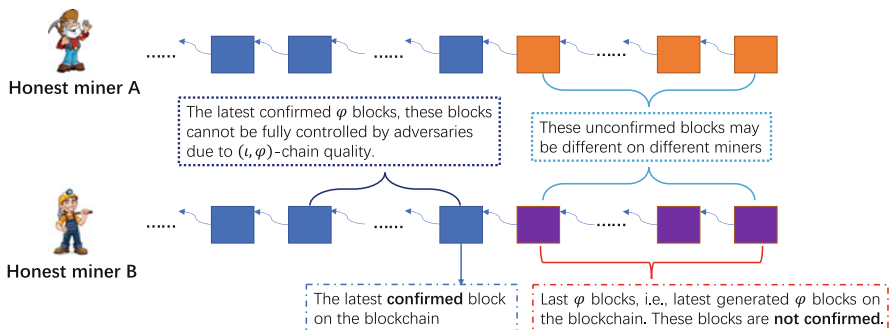


Fig. 2.2 Illustration of φ -chain consistency

- Chain growth. The number of blocks that are added to the blockchain during any given time interval is deterministic. In other words, the blockchain height can be trusted to steadily increase with respect to either short or long term.

With the above fundamental properties as well as the inherent characteristics of PoW-based consensus algorithm, we derive two properties from the Ethereum blockchain as follows [23–26], which will be utilized in subsequent sections.

- Unpredicted hash value. The hash value (denoted by `Blockhash` in Fig. 2.1) of each block on the chain only can be determined after a valid nonce is computed and verified by all miners. Once the block is appended to the chain, its hash value is deterministic and would never be changed. This means that given a time t , if t is a past time, the hash value of the latest block that has appeared since t in the blockchain is deterministic, and can be extracted efficiently; if t is a future time, the hash value is computationally unpredictable. We stress that *the fact that the hash value of blocks generated in the future cannot be predicted does not mean that the hash value cannot be biased by an adversary who has infinite budget*. Since the adversary can incentivize a miner who first mines a block to throw the newly mined block away and continue to mine if the hash value of the block does not meet the adversary’s requirement [27]. We also notice that other PoW-based on-chain currencies, such as Bitcoin, also have this feature [28, 29].
- Time-sensitive data state. For a transaction with a string Ti as its Data value, if a block including this transaction is accepted by a majority of miners and is chained to the blockchain, the string Ti is then time-stamped. It means that Ti is generated no later than the time that the block is chained to the blockchain. Thus Ti is time-sensitive.

2.6 Trusted Execution Environments

Most smartphones support system-wide trusted execution environments (short for TEEs), such as ARM TrustZone (<http://www.arm.com>). Such devices have two processors: an application processor and a baseband processor.

The application processor runs the mobile operating system (e.g., Android) and the applications on top of it. The application processor also runs a small layer of software called trusted operation system (short for OS) and the trusted applications on top of it. Consequently, the application processor supports two execution states: normal world and secure world, and only one state is active at a time. A trusted application can be executed in the secure world, only if it is certified by the device manufacturer.

The baseband processor runs the baseband operating system and handles cellular communications and mediates communications between the application processor and the subscriber identity module (short for SIM).

The architecture of a mobile device with the TrustZone system is described in Fig. 2.3, where we omit peripherals such as GPS, Bluetooth, etc. The smartphone

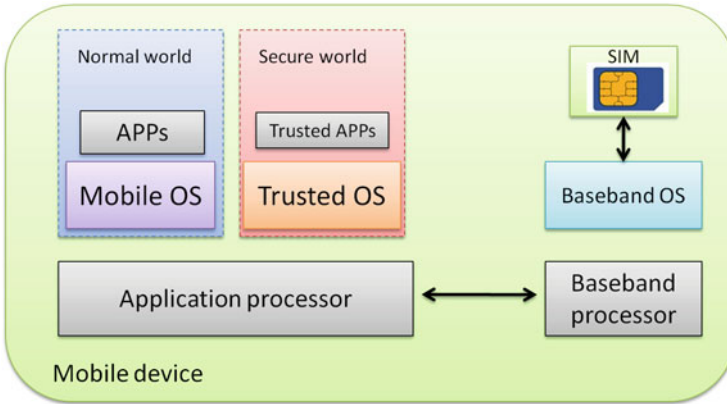


Fig. 2.3 The architecture of TrustZone-enabled smartphones

manufacturer embeds a device-specific key-pair in each smartphone, and also certifies each smartphone. Each smartphone has a device certificate, which contains an immutable device identifier (e.g., International Mobile Equipment Identity, IMEI) and the public key of the smartphone. The secret key embedded in the device is only accessible by applications that run in the secure world [30, 31].

2.7 Summary and Further Reading

In this chapter, we have introduced basic theorems, cryptographic primitives, and techniques for the protection of data security, which serves as the preliminary knowledge for the subsequent chapters.

In this chapter, all theorems, cryptographic primitives, and techniques are only briefly introduced. Readers are advised to related references for more technical details. Specifically, in [32], Menezes provides a survey on the bilinear pairing. The security definition for MAC is adapted by Bellare et al. [2] from the security definition for digital signatures provided by Goldwasser et al. [33]. The signature scheme utilized in the subsequent chapters is the BLS signature [34, 35], it is very easy to be understood. The pros and cons of PKI-based cryptosystems are detailed in [36, 37]. A comprehensive survey on the blockchain can be found in [38, 39]. An introduction of TEEs is provided in [40].

References

1. Katz J, Lindell Y (2014) Introduction to modern cryptography. CRC Press, New York
2. Bellare M, Kilian J, Rogaway P (2000) The security of the cipher block chaining message authentication code. *J Comput Syst Sci* 61(3):362–399

3. Carter JL, Wegman MN (1979) Universal classes of hash functions. *J Comput Syst Sci* 18(2):143–154
4. Damgård IB (1989) A design principle for hash functions. In: *International conference on the theory and application of cryptography*, pp 416–427
5. Stinson DR (2005) *Cryptography: theory and practice*. Chapman and Hall/CRC, London
6. Chaum D, Heyst EV (1991) Group signatures. In: *Workshop on the theory and application of cryptographic techniques*, pp 257–265
7. Camenisch J, Stadler M (1997) Efficient group signature schemes for large groups. In: *Annual cryptology conference*, pp 410–424
8. Shamir A (1979) How to share a secret. *Commun ACM* 22(11), pp 612–613
9. Zhang K (1997) Threshold proxy signature schemes. In: *International workshop on information security*. Springer, Berlin, pp 282–290
10. Chor B, Goldwasser S, Micali S, Awerbuch B (1985) Verifiable secret sharing and achieving simultaneity in the presence of faults. In: *IEEE annual symposium on foundations of computer science*, pp 383–395
11. Herzberg A, Jarecki S, Krawczyk H, Yung M (1995) Proactive secret sharing or: how to cope with perpetual leakage. In: *Annual cryptology conference*, pp 339–352
12. Zhang Y, Xu C, Li H, Yang K, Cheng N, Shen X (2020) PROTECT: efficient password-based threshold single-sign-on authentication for mobile users against perpetual leakage. *IEEE Trans Mob Comput*. <https://doi.org/10.1109/TMC.2020.2975792>
13. Shamir A (1984) Identity-based cryptosystems and signature schemes. In: *Workshop on the theory and application of cryptographic techniques*, pp 47–53
14. Boneh D, Franklin M (2001) Identity-based encryption from the weil pairing. In: *Annual cryptology conference*, pp 213–229
15. Al-Riyami SS, Paterson KG (2003) Certificateless public key cryptography. In: *International conference on the theory and application of cryptography and information security*, pp 452–473
16. Nakamoto S, Bitcoin: a peer-to-peer electronic cash system. <https://bitcoin.org/bitcoin.pdf>
17. Wood G (2014) Ethereum: a secure decentralised generalised transaction ledger. *Ethereum Project Yellow Pap* 151:1–32
18. Kiayias A, Russell A, David B, Oliynykov R (2017) Ouroboros: a provably secure proof-of-stake blockchain protocol. In: *Annual cryptology conference*, pp 357–388
19. Dziembowski S, Faust S, Kolmogorov V, Pietrzak K (2015) Proofs of space. In: *Annual cryptology conference*, pp 585–605
20. Garay J, Kiayias A, Leonardos N (2015) The bitcoin backbone protocol: analysis and applications. In: *International conference on the theory and applications of cryptographic techniques*, pp 281–310
21. Kiayias A, Panagiotakos G (2015) Speed-security tradeoffs in blockchain protocols. *IACR Cryptol ePrint Archive* 2015:1–19
22. Badertscher C, Maurer U, Tschudi D, Zikas V (2017) Bitcoin as a transaction ledger: a composable treatment. In: *Annual cryptology conference*, pp 324–356
23. Zhang Y, Xu C, Lin X, Shen X (2019) Blockchain-based public integrity verification for cloud storage against procrastinating auditors. *IEEE Trans Cloud Comput*. <https://doi.org/10.1109/TCC.2019.2908400>
24. Zhang Y, Xu C, Ni J, Li H, Shen X (2019) Blockchain-assisted public-key encryption with keyword search against keyword guessing attacks for cloud storage. *IEEE Trans Cloud Comput*. <https://doi.org/10.1109/TCC.2019.2923222>
25. Zhang Y, Xu C, Cheng N, Li H, Yang H, Shen X (2019) Chronos+: an accurate blockchain-based time-stamping scheme for cloud storage. *IEEE Trans Serv Comput*. <https://doi.org/10.1109/TSC.2019.2947476>
26. Zhang Y, Xu C, Li H, Yang H, She X (2019) Chronos: secure and accurate time-stamping scheme for digital files via blockchain. In: *IEEE international conference on communications*, pp 1–6
27. Pierrot C, Wesolowski B (2018) Malleability of the blockchain’s entropy. *Cryptography Commun* 10(1):211–233

28. Armknecht F, Bohli J, Karame GO, Liu Z, Reuter CA (2014) Outsourced proofs of retrievability. In: ACM conference on computer and communications security, pp 831–843
29. Zhang Y, Xu C, Yu S, Li H, Zhang X (2015) SCLPV: secure certificateless public verification for cloud-based cyber-physical-social systems against malicious auditors. *IEEE Trans Comput Soc Syst* 2(4):159–170
30. Kostiainen K, E R, Ekberg J, Asokan N (2011) Old, new, borrowed, blue—a perspective on the evolution of mobile platform security architectures. In: ACM conference on data and application security and privacy, pp 13–24
31. Zhang Y, Xu C, Li H, Yang K, Zhou J, Lin X (2018) HealthDep: an efficient and secure deduplication scheme for cloud-assisted eHealth systems. *IEEE Trans Ind Inf* 14(9):4101–4112
32. Menezes A (2009) An introduction to pairing-based cryptography. *Recent Trends Cryptography* 477:47–65
33. Goldwasser S, Micali S, Rivest RL (1988) A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J Comput* 17(2):281–308
34. Boneh D, Lynn B, Shacham H (2001) Short signatures from the weil pairing. In: International conference on the theory and application of cryptology and information security, pp 514–532
35. Boneh D, Lynn B, Shacham H (2004) Short signatures from the weil pairing. *J Cryptol* 17(4):297–319
36. Gutmann P (2002) PKI: it’s not dead, just resting. *Computer* 35(8):41–49
37. Adams C, Lloyd S (2003) Understanding PKI: concepts, standards, and deployment considerations. Addison-Wesley Professional, New York
38. Tschorsch F, Scheuermann B (2016) Bitcoin and beyond: a technical survey on decentralized digital currencies. *IEEE Commun Surv Tutor* 18(3):2084–2123
39. Conti M, Kumar S, Lal C, Ruj S (2018) A survey on security and privacy issues of bitcoin. *IEEE Commun Surv Tutor* 20(4):3416–3452
40. Sabt M, Achemlal M, Bouabdallah A (2015) Trusted execution environment: what it is, and what it is not. In: IEEE international conference on trust, security and privacy in computing and communications, vol 1, pp 57–64

Chapter 3

Cloud Storage Reliability



This chapter introduces the data integrity verification technique for cloud storage systems. First, the basic paradigms and principles of the data integrity verification technique are presented. Then, a comprehensive survey on the data integrity verification technique is provided, where existing data integrity verification schemes are introduced, and their pros and cons are analyzed. Finally, the latest advances in the data integrity verification technique are studied.

3.1 Data Integrity

Cloud storage services provide users an efficient and flexible way to manage their data such that users are free from heavy local storage costs. In a cloud storage system, the storage service is provided by a cloud server which is subject to a cloud service provider, where users outsource their data to cloud servers and access their data remotely.

Although users enjoy great benefits from these services, data outsourcing has also incurred critical threats towards data integrity. Unlike traditional data management paradigm, where users store their data locally, users would not physically own their data once having outsourced the data to the cloud server. As such, the integrity of outsourced data is being put at risk in reality. Specifically, although cloud service providers put their efforts into enhancing the reliability of their storage infrastructures in terms of hardware, software, and devices, they still suffer from the broad range of both internal and external threats towards the integrity of outsourced data. On the one hand, the cloud service provider may always conceal incidents of data corruption for a good reputation. A typical incident¹ appeared recently is that QCloud completely lost a part of the outsourced data from a business user due

¹http://www.xinhuanet.com/finance/2018-08/13/c_129931885.htm.

to a series of maloperations made by employees working at the service provider, which causes more than 10 million dollars loss (evaluated by the business user). In addition, a cloud service provider who is greedy for profit may delete a part of outsourced data that is never accessed by users to reduce the storage costs. On the other hand, an external adversary motivated by financial or political reward may be interested in distorting the outsourced data but attempt to convince the users that their data are still maintained intact.

As a consequence, data integrity becomes the fundamental factor that affects the reliability of cloud storage systems. If the integrity of outsourced data cannot be guaranteed, the cloud service provider fails to ensure the normal archiving service, let alone provides other services (such as data deduplication, retrieval, and investigations) based on the outsourced data.

The most effective way to ensure the integrity of outsourced data is the data integrity verification technique, where the integrity of outsourced data is verified periodically. Once the data corruption occurs, users would be informed as soon as possible, stop using the outsourced data at once, and recover the corrupted data with the aid of the cloud server.

In this chapter, we introduce the data integrity verification technique for cloud storage systems.

3.2 Proofs of Storage: Definition and Criteria

The concept of proofs of storage is to construct a storage auditing mechanism to allow data owners to verify that their data remains intact on an untrusted server, which reassures them that their data are correct and indeed available. To construct secure and practical proof-of-storage systems, two cryptographic primitives were proposed at the same time. The first one is proofs of retrievability (PoR) [1] and the second one is provable data possession (PDP) [2]. Both PoR and PDP are built on a challenge-response model: the verifier challenges the storage server to the data integrity, and the storage server responds with a proof. Once the proof is verified, the verifier considers that the data are well maintained. Juels et al. [1] first define the security model of proofs of storage: in a secure proof-of-storage system, if a storage server can pass the verification, a special extractor algorithm is able to extract the outsourced data with interacting with the server. The difference between PoR and PDP would be discussed later.

Existing PoR/PDP-based proof-of-storage schemes can be mainly classified into two categories: private verification and public verification. The main difference between these two categories of verification schemes is that who verifies the data integrity. In a private verification scheme, the verification is performed by the data owner herself/himself, which can be constructed on symmetric-key cryptosystems. In a public verification scheme, the verification is performed by an independent entity (who is called third-party auditor, short for TPA), and the scheme is constructed on public-key cryptosystems.

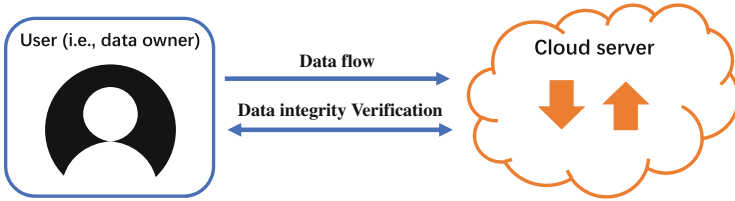


Fig. 3.1 System model of private verification

3.2.1 Threat Models

Generally, as shown in Fig. 3.1, there are two entities in a private verification scheme: users and cloud server.

Users are data owners, they outsource their data to the cloud server, access the outsourced data as needed, and periodically verify the data integrity. After data outsourcing, the user will periodically interact with the cloud server to verify data integrity.

The cloud server is subject to the cloud service provider, and provides cloud storage services. It has not only significant storage space, but also a massive amount of computing power.

The procedure of private verification is straightforward: a user generates his/her data, processes the data using some algorithms to support the data integrity verification, and outsources processed data to the cloud server. The cloud server well maintains the outsourced data and periodically proves the data integrity to the user.

Integrating private verification into cloud storage systems surely guarantees the integrity of outsourced data, but it also suffers from the following problems.

First, the private verification paradigm relies on users themselves to periodically execute the integrity verification, which requires users to keep online to interact with the cloud server for the verification. As a consequence, users have to bear heavy communication burden to handle verification tasks.

Second, the usage of cloud storage services for users is further complicated and the overhead of using cloud storage services is increased. From the users' perspective, it is best to leverage cloud storage services in a black-box way with minimal overhead and simplest operations, such that the users can retrieve the outsourced data without performing too many additional operations.

Third, a cloud server needs to serve multiple users in reality, and would be confronted with heavy costs if it periodically proves the integrity of different parts of data to corresponding users. Actually, it is more advantageous for the cloud server to only entertain verification requests from a single designated entity.

Last but not least, data verification results not only reflect the corresponding states of integrity for the outsourced data, but also serve as the most important criterion that is used to evaluate the quality of the cloud storage service. In private

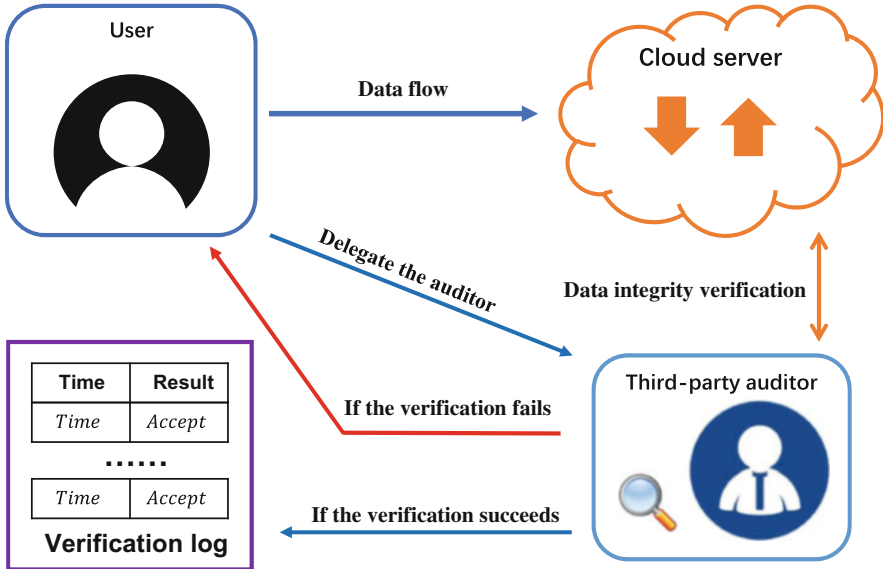


Fig. 3.2 System model of public verification

verification schemes, collecting the verification results from different users also incurs additional and heavy costs.

To address the above problems, a public verification paradigm has been proposed [2]. The idea is to employ an external and independent auditor, who has expertise and capabilities that users do not, to verify the data integrity on behalf of users.

Generally, there are three entities in a public verification scheme, as shown in Fig. 3.2. Compared with its private counterpart, a third-party auditor is employed to periodically interact with the cloud server for the data integrity verification.

The procedure of public verification is more complex than that of private verification: the user generates his/her data, processes the data using some algorithms to support the data integrity verification, and outsources processed data to the cloud server. After data outsourcing, the user sets a verification period (i.e., the frequency at which the auditor performs the verification). Then the auditor interacts with the cloud server to verify the outsourced data integrity at the corresponding time. In practice, the auditor generates a verification report containing all verification results. If in any period the verification result is “Reject,” it means that the data may be corrupted, and the auditor needs to inform the user at once. Otherwise, the auditor generates a verification log and stores the log into the report. The user is able to access this verification report as needed.

According to the procedure of public verification, we can observe that in public verification schemes, from the user’s perspective, if the outsourced data is corrupted, the longest delay within which she/he needs to find the data corruption should be the verification period.

3.2.2 Security Criteria

Both private and public verification schemes should be evaluated by criteria in terms of practicality and security. The criteria can be divided into two parts: system criteria and crypto criteria.

System criteria include:

- *Efficiency.* A proof-of-storage scheme should be as efficient as possible in terms of communication and computation overhead. An auditor (in private proof-of-storage schemes, the data owner serves as the auditor) can verify the data integrity without downloading the entire data set.
- *Boundless verification.* The auditor is able to verify the data integrity without prior bounds on the number of verification interactions.
- *Stateless auditor.* The auditor should be stateless and should not need to maintain and update state during verification.

Crypto criteria include:

- *Soundness.* Any time a storage server passes the auditor's verification, it must possess the specified data intact.
- *Resistance against external adversaries.* A proof-of-storage scheme should resist a common type of attack, where an external, active, and online adversary modifies the outsourced data and tampers with the interaction messages between the cloud server and auditor to pass the verification.

In regards to public verification schemes, there are some additional requirements in terms of efficiency and security.

- The auditor, who has the system (public) parameter, is able to verify the data integrity without the data owner's participation. This requirement is a basis of public verification schemes, since if each verification requires the data owner's participation, the data owner can leverage private verification schemes to verify the data integrity.
- The auditor cannot impersonate the user to generate and outsource valid data, even if it colludes with the cloud server. This requirement is very important in reality. If the auditor can impersonate the user, i.e., the auditor has the user's secret key to process the data, the user cannot further control the auditor. As a consequence, the auditor can delegate the verification to others without the user's permission. Furthermore, if the user attempts to employ another auditor, she/he needs to download the entire data set, re-process the data using a new secret key, and outsource the processed data to the cloud server. This introduces heavy costs in terms of communication and computation on both cloud server and user sides.

3.3 Proofs of Storage for Cloud Storage Systems

Early systems that implement proofs of storage were proposed in literature [3–5], where these works consider the scenario that data are stored on an untrusted server and the integrity of data should be checked. We stress that these prior works focus on the system that users' data are stored on a remote storage server which can be considered as a nascent stage of cloud storage systems.

We briefly review these prior works and give an analysis in terms of security and efficiency.

Let M be the data stored on a server \mathcal{S} and \mathcal{V} be the verifier to check the data integrity. Since \mathcal{S} are vulnerable to attacks, the trustworthiness of integrity verification results on the data reported by \mathcal{S} is arguable; downloading the entire data set from \mathcal{S} would cause prohibitive communication costs.

In [3], the authors propose a remote integrity checking scheme enabling a verifier to check the data integrity in a challenge-response way: \mathcal{V} periodically generates a random element R as a challenge, and sends R to \mathcal{S} ; \mathcal{S} generates the hash value $prf = h(R||M)$ as the corresponding proof and sends prf to \mathcal{V} ; \mathcal{V} verifies the data integrity by checking the validity of prf . To verify the validity of prf , \mathcal{V} has to pre-compute all possible prf and maintain them locally. To free \mathcal{V} from maintaining large number of pre-computed proofs, the challenges are generated using the hashchain, as shown in Fig. 3.3: the last challenge $R_{|\ell|}$ is randomly chosen from Z_p and kept securely; For $i = 1, 2, \dots, |\ell| - 1$, $R_i = h(R_{i+1})$. With $R_{|\ell|}$, \mathcal{V} can compute all other challenges for verification, but given R_i , it is computationally infeasible to compute R_{i+1} , due to the preimage resistance of hash function h .

Y. Deswarte et al. [3] also propose an alternative scheme based on Diffie–Hellman key exchange protocol [6]. Specifically, the data M is represented by an integer, N is the RSA modulus which is the product of two distinct odd primes p and q , $\phi(N) = (p - 1)(q - 1)$ is the Euler function. \mathcal{V} randomly chooses $a \in [2, N - 2]$ and pre-computes $Pre = a^M \pmod{N}$. a is published. When \mathcal{V} verifies the data integrity, it randomly chooses $r \in [2, N - 2]$, computes $A = a^r \pmod{N}$ as a challenge, and sends A to \mathcal{S} . \mathcal{S} computes $B = A^M \pmod{N}$ as the proof and sends B to \mathcal{V} . \mathcal{V} verifies the integrity of M by verifying whether $Pre^r \pmod{N} = B$ holds, if it holds, the data is well maintained.

Although the schemes proposed by Y. Deswarte et al. [3] provide the data integrity assurance, they are also confronted with some efficiency issues. In particu-

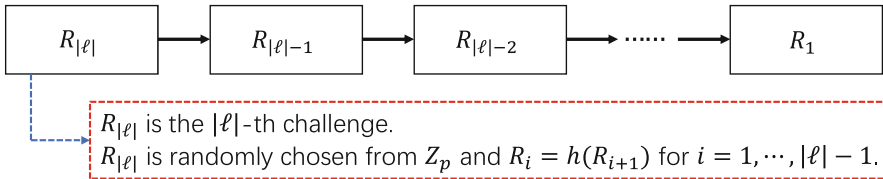


Fig. 3.3 The generation challenge in [3]

lar, the basic scheme constructed on the hash function cannot achieve the boundless verification described in Sect. 3.2.2. It means that the data owner needs to download the entire data set and re-generate the hash values for data integrity verification, which introduce heavy costs in terms of communication and computation. The improved scheme constructed on Diffie–Hellman key exchange protocol achieves the boundless verification. However, in this scheme, the data is represented by an integer that is less than the RSA modulus N . If $|N| = 1024$ bits, the data size $|M| < 1024$ bits. If the size of M is large, the data owner needs to split M into multiple blocks, e.g., $M = \{M_1, M_2, \dots, M_n\}$, such that the size of each block is less than 1024 bits. The data owner then pre-computes a verification value Pre_i for each block $M_i, i = 1, 2, \dots, n$. For a verification task, \mathcal{S} has to compute n proofs and send them to \mathcal{V} to prove the data integrity, this introduces prohibitive costs in terms of storage, communication, and computation. Subsequent schemes [4, 5] are constructed on signature schemes to free from pre-computing values used for verification, which improves efficiency. However, in these schemes, the proof generated by \mathcal{S} includes one signature per each block. This makes the size of the proof linearly increases with the number of blocks. In addition, in the above schemes [3–5], to prove the data integrity, \mathcal{S} needs to access all the data blocks (we assume the data consists of n blocks as described above) to compute the proof, this also causes the expensiveness in input/output (I/O). Furthermore, all these schemes are in the absence of a theoretical treatment and there is no precise indication of what proof-of-storage systems do or do not accomplish.

In CCS’07, two systematic works on proof-of-storage were proposed by Ateniese et al. [2] and Juels et al. [1], separately. We next briefly introduce them one by one and analyze their cons and pros.

In [1], Juels et al. first define a cryptographic primitive called proofs of retrievability (PoRs). A PoR scheme enables \mathcal{S} to produce a concise proof that \mathcal{V} can retrieve the object data M . The authors define the formal security model of PoRs. Intuitively, the security of PoRs ensures that \mathcal{S} cannot “cheat” \mathcal{V} , where “cheat” means that \mathcal{S} does not store the target data M but convinces \mathcal{V} that M can be retrieved. Juels et al. also propose a sentinel-based PoR scheme. In the scheme M is processed as follows. First, the data owner (i.e., \mathcal{V}) processes M using an error correcting code algorithm such that $M = \{M_i\}(i = 1, \dots, n)$ and \mathcal{V} can retrieve M if she/he has any t blocks of $\{M_1, \dots, M_n\}$. Then, \mathcal{V} encrypts M using a symmetric-key encryption with a random key k : $C = E(k, M) = \{E(k, M_1), E(k, M_2), \dots, E(k, M_n)\} = \{C_1, C_2, \dots, C_n\}$. \mathcal{V} computes a set of s sentinels $\{a_1, a_2, \dots, a_s\}$, where for $j = 1, 2, \dots, s$, a_j is a random string with the same size of each sentinel. \mathcal{V} randomly inserts these sentinels into $\{C_1, \dots, C_n\}$ to yield \hat{C} , records the positions of all sentinels, and outsources \hat{C} to \mathcal{S} . When \mathcal{V} verifies the integrity of \hat{C} , she/he randomly selects a subset of a_1, a_2, \dots, a_s , sets the indexes (i.e., positions) of these sentinels in \hat{C} as the challenge, and sends the challenge to \mathcal{S} . If \mathcal{S} sends the corresponding sentinels to \mathcal{V} . If and only if all the sentinels received from \mathcal{S} are valid, the retrievability of \hat{C} is ensured.

The sentinel-based PoR scheme [1] also suffers from some issues. First, it cannot achieve the boundless verification, which causes heavy costs in terms of

communication and computation as discussed before. Second, it cannot support public verification, i.e., \mathcal{V} is the data owner and cannot be delegated by other data owners. Actually, if a data owner delegates the verification to an independent auditor, the auditor is able to impersonate the user to generate the valid data \hat{C} .

In [2], Ateniese et al. propose a cryptographic primitive called provable data possession (PDP). A PDP scheme allows a user that has outsourced data on an untrusted server to verify that the server possesses the original data without retrieving it. Ateniese et al. construct a PDP scheme on homomorphic verifiable tags (HVTs) that are derived from RSA-based homomorphic signatures [7]. In this work, the concept of sampling verification is proposed to improve the efficiency, which also serves as a general principle for subsequent verification schemes. Sampling verification refers to a strategy that \mathcal{V} only chooses a random subset of all data blocks (e.g., sample 300 blocks from 10,000 ones) and verifies the integrity of the sampled blocks. If the verification passes, the integrity of the entire data set is ensured. Due to the homomorphism of HVTs, \mathcal{V} can check multiple data blocks simultaneously without retrieving them. Specifically, M is first split into n blocks $M = \{M_i\}(i = 1, \dots, n)$. For each block M_i , \mathcal{V} computes a HVT as $\sigma_i = (h(i||name) \cdot g^{M_i})^d \pmod{N}$, where $name \in Z_N$ is the name of M , g is a generator of QR_N (which is the set of quadratic residues modulo N), $sk = (e, d, v)$ is secret key, and $pk = (N, g)$ is the corresponding public key. \mathcal{V} outsources $\hat{M} = \{M_1, M_2, \dots, M_n, \sigma_1, \sigma_2, \dots, \sigma_n\}$ to \mathcal{S} . To prove the integrity of \hat{M} , \mathcal{S} only needs to send a proof $\sigma = \prod_{i \in I} \sigma_i^{v_i}$ (and auxiliary information) to \mathcal{V} , where I is the set of challenged blocks' indexes and v_i is random value used to ensure the freshness of the proof. \mathcal{V} can verify the integrity of \hat{M} by verifying σ without downloading the corresponding blocks from \mathcal{S} .

In [2], the proposed PDP scheme is the first one that supports public verification, where an auditor who has the public parameters $(name, pk, n)$ can verify the integrity of \hat{M} on behalf of \mathcal{V} . On the other hand, the proposed PDP scheme is constructed on the RSA cryptosystem. To achieve 80-bit security, the RSA modulus N should be at least 2048 bits, which causes substantial storage costs to store HVTs. Here, we stress that HVTs is essentially a signature on a data block, but generating HVTs cannot leverage the hash-and-sign paradigm, because if HVT is computed from the hash value of the block, \mathcal{S} , who only stores the hash value of the block rather the block itself, can pass the \mathcal{V} 's verification. As a consequence, the size of each block relies on the security parameter and would be restricted to a small size, and for the data M , if the size of each block is reduced, the number of data blocks increases, which increases the number of HVTs and thereby increases storage costs.

The main difference between PoR and PDP is the notion of security that they achieve. Specifically, the security of PoR ensures two fundamental properties in a proof-of-storage system [8].

- **Authenticated storage.** \mathcal{V} can verify that data fetched from \mathcal{S} are correct, where correctness is equivalent to authenticity and freshness.
- **Retrievability.** \mathcal{V} can ensure that the specified data can be fully retrieved.

PDP only focuses on verifying the integrity of the data and does not focus on data retrievability. In the PDP scheme, if \mathcal{S} has lost a small number of data blocks, it can pass the \mathcal{V} 's verification with significant probability. Particularly, in the sampling verification mechanism proposed by Ateniese et al. [2], if ρ fraction of data is corrupted, then randomly (uniformly) sampling c blocks would reach the detection probability $P_{detec} = 1 - (1 - \rho)^c$. If $\rho = 0.1\%$, $P_{detec} = 0.2593$, which means that if the total number of data blocks is 10,000 and 10 blocks are corrupted, \mathcal{S} can pass \mathcal{V} 's verification with probability more than 75%.

From the perspective of technique, the only difference between PoR and PDP is the use of erasure coding. In PoR, the data owner first uses an erasure coding algorithm (e.g., Reed-Solomon codes [9]) to process the data M and gets $M = \{M_1, M_2, \dots, M_n\}$, such that any s ($s < n$) blocks of $\{M_1, M_2, \dots, M_n\}$ can retrieve M . Recall the above example, if the total number of data blocks is 10,000 and 10 blocks are corrupted, although the detection probability in a PoR scheme is still less than 26%, the original data M can be retrieved under a reasonable choice of s (e.g., $s/n = 90\%$). By comparison, in this case, M protected by the PDP scheme cannot be retrieved.

In ASIACRYPT'08, Shacham et al. [10, 11] propose two compact PoR schemes with full proofs of security against adversaries in the security model proposed by Juels et al. [1], where the first PoR scheme only supports private verification, and the second one supports public verification. Both the schemes achieve the system criteria proposed in Sect. 3.2.2 and are provably secure. The private verification is based on HVTs derived from pseudorandom function $F(\cdot)$. Specifically, the data M is split into n blocks using the erasure code as $M = \{M_1, M_2, \dots, M_n\}$. The HVT for data block M_i is $\sigma_i = F(k, i) + \alpha M_i$, where α and k are secret keys. We assume that Q is the set $\{(i, v_i)\}$ of challenge index-coefficient pairs, the corresponding proof is $\{\sigma, \mu\}$, in which $\sigma = \sum_{(i, v_i) \in Q} v_i \sigma_i$ and $\mu = \sum_{(i, v_i) \in Q} v_i M_i$. The data owner checks the following equation to verify the data integrity:

$$\sigma \stackrel{?}{=} \alpha \cdot \mu + \sum_{(i, v_i) \in Q} v_i \cdot F(k, i).$$

For the public verification scheme in [10, 11] (hereinafter, this scheme is called SWP), the HVTs are built on the BLS signature [12]. Compared with its RSA counterpart, the BLS-based HVT is considerably shorter. We briefly introduce SWP as follows. Let $e : G \times G \rightarrow G_T$ be a computable bilinear map. The data owner's private key is $\alpha \in Z_p$, the corresponding public key is $v = \alpha P \in G$, where P is the generator of G . The data owner applies the erasure code on M to split it to n blocks, and further split each block into s sectors, i.e., $M = \{M_{ij}\}_{1 \leq i \leq n, 1 \leq j \leq s}$. The data owner randomly chooses s random elements $u_1, \dots, u_s \in G$, the HVT for the i -th block is

$$\sigma_i = \alpha \cdot \left(H(i || name) + \sum_{j=1}^s M_{ij} u_j \right),$$

where $name \in Z_p$ is the name of M , $H : \{0, 1\}^* \rightarrow G$ is a secure hash function. Let $Q = \{(i, v_i)\}$ be the challenging information as described before, the corresponding proof generated by \mathcal{S} is $\{\sigma, \mu_j\}_{j=1,2,\dots,s}$, where $\sigma = \sum_{(i,v_i) \in Q} v_i \sigma_i$ and $\mu_j = \sum_{(i,v_i) \in Q} v_i M_{ij}$. The verification equation becomes:

$$e(\sigma, P) \stackrel{?}{=} e \left(\sum_{(i,v_i) \in Q} v_i H(i || name) + \sum_{j=1}^s \mu_j u_j, v \right).$$

A third-party auditor, who holds the public key and the data name, can verify the data integrity without the data owner's participation. In SWP, each block is split into multiple sectors to balance the tradeoff between storage and communication: each sector is an element of Z_p and there is one HVT per block, which reduces the storage costs to $(1 + 1/s) \times$; while the proof is one aggregated block and HVT, and is $(1 + s) \times$ as long as an authenticator. Therefore, the larger s , the less storage costs, and the higher communication costs.

Since the public verification paradigm is more expressive than its private counterpart, subsequent works mainly focus on constructing public verification of data integrity schemes for cloud storage systems with different characteristics. On a very high level, most of the public verification schemes share essentially the same common structure as SWP proposed by Shacham et al. [10, 11]. These schemes can be mainly classified into three types.

3.3.1 Proofs of Storage for Dynamic Data

In cloud storage systems, outsourced data might be updated by the data owner in reality, this requires proof-of-storage schemes supporting dynamic data operations (e.g., addition, deletion, and updation) [13]. Generally, there are two types of data integrity verification schemes. The first one is based on PDP, and the second one is based on PoR. In the following, we will introduce these two types of schemes one by one and discuss their differences in detail.

3.3.1.1 PDP-Based Integrity Verification of Dynamic Data

Most PDP-based proof-of-storage schemes [14–17] that support data dynamic operations share essentially the same common structure: initially, the data owner generates an HVT for each data block and records the abstract information of each block as well as its HVT using an authenticated data structure, e.g., hash table, hash chain, hash tree, etc. After outsourcing the data to the cloud server, the data owner also sends the recorded abstract information to the auditor. At any point in time, the data owner performs an operation (e.g., addition, deletion, and updation) on the

outsourced data, she/he sends the details of the operation to the auditor, and the auditor updates the recorded abstract information for subsequent verifications.

3.3.1.2 PoR-Based Integrity Verification of Dynamic Data

Constructing a proof-of-storage scheme on PoR to support dynamic operations on outsourced data is more challenging than that on PDP. We give an example to illustrate the challenge of performing the dynamic operations in a PoR-based proof-of-storage scheme. Let M' be the data that the data owner wants to outsource. Originally, $M' = \{M'_1, M'_2, \dots, M'_n\}$. Let Σ denote an erasure coding algorithm such that $M = \Sigma(M') = \{M_1, M_2, \dots, M_n\}$ and any $n/2$ blocks from $\{M_1, M_2, \dots, M_n\}$ can be used to recover M . The data owner outsources M and the corresponding HVTs to the cloud server. If the data owner wants to modify even a block of M , e.g., M_i for $i \in [1, n]$, she/he needs to change at least half of the blocks in M outsourced on the cloud server, which requires heavy communication and computation costs.

Intuitively, addressing the above problem is very straightforward: the data owner can split the original data M' into multiple blocks as $M' = \{M'_1, M'_2, \dots, M'_n\}$ and further split each data block, i.e., M'_i for $i = 1, 2, \dots, n$, using erasure coding algorithm as $M_i = \Sigma(M'_i) = \{M_{i1}, M_{i2}, \dots, M_{is}\}$. Finally, the data owner concatenates these blocks to form $M = \{M_1, M_2, \dots, M_n\}$, computes an HVT for each block, and outsources M as well as the corresponding HVTs to the cloud server. The data integrity verification is the same as before: the auditor (i.e., verifier) randomly chooses c blocks from all $n \cdot s$ blocks of M and verifies the integrity of these c blocks. Now, if the data owner wants to read/write to any data block of M , he only needs to read/write to the s relevant coded blocks on the cloud server, which reduces the communication and computation costs significantly.

We assume that λ denotes the size of M' , we introduce a parameter k called block size, which is computed by $k = \lambda/n$ and determines the complexity of reads/updates. c determines the complexity of verification. The above scheme can be secure under the security definition of PoR (i.e., achieving authenticated storage and retrievability), only if k and c are both set to $\Omega(\sqrt{\lambda})$. This still incurs high communication and computation costs. If both k and c are set to be small (e.g., $k, s = o(\sqrt{\lambda})$), the scheme cannot achieve retrievability. In particular, when $k, s = o(\sqrt{\lambda})$, the server can delete a single $M_i = \{M_{i1}, M_{i2}, \dots, M_{is}\}$ from M entirely, but can pass the auditor's verification with a high probability. However, the data block M_i is totally lost and cannot be retrieved. As a consequence, the scheme cannot guarantee retrievability.

To achieve provable security under the PoR model and high efficiency simultaneously, the data owner in the above scheme needs to make the data “unintelligible” to the cloud server, such that the cloud server cannot identify the locations within M that correspond to a single data block. Note that pseudorandomly permute the locations of each data blocks cannot achieve the above requirement, since after the

data owner performs an operation on a block of M that has been pseudorandomly permuted, the cloud server can exactly identify the corresponding block from M . To address this problem, the data owner can utilize an oblivious way to outsource and access data, such that she/he can outsource the data to the cloud server and access the outsourced data without leaking access patterns to the server. Here, access patterns include the identity, index, and other information of accessed data block, the frequency of the data owner accessing the same data block, and the sequence of accessing multiple data blocks. Existing schemes [8, 18, 19] utilize oblivious RAM (short for ORAM) [20] and its variants to implement the oblivious storage.

3.3.2 Enhancement of Security

This type of works can be further classified into two parts.

3.3.2.1 Privacy-Preserving Public Verification

In a public verification of data integrity scheme, a delegated auditor is employed to verify the integrity of outsourced data on behalf of the data owner. During the verification, the auditor is able to extract the data content from the proofs generated by the storage server. For instance, in SWP, the proof is $\{\sigma, \mu\}$ (hereinafter, for the sake of brevity, the number of sectors $s = 1$ by default, i.e., $M = \{M_i\}_{1 \leq i \leq n}$, $\sigma_i = \alpha \cdot (H(i || name) + M_i u)$, and $u \in G$ is a public parameter), where $\sigma = \sum_{(i, v_i) \in Q} v_i \sigma_i$ and $\mu = \sum_{(i, v_i) \in Q} v_i M_i$. Since i and v_i are generated by the auditor, if the auditor has lots of μ under different $Q = \{(i, v_i)\}$, it can compute M_i for some $i \in Q$.

However, from the perspective of protecting users' privacy, the users, who rely on the auditor just for ensuring the integrity of outsourced data, do not want the employed security mechanism that introduces new threats towards the outsourced data. Therefore, protecting data contents against the auditor and other external adversaries (who may eavesdrop on the interactions between the storage server and the auditor) should be considered in constructing proof-of-storage systems.

To address the above problem, the notion of privacy-preserving public verification is proposed [21]. A privacy-preserving public verification scheme enables a third-party auditor to verify the integrity of data outsourced to a remote server (i.e., cloud server) without extracting the data contents.

Obliviously, privacy-preserving public verification schemes can be directly derived from existing PoR/PDP-based scheme by adopting symmetric encryption algorithms (e.g., AES): the user encrypts her/his data before outsourcing. Since PoR and PDP themselves do not require data confidentiality, and the symmetric encryption can serve as an independent component to be integrated into existing PoR and PDP schemes to protect data contents against both the cloud server and the auditor. However, such the encryption-based privacy-preserving scheme could

be an overkill for unencrypted/public data (e.g., outsourced libraries and scientific data sets) [2, 22].

Another way to construct privacy-preserving public verification schemes is to let the cloud server blind the proof using a random mask, which guarantees that the auditor cannot extract any knowledge about the data content from the proof generated by the cloud server. The first privacy-preserving public verification scheme using the random masking technique is proposed by Wang et al. [21, 22].

After the Wang et al.'s scheme, protection data content against auditors has been considered as a fundamental property for public verification schemes, and most privacy-preserving public verification schemes are based on the random masking technique [23, 24]. It is worth discussing the relationship between the random masking technique and symmetric encryption in constructing privacy-preserving public verification schemes. As discussed in [21, 22], public verification based on the random masking technique to protect data content against auditors is more efficient than that based on symmetric encryption. In the nascent stage of cloud storage services, only a few interfaces that support simple operations, such as data outsourcing and access, are provided to cloud users. At that time, data are always outsourced in the plaintext form, and the public verification technique can be employed in a stand-alone fashion to ensure data integrity. Nonetheless, in today's cloud storage services, protection data content against anyone who does not own the data is an inherent requirement for users, which also serves as one of the most important selling points in reality. This is further intensified by the incidents of outsourced data leakage occurred in recent years.² Consequently, in current cloud storage systems, users' data are always encrypted before outsourcing and are stored in the ciphertext form on the cloud server, and the public verification, encryption as well as other security mechanisms are often integrated to protect the overall security of a cloud storage system (how to integrate multiple security mechanisms into one system to achieves multiple security goals is one of our objectives in this monograph).

3.3.2.2 Resistance Against External Adversaries

The threat model in [2, 10, 11] does not consider external adversaries. We first show how an external adversary \mathcal{A} invalidates SWP [10, 11] as follows.

We assume the data $M = \{M_i\}_{i \in [1, n]}$ is stored on the cloud server \mathcal{CS} . \mathcal{A} first intrudes into \mathcal{CS} , modifies each data block M_i to $\hat{M}_i = M_i + L_i$ for $i \in [1, n]$, and records $L_i, i \in [1, n]$. In the data verification phase, \mathcal{A} eavesdrops on the challenging message $Q = \{(i, v_i)\}$. In the proving phase, \mathcal{CS} computes the proof information Prf as: $\hat{\mu} = \sum_{(i, v_i) \in Q} v_i \hat{M}_i, \sigma = \sum_{(i, v_i) \in Q} v_i \sigma_i, Prf = \{\hat{\mu}, \sigma\}$. Then

²https://en.wikipedia.org/wiki/List_of_data_breaches.

\mathcal{S} sends \hat{Prf} to the auditor. \mathcal{A} intercepts \hat{Prf} and computes³: $\gamma = \sum_{(i,v_i) \in Q} v_i L_i$, $\mu = \hat{\mu} - \gamma$, $Prf = \{\mu, \sigma\}$. Finally, \mathcal{A} sends Prf to the auditor. In this way, \mathcal{A} tampers with the outsourced data and deceives the auditor successfully.

Such an external adversary can invalidate lots of public verification schemes [25, 26]. A straightforward way to thwart the adversary is to establish a secure channel between the cloud server and the auditor such that the external can neither obtain the challenge Q nor tamper with the proof information. However, establishing a secure channel between the cloud server and the auditor is a costly operation.

To address the problem, Xu et al. [27] propose a public verification scheme based on the random masking technique. Specifically, the HVT of M_i is the same as SWP, i.e., $\sigma_i = \alpha(H(i||name) + M_i u)$. The proof information is $prf = \{\sigma, \mu, R\}$, where μ is blinded by a random $r \in \mathbb{Z}_p$ as $\mu = r^{-1}(\sum_{(i,v_i) \in Q} v_i M_i + h(R))$ and $R = r \cdot u \in G$. The verification equation is

$$e(\sigma, P) = e \left(\sum_{(i,v_i) \in Q} v_i H(i||name) + \mu R + (-h(R))u, v \right).$$

Now, if \mathcal{A} launches the above attack to invalidate the scheme, he has to compute $\mu = \hat{\mu} - r^{-1} \sum_{(i,v_i) \in Q} L_i v_i$. Since r is a random element and is unknown to \mathcal{A} , it is computationally infeasible to compute μ . Therefore, the proposed scheme is secure against external adversaries. Furthermore, with the leverage of the random masking technique, the scheme also protects the data content against the auditor, i.e., it achieves privacy-preserving public verification.

3.3.3 Constructing Public Verification on Different Cryptosystems

As discussed before, PoR/PDP-based public verification schemes are based on homomorphic verifiable tags which are essentially homomorphic signatures (no matter what the hard problem the algebraic structure they are built on). To verify the data integrity of outsourced data, the auditor needs to maintain the users' certificates to choose the correct public keys. As a consequence, the security of the above public verification schemes relies on the security of Public Key Infrastructure (PKI).

In traditional PKI, a user's certificate is issued by a Certificate Authority (CA), which binds the user's public/private key to her/his identity. Despite the wide development of PKI in constructing public-key cryptosystems, it also suffers from some security and efficiency issues. One of the most fundamental issues is the certificate management. Particularly, certificate management, which includes

³In fact, γ can be pre-computed after obtaining Q .

certificate revocation, storage, distribution, and verification, is very cumbersome and even confronted with threats in practice. For example, an issued certificate can be valid if and only if the corresponding root certificate is trustworthy. Whereas, since the root certificate is signed by CA itself, “verifying” the trustworthiness of the root certificate is not easy in practice. In most cases, users only accept the given root certificate without further verification. Furthermore, once an issued certificate needs to be revoked or re-issued (i.e., re-distribution), the revocation information should be broadcast to all users in the system, and the certificate should be invalid from now on. However, this is not an easy task when the system has a large number of users. To make matters worse, in actually public-key cryptosystems, multiple CAs are introduced to the different requirements, which makes the certificate management more complex than the case of single CA.

To address the certificate management problem, some researchers [28, 29] construct HVTs on identity-based cryptosystems [30] to propose identity-based public verification scheme, where a Private Key Generator (PKG) is employed to utilize a system-wide master key and the user’s identity to generate her/his privacy key. Consequently, the auditor, who only holds the system parameters and the user’s identity, is able to verify the integrity of outsourced data on behalf of the user without maintaining the user’s certificate.

Identity-based public verification schemes inevitably inherit security issues from identity-based public-key cryptography (ID-PKC). The most fundamental one is the key escrow problem [31]. In particular, in ID-PKC, a user’s private key is generated by PKG. It means that both PKG and the user have the user’s private key. Thus, the identity-based public verification cannot provide true non-repudiation in the way that the PKI-based one can.

To address the key escrow problem, subsequent public verification schemes [32–34] are based on certificateless cryptography [31]. In a certificateless public verification scheme, the user’s secret key consists of two parts: the first one is a secret key randomly chosen by the user herself/himself, the second one is called partial private key that is generated by a Key Generation Center (KGC) who holds a system-wide master key. As a result, the security of public verification can be ensured even if KGC is compromised.

3.3.4 Other Works

In addition to the above works, some other public verification schemes are proposed to enrich the functionalities of proof-of-storage systems. For instance, some public verification schemes focus on supporting data sharing [35, 36] for cloud storage systems, where the data are owned by a set of users and different users can outsource their data to the cloud server and share the data with other users. In this scenario, new privacy issues are introduced. Since the outsourced data are generated by multiple users, users’ identity privacy could be leaked to the auditor during the verification. The auditor can exactly learn the identity of the generator on each data

block. By utilizing this information, the auditor is able to extract significant (private) information on the users, such as the group membership, the most valuable data block, and the roles of different users in the group. To address this problem, HVTs in the public verification scheme can be constructed by the ring signature [37] and group signature [38], which protect the signers' identities against the auditor.

Another line of work focuses on constructing public verification schemes to be compatible with other cryptographic building blocks to enhance cloud storage services in terms of reliability, functionality, and efficiency. In the following chapters, we will introduce these works in detail.

3.4 Latest Advances in Proofs of Storage

In this section, we introduce the latest advances in public verification schemes.

3.4.1 Proofs of Storage Based on Indistinguishability Obfuscation

Indistinguishability obfuscation ($i\mathcal{O}$) is an important cryptographic primitive to make computer programs “unintelligible” while preserving their functionality. Recent works [39, 40] have shown great potentials for enhancing cloud storage services in terms of security and privacy and for ensuring the data security from $i\mathcal{O}$. With the definition of $i\mathcal{O}$, we can prove that given an obfuscated program, the secrets embedded in it (i.e., the constants of the original program) cannot be extracted by adversaries.

As discussed before, the PDP/PoR-based public verification schemes are based on HVTs, and the auditor can verify the data integrity by checking the HVTs. This introduces heavy computational costs on both the data owner and the auditor. Specifically, the data owner needs to compute an HVT for each data block to enable the auditor to verify the data integrity on behalf of her/him. The computation costs on the data owner side linearly increase with the size of the data to be outsourced. For example, in SWP, if the security level is chosen to be 80 bits, a data M with the size of only 1 MB has more than 52,000 blocks (each block is not further split into sectors). In this case, the computational delay to compute the HVTs is more than 1 min for the data owner who equips a laptop with a macOS system, an Intel Core i7 CPU, and 16 GB DDR3 of RAM. Moreover, the auditor also bears heavy computation costs to verify the data integrity since the data integrity verification essentially requires the auditors to verify multiple signatures. For an auditor equipping the above laptop, if the number of challenged data is set to 300, the verification delay would be 0.38 s. Note that the auditor would serve lots of users

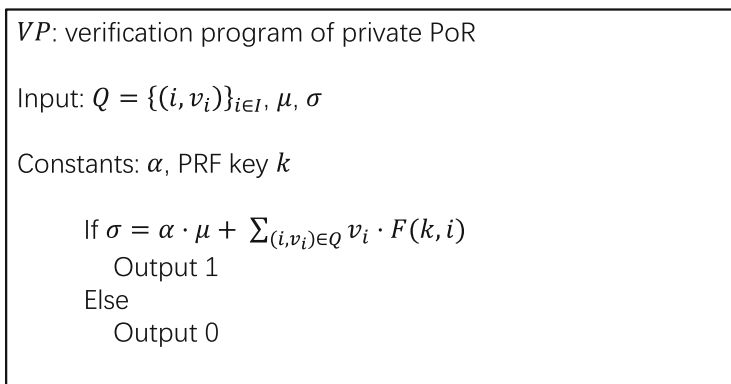


Fig. 3.4 Verification Program of Private PoR

in cloud storage systems, and this delay would be prohibitively long if the number of users is huge, e.g., more than 6 min for 1000 users.

More recently, some researchers [41, 42] investigate how to use $i\mathcal{O}$ to improve the efficiency of public verification schemes.

In [41], the authors utilize $i\mathcal{O}$ to construct a private PoR scheme supporting public verification. Recall that traditional private PoR [10] cannot support public verification since one cannot verify the data integrity without the secret key used to compute HVTs. If the auditor is given this secret key, it can impersonate the user to generate and outsource valid data. However, since private PoR is constructed on symmetric-key cryptosystems, it is highly lightweight on the data owner side. The key technique of [41] is that the data owner generates a verification program VP (as shown in Fig. 3.4) of private PoR scheme, where the secret key used to generate HVTs is embedded into the program as the constant. The data owner obfuscates VP using $i\mathcal{O}$ and sends the obfuscated program to a third-party auditor. Due to the security and functionality of $i\mathcal{O}$, the auditor can verify the data integrity without extracting the secret. To ensure security, the PRF used in Fig. 3.4 is a puncturable PRF [43].

By doing so, the data owner in [41] only needs to perform symmetric-key operations to outsource the data, which reduces the computational costs significantly.

Another line of work focuses on improving the efficiency on the auditor side from $i\mathcal{O}$. Particularly, in [42], the authors propose a public verification scheme called EPVDI for cloud storage services using $i\mathcal{O}$ to reduce the computation overhead on the auditor's side to a MAC tag computation. The key idea behind EPVDI is to delegate the heavy computation operations (originally performed by the auditor) to the cloud server. To this end, the auditor first determines the challenged blocks from the entire data set, and the cloud server verifies the integrity of the challenged blocks. Only if the verification succeeds, the cloud server generates a commitment on the challenged data, and sends the commitment to the auditor. The auditor only needs to verify the validity of the commitment to check data integrity. The commitment is

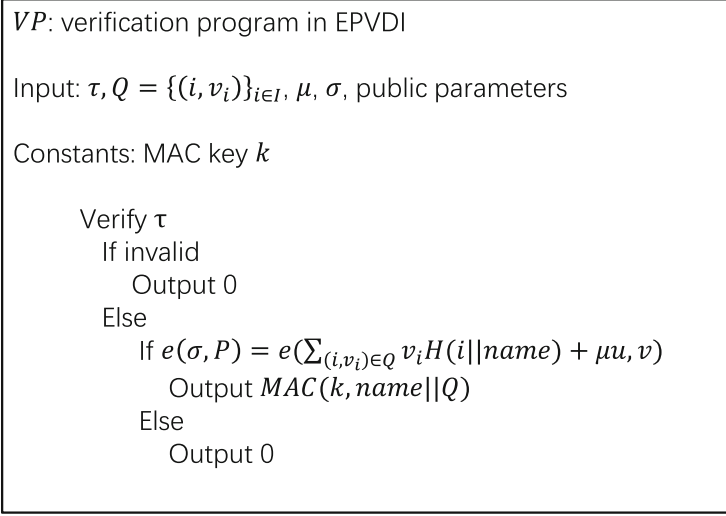


Fig. 3.5 Verification program of EPVDI

constructed on the message authentication code (MAC) technique, such that the auditor can complete the verification efficiently. The main challenge here is to resist the malicious cloud server which generates a fake commitment to deceive the auditor. To protect the scheme against the malicious cloud server, the data integrity verification and commitment generation executed by the cloud server are integrated into a program as a verification program VP , and this program is obfuscated by using $i\mathcal{O}$. The cloud server is only given the obfuscated program such that if and only if it well maintains the data, it can output a valid commitment.

EPVDI is based on SWP. In the setup phase, the data owner generates the verification program VP as shown in Fig. 3.5 (we would not split each block into sectors for the sake of brevity), where τ is a data tag to authenticate the trustworthiness of the public parameter and $name$ is the data identity.

The data owner then obfuscates VP using $i\mathcal{O}$. She/he sends the obfuscated program to the cloud server and securely sends the PRF key k to the auditor. When the auditor verifies the data integrity, it generates the challenging message Q and sends Q to the cloud server. With Q , the cloud server computes the corresponding proof information μ and σ and executes the obfuscated program. If and only if the proof information is valid, it can output a MAC tag. Once the cloud server computes the MAC tag, it sends the tag to the auditor. With the PRF k , the auditor can verify the validity of the tag efficiently, even if it only equips a lightweight device [44].

We have to accept for the time being that existing obfuscation candidates are inefficient to generate, due to their impractical polynomial-time constructions. However, $i\mathcal{O}$ has been envisioned as one of the most important cryptographic primitives. Lots of researchers devote themselves to making $i\mathcal{O}$ practical, and it is plausible that $i\mathcal{O}$ with reasonable performance will be realized in the not too distant

future. Therefore, it is promising to leverage iO to enhance cloud storage services in terms of security, efficiency, and functionality.

3.4.2 Proofs of Storage Based on Blockchain

We first review the public verification scheme and analyze their vulnerability against dishonest auditors, which motivates the researchers to construct new public verification schemes on blockchains.

The key idea of the public verification technique (i.e., PDP/PoR) is that the user (i.e., data owner) splits the data into multiple blocks, computes a signature for each one, and outsources the data blocks as well as corresponding signatures to the cloud server. When the auditor verifies the data integrity, it chooses a random subset of all data blocks (e.g., sample 300 blocks from 10,000 ones) and sends the sampled blocks' indexes (as a challenging message) to the cloud server. The cloud server responds with the corresponding proof, the auditor checks the integrity of challenged blocks by verifying the validity of the proof. If the verification succeeds, the integrity of the entire data set is ensured.

In public verification schemes, after data outsourcing, the user sets a verification period (i.e., the frequency at which the auditor performs the verification). Then the auditor verifies the outsourced data integrity at the corresponding time. In reality, the auditor generates a verification report containing multiple verification results (corresponding to multiple periods, we call these periods an epoch). If, in any period, the verification result is "Reject," it means that the data may be corrupted, and the auditor needs to inform the user at once. Otherwise, the auditor generates a verification log and provides the user with the log at the end of each epoch. Since the auditor is able to verify the data integrity without the user's participation, the user can assign the auditor to perform the verification with any period as needed. In other words, from the user's perspective, if the outsourced data is corrupted, the longest delay within which she/he needs to find the data corruption should be the verification period.

3.4.2.1 On the Vulnerability of Existing Public Verification Schemes Against Dishonest Auditors

All public verification schemes we have introduced so far bear an assumption that the auditor is honest and reliable. This means that the auditor would honestly follow the prescribed schemes, and performs the verification reliably. Note that some works [15, 22, 23] assume that auditors are honest but curious, however, from the perspective of data integrity verification, there is no difference between these two assumptions, since the auditors would not deviate from the prescribed schemes.

These schemes cannot resist dishonest auditors. Dishonest auditors can be divided into two types: the first one is called a malicious auditor, the second one is

called a procrastinating auditor. We will analyze how they invalidate these schemes in the following.

For the malicious auditor, the most trivial attack it can launch is that it always generates a good integrity report without verifying the data integrity to avoid the verification burden. In such a way, the auditor is virtually non-existent. In addition, a malicious auditor may generate a bias verification result to deceive the users for profits. For example, the auditor colludes with the cloud server and always creates bias challenging messages such that only the data blocks which are well maintained are verified, which could conceal the data corruption.

The procrastinating auditor may procrastinate on the scheduled verification to invalidate the public verification schemes. Specifically, assuming the agreed verification period in the public verification scheme is 1 day, and an epoch is 1 month (i.e., 30 days), this means that the auditor checks the outsourced data integrity one time per day, and the user checks the auditor's log file recording the verification results one time per month. Normally, the auditor would perform the verification every day and generate a verification report every 30 days. For a procrastinating auditor, it would not perform the verification on the first 29 days and would perform the verification 30 times on the last day, where the challenging messages in each verification of the first 29 days can be regenerated in the 30th day. As such, the verification report only reflects the most recent (the 30th day's) state of integrity for the outsourced data. The reason why the auditor procrastinates on the scheduled verification is straightforward: when the data corruption occurs, the cloud server may collude with the auditor, where it asks the auditor for halting the scheduled verification and gains much more time to retrieval the outsourced data for a good reputation. This deviates from the public verification's original target: if the outsourced data are corrupted, the data owner is able to find it within 1 day (i.e., one verification period).

3.4.2.2 Blockchain-Based Public Verification Against Dishonest Auditors

To resist the malicious auditor who only generates a good integrity report without verifying the data integrity, a straightforward solution is to require the user to audit the auditor at the end of each epoch. The auditor's behavior can be reflected by the challenging messages it generated and the proof information it received in each verification task. Due to the security of the public verification schemes, it is computationally infeasible for the auditor to forge proof information that could be valid under any challenging messages selected by it.

However, such a strategy cannot resist malicious auditors who collude with the cloud server and only checks the data blocks that are well maintained on the cloud server. To resist such the malicious auditor, in addition to audit the auditor, the data owner should also check the randomness of challenged blocks in each verification. However, ensuring the randomness of challenged blocks in public verification schemes is very challenging due to the following reasons. First, it is insecure to require the data owner to pre-generate the challenging messages at the

beginning of each epoch, since the auditor who colludes with the cloud server can leak all challenging messages to the cloud server such that the cloud server either can delete the blocks that would not be verified in subsequent verifications or can pre-generate all corresponding proofs to enable the auditor to pass the user's auditing. Second, the interactive generation of challenging messages among the auditor and the cloud server would also suffer from the collusion attacks. Furthermore, to allow the user to audit the auditor's behavior, the randomness used to generate challenging messages should be publicly verifiable after it is generated. The requirement on the randomness "seed" in public verification is akin to those in a lottery where players guess a target number and a banker periodically publish a winning number; If a player's number is equal to the winning number, the player is the lottery winner. This seed should be time-dependent, i.e., given a determinate time t , if t is a future time, the seed generated in t is unpredictable; If t is a past time, the seed generated in t can be easily verifiable and is resistant to modification.

Such a time-dependent random seed can be derived from public blockchain systems. A straightforward way to obtain the seed is to use the hash value of block that is latest confirmed on the blockchain. However, the security of such a way is too weak. Specifically, if the outsourced data are corrupted, and the indexes of corrupted data form a set CR . The target of the cloud server who compromises the auditor is to conceal the data corruption by sampling biased challenging messages. The cloud server knows the next time when the challenging message would be generated. When the time gets close, the cloud server can incentivize the miners who mine a new block to throw the newly mined block away and continue to mine if the challenging message derived from the hash value of the block (denoted by Bl_t) covers the corrupted data blocks, i.e., the indexes of blocks to be verified exist in CR . Essentially, the malicious cloud server can launch the above attack since the entropy of the challenging message derived from the hash value of a single block on the blockchain is low.

To address the above problem, the existing scheme [45] derives the challenging messages from φ successive blocks that are the latest ones confirmed on the Ethereum blockchain, where φ denotes the number of blocks deep used to confirm a transaction. By doing so, the security of the challenging messages is based on the (t, φ) -chain quality of secure blockchains.

The above mechanism is still vulnerable to procrastinating auditors. To resist the procrastinating auditor, a trivial solution is to let the user audit the auditor's behaviors in a random time interval. Whereas, before the user audits the correctness of the auditor's behaviors, she/he needs to interact with the auditor to obtain the log file that records the challenging message and proof information in each verification task. This sufficiently gives rise to forge the challenging messages for the auditor and cloud server, since it is easy to re-compute the challenging messages and the corresponding proofs given access to the blockchain. As such, a procrastinating auditor is able to pass the user's auditing by colluding with the cloud server. Another trivial solution is to introduce a trusted service provider (short for TSP) who provides a time-stamping service [46]. After each verification, the auditor is required to request the time-stamping service on the challenging message and

proof from TSP. This makes the verification performed by the auditor time-sensitive and allows the user to check the timeliness of each verification, which thwarts the procrastinating auditor. Nevertheless, the security of such mechanisms relies on the security and reliability of TSP, and TSP here becomes a single point of failure. Furthermore, TSP has to bear heavy communication and computation burden in the case of multiple users and auditors. As such, how to resist the procrastinating auditor without introducing any trusted entity is a very challenging problem.

The blockchain now becomes a panacea for public verification to resist dishonest auditors [45]. To resist the procrastinating auditor, the key idea is to use a public blockchain and integrate the challenging message and proof information generated during each verification into a transaction on the blockchain. After that, the user is able to verify the time when the auditor performs the verification by checking the generation time of the transaction. The security of such mechanism is based on the φ -chain consistency of secure blockchains.

This yields the final blockchain-based public verification scheme that resists both malicious and procrastinating auditors. The scheme is shown in Fig. 3.6. The scheme consists of two phases. In the first phase, the auditor verifies the integrity of outsourced data on behalf of the user. In the second phase, the user audits the auditor's behavior.

Specifically, in the first phase, the verification period is determined by the user. For a point in time when the data integrity should be verified, the auditor first extracts the hash values of φ successive blocks that are the latest ones confirmed on the public blockchain, and these hash values are denoted by $\{Bl_{t-\varphi+1}, Bl_{t-\varphi+2}, \dots, Bl_t\}$, where t is the height of the latest confirmed block (i.e., the current height of the blockchain). Then the auditor generates a challenging message on $\{Bl_{t-\varphi+1}, Bl_{t-\varphi+2}, \dots, Bl_t\}$, and sends the challenging message to the cloud server. Upon receiving the challenging message, the cloud server computes

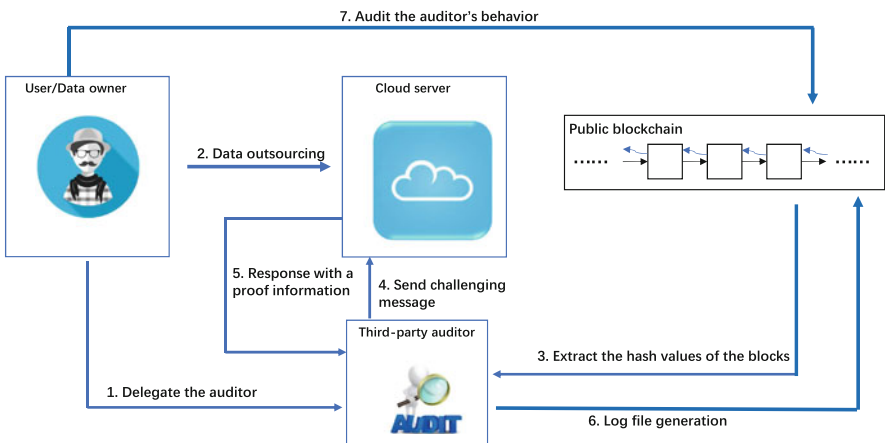


Fig. 3.6 Blockchain-based public verification scheme

the corresponding proof. The auditor checks the validity of the proof to verify data integrity. If the checking fails, the auditor informs the user that the data may be corrupted; Otherwise, the auditor sets $\{Bl_{t-\varphi+1}, Bl_{t-\varphi+2}, \dots, Bl_t\}$ and the proof as a log entry, stores the entry to a log file, and creates a transaction that transfers 0 deposit from its account to the user's account,⁴ wherein the data field is set to the hash value of the entry. Ideally, this transaction would be recorded to the block whose height is $t + \varphi + 1$.

In the second phase, the user audits the auditor's behavior in a much longer period compared with the verification period. We first show how is a single entry (without loss of generality, $\{Bl_{t-\varphi+1}, Bl_{t-\varphi+2}, \dots, Bl_t\}$ and the corresponding proof) in the log file audited by the user. The user first determines the verification time that the auditor should verify the data integrity. Then she/he obtains $\{Bl_{t-\varphi+1}, Bl_{t-\varphi+2}, \dots, Bl_t\}$ from the public blockchain according to the agreed verification time, and extracts the hash value of the entry from the transaction. Next, she/he regenerates the challenging message on $\{Bl_{t-\varphi+1}, Bl_{t-\varphi+2}, \dots, Bl_t\}$, and checks the validity of the corresponding proof by using the challenging message generated by herself/himself. If the checking passes, it means that the auditor performs the verification correctly. Due to the homomorphism of HVTs utilized in the underlying public verification scheme, multiple entries can be audited simultaneously, and the auditing costs can be amortized over these entries to reduce the computational costs.

3.5 Summary and Further Reading

In this chapter, we have introduced the data integrity verification technique for cloud storage systems. We have provided a comprehensive overview of existing data integrity verification schemes and analyzed their pros and cons, and also conducted a comparison between them. Finally, we have introduced the latest advances in data integrity verification technique for cloud storage systems.

There are also some survey papers to introduce the data integrity verification technique from different aspects, such as [47, 48]. Data integrity is one of the most important factors affecting the reliability of cloud storage systems. Based on the data integrity verification technique, some mechanisms are also proposed to enhance the reliability of cloud storage services. We refer the reader to papers [49, 50] for further details.

⁴Most of public blockchains can allow a payer to conduct a transaction wherein the transaction value is 0.

References

1. Juels A, Kaliski BS Jr (2007) Pors: proofs of retrievability for large files. In: ACM conference on computer and communications security, pp 583–597
2. Ateniese G, Burns R, Curtmola R, Herring J, Kissner L, Peterson Z, Song D (2007) Provable data possession at untrusted stores. In: ACM conference on computer and communications security, pp 598–609
3. Deswarte Y, Quisquater J-J, Saïdane A (2003) Remote integrity checking. In: Working conference on integrity and internal control in information systems, pp 1–11
4. Filho DG, Barreto P (2006) Demonstrating data possession and uncheatable data transfer. Cryptol ePrint Archive. Report 2006/150, 1–9
5. Schwarz T, Miller E (2006) Store, forget, and check: using algebraic signatures to check remotely administered storage. In: IEEE international conference on distributed computing systems, pp 1–10
6. Diffie W, Hellman M (1976) New directions in cryptography. *IEEE Trans Inf Theory* 22(6):644–654
7. Johnson R, Molnar D, Song D, Wagner D (2002) Homomorphic signature schemes. In: Cryptographers’ track at the RSA conference, pp 244–262
8. Shi E, Stefanov E, Papamanthou C (2013) Practical dynamic proofs of retrievability. In: ACM conference on computer and communications security, pp 325–336
9. Reed IS, Solomon G (1960) Polynomial codes over certain finite fields. *J Soc Indust Appl Math* 8(2):300–304
10. Shacham H, Waters B (2008) Compact proofs of retrievability. In: International conference on the theory and application of cryptography and information security, pp 90–107
11. Shacham H, Waters B (2013) Compact proofs of retrievability. *J Cryptol* 26(3):442–483
12. Boneh D, Lynn B, Shacham H (2001) Short signatures from the weil pairing. In: International conference on the theory and application of cryptography and information security, pp 514–532
13. Ateniese G, Pietro RD, Mancini LV, Tsudik G (2008) Scalable and efficient provable data possession. In: International conference on security and privacy in communication networks, pp 1–9
14. Erway CC, K p c  A, Papamanthou C, Tamassia R (2015) Dynamic provable data possession. *ACM Trans Inf Syst Secur* 17(4):15
15. Yang K, Jia X (2013) An efficient and secure dynamic auditing protocol for data storage in cloud computing. *IEEE Trans Parallel Distributed Syst* 24(9): 1717–1726
16. Yang A, Xu J, Weng J, Zhou J, Wong DS (2018) Lightweight and privacy-preserving delegatable proofs of storage with data dynamics in cloud storage. *IEEE Trans Cloud Comput*. <https://doi.org/10.1109/TCC.2018.2851256>
17. Wang Q, Wang C, Li J, Ren K, Lou W (2009) Enabling public verifiability and data dynamics for storage security in cloud computing. In: European symposium on research in computer security, pp 355–370
18. Cash D, K p c  A, Wichs D (2013) Dynamic proofs of retrievability via oblivious ram. In: International conference on the theory and applications of cryptographic techniques, pp 279–295
19. Cash D, K p c  A, Wichs D (2017) Dynamic proofs of retrievability via oblivious ram. *J Cryptol* 30(1):22–57
20. Pinkas B, Reinman T (2010) Oblivious ram revisited. In: Annual cryptology conference, pp 502–519
21. Wang C, Wang Q, Ren K, Lou W (2010) Privacy-preserving public auditing for data storage security in cloud computing. In: IEEE international conference on computer communications, pp 1–9
22. Wang C, Chow SS, Wang Q, Ren K, Lou W (2013) Privacy-preserving public auditing for secure cloud storage. *IEEE Trans Comput* 62(2):362–375

23. Worku S, Xu C, Zhao J, He X (2014) Secure and efficient privacy-preserving public auditing scheme for cloud storage. *Comput Elect Eng* 40(5):1703–1713
24. Li J, Zhang L, Liu JK, Qian H, Dong Z (2016) Privacy-preserving public auditing protocol for low-performance end devices in cloud. *IEEE Trans Inf Forensics Secur* 11(11):2572–2583
25. Ni J, Yu Y, Mu Y, Xia Q (2013) On the security of an efficient dynamic auditing protocol in cloud storage. *IEEE Trans Parallel Distrib Syst* 25(10):2760–2761
26. Zhang Y, Xu C, Zhao J, Zhang X, Wen J (2015) Cryptanalysis of an integrity checking scheme for cloud data sharing. *J Inf Secur Appl* 23:68–73
27. Xu C, Zhang Y, Yu Y, Zhang X, Wen J (2014) An efficient provable secure public auditing scheme for cloud storage. *KSII Trans Int Inf Syst* 8(11):4226–4241
28. Wang H, Wu Q, Qin B, Domingo-Ferrer J (2013) Identity-based remote data possession checking in public clouds. *IET Inf Secur* 8(2):114–121
29. Zhao J, Xu C, Li F, Zhang W (2013) Identity-based public verification with privacy-preserving for data storage security in cloud computing *IEICE Trans Fundamen Electron Commun Comput Sci* 96(12):2709–2716
30. Boneh D, Franklin M (2001) Identity-based encryption from the weil pairing. In: Annual cryptology conference, pp 213–229
31. Al-Riyami SS, Paterson KG (2003) Certificateless public key cryptography. In: International conference on the theory and application of cryptology and information security, pp 452–473
32. Wang B, Li B, Li H, Li F (2013) Certificateless public auditing for data integrity in the cloud. In: IEEE conference on communications and network security, pp 136–144
33. Zhang Y, Xu C, Yu S, Li H, Zhang X (2015) SCLPV: secure certificateless public verification for cloud-based cyber-physical-social systems against malicious auditors. *IEEE Trans Comput Soc Syst* 2(4):159–170
34. He D, Zeadally S, Wu L (2018) Certificateless public auditing scheme for cloud-assisted wireless body area networks. *IEEE Syst J* 12(1):64–73
35. Wang B, Li B, Li H (2014) Oruta: privacy-preserving public auditing for shared data in the cloud. *IEEE Trans Cloud Comput* 2(1):43–56
36. Wang B, Li B, Li H (2015) Panda: public auditing for shared data with efficient user revocation in the cloud. *IEEE Trans Serv Comput* 8(1):92–106
37. Rivest RL, Shamir A, Tauman Y (2001) How to leak a secret. In: International conference on the theory and application of cryptology and information security, pp 552–565
38. Camenisch J, Stadler M (1997) Efficient group signature schemes for large groups. In: Annual cryptology conference, pp 410–424
39. Sahai A, Waters B (2014) How to use indistinguishability obfuscation: deniable encryption, and more. In: ACM symposium on theory of computing, pp 475–484
40. Boneh D, Gupta D, Mironov I, Sahai A (2015) Hosting services on an untrusted cloud. In: International conference on the theory and applications of cryptographic techniques, pp 404–436
41. Guan C, Ren K, Zhang F, Kerschbaum F, Yu J (2015) Symmetric-key based proofs of retrievability supporting public verification. In: European symposium on research in computer security, pp 203–223
42. Zhang Y, Xu C, Liang X, Li H, Mu Y, Zhang X (2017) Efficient public verification of data integrity for cloud storage systems from indistinguishability obfuscation. *IEEE Trans Inf Forensics Secur* 12(3):676–688
43. Boneh D, Waters B (2013) Constrained pseudorandom functions and their applications. In: International Conference on the theory and application of cryptology and information security, pp 280–300
44. Zhang Y, Xu C, Li H, Yang K, Cheng N, Shen X (2020) PROTECT: efficient password-based threshold single-sign-on authentication for mobile users against perpetual leakage. *IEEE Trans Mob Comput*. <https://doi.org/10.1109/TMC.2020.2975792>
45. Zhang Y, Xu C, Lin X, Shen X (2019) Blockchain-based public integrity verification for cloud storage against procrastinating auditors. *IEEE Trans Cloud Comput*, 1–15. <https://doi.org/10.1109/TCC.2019.2908400>

46. Haber S, Stornetta WS (1990) How to time-stamp a digital document. In: Annual cryptology conference, pp 437–455
47. Sookhak M, Gani A, Talebian H, Akhuzada A, Khan SU, Buyya R, Zomaya AY (2015) Remote data auditing in cloud computing environments: a survey, taxonomy, and open issues. *ACM Comput Surv* 47(4):1–34
48. Wang C, Ren K, Lou W, Li J (2010) Toward publicly auditable secure cloud data storage services. *IEEE Netw* 24(4):19–24
49. Armknecht F, Barman L, Bohli J-M, Karame GO (2016) Mirror: enabling proofs of data replication and retrievability in the cloud. In: *USENIX security symposium*, pp 1051–1068
50. Curtmola R, Khan O, Burns R, Ateniese G (2008) MR-PDP: multiple-replica provable data possession. In: *IEEE international conference on distributed computing systems*, pp 411–420

Chapter 4

Secure Deduplication



This chapter introduces the secure data deduplication technique for cloud storage systems. First, the basic paradigms, principles, and classification of the secure data deduplication technique are presented. Then, the secure deduplication over outsourced data (which are in the plaintext form) is reviewed, where the threats and countermeasures are introduced and analyzed. Next, the secure deduplication over outsourced encrypted data is introduced, and a comprehensive survey on encrypted deduplication systems is provided. Finally, the latest advances in the secure encrypted data deduplication technique are studied, which shows the potentials and benefits of applying the encrypted deduplication technique to eHealth systems.

4.1 Deduplication Classification

With the significant development of cloud storage, people are increasingly outsourcing their data to cloud servers, which enables them to efficiently manage their data without deploying infrastructures and maintaining local devices. Commercial cloud service providers always perform data deduplication across their users to save storage space, where a service provider checks duplicated data, only stores a single copy of duplicated data, and provides links to that copy instead of storing other copies, as shown in Fig. 4.1. According to the investigation from recent literature [1, 2], such a deduplication strategy can reduce storage costs by more than 65% in electronic health (eHealth) systems and 90% in backup systems.

From the perspective of the form of target data, deduplication schemes can be divided into two types:

- Deduplication over plaintext data. Users outsource their data in the plaintext form and the cloud server performs data deduplication over plaintexts.

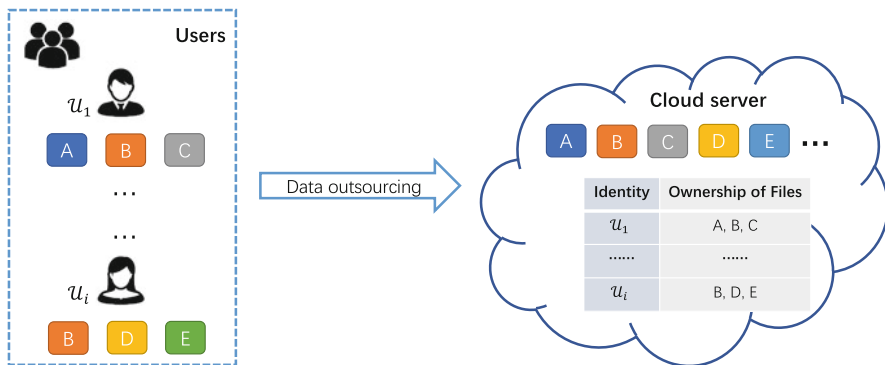


Fig. 4.1 Illustration of deduplication in cloud storage

- Deduplication over encrypted data. Users encrypt their data and outsource the ciphertexts to the cloud server. The cloud server performs data deduplication over ciphertexts.

From the perspective of deduplication granularity, deduplication schemes can be divided into two types:

- File-level deduplication. The data redundancy is exploited on the file level, only one instance of the data is outsourced, and subsequent copies are replaced with a pointer that points to the instance.
- Block-level deduplication. The data is split into multiple blocks, and the cloud server performs deduplication over all blocks. Compared with file-level deduplication, utilizing block-level deduplication can achieve a higher “deduplication ratio,” but requires more computational costs.

From the perspective of deduplication architecture, deduplication schemes can be divided into two types:

- Server-side deduplication. Data deduplication is only performed by the cloud server and users are unaware of deduplication that might occur. Such a deduplication strategy can only reduce storage costs.
- User-side deduplication. Data deduplication is performed by the cloud server in cooperation with users. Specifically, when a user wants to outsource a file M , she/he first interacts with the cloud server to check whether M has been outsourced before. If M has been outsourced previously, the user does not need to upload M to the cloud server. The cloud server labels the user with the “owner” of M . Subsequently, M can be accessed by the user. Since users do not need to upload duplicated data to the cloud server, such a deduplication strategy can reduce the storage costs and the communication costs simultaneously.

Despite the great benefits brought by data deduplication, it also introduces new threats towards the security of cloud storage systems. In this chapter, we

first introduce the deduplication technique for cloud storage systems from the perspective of security, then introduce the state-of-the-art research on encrypted data deduplication.

4.2 Secure Deduplication: Threats and Countermeasures

In the nascent stage of cloud storage services, data are outsourced to the cloud server in plaintext form. It is easy to check duplicate data across its users for the cloud service provider. In this case, the cloud service provider mainly utilizes the user-side deduplication to save both communication and storage costs.

Specifically, for an outsourced file M , the cloud server computes $t_M = h(M)$ as a fingerprint to support efficient deduplication. For a subsequent user who wants to outsource a file M' , she/he first sends $h(M')$ to the cloud server. Then, the cloud server checks whether $h(M') = t_M$ holds, if yes, it means that M' has been outsourced before. As such, the user does not need to upload M' to the cloud server. She/he is labeled with the owner of M' and deletes M' in her/his local storage.

Despite large amounts of savings in terms of communication costs and storage space, critical security concerns in data deduplication have been raised seriously. In the above deduplication scheme, an adversary can utilize the deduplication services to construct an unintended content distribution network (CDN). Particularly, the adversary can outsource M to the cloud server and broadcast the deduplication fingerprint t_M to everyone who wishes to obtain M . Subsequently, a user who does not own M can request M from the cloud server. This attack can contribute to piracy and copy infringing behavior, since it is easy to be launched and hard to be traced in reality. Furthermore, note that the cloud storage is an on-demand service, in the above attack, when the size of M is large, the adversary is able to launch the attack to share the data to others without bearing additional costs. More critical, if the adversary is able to temporarily compromise the cloud server (or he colludes with a malicious insider working at the service provider), he can obtain all deduplication fingerprints of outsourced files and access these files from the cloud server, which violates the profits of both the cloud server and the users.

By using the attack discussed above, the adversary essentially uses the deduplication service itself as a converted channel to transfer some information without payment and being detected. The success of the attack is based on the fact that if a user has a file's fingerprint, she/he is considered as an owner of the file. This assumption is too strong such that the adversary can easily impersonate an "owner" to access the data that he does not actually own. To resist the attack, a straightforward solution is to stop user-side deduplication. However, it makes savings on communication costs impossible.

4.2.1 Proofs of Ownership

To thwart the attack, when the cloud server receives the fingerprint t_M of M from a user \mathcal{U} , it could not directly label \mathcal{U} with the owner of M . It should verify that \mathcal{U} actually possesses M . As such, a proof mechanism is required to enable \mathcal{U} to prove her/his ownership of M . Such the proof mechanism is called Proof of Ownership (POW), (hereinafter, to avoid misunderstanding and equivocation, Proof of Ownership is abbreviated to POW, while Proof of Work is abbreviated to PoW) and is first proposed by Halevi et al. [3].

Essentially, POW is an independent component that is integrated into the deduplicated cloud storage system to enable a user who claims herself/himself is the owner of data M to prove her/his ownership to the cloud server. The “claim” is achieved by providing the correct fingerprint, and resistance against the above attack is ensured by POW. The security of POW guarantees that if a user does not possess M (even if she/he loses one bit), she/he cannot pass the server’s verification.

Recall the PoR/PDP-based proof of storage schemes introduced in the previous chapter, it seems that a secure proof of storage scheme can be utilized directly to construct a POW scheme via a role reversal: the user in POW becomes the prover in the proof of storage scheme and the cloud server in POW becomes the auditor in the proof of storage scheme. However, such a conversation is impractical due to the following reason.

In PoR/PDP-based proof of storage schemes, a pre-processing step is required to utilizing some secrets to process the data, which enables the auditor to verify the data integrity after data outsourcing with resistance against cheating from the cloud server. Whereas, in POW, users need to prove the cloud server their ownership of some data which have been outsourced before. If the first user who uploads a file M to the cloud server processes M using a secret sk , the subsequent users who want to outsource M to the cloud server cannot prove their ownership of M , since they do not know sk .

In the remainder of this subsection, we will overview the first POW scheme [3] and elaborate on its construction.

A user \mathcal{U} and a cloud server \mathcal{CS} are involved in the POW scheme. Notice that traditional hash-based user-side deduplication over outsourced data suffers from malicious content distribution, since the hash value of the file serves as the fingerprint of the file to represent the “proxy” of the file’s ownership. A straightforward way to construct POW scheme is to use application-specific hash function and salt: the fingerprint of the file M is $t_M = \mathfrak{h}(ID_{\mathcal{CS}}||salt||M)$, where $salt$ is selected by \mathcal{CS} for all files, and \mathfrak{h} is a secure hash function that would not be used by other applications. However, the above mechanism does not address the problem that a large file is represented by a short string. Although one attack can be resisted, there could be others that are more tricky to cause the vulnerability.

To resolve the above problem, a challenge-response mechanism is introduced to verify the ownership of M for \mathcal{U} . Specifically, we assume that M has been outsourced to \mathcal{CS} , when \mathcal{U} proves her/his ownership of M , \mathcal{CS} first randomly selects a nonce $nonce$ and sends $nonce$ to \mathcal{U} ; \mathcal{U} computes $t_M = h(nonce||M)$ as the proof her/his ownership of M ; \mathcal{CS} verifies the ownership by checking the validity of t_M . Such a challenge-response mechanism ensures that if a user can pass the cloud server's verification, she/he must possess the data intact. Despite the high-level security guarantee that the above challenge-response mechanism can achieve, it is confronted with an efficiency problem. Note that to verify the user's ownership, the cloud server has to retrieve the entire data from its secondary storage, it introduces considerable costs, especially when there is a large number of data items and users.

With the above efficiency problem, a practical POW scheme should achieve the following efficiency objective: the cloud server should only store a piece of short information computed from the data itself for the ownership verification; The proof of ownership of the data should be much shorter than the data size. This efficiency objective is important in practice, since it ensures that the user and the cloud server are able to execute the POW scheme with high efficiency.

An efficient POW scheme can be directly derived from the Merkle hash tree (short for MHT): a file M is first split into multiple blocks as $M = \{M_1, M_2, \dots, M_n\}$; Each block is hashed to be a leaf of MHT, such that all blocks form the MHT and the cloud server \mathcal{CS} can obtain the root of MHT. When a user \mathcal{U} wants to prove her/his ownership of M , \mathcal{CS} asks \mathcal{U} to provide paths to c random leaves; With the proof information received from \mathcal{U} , \mathcal{CS} is easy to verify the ownership using the outsourced data. By doing so, both \mathcal{U} and \mathcal{CS} only need to perform hash operations to generate and verify the proof of ownership, which is highly efficient. However, the security of such a mechanism is arguable. Actually, in the above mechanism, if an adversary only has a ρ fraction of M , he can pass \mathcal{CS} 's verification with a probability $Pr < \rho^c$. Specifically, if $\rho = 95\%$, the adversary can pass \mathcal{CS} 's verification with the probability of 0.6.

To address the problem, the erasure code can be used: M is first processed by using the erasure code to obtain M' ; M' is split into multiple blocks to form the MHT, which is the same as before. By the way, the adversary, who does not have M (even he loses a single block of M), cannot obtain M' . Nonetheless, processing M using erasure code is inefficient, especially when the size of M is large. Furthermore, if M stored on the user side does not fit in memory, the user needs to perform a large number of disk-seeking operations. Hence, for a practical POW scheme, the costs in terms of computation and storage should be reduced significantly while retaining the security. In [3], this is achieved by three phases: reducing, mixing, and authenticating by MHT. Specifically, in the reducing phase, M (assuming M consists of n blocks and each block has x bits) is first mapped to a y bits buffer, where each block is XORed to a constant number of random locations in the buffer; in the mixing phase, a Feistel-like structure is introduced to diffuse the reduced data (which is stored in the buffer); finally, an MHT is built on the reduced and mixed file for proving the ownership.

Following the POW scheme proposed in [3], some POW schemes are presented to enhance the functionality. In the subsequent work, researchers focus on designing secure POW schemes for encrypted data, which we will introduce in Sect. 4.4.

4.2.2 *Randomized Deduplication*

We now introduce a more tricky attack towards the deduplication scheme. Compared with the cloud server, the devices (e.g., laptops and mobile phones) that users equip are more vulnerable to be compromised in reality. If an adversary can temporarily compromise a user's device, he can utilize the deduplication service as an oracle to obtain some incentive information about the user. Particularly, the adversary first installs some malicious software on the target user's device. The software first generates two random long binary strings r_0 and r_1 . At some point in time, the user may generate some files locally and outsources a part of these files to the cloud server. The software monitors the generated files, if a predetermined file M occurs, it outsources r_1 to the cloud server, if M does not occur, it outsources r_0 to the cloud server. The adversary can extract the existence of M by outsourcing both r_0 and r_1 to the cloud server and checking which one is duplicated. By doing so, the deduplication service is leveraged as an oracle to respond to the adversary, and sensitive information can be leaked. Such an attack is well scalable, which enables the adversary to launch more expressive attacks than the above one.

To resist such an information leakage, a paradigm of randomized deduplication is proposed: the cloud server keeps an independent random threshold for every file. The threshold is selected uniformly at random in a range $[2, x]$, where x is a parameter that might be public. When the upload number of the file is less than the threshold, the cloud server would only perform server-side deduplication; When the number achieves the threshold, user-side deduplication is triggered, and the cloud server performs deduplication over the outsourced data. Note that the threshold is randomly chosen and is unknown to the user, the adversary cannot directly leverage the user-side deduplication service to extract information on the victim's device from the cloud server.

4.3 Message-Locked Encryption

The deduplication technique plays an important role in commercial cloud storage services, due to the large savings on storage costs. However, from the perspective of data owners, the content of outsourced data should not be leaked for security reasons. Therefore, the privacy protection of the data content against anyone who does not own the data should be guaranteed. Actually, such a guarantee has become one of the most important selling points for current cloud storage systems. To this

end, the data are always encrypted by using conventional encryption algorithms before outsourcing.

Generally, if users encrypt their data using conventional encryption schemes, deduplication is impeded. Specifically, if a user \mathcal{U}_1 wants to outsource a file M to the cloud server \mathcal{CS} , she/he first encrypts M using a symmetric encryption algorithm E with a random key k_1 as $C_1 = E(k_1, M)$, and outsources C_1 to \mathcal{CS} . Subsequently, when another user (say \mathcal{U}_2) wants to outsource the same file to \mathcal{CS} , she/he would upload $C_2 = E(k_2, M)$ to \mathcal{CS} , where k_2 is randomly chosen by \mathcal{U}_2 . Due to the following reasons, deduplication is impeded.

First, since different users would choose different encryption keys, and thereby output different ciphertexts for the same file, the cloud server cannot detect that the file underlying the two ciphertexts is the same. Second, even though the cloud server can so detect, either \mathcal{U}_1 or \mathcal{U}_2 cannot decrypt the ciphertext outsourced to the cloud server using the key stored locally.

Message-locked encryption (MLE) is a special type of symmetric encryption, in which the MLE key (i.e., the encryption and decryption key) is derived from the plaintext itself. As the name indicates, in an MLE scheme, the message is locked under itself. This guarantees that different users output the same ciphertext for the same plaintext, and enables the cloud server to perform deduplication over encrypted data. MLE serves the key component in encrypted deduplication cloud storage systems.

4.3.1 Overview

An MLE scheme generally consists of four algorithms, as shown in Fig. 4.2. Particularly, an MLE scheme $\text{MLE} = \{\text{MLEKey}, \text{Enc}, \text{Dec}, \text{Tag}\}$ is a four-tuple of probabilistic-time algorithms (for the sake of brevity, we omit the public parameter generation algorithm hereinafter), where MLEKey is the key generation algorithm which takes the public parameter PP and a message M as inputs and outputs an MLE key k ; Enc is an encryption algorithm which takes PP , M , and k as inputs and outputs the ciphertext C ; Dec is a decryption algorithm which takes PP , k , and C as inputs and outputs M (if decryption succeeds) or \perp (if decryption fails); Tag is a tag generation algorithm which takes PP and C as inputs and outputs a tag τ (i.e., deduplication fingerprint) that supports checking duplicate data.

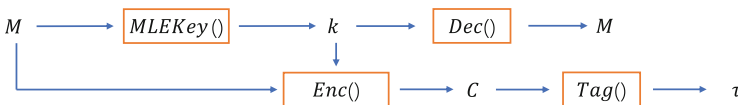


Fig. 4.2 General description of MLE

Theoretically, any encryption algorithm that supports deduplication across multiple users can be subsumed into MLE. However, MLE is used to reduce storage costs from deduplication, and thereby several practical considerations should be taken.

First, an MLE key that is generated in *MLEKey* should be much shorter than the message. Notice that a trivial construction of an MLE scheme is that the message is encrypted under itself, i.e., $k = M$. However, this scheme is of no use for deduplication since each owner of M should store the entire data as the MLE key and no storage savings are achieved.

Second, *MLEKey* can be executed by the data owner without interacting with other users who also own M . In reality, requiring a user to interact with other users for data outsourcing not only changes the user's interaction pattern, but also introduces additional communication costs, which is cumbersome and impractical. In most cases, users expect to utilize the cloud storage service in a black-box way with minimal monetary costs.

With the above considerations, there are generally four types of MLE schemes.

The first one is called convergent encryption (short for CE), which is first proposed by Douceur et al. [4] and shown in Fig. 4.3. In CE, the MLE key is just the hash value of the message and the tag is the hash value of the ciphertext, i.e., $k = h(M)$, $\tau = h(C)$, where $h : \{0, 1\}^* \rightarrow Z_p$ is a secure hash function.

In a CE scheme, generating the tag (i.e., the deduplication fingerprint) requires the cloud server to access the entire ciphertext C , which is inefficient. To improve efficiency, a variant of CE called HCE1 is proposed. Specifically, as shown in Fig. 4.4, HCE1 sets the tag as $\tau = h(k)$ and $C = E(k, M) || \tau$. By doing so, the cloud server only needs to use the hash value of MLE key k received from the user as the tag, which improves the performance significantly.

Despite the performance improvement, HCE1 suffers from duplicate faking attacks. Specifically, assume an adversary \mathcal{A} has a file $M_{\mathcal{A}}$ and user \mathcal{U} has a file M . Both of them encrypt their data using HCE1. \mathcal{A} outsources not an honest encryption of $M_{\mathcal{A}}$ but a maliciously generated ciphertext $C_{\mathcal{A}}$ such that, when \mathcal{U} attempts to

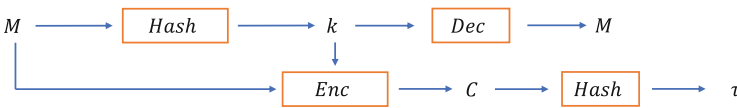


Fig. 4.3 Description of CE

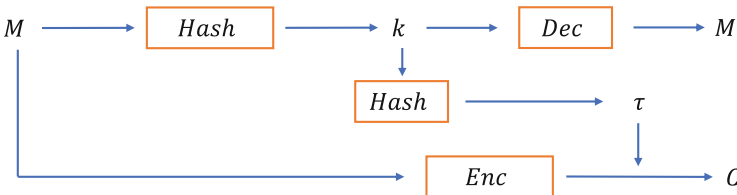


Fig. 4.4 Description of HCE1

outsource M , the cloud server sees that the tag (i.e., deduplication fingerprint) $\tau_{\mathcal{A}}$ of $C_{\mathcal{A}}$ is equal to the tag τ of C (which is the ciphertext of M). The cloud server thus only stores $C_{\mathcal{A}}$ and deletes C . However, when \mathcal{U} subsequently downloads the ciphertext from the cloud server and decrypts it, she/he only obtains $M_{\mathcal{A}}$ rather than M , which means that the data has been corrupted. Resistance against duplicate faking attacks is related to a security notion of *tag consistency* (short for TC). TC asks that it should be hard to generate a pair of plaintext and ciphertext (M, C) such that $Tag(C) = Tag(Enc(MLEKey(M), M))$ but $Dec(MLEKey(M), C)$ is a string different from M .

To resist duplicate faking attacks, a variant of HCE1, called HCE2, is proposed. In HCE2, an additional mechanism, called guarded decryption, is integrated into HCE1: After \mathcal{U} downloads C from the cloud server, she/he decrypts the ciphertext to obtain a file M' , recomputes the tag using M' , and checks whether the recomputed tag is equal to the one embedded in the ciphertext. If the checking fails, \mathcal{U} returns \perp and rejects the ciphertext.

Note that in cloud storage systems, once a user outsources her/his data to the cloud server, she/he would delete the data in the local storage. Consequently, in HCE2, if the adversary launches the duplication faking attack, the outsourced data of subsequent users would be substituted, and the original data (i.e., the correct data) might not be recovered, even if the duplication faking attack can be detected. To mitigate this problem, a variant of HCE2, called randomized convergent encryption (RCE), has been proposed.

As shown in Fig. 4.5, in RCE, the first user who uploads the data M randomly chooses an encryption key L and computes k that is the hash value of M . L is utilized to encrypt M . k is used to compute the tag τ and is used to be the encryption key to encrypt L . With M , anyone is able to compute the tag, which enables the cloud server to check duplicate data. Subsequent users, who have proven their ownership of M , can download C_2 from the cloud server and decrypt it using the newly computed k to obtain L , and encrypt locally stored M using L and sends the hash value of the ciphertext to the cloud server. With the hash value of the ciphertext, the cloud server can check whether C_1 is the actual ciphertext of M . RCE is an elegant design thwarting duplication faking attacks and can be directly utilized to construct a secure cloud storage system with user-side deduplication.

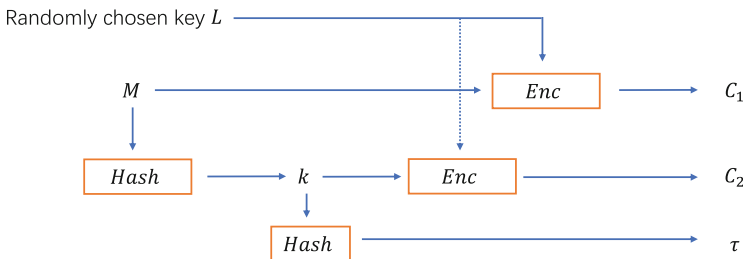


Fig. 4.5 Description of RCE

We stress that RCE is confronted with the information leakage problem introduced in Sect. 4.2.2, and the randomized deduplication strategy can be integrated directly to address the problem.

MLE serves as the key component cloud storage systems to ensure the confidentiality of outsourced data and reduce the storage costs via deduplication. In the following sections of this chapter, we will focus on introducing cloud storage systems with encrypted deduplication.

4.3.2 Threat Models of Encrypted Deduplication Storage Systems

We first introduce the system model of encrypted deduplication storage systems. Two entities are involved in the system: users and a cloud server. The system model is similar to that shown in Fig. 4.1, with one difference that the data are outsourced to the cloud server in the ciphertext form.

Users. The users are data owners and outsource files to the same storage server. They never communicate directly, but they desire to reduce the storage costs from deduplication. For the purpose of privacy protection, the contents of files should not be leaked to anyone who does not own the data. Therefore, the files should be encrypted before outsourcing.

Cloud server. The cloud server is subject to the cloud storage service provider and provides storage services for users. It checks the duplicate file across its users, stores only a single copy of the duplicate file to reduce the storage costs.

Two types of adversaries are considered in the threat model.

The first one is a malicious but rational cloud server. The goal of the malicious cloud server is to violate the users' privacy by extracting the contents of outsourced files. To this end, the malicious cloud server might launch various attacks. On the other hand, the cloud server is also a rational party [5]: it would not launch attacks if its profits cannot be increased.

The second one is the malicious user. Notice that in a cloud storage system, anyone can become a valid user by registering with the cloud server. Therefore, any adversary can first become a valid user in the system to increase his advantage that compromises the security of the system. Different from the malicious cloud server which only targets at breaking the confidentiality of outsourced data, a malicious user may launch different attacks to achieve different goals. We stress that the attacks towards plaintext deduplication services introduced in Sects. 4.2.1 and 4.2.2 can also be utilized to break the security of encrypted deduplication services.

4.3.3 Security Definition

As described in Sect. 4.3.1, MLE schemes essentially protect the data using the data itself. As such, MLE schemes cannot achieve semantic-security-style privacy in the spirit of [6]. Actually, if the target data M is drawn from a data space S of size $|S|$, given the ciphertext C of M produced using an MLE scheme, M can be recovered by an adversary who only tests $O(|S|)$ times.

To measure the security of MLE schemes, new security notions are defined by Bellare et al. [7], where the best possible privacy that MLE schemes can achieve is investigated. In particular, four different security notions are proposed, and we elaborate on them in the following.

The first one is called privacy against chosen-distribution attacks (short for PRV-CDA). It indicates that ciphertexts of two unpredictable messages should be indistinguishable. The second one is PRV $\$$ -CDA, which is stronger than PRV-CDA. PRV $\$$ -CDA indicates that the ciphertext of an unpredictable message should be indistinguishable from a (same-length) random string. The above two security notions are formalized to protect the confidentiality of data protected by MLE. The third notion is called tag consistency (short for TC), which has been discussed before. It indicates that it should be hard to generate a pair of plaintext and ciphertext (M, C) such that $Tag(C) = Tag(Enc(MLEKey(M), M))$ but $Dec(MLEKey(M), C)$ is a string different from M . The fourth one is a stronger version of TC, short for STC, which indicates that it should be hard to generate a pair of plaintext and ciphertext (M, C) such that $Tag(C) = Tag(Enc(MLEKey(M), M))$ but $Dec(MLEKey(M), C) = \perp$. STC ensures that an adversary cannot erase an honest user's data that would be outsourced to the cloud server.

4.4 Encrypted Deduplication Systems

The problem of deduplication over encrypted data (hereinafter, it is also called encrypted deduplication) is first defined by Douceur et al. [4]. In this work, the authors propose an elegant encryption scheme called convergent encryption (CE) to support encrypted deduplication. A CE scheme is constructed on a deterministic symmetric encryption algorithm (e.g., AES with CTR mode), but the encryption key is the hash value of the data to be encrypted rather than a randomly chosen value. By doing so, different users would produce the same ciphertext for the same data.

Following CE, many variants of CE are proposed and integrated into practical storage systems to support encrypted deduplication. These encrypted deduplication systems can be mainly divided into two parts: the first line of the work targets at enhancing the security, and another line of the work aims to take practical concerns into consideration to present expressive encrypted deduplication schemes for cloud storage systems.

4.4.1 Enhancement of Security

The security notions introduced in Sect. 4.3.3 essentially bear a common assumption that the data to be encrypted is unpredictable, i.e., the data has a high min-entropy. However, in reality, outsourced data are often predictable. For example, most outsourced data are well-formatted, and the format is always publicly known. As a consequence, an adversary, who has sufficient contextual information and wants to break the confidentiality of the data, just needs to “guess” a small part of the data. In this case, the success of the “brute-force” attack is not as hard as we might think. In addition, some data, e.g., very short documents, are inherently low-entropy. In reality, an adversary (e.g., a malicious cloud server) can encrypt all plaintext candidates by using MLE and identify the matched ciphertext to recover the content of target data M protected by MLE. Therefore, security against brute-force attacks should be considered in practice.

The vulnerability of MLE-based deduplication systems against brute-force attacks is first pointed out by Bellare et al. [8], who also propose a server-aided encrypted deduplication scheme, namely DupLESS, to mitigate such attacks. Specifically, an independent key server that holds a server-side secret key is introduced to assist users in generating MLE keys using the server-side secret. Users request an MLE key for each file to be outsourced from the key server. Furthermore, the interaction between the user and the key server is oblivious such that the user can generate the MLE key with the aid of the key server without leaking any information, which protects the data information against the key server and resists the brute-force attacks launched by the key server. The key technique to achieve this is an oblivious pseudorandom function (short for OPRF).

In [8], the authors attempt to design an easily deployed deduplication scheme that resists brute-force attacks and works transparently with existing cloud storage systems. To this end, DupLESS is designed to be compatible with any variant of MLE (i.e., any variant of MLE can be integrated into DupLESS to serve as the underlying encryption scheme) and also supports both server-side deduplication and user-side deduplication. In this section, we take CE as the underlying encryption scheme to show how DupLESS (as well as subsequent schemes) works for the sake of brevity.

In DupLESS, three entities are involved: users, the cloud server \mathcal{CS} , and the key server \mathcal{KS} . DupLESS is constructed on the RSA cryptosystem [9], and we assume the RSA exponent e and the RSA module N are determined with the security parameter ℓ . With e and N , the secret key d can be computed as $ed \equiv 1 \pmod{\phi(N)}$, where ϕ denotes the Euler’s totient function.

Each time a user \mathcal{U} wants to encrypt her/his data M , she/he first blinds M as $M' = r \cdot H(M)^e \pmod{N}$, where $r \in Z_N$ is randomly chosen by \mathcal{U} and $H : \{0, 1\}^* \rightarrow Z_N$ is a secure hash function. Then, \mathcal{U} sends M' to \mathcal{CS} . Upon receiving M' , \mathcal{CS} computes $\sigma'_M = M'^d \pmod{N}$ and sends σ'_M to \mathcal{U} , where d is the server-side secret. After receiving σ'_M , \mathcal{U} removes the blinding by calculating $\sigma_M = \sigma'_M \cdot r^{-1}$. To encrypt M , the encryption key here is $k = h(F(\sigma_M, M))$, where

$h : Z_N \rightarrow Key.Space$, $Key.Space$ is the key space of the underlying symmetric-key encryption scheme $E()$, and F is a pseudorandom function. Now, \mathcal{U} can encrypt M as $C = E(k, M)$ and outsources C to \mathcal{CS} .

The server-aided MLE utilized in DupLESS supports deduplication over encrypted data, since the MLE key is derived from the data itself and the server-side secret, and the server-side secret would not be changed for different users. Therefore, \mathcal{CS} is able to check duplicated data across all its users. On the other hand, since the MLE key is computed on two secrets (i.e., the one is the data, and another one is the server-side secret), as long as the key server remains inaccessible to attackers (e.g., the malicious cloud server), the security against brute-force attacks is achieved.

DupLESS is the first encrypted deduplication system with resistance against brute-force attacks, which provides a new framework and method to ensure the security of low-entropy data protected by MLE. However, it also remains some problems.

The first one is that the formal security treatment on the server-aided MLE is lacked. From DupLESS, we have no precise indication of what the server-aided MLE does or does not accomplish, and cannot clearly treat what precisely is the underlying security goal. To capture the security of MLE under brute-force attacks, a variant of PRV-CDA, which is called D-IND $\$$ -CPA, is proposed [10]. D-IND $\$$ -CPA refers to the security property that the ciphertext of a message (we would not require the message to be unpredictable here) is indistinguishable from a (same-length) random string.

The second one is that DupLESS actually bears a strong assumption that the key server is honest and reliable. Once the key server is compromised, i.e., the server-side secret is leaked, the security against brute-force attacks is broken. As a result, the key server becomes the single point of failure in the system, and DupLESS suffers from the single-point-of-failure problem. To address the single-point-of-failure problem, an effective way is to distribute the generation of MLE keys from the single key server to multiple ones using a threshold protocol [11]. Such a mechanism is first proposed by Duan [10] and is further enhanced in [2, 12]. We would give the technical details later and only review these works briefly to make it easy to understand. The main differences between the Duan's scheme [10] and those in [2, 12] are twofold as follows.

On the one hand, the Duan's scheme [10] relies on a trusted dealer to generate and distribute the server-side secret among all key servers, and the trusted dealer becomes a new single point of failure in the system. In contrast, the server-side secret in [2, 12] is generated by all key servers in a distributed way, which completely addresses the single-point-of-failure problem.

On the other hand, the Duan's scheme [10], following DupLESS, is constructed on the RSA cryptosystem [9], while the schemes in [2, 12] are built on the BLS cryptosystem [13, 14]. Compared with its RSA counterpart, the BLS signature is considerably shorter and more efficient on the key servers. In reality, key servers would serve multiple users simultaneously, and hence improving the key servers' computational efficiency could be economical and favorable.

As discussed before, the above mechanism, i.e., generating MLE keys by multiple key servers using a threshold protocol, surely makes performing brute-force attacks harder, since even if one or more (less than the threshold number of) key servers are compromised, the adversary who launches brute-force attacks cannot recover the data content. However, such a servers-aided MLE does not resolve the fundamental issue of trusting a specific group of key servers during the lifetime of protected data. Actually, it is feasible and practical for a sophisticated adversary to corrupt these key servers given enough time. Hence, for long-lived data, protection provided by servers-aided MLE could be insufficient [15, 16].

A straightforward way to remedy this problem is to periodically replace key servers by some new ones and let the new key servers re-share a new server-side secret. However, it makes the deduplication on the same file protected under different server-side secrets impossible.

In [17], the authors present an encrypted data deduplication scheme for cloud storage systems against compromised key servers and realize it in a system called DECKS. The security protection of DECKS is periodically renewed to free from the reliance on a specific group of key servers in a long period of time. Particularly, time in DECKS is divided into fixed intervals of predetermined length called epochs. In different epochs, the key servers are changed such that an adversary who compromises some key servers in the previous epoch cannot help in attacking in the current epoch. To ensure the deduplication on files outsourced in different epochs, a handoff mechanism is employed, where the server-side secret is transferred from some key servers that are not compromised in the current epoch to all key servers in the next epoch.

DECKS goes one step beyond the schemes in [2, 12], but its basic scheme (excluding the handoff mechanism) is the same as those of [2, 12]. With the introduction to DECKS, readers are easy to learn how the schemes in [2, 12] work. We provide the technical details of DECKS in the following.

DECKS consists of four algorithms: **Setup**, **MLEKeyGen**, **Deduplication**, and **Proactivation** (DECKS and the schemes in [2, 12] essentially share the same algorithms of **Setup**, **MLEKeyGen**, **Deduplication**). We assume that there are multiple users $\mathcal{U} = \{\mathcal{U}_1, \mathcal{U}_2, \dots\}$ (there is no upper bound of the number of users), n key servers $\{\mathcal{K}_1, \mathcal{K}_2, \dots, \mathcal{K}_n\}$, and a cloud server \mathcal{CS} in DECKS.

Setup

- With ℓ , public parameters $\{p, P, G, G_T, e, H, h, \tilde{h}, E(\cdot), Enc(\cdot), Sig(\cdot), t, n, \rho\}$ are determined, where $e : G \times G \rightarrow G_T$ is the bilinear pairing, P is a generator of G , $H : \{0, 1\}^* \rightarrow G$, $h : G \rightarrow \{0, 1\}^\ell$, $\tilde{h} : \{0, 1\}^* \rightarrow Z_p$ are secure hash functions, $E(k, F)$ is a symmetric encryption algorithm (CTR[AES]) to encrypt F using k , $Enc(epk, F)$ is a secure public-key encryption algorithm to encrypt F using epk , $Sig(ssk, F)$ is a secure signature algorithm [13] to sign F using ssk , ρ is the upper bound of MLE key requests made by a user in an epoch, t is a threshold, and n is the number of key servers.
- Each \mathcal{K}_i ($i \in [1, n]$) has a signing key pair (ssk_i, spk_i) and a public-key encryption key pair (esk_i, epk_i) . \mathcal{K}_i selects a nonce N_i , $a_{i,0} \in Z_p^*$, and a

$(t - 1)$ -degree polynomial $f_i(x) \in Z_p$, s.t. $f_i(0) = a_{i,0}$, where $f_i(x) = a_{i,0} + a_{i,1}x + \dots + a_{i,t-1}x^{t-1}$. \mathcal{K}_i publishes N_i .

- \mathcal{K}_i generates a session identity $SID^{(0)} = \{(\mathcal{K}_1, N_1), (\mathcal{K}_2, N_2), \dots, (\mathcal{K}_n, N_n)\}$, computes $a_{i,0}P$ and $a_{i,\gamma}P$ for $\gamma = 1, 2, \dots, t - 1$, and calculates $f_i(j)$ for $j = 1, 2, \dots, n; j \neq i$.
- For $\gamma = 1, 2, \dots, t - 1$ and $j = 1, 2, \dots, n; j \neq i$, \mathcal{K}_i computes

$$\theta_i = \{SID^{(0)}, a_{i,0}P, \{a_{i,\gamma}P\}, \{Enc(epk_j, f_i(j))\}\},$$

$$\Phi_i = Sig(ssk_i, \theta_i).$$

\mathcal{K}_i publishes $\{\theta_i, \Phi_i\}$.

- For $j = 1, 2, \dots, n; j \neq i$, \mathcal{K}_i obtains $\{\theta_j, \Phi_j\}$, accepts it if $SID^{(0)}$ and Φ_j are valid, and decrypts $Enc(epk_j, f_j(i))$ to get $f_j(i)$. \mathcal{K}_i checks $f_j(i)$ by verifying

$$f_j(i)P \stackrel{?}{=} \sum_{\gamma=0}^{t-1} i^\gamma \cdot a_{j,\gamma}P.$$

If the checking fails, \mathcal{K}_i rejects $f_j(i)$.

- \mathcal{K}_i computes its secret share $s_i = \sum_{j=1}^n f_j(i)$, the corresponding public share $Q_i = s_iP$. The server-side secret is $s = \sum_{i=1}^n a_{i,0}$ shared among all key servers, the public key $Q = sP = \sum_{i=1}^n a_{i,0}P$.
- For $\epsilon = 1, 2, \dots, m$, \mathcal{K}_i maintains ρ_ϵ to count up how many times \mathcal{U}_ϵ requests MLE keys in current epoch.

MLEKeyGen Without loss of generality, we assume that $\mathcal{U}_\epsilon (\epsilon \in \mathcal{U})$ is the first user who outsources a file M . For the subsequent users, they derive the MLE key on M as \mathcal{U}_ϵ does. \mathcal{U}_ϵ computes the MLE key on M as follows.

- \mathcal{U}_ϵ randomly selects $r \in Z_p^*$, computes $M' = rH(M)$, and sends F' to \mathcal{K}_i for $i = 1, 2, \dots, n$.
- \mathcal{K}_i verifies $\rho_\epsilon \leq \rho$ and aborts if the verification fails. Then, it computes a signature $\sigma_i = s_iM'$, sets $\rho_\epsilon ++$, and sends σ_i to \mathcal{U}_ϵ .
- After receiving σ_i , \mathcal{U}_ϵ checks its validity by verifying

$$e(\sigma_i, P) \stackrel{?}{=} e(M', Q_i).$$

If the checking fails, \mathcal{U}_ϵ rejects σ_i . After receiving t valid signatures $\{\sigma_{i_1}, \sigma_{i_2}, \dots, \sigma_{i_t}\}$ (we assume that the indexes of these signatures form a set $T = \{i_1, i_2, \dots, i_t\}$ with an ascending order and $|T| = t$), \mathcal{U}_ϵ computes

$$w_{i_k} = \prod_{\substack{i_1 \leq \eta \leq i_t \\ \eta \in T, \eta \neq i_k}} \frac{\eta}{\eta - i_k}$$

$$\sigma = r^{-1} \sum_{k=1}^t w_{i_k} \sigma_{i_k}.$$

\mathcal{U}_ϵ checks whether $e(\sigma, P) = e(H(M), Q)$ holds, she/he rejects σ if the checking fails.

- \mathcal{U}_ϵ computes MLE key $mk_M = h(\sigma)$.

Deduplication With mk_M , \mathcal{U}_ϵ encrypts M as $C_M = E(mk_M, F)$ and outsources C_M to \mathcal{CS} .

For \mathcal{CS} , some methods can be utilized to check duplicates. One of the most efficient ways is to check the hash value of the ciphertext. Specifically, \mathcal{CS} computes $\tau_M = \mathfrak{h}(C_M)$ and maintains $\{C_M, \tau_M\}$ locally. For a subsequent user who uploads a ciphertext C' , \mathcal{CS} verifies $\mathfrak{h}(C') \stackrel{?}{=} \tau_M$. If the verification passes, \mathcal{CS} performs deduplication; Otherwise, \mathcal{CS} maintains C' locally.

Proactivation At the end of an epoch, the key servers are replaced by n new key servers. For the sake of brevity, we assume that the key servers in the χ -th epoch form a committee $\mathcal{K}^{(\chi)} = \{\mathcal{K}_1^{(\chi)}, \mathcal{K}_2^{(\chi)}, \dots, \mathcal{K}_n^{(\chi)}\}$ and the new key servers in the next epoch, i.e., the $(\chi + 1)$ -th epoch, form a committee $\mathcal{K}^{(\chi+1)} = \{\mathcal{K}_1^{(\chi+1)}, \mathcal{K}_2^{(\chi+1)}, \dots, \mathcal{K}_n^{(\chi+1)}\}$, where the secret share of $\mathcal{K}_i^{(\chi)}$ ($i = 1, 2, \dots, n$) is denoted by $s_i^{(\chi)}$. The replacement is achieved by a handoff process where the server-side secret s is redistributed among $\mathcal{K}^{(\chi+1)}$. This process is shown in the following (unless specified otherwise hereinafter, $k = 1, 2, \dots, t$ and $j = 1, 2, \dots, n$).

- t honest and reliable key servers $\{\mathcal{K}_{i_1}^{(\chi)}, \mathcal{K}_{i_2}^{(\chi)}, \dots, \mathcal{K}_{i_t}^{(\chi)}\}$ are selected. Their indexes form a set $T^{(\chi)} = \{i_1, \dots, i_t\}$.
- $\mathcal{K}_{i_k}^{(\chi)}$ generates a nonce $N_{i_k}^{(\chi)}$ and publishes it. $\mathcal{K}_j^{(\chi+1)}$ generates a nonce $N_j^{(\chi+1)}$ and publishes it.
- $\mathcal{K}_{i_k}^{(\chi)}$ randomly chooses $b_{i_k,1}, b_{i_k,2}, \dots, b_{i_k,t-1} \in \mathbb{Z}_p$ and generates

$$g_{i_k}^{(\chi)}(x) = s_{i_k}^{(\chi)} + b_{i_k,1}x + b_{i_k,2}x^2 + \dots + b_{i_k,t-1}x^{t-1}. \quad (4.1)$$

- $\mathcal{K}_{i_k}^{(\chi)}$ generates a session identity

$$SID^{(\chi)} = \{ \{(\mathcal{K}_{i_k}^{(\chi)}, N_{i_k}^{(\chi)})\}, \{(\mathcal{K}_j^{(\chi+1)}, N_j^{(\chi+1)})\} \},$$

computes $b_{i_k,1}P, b_{i_k,2}P, \dots, b_{i_k,t-1}P, s_{i_k,j}^{(\chi)} = g_{i_k}^{(\chi)}(j)$.

- $\mathcal{K}_{i_k}^{(\chi)}$ computes

$$\begin{aligned} \theta_{i_k}^{(\chi)} &= \left\{ SID^{(\chi)}, \{b_{i_k,\gamma}P\}_{\gamma=1,\dots,t-1}, \left\{ Enc \left(epk_j^{(\chi+1)}, s_{i_k,j}^{(\chi)} \right) \right\} \right\}, \\ \Phi_{i_k}^{(\chi)} &= Sig \left(ssk_{i_k}^{(\chi)}, \theta_{i_k}^{(\chi)} \right). \end{aligned}$$

$\mathcal{K}_{i_k}^{(\chi)}$ publishes $\{\theta_{i_k}^{(\chi)}, \Phi_{i_k}^{(\chi)}\}$. Here, $Q_{i_k}^{(\chi)} = s_{i_k}^{(\chi)}P$ is the public share and has been published.

- $\mathcal{K}_j^{(\chi+1)}$ obtains $\{\theta_{i_k}^{(\chi)}, \Phi_{i_k}^{(\chi)}\}$, accepts it if $SID^{(\chi)}$ and $\Phi_{i_k}^{(\chi)}$ are valid, and decrypts $Enc(epk_j^{(\chi+1)}, s_{i_k, j}^{(\chi)})$ to get $s_{i_k, j}^{(\chi)}$.
- $\mathcal{K}_j^{(\chi+1)}$ computes the Lagrange coefficients $w_{i_k} = \prod_{\substack{i_1 \leq \eta \leq i_t \\ \eta \in T^{(\chi)}, \eta \neq i_k}} \frac{\eta}{\eta - i_k}$, and verifies

$$s_{i_k, j}^{(\chi)} P \stackrel{?}{=} s_{i_k}^{(\chi)} P + \sum_{\gamma=1}^{t-1} j^\gamma \cdot b_{i_k, \gamma} P, \quad (4.2)$$

$$Q \stackrel{?}{=} \sum_{k=1}^t w_{i_k} Q_{i_k}^{(\chi)}. \quad (4.3)$$

If the verification fails, $\mathcal{K}_j^{(\chi+1)}$ aborts; Otherwise, it sends an ‘‘Accept’’ message to other key servers.

- After receiving ‘‘Accept’’ messages from all other key servers, $\mathcal{K}_j^{(\chi+1)}$ computes its secret share $s_j^{(\chi+1)}$ as

$$s_j^{(\chi+1)} = \sum_{k=1}^t w_{i_k} s_{i_k, j}^{(\chi)}. \quad (4.4)$$

The corresponding public share is $Q_j^{(\chi+1)} = s_j^{(\chi+1)} P$.

The server-side secret s would not be changed after **Proactivation**, which ensures that deduplication can work on data outsourced in different epochs. The proof is provided as follows. For the sake of brevity, we assume that the selected honest key servers are $\{\mathcal{K}_1^{(\chi)}, \mathcal{K}_2^{(\chi)}, \dots, \mathcal{K}_t^{(\chi)}\}$, where $\mathcal{K}_k^{(\chi)}$'s secret share is $s_k^{(\chi)}$. Note that s has the form:

$$s = \sum_{k=1}^t w_k s_k^{(\chi)} = \sum_{\zeta=1}^t w'_\zeta s_\zeta^{(\chi+1)}, \quad (4.5)$$

where both w_k and w'_ζ are Lagrange coefficients. if $s_k^{(\chi)}$ and $s_\zeta^{(\chi+1)}$ are valid, then

$$\begin{aligned} s &= \sum_{k=1}^t w_k s_k^{(\chi)} \\ &= \sum_{k=1}^t \left(w_k \sum_{\zeta=1}^t w'_\zeta s_{k, \zeta}^{(\chi)} \right) \end{aligned}$$

$$\begin{aligned}
&= \sum_{\zeta=1}^t \sum_{k=1}^t w'_\zeta w_k s_{k,\zeta}^{(\chi)} \\
&= \sum_{\zeta=1}^t w'_\zeta s_\zeta^{(\chi+1)}.
\end{aligned}$$

DECKS also supports that committees in different epochs would intersect, although the above description only shows the case of disjoint committees. Furthermore, the total number of key servers n and the threshold number t can also be changed in different epochs to satisfy different levels of security. This is achieved by choosing a corresponding degree of the polynomial in Eq. (4.1).

Another work [18] to resolve the fundamental issue of trusting the key server(s) is somewhat straightforward: as the employment of key server(s) would cause the trust issue, the users who have outsourced a file M assist the subsequent users who attempt to upload the same file in encrypting M . In [18], if a user \mathcal{U} wants to outsource M using MLE and M has been outsourced by other users \mathbf{U} , i.e., deduplication would occur, \mathcal{U} would interact with a subset of users of \mathbf{U} to encrypt M . If \mathcal{U} is a brute-force adversary, he only obtains a random key to encrypt M , and if \mathcal{U} is honest, she/he will compute an MLE key from the interaction, which enables her/him to benefit from deduplication. However, the scheme in [18] bears a strong assumption that a number of users in \mathbf{U} should keep online when \mathcal{U} outsources M . This actually requires users in \mathbf{U} to perform too many operations to use their data and to enjoy the deduplication service.

Another line of work focusing on enhancing the security of encrypted deduplication schemes is to consider a special type of data protected by MLE. In the above discussion, we only introduce two types of data to be encrypted: the one has a high min-entropy which inherently resists brute-force attacks, and the other one has a low min-entropy which inherently vulnerable to brute-force attacks. However, in reality, a special type of data called lock-dependent message exists, where the plaintext distributions that may depend on the public parameters of the schemes. Examples of the lock-dependent message including those that the tags generated using MLE share a particular property, such as that they all start with a zero bit, or that the first bit of the tag reveals the first bit of the message. With such a property, the adversary is able to extract additional information from the ciphertext protected under the four types of MLE introduced before to violate the privacy of data owners. However, all the MLE schemes [19, 20] that well protects lock-dependent message against adversaries incurs prohibitive costs on deduplication, since heavy cryptographic operations, e.g., zero-knowledge proofs, are leveraged.

All above schemes do not explicitly target at resistance against other attacks introduced in Sect. 4.2, such as traffic analysis [21], side-channel attacks [22], and leakage of the hashed plaintext [23], that may be utilized by adversaries to violate users' privacy and cloud server's security. However, these schemes are well compatible with orthogonal techniques [2, 3] to thwart these attacks.

It is worth stress that POW also serves an important component in encrypted deduplication. Recall that the primary motivation of introducing POW is to resist adversaries who do not own the data but attempt to “steal” the data from the cloud server leveraging the deduplication service. However, with the use of MLE, encrypted deduplication schemes are inherently resistant to such a stealing attack. This does not mean that POW is useless and would not be needed in encrypted deduplication systems. As a matter of fact, the attacks that utilize the deduplication service to construct an unintended content distribution network still work in encrypted deduplication systems, and even they are harder to be detected compared with those in the case of plaintext. With the construction of MLE, designing POW schemes for encrypted deduplication is not so challenging. Such a POW scheme can be directly derived from the public verification scheme proposed by Shacham et al. [24, 25] via a role reversal with the modification that the secret key utilized to compute σ_i is the hash value of the data block m_i , rather than a randomly chosen one. Encrypted deduplication systems with the integration of POW have been investigated by different researchers.

4.4.2 *Practical Concern*

The encrypted deduplication schemes we have studied so far just consider the data protected by MLE as an element in Z_p . Although some of them consider that the data are low-entropy, the inherent characteristics of target data and storage systems are not considered. With the integration of these inherent characteristics, additional features can be achieved.

The first work that considers the inherent characteristics of target data and storage systems is proposed by Stanek et al. [26, 27]. In this work, data to be protected by MLE are generally divided into two categories according to the popularity. The one is called popular data, which means that the number of users who outsource the data is more than a threshold (this threshold depends on the underlying system); The other one is called unpopular data, which means that the number of users who outsource the data is less than the threshold. The key observation of [26, 27] is that different data requires different levels of security protection. For example, data shared by lots of users, e.g., a popular song or video, arguably requires less protection than individual documents which might be only owned by the data owners themselves. As such, deduplication should only occur when the data are popular. With this observation, there is a state for each outsourced file to indicate its popularity. This state is dynamically updated with data outsourcing. To enable the encrypted deduplication with “fine-grained security-to-efficiency tradeoff,” two trusted entities are introduced as follows:

- Identity provider (IdP): IdP is introduced to identify users when a user joins, which protects the system from Sybil attacks.

- Index repository service (IRS): IRS is introduced to provide secure indexation for unpopular files.

In reality, an adversary may control multiple users. The goal of the adversary inherits that in encrypted deduplication schemes introduced before, i.e., breaking the confidentiality of outsourced data. Let n_A be the upper bound of the number of users that can be controlled by the adversary, and p_{lim} be a system-wide popularity limit that represents the smallest number of distinct and legitimate users that need to upload a given file M for that file to be declared popular. The threshold used to decide the popularity t should be set to be $t \geq p_{lim} + n_A$, which ensures both the security and efficiency.

The data M in [26, 27] is protected by a two-layer encryption algorithm: The inner layer is obtained through MLE that generates identical ciphertext at each invocation; The outer layer is obtained through a semantically secure encryption algorithm. To design such a two-layer encryption algorithm, a cryptosystem, called the convergent threshold cryptosystem, is introduced. The threshold and message-locked nature of the convergent threshold cryptosystem make it suitable for the secure deduplication scheme. Specifically, the number of users who outsource M is recorded by IRS; Once this number is less than t (i.e., M is an unpopular data), M is protected by two-layer encryption which is semantically secure; Once this number achieves t (i.e., M become popular data), the outer layer of encryption is decrypted, and M is just protected by MLE, which supports encrypted deduplication.

The work in [26, 27] has demonstrated that designing encrypted deduplication schemes with the integration of the inherent characteristics of target data and storage systems is very promising and may significantly enhance the systems in terms of security, efficiency, and functionality.

One of the most important motivations to employ cloud storage services is to manage big data that has a large volume. However, the encrypted deduplication schemes introduced so far suffer from either low efficiency or low space savings for large files in cloud storage systems.

Particularly, in the above schemes, when an entity, say Alice, outsources a large file M to the cloud server CS , where M is protected by MLE. Subsequently, another entity, say Bob, wants to outsource the same file M to CS using the same MLE scheme. CS would perform deduplication and only store a single copy of the ciphertext of M , denoted by C locally. Later, Bob may only append some new data to M and outsources the updated file (denoted by M') to CS . Since M is modified to M' , the ciphertext C is also changed to C' . CS needs to maintain both C and C' locally, which is very cumbersome and costly.

To address the above problem, block-level deduplication can be utilized. M is first split into multiple blocks $M = \{M_1, M_2, \dots, M_n\}$, and each block M_i is encrypted using MLE. As a result, if Bob wants to update M , CS only needs to additionally maintain the newly generated blocks, which achieves high space savings.

The above block-level deduplication scheme is directly extended from existing MLE schemes. However, it is confronted with the key management problem: each

block corresponds to an MLE key, and both Alice and Bob need to store all these MLE keys locally. When the number of blocks is large, maintaining these MLE keys is a heavy burden for Alice and Bob. A trivial solution is to encrypt all blocks' MLE keys using a master key and outsource these ciphertexts together with the encrypted data to CS . Whereas, this causes a significant extension on the storage costs.

To address this problem, Chen et al. propose a dual-level user-side encrypted deduplication scheme, called BL-MLE [28]. The key technique underlying BL-MLE is threefold: Each block is further split into multiple sectors which reduces the number of blocks' MLE keys significantly (This is essentially the same as those in public data integrity verification schemes discussed in Sect. 3.3); Each block MLE key is encapsulated into the block identifier which is used to identify the redundant blocks; The master key is derived from the entire data M and is protected under public-key cryptosystem. In addition, BL-MLE employs the user-side deduplication strategy, where POW is also leveraged to ensure security.

Based on Chen et al.'s work, a variant of MLE is formalized as updatable block-level MLE (UMLE) [29, 30]. The security is formally defined, and schemes with high efficiency are proposed. However, we stress that UMLE is only used to perform deduplication over large files. Compared with MLE, UMLE achieves higher space savings but sacrifices the computational efficiency and makes the system more complex. Furthermore, in reality, the block size should be carefully selected; Otherwise, it may fail to check the duplicate blocks.

Another line of work [31, 32] targets at designing transparent deduplication schemes. The motivation of these works is that encrypted deduplication schemes surely increase the profitability of the cloud service provider, but do not enable users to directly benefit from the savings of deduplication over their data. The service charge of using cloud storage service is calculated by the volume of the outsourced data, without consideration of the deduplication level (i.e., how many users that outsource the same data to the cloud server) of the data. The deduplication level for one data file is formalized as the “deduplication pattern.”

To address the above problem, a straightforward solution is to require the cloud server to periodically release the deduplication pattern for each outsourced file. However, the cloud service provider may forge a deduplication pattern and only release this forged one to users for profits. For example, if the service charge of outsourcing a data file M is 100 dollars. When the number of users that outsource M to the cloud server is 100, the deduplication pattern of M is 100, and each user only needs to pay the cloud service provider 1 dollar for the data outsourcing. To gain more service charge, the cloud service provider may claim a forged deduplication pattern of 50, and now each user needs to pay the cloud service provider 2 dollars. After that, the cloud service provider can gain additional 100 dollars from users.

To resist such a misbehaved cloud service provider, existing schemes [31, 32] introduce an independent entity, called a gateway, to periodically verify the deduplication pattern of each file. To improve verification efficiency, the gateway leverages a sampling verification strategy that randomly chooses a subset of all outsourced data to verify the deduplication pattern of the data in the subset. To resist the collusion between the gateway and the cloud server to forge deduplication pattern,

the key technique used here is to utilize a public blockchain to construct a time-dependent random source to ensure the randomness of the subset while ensuring the public verifiability, which follows the idea introduced in Sect. 3.4.2.2. The proof of deduplication pattern is based on the cryptographic accumulators [33–35] which can be used to verify whether a given element belongs to a set.

4.4.3 Other Works

In addition to the above works, the literature features some proposals for integrating encrypted deduplication and public verification (e.g., [36, 37]) in cloud storage systems. Actually, proofs of storage, MLE and proofs of ownership can be integrated into one system to ensure the confidentiality and integrity of outsourced data while achieving space savings from deduplication. Moreover, the idea of designing secure encrypted data deduplication schemes can also be used in other systems-security applications. In particular, a notion of “public-key encryption with keyword search” is proposed in [38] for the problem of retrieving target data using a keyword from the entire data set protected under a public-key encryption scheme, which will be introduced in the next chapter.

4.5 When Secure Deduplication Meets eHealth: A Case Study

Compared with traditional paper-based systems, electronic health (eHealth) systems provide a more efficient, less error-prone, and more flexible service for both doctors and patients. As such, eHealth systems have replaced paper-based medical systems in recent years and become a central hub of hospital systems. For example, in China, if a hospital’s eHealth system is not qualified to provide services, consultation fees of any patient visiting the hospital cannot be reimbursed.

In fact, eHealth systems are data-intensive [39]. Typically, eHealth systems allow doctors to generate and access their patients’ electronic medical records (EMRs), such as prescriptions. With a numerous EMRs in eHealth systems’ generation, medical institutions (e.g., clinics and hospitals) who store large data sets locally incur substantial hardware, software, and personnel costs involved in deploying and maintaining applications in practice [40]. Furthermore, the local EMRs store makes no contribution to the judgment and dispute resolution in medical malpractice. At this point, outsourcing EMRs to cloud servers is a practical choice.

Generally, the storage server needs to store the outsourced EMRs for a prolonged period of time to satisfy several government regulations or hospital requirements on EMRs archiving, while the volume of EMRs generated from eHealth systems grows over time, which causes the sustained growth in the costs of store EMRs. Actually,

the storage costs can be reduced significantly after deduplication, where the storage server checks duplicate EMRs and deletes the redundant ones. According to our analysis, performing deduplication of EMRs can save the storage costs by more than 65% in cloud-assisted eHealth systems, which will be elaborated in Sect. 4.5.3. However, from the perspective of data owners, including both medical institutions and patients, the content of EMRs should not be leaked for security reasons [41]. Therefore, the privacy protection of the EMRs' content against anyone who does not own the EMRs should be guaranteed. This can be achieved by conventional encryption, but it makes deduplication impossible.

MLE is a cryptographic primitive that supports encrypted data deduplication. In cloud-assisted eHealth systems, when consulting a doctor, the patient delegates his/her doctor to generate EMRs, the doctor encrypts the generated EMRs by using MLE, outsources the ciphertexts to the cloud storage, and sends the MLE keys to the patient [42]. The storage server checks duplicate (encrypted) EMRs and stores only a single copy of them to reduce the storage costs. However, EMRs are inherently low entropy. For example, a list of most existing antibiotics can be found in [43], the list only involves about 100 items. Actually, most EMR candidates can be enumerated quickly by adversaries. This problem is further exacerbated by the fact that an adversary has sufficient contextual information (e.g., patients' symptoms). As a consequence, the outsourced EMRs protected by MLE is vulnerable to brute-force ciphertext recovery.

Although server(s)-aided encrypted deduplication schemes can be employed to mitigate the brute-force attack, two problems still exist.

- As discussed before, the patients need to delegate his/her doctor to generate EMRs, and are required to store MLE keys locally, therefore, how to make secure delegation and maintain MLE keys well on the patient side should be considered;
- As the number of EMR items is enormous, checking duplicate EMRs requires the storage server to scan the entire EMR set and check the EMR items one by one. Consequently, employing existing schemes to check duplicate EMRs incurs a considerable delay and becomes a bottleneck in applications.

To address the above problems, HealthDep, a secure and efficient encrypted deduplication scheme for cloud-based eHealth systems, is proposed in [2]. In this section, we study HealthDep and explore the potentials of integrating the encrypted deduplication technique into eHealth systems.

4.5.1 *Cloud-Based eHealth Systems*

A model of the cloud-based eHealth system is shown in Fig. 4.6. There are five different entities in it: patients, hospital, doctor, key servers, and storage server.

The hospital involves multiple departments, such as Cardiology, Gastroenterology, Orthopedics, and so on. Each doctor attaches herself/himself to a department. Each patient has a smartphone with system-wide Trusted Execution Environments

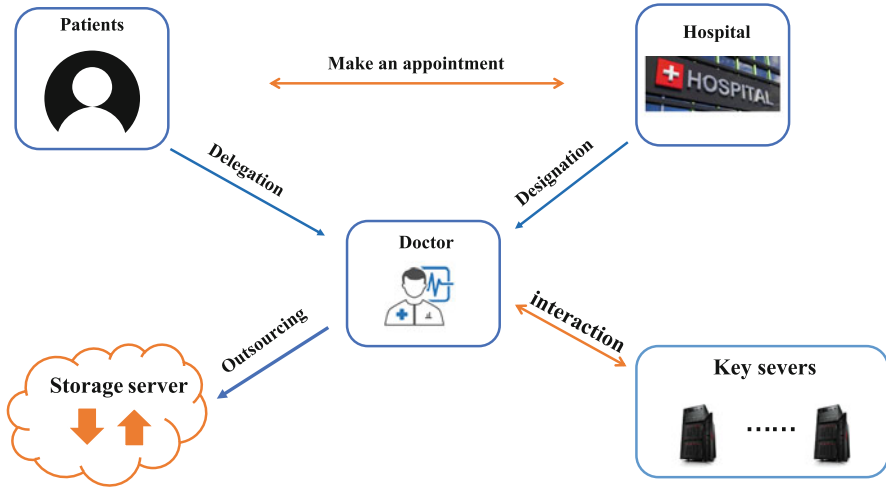


Fig. 4.6 System model of HealthDep

(TEEs) (e.g., ARM TrustZone [44]). Generally, the patients, hospital, and doctor are cloud clients; the storage server is the cloud server and is subject to cloud service providers.

The procedure when a patient consults a doctor in the eHealth system is described as follows. First, the patient registers with a hospital, and the hospital determines that the patient is subject to which department. Then the hospital designates a doctor for diagnosing, and the patient makes an appointment with the hospital to obtain the diagnosing information (e.g., time and place). At the corresponding time, the patient delegates to the doctor and is diagnosed and treated. Then the doctor generates the EMRs for the patient, performs a server-aided MLE to encrypt the EHRs, and outsources the ciphertexts to the storage server. Finally, the storage server checks duplicate EMRs across all the patients and stores only a single copy of redundant EMRs.

4.5.2 Adversary Model and Security Goals

In the adversary model, threats from two different angles are considered: internal adversaries and external adversaries. We also assume that both internal and external adversaries are able to access the outsourced data.

Internal Adversaries

- Compromised key servers. The adversary may control the key servers, extract the secrets from them, and record their interactive message to retrieve the EHRs from the ciphertexts generated by the doctor. Here, we assume the cost of

compromising a threshold number of key servers is higher than the value of the EHRs that it protects.

- Rational storage server. The storage server is a rational entity. By rational, the storage server will only deviate from the scheme if such a strategy increases their profit in the system. Furthermore, the storage server (including malicious insiders working at the cloud service provider) always attempts to violate the confidentiality of the outsourced EHRs to retrieve the patients' privacy.
- Semi-trusted doctor. The doctor is an honest entity during the diagnosing period. However, the doctor may perform two attacks:
 1. Outsourcing forged EMRs. After the diagnosing period, the doctor may outsource forged EMRs to the storage server to conceal his mistake in medical malpractice.
 2. Violating the confidentiality of the EMRs outsourced by other doctors. The only way to do this is that the doctor performs online brute-force attacks, where he requests the MLE keys of EMRs' candidates from the key servers, and then performs brute-force ciphertext recovery.

External Adversaries

- Hardware adversaries. The hardware adversary is the strongest adversary whose target is to break the security of the patient's device. We assume that the hardware adversary cannot physically access the patients' device.
- Online adversaries. Such an adversary can control the communications between the storage server and other entities. He is able to eavesdrop on the interaction messages between the storage server and the doctor, and tamper with them. Moreover, the active adversaries also can impersonate the doctor to forge EMRs and outsource them.

Since EHRs are inherently low-entropy, all the above adversaries can predicate them. HealthDep does not consider denial-of-service attacks. The hospital has a secure space to store secrets.

4.5.3 Analysis of EMRs in Actual eHealth Systems

We first analyze the inherent characteristics of EMRs from actual eHealth systems.

We show three prescriptions in Figs. 4.7, 4.8, and 4.9, where **Rp.**, *Sig*, *po*, *qd*, *bid*, and *qn* come from Latin and denote "get the medicine," "the usage and dosage," "oral," "four times a day," "twice a day," and "every night," respectively.

Figure 4.7 shows the prescription of a patient who is diagnosed with coronary heart disease and stable angina pectoris. Figure 4.8 shows the prescription of another patient who is diagnosed with hypertension. These two prescriptions are generated by the same doctor from the Department of Cardiology. We can see that although these two patients are diagnosed with different diseases, the common medicines with the same usage and dosage are used, that is, "Aspirin Enteric-coated

Fig. 4.7 The prescription of patient with coronary heart disease

XXX Hospital' s Prescription
Prescription ID: XXX

Patient Information: XXX
Clinical Diagnosis:
 Coronary Heart Disease;
 Stable Angina Pectoris.

RP:
 Aspirin Enteric-coated Tablets,
 0.1g * 30 pills; Sig: 0.1 g. po. qd.

Isosorbide Mononitrate Tablets,
 20mg * 48 pills; Sig: 20 mg. po. qn.

Metoprolol Tartrate Tablets,
 25mg * 20 pills; Sig: 25 mg. po. bid.

Nifedipine Sustained-release Tablets
 20mg * 50 pills; Sig: 20 mg. po. bid.

Doctor' s Signature: XXX.
 Doctor' s ID: XXX.

Tablets,” “Metoprolol Tartrate Tablets,” and “Nifedipine Sustained-release Tablets.” By comparison, Fig. 4.9 shows the prescription of a patient, who is diagnosed with gastric ulcer, and the prescription is generated by a doctor from the Department of Gastroenterology. There is no same medicine between the prescription described in Fig. 4.9 and those described in Figs. 4.7 and 4.8, since these prescriptions are generated by different doctors from different departments. Furthermore, each item should be coded before storing it. In China, each medicine corresponds with a *YP* code formulated by the National Health and Family Planning Commission of the People’s Republic of China. We would omit the coding processing hereinafter.

From the above three figures, we also observe that a portion of EMRs would be duplicated, which are contained within the blue rectangle, while a portion of them (contained within the red rectangle) would not. For example, each prescription is first divided into multiple blocks before encrypting, as Fig. 4.10 shows (since the prescription ID, and the doctor’s ID and signature do not require encryption, and are not highlighted in the figure). Actually, different patients would not have the same prescription ID and patient information. Therefore, to further improve the efficiency that checks duplicate EMRs, the storage server only needs to perform deduplication on $m_1 \sim m_4$ shown in Fig. 4.10. Furthermore, the patient information and the clinical diagnosis are the most sensitive information, and these two parts

Fig. 4.8 The prescription of patient with hypertension

XXX Hospital' s Prescription	
Prescription ID: XXX	
Patient Information: XXX	
Clinical Diagnosis: Hypertension.	
RP:	
Aspirin Enteric-coated Tablets, 0.1g * 30 pills; Sig: 0.1 g. po. qd.	
Metoprolol Tartrate Tablets, 25mg * 20 pills; Sig: 25 mg. po. bid.	
Nifedipine Sustained-release Tablets 20mg * 50 pills; Sig: 20 mg. po. bid.	
Doctor' s Signature: XXX.	
Doctor' s ID: XXX.	

(m_0 in Fig. 4.10) should be protected by a symmetric-key encryption with semantic security.

We first perform deduplication on 200 prescriptions from an actual eHealth system. These prescriptions are selected randomly from 10,000 prescriptions generated by doctors from the Department of Cardiology during 2013–2017. We show the results in Table 4.1, which demonstrates that performing deduplication can save storage costs by more than 50% in this case. We further analyze more prescriptions and observe that performing deduplication is able to save the storage costs about 66% in the case of 500 prescriptions and more.

4.5.4 Study of HealthDep

Since a patient always consults a doctor without heavy luggage, it is impractical to require patients to be well equipped in eHealth systems [45]. As most persons already have equipped with smartphones, deployment of the mobile device to make delegation and store MLE keys is practical. To ensure the security, the key technique used in HealthDep is system-wide TEEs [46], such as ARM TrustZone [44]. Prior to making an appointment with the hospital and see a doctor, assuming

Fig. 4.9 The prescription of patient with gastric ulcer

XXX Hospital' s Prescription
Prescription ID: XXX

Patient Information: XXX
Clinical Diagnosis:
Gastric Ulcer.

RP: Rabeprazole Sodium Enteric-coated Capsules, 10mg * 21 pills; Sig: 10 mg. po. bid.
Colloidal Bismuth Pectin Capsules, 50mg * 48 pills; Sig: 100 mg. po. qd.
Clarithromycin Tablets, 0.25g * 40 pills; Sig: 0.5 g. po. bid.
Tinidazole Tablets, 0.2g * 40 pills; Sig: 0.4 g. po. bid.

Doctor' s Signature: XXX.
Doctor' s ID: XXX.

Fig. 4.10 Example of processing prescription

XXX Hospital' s Prescription
Prescription ID: XXX

Patient Information: XXX
Clinical Diagnosis:
Coronary Heart Disease;
Stable Angina Pectoris.

RP:

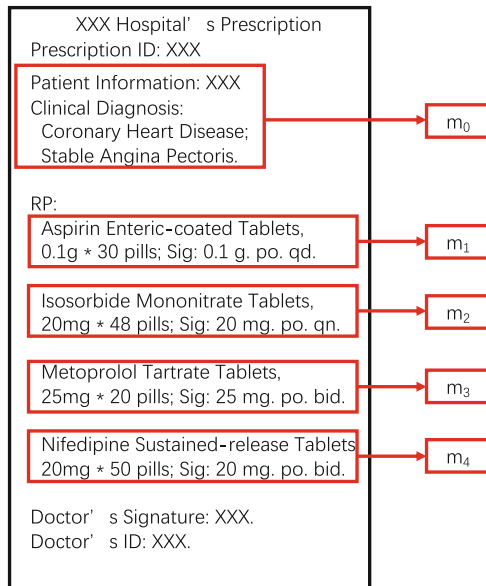
Aspirin Enteric-coated Tablets,
0.1g * 30 pills; Sig: 0.1 g. po. qd.

Isosorbide Mononitrate Tablets,
20mg * 48 pills; Sig: 20 mg. po. qn.

Metoprolol Tartrate Tablets,
25mg * 20 pills; Sig: 25 mg. po. bid.

Nifedipine Sustained-release Tablets
20mg * 50 pills; Sig: 20 mg. po. bid.

Doctor' s Signature: XXX.
Doctor' s ID: XXX.



that each patient has already installed two applications provided by the hospital on his device: a companion application running in the normal world and a trusted application running in the secure world. HealthDep also assumes that each patient has completed registration, that is, the hospital has stored the device certificate of the

Table 4.1 Savings of storage costs after deduplication

Prescriptions number	100	150	200	500
Savings of storage costs	38.8%	43.3%	50.06%	66%

patient’s smartphone. HealthDep makes use of the International Mobile Equipment Identity (IMEI) of the patient’s device as the identity, since the IMEI is written by the device manufacturer and stored in a read-only memory on the device. This binds the patient to his/her device TEE.

Each patient first obtains a treatment key from the hospital and seals the treatment key in the secure world of his/her smartphone TEE. All the subsequent interactive messages between the patient and the hospital are protected under the treatment key. The patient first makes an appointment with the hospital and receives his/her diagnosing information. At the treatment time, the patient delegates to the doctor, and the doctor generates the EMRs. Next, the doctor divides the EMRs into two parts: the one involves the individual information, such as patient information and clinical diagnosis, which is the most sensitive data; the other one involves the medical records, such as medicines and their usage and dosage, which would be duplicate and can be deduplicated to reduce the storage costs. Then the doctor encrypts the first part (e.g., the content contained within the red rectangle described in Fig. 4.7) by using conventional encryption (e.g., AES), encrypts the later part (e.g., the content contained within the blue rectangle shown in Fig. 4.7) by using the server-aided MLE, outsources the ciphertexts as well as some auxiliary information corresponding himself/herself to the storage server, and sends the keys to the patient. The storage server first checks the validity of the patient’s delegation to authenticate the doctor. If the checking passes, it accepts the outsourced ciphertext. Finally, the storage server determines whether performing deduplication by the auxiliary information and performs deduplication on the ciphertexts protected under the server-aided MLE. In actual eHealth systems, the size of EMRs’ first part is always small, and therefore can be represented by one data block. HealthDep assumes that the first block is the data that would not be duplicated.

To enable HealthDep to resist offline brute-force ciphertext recovery without the assumption that the key server is fully trusted, multiple key servers are introduced to assist the doctor in generating the MLE keys and HealthDep employs a (t, n) -threshold blind signature scheme [47] between the doctor and the key servers. The key observation here is that compromising t key servers is much harder than compromising a single one, as discussed before.

In HealthDep, to thwart online brute-force attacks, in which attackers (curious doctors) impersonate a valid doctor to request MLE keys and further violate the confidentiality of the EMRs, it requires that the key servers limit the number of MLE keys request for each doctor in an epoch, as [8] does. A bound ρ is pre-defined at the initialization phase; Each key server keeps track of the total number of the queries made by each doctor and stops responding after ρ is reached.

In practice, each patient’s treatment key can be considered as a long-term one. Thus, to improve efficiency, HealthDep supports that each patient needs to perform

the registration only once when he/she first visits the hospital in person and reuses the treatment key.

In actual eHealth systems, for a specific department, the number of common medicines is determined. Therefore, when obtaining a new MLE key from the key servers, the doctor is able to maintain this MLE key well and reuse it in the subsequent treatments. This can reduce the communication and computation overhead at the expense of slight storage costs.

HealthDep also can be used to a scenario where multiple medicine institutions outsource their patients' EMRs to the same cloud storage, since our observation still holds in this condition.

HealthDep is the first secure and efficient encrypted EMRs deduplication scheme for cloud-based eHealth systems. It not only integrates the inherent characteristics of EMRs to improve the efficiency of performing deduplication, but also employs servers-aided MLE to enhance the security.

It has demonstrated the promising of applying advanced encrypted deduplication techniques in reality, and also serves as a key reference to inspire subsequent researchers to design secure and efficient cryptographic schemes to enhancing the cloud storage systems in terms of security, efficiency, and functionality.

4.6 Summary and Further Reading

In this chapter, we have introduced the data deduplication technique for cloud storage systems. We have provided a comprehensive overview of existing data deduplication schemes and analyzed their pros and cons, and also conducted a comparison between them. Finally, we have studied the latest advances in encrypted deduplication techniques and discussed their potentials to enhance cloud storage services.

There are also some survey papers to introduce the data deduplication technique from different aspects, such as [1, 48]. There is also a survey paper [49] to only focus on the secure deduplication technique.

References

1. Meyer DT, Bolosky WJ (2012) A study of practical deduplication. *ACM Trans Storage* 7(4):1–20
2. Zhang Y, Xu C, Li H, Yang K, Zhou J, Lin X (2018) HealthDep: an efficient and secure deduplication scheme for cloud-assisted eHealth systems. *IEEE Trans Ind Inf* 14(9):4101–4112
3. Halevi S, Harnik D, Pinkas B, Shulman-Peleg A (2011) Proofs of ownership in remote storage systems. In: *ACM conference on computer and communications security*, pp 491–500
4. Douceur JR, Adya A, Bolosky WJ, Simon D, Theimer M (2002) Reclaiming space from duplicate files in a serverless distributed file system. In: *International conference on distributed computing systems*, pp 617–624.

5. Zhang Y, Xu C, Lin X, Shen X (2019) Blockchain-based public integrity verification for cloud storage against procrastinating auditors. *IEEE Trans Cloud Comput*. <https://doi.org/10.1109/TCC.2019.2908400>
6. Bellare M, Desai A, Jokipii E, Rogaway P (1997) A concrete security treatment of symmetric encryption. In: *IEEE annual symposium on foundations of computer science*, pp 394–403
7. Bellare M, Keelveedhi S, Ristenpart T (2013) Message-locked encryption and secure deduplication. In: *International conference on the theory and applications of cryptographic techniques*, pp 296–312
8. Bellare M, Keelveedhi S, Ristenpart T (2013) DupLESS: server-aided encryption for deduplicated storage. In: *USENIX security symposium*, pp 179–194
9. Rivest RL, Shamir A, Adleman L (1978) A method for obtaining digital signatures and public-key cryptosystems. *Commun ACM* 21(2):120–126
10. Duan Y (2014) Distributed key generation for encrypted deduplication achieving the strongest privacy. In: *ACM workshop on cloud computing security*, pp 57–68
11. Shamir A (1979) How to share a secret. *Commun ACM* 22(11):612–613
12. Miao M, Wang J, Li H, Chen X (2015) Secure multi-server-aided data deduplication in cloud computing. *Pervasive Mob Comput* 24:129–137
13. Boneh D, Lynn B, Shacham H (2001) Short signatures from the weil pairing. In: *International conference on the theory and application of cryptology and information security*, pp 514–532
14. Boneh D, Lynn B, Shacham H (2004) Short signatures from the weil pairing. *J Cryptol* 17(4):297–319
15. Wong TM, Wang C, Wing JM (2002) Verifiable secret redistribution for archive systems. In: *IEEE international security in storage workshop*, pp 94–105
16. Zhang Y, Xu C, Li H, Yang K, Cheng N, Shen X (2020) PROTECT: efficient password-based threshold single-sign-on authentication for mobile users against perpetual leakage. *IEEE Trans Mob Comput*. <https://doi.org/10.1109/TMC.2020.2975792>
17. Zhang Y, Xu C, Cheng N, Shen X (2019) Secure encrypted data deduplication for cloud storage against compromised key servers. In: *IEEE global communications conference*, pp 1–6
18. Liu J, Asokan N, Pinkas B (2015) Secure deduplication of encrypted data without additional independent servers. In: *ACM conference on computer and communications security*, pp 874–885
19. Abadi M, Boneh D, Mironov I, Raghunathan A, Segev G (2013) Message-locked encryption for lock-dependent messages. In: *Annual cryptology conference*, pp 374–391
20. Bellare M, Keelveedhi S (2015) Interactive message-locked encryption and secure deduplication. In: *IACR international workshop on public key cryptography*, pp 516–538
21. Islam MS, Kuzu M, Kantarcioglu M (2012) Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In: *Network and distributed system security symposium*, pp 1–15
22. Harnik D, Pinkas B, Shulman-Peleg A (2010) Side channels in cloud services: deduplication in cloud storage. *IEEE Security Privacy* 8(6):40–47
23. Li S, Xu C, Zhang Y (2019) CSED: client-side encrypted deduplication scheme based on proofs of ownership for cloud storage. *J Inf Secur Appl* 46:250–258
24. Shacham H, Waters B (2008) Compact proofs of retrievability. In: *International conference on the theory and application of cryptology and information security*, pp 90–107
25. Shacham H, Waters B (2013) Compact proofs of retrievability. *J Cryptol* 26(3):442–483
26. Stanek J, Sorniotti A, Androulaki E, Kencl L (2014) A secure data deduplication scheme for cloud storage. In: *International conference on financial cryptography and data security*, pp 99–118
27. Stanek J, Kencl L (2016) Enhanced secure thresholded data deduplication scheme for cloud storage. *IEEE Trans Dependable Secure Comput* 15(4):694–707. <https://doi.org/10.1109/TDSC.2016.2603501>
28. Chen R, Mu Y, Yang G, Guo F (2015) BL-MLE: block-level message-locked encryption for secure large file deduplication. *IEEE Trans Inf Forensics Secur* 10(12):2643–2652

29. Zhao Y, Chow SSM (2019) Updatable block-level message-locked encryption. *IEEE Trans Dependable Secure Comput.* <https://doi.org/10.1109/TDSC.2019.2922403>
30. Kandekele S, Paul S (2018) Message-locked encryption with file update. In: International conference on applied cryptography and network security, pp 678–695
31. Armknecht F, Bohli J, Karame GO, Youssef F (2014) Transparent data deduplication in the cloud. In: ACM conference on computer and communications security, pp 831–843
32. Leontiadis I, Curtmola R (2018) Secure storage with replication and transparent deduplication. In: ACM conference on data and application security and privacy, pp 13–23
33. Baric N, Pfitzmann B (1997) Collision-free accumulators and fail-stop signature schemes without trees. In: International conference on the theory and applications of cryptographic techniques, pp 480–494
34. Camenisch J, Lysyanskaya A (2002) Dynamic accumulators and application to efficient revocation of anonymous credentials. In: Annual cryptology conference, pp 61–76
35. Kate A, Zaverucha GM, Goldberg I (2010) Constant-size commitments to polynomials and their applications. In: International conference on the theory and application of cryptology and information security, pp.177–194
36. Li J, Li J, Xie D, Cai Z (2016) Secure auditing and deduplicating data in cloud. *IEEE Trans Comput* 65(8):2386–2396
37. Liu X, Sun W, Lou W, Pei Q, Zhang Y (2017) One-tag checker: message-locked integrity auditing on encrypted cloud deduplication storage. In: IEEE conference on computer communications, pp 1–9
38. Boneh D, Crescenzo GD, Ostrovsky R, Persiano G (2004) Public key encryption with keyword search. In: International conference on the theory and applications of cryptographic techniques, pp 506–522
39. Casola V, Castiglione A, Choo KR, Esposito C (2016) Healthcare-related data in the cloud: challenges and opportunities. *IEEE Cloud Comput* 3(6):10–14
40. Sun J, Fang Y (2010) Cross-domain data sharing in distributed electronic health record systems. *IEEE Trans Parallel Distrib Syst* 21(6):754–764
41. Liang X, Li X, Shen Q, Lu R, Lin X, Shen X, Zhuang W (2012) Exploiting prediction to enable secure and reliable routing in wireless body area networks. In: International conference on computer communications, pp 388–396
42. Wang Y, Wu Q, Qin B, Shi W, Deng RH, Hu J (2017) Identity-based data outsourcing with comprehensive auditing in clouds. *IEEE Trans Inf Forensics Secur* 12(14):940–952
43. List of antibiotics. https://en.wikipedia.org/wiki/List_of_antibiotics
44. ARM, Building a secure system using TrustZone technology. <http://www.arm.com>
45. Lin H, Shao J, Zhang C, Fang Y (2013) CAM: cloud-assisted privacy preserving mobile health monitoring. *IEEE Trans Inf Forensics Secur* 8(6):985–997
46. Ekberg J, Kostiainen K, Asokan N (2013) Trusted execution environments on mobile devices. In: ACM conference on computer and communications security, pp 1497–1498
47. Vo DL, Zhang F, Kim K (2003) A new threshold blind signature scheme from pairings. In: IEICE symposium on cryptography and information security
48. Xia W, Jiang H, Feng D, Douglis F, Shilane P, Hua Y, Fu M, Zhang Y, Zhou Y (2018) A comprehensive study of the past, present, and future of data deduplication. *Proc IEEE* 104(9):1681–1710
49. Shin Y, Koo D, Hur J (2017) A survey of secure data deduplication schemes for cloud storage systems. *ACM Comput Surv* 49(4):1–38

Chapter 5

Secure Keyword Search



This chapter introduces searchable encryption (SE), a cryptographic primitive supporting search over encrypted data using keywords. SE plays an important role in current cloud storage systems. It enables users to retrieve target data from their entire data set in an efficient way without leakage of the data contents. In the following sections, the motivation of SE and its classification are first reviewed. Then, a comprehensive survey on each class of SE is provided. Finally, the latest advances in SE is studied.

5.1 Keyword Search Over Encrypted Data

Data confidentiality is one of the most important requirements in cloud storage systems, which protects the contents of outsourced data against anyone who does not own the data. To achieve data confidentiality, data owners always encrypt their data before outsourcing, and the cloud server only maintains the ciphertexts. This can be achieved by leveraging conventional encryption schemes, but it makes efficient searches over ciphertext by keyword impossible. Particularly, after a data owner \mathcal{U} generates a data set M , \mathcal{U} first encrypts M using an encryption scheme (both the symmetric-key encryption and public-key encryption can be used) and obtains the ciphertext C , and outsources C to the cloud server. Subsequently, when \mathcal{U} only needs to retrieve some segments of M , she/he has to download C from the cloud server, decrypt C to obtain M , and retrieve the target segments from M . This introduces prohibitive costs in terms of communication and computation.

Keywords firmly remain the most prevalent mechanism for searching data that are outsourced remotely in the plaintext form. A user can conclude some keywords for each data file and subsequently identify the target file using the corresponding keywords. However, such a keyword searching mechanism cannot be directly

integrated into the above encrypted data outsourcing system to support searching over ciphertexts, since it will introduce new vulnerabilities and considerable costs.

Specifically, assuming the outsourced data set is $M = \{M_1, M_2, \dots, M_n\}$, each of $M_i (i \in [1, n])$ contains some keyword w_i . For $i = 1, 2, \dots, n$, the user \mathcal{U} first encrypts M_i as well as w_i and obtains the ciphertexts C_i and c_i . Then \mathcal{U} outsources $\{C_i, c_i\}$ to the cloud server. A critical security concern occurs: Note that for $i \in [1, n], j \in [1, n], i \neq j$, c_i and c_j may intersect, the cloud server is easy to learn which files include the same keyword. Even if the cloud server does not know the content of keyword, it is enough to violate the user's privacy from the learned information. We stress that although this problem can be addressed by utilizing a randomized encryption algorithm, \mathcal{U} needs to securely maintain all random elements used to encrypt the files and keywords for subsequent searching, which is inefficient in reality.

Searchable encryption (SE) is a cryptographic primitive supporting search on encrypted data using keywords. In an SE scheme, both the file and its keywords are encrypted and outsourced to the cloud server. To search a target file, the user first takes a keyword and her/his secret key as inputs to obtain a *trapdoor*, then she/he sends the trapdoor to the cloud server. With the trapdoor and the outsourced files and keywords, the cloud server can identify the file containing the corresponding keyword and send the ciphertext to the user.

The security of SE requires that the leakage information after the user's searches should not be more than "search pattern" and "access pattern." Here, the search pattern refers to some information extracted from the outcomes of searches (i.e., whether searches were for the same word or not), and the access pattern refers to the identifiers of the documents that contain a keyword. We stress that an inherent requirement of SE is that any information that can be extracted by adversaries (e.g., a malicious cloud server) can only be obtained after the user has searched the relevant files, and if the user does not perform any search operation, nothing should be leaked.

Generally, SE can be divided into two categories: symmetric-key searchable encryption (SSE) and public-key searchable encryption (PSE). They are designed for different scenarios.

SSE is designed for a scenario that a user herself/himself encrypts her/his files using a symmetric-key encryption scheme, outsources the ciphertext to a cloud server, and retrieves target file(s) leveraging some keywords and her/his secret key. It can be extended to the multiple-user case that the data owner can delegate other users to access and search her/his outsourced files.

PSE is designed for a different scenario that some users (called senders) encrypt their files using a public-key encryption scheme under a receiver's public key, and outsource the ciphertexts to a cloud server; The receiver can retrieve target file(s) leveraging some keywords and her/his secret key.

In the remainder of this chapter, we will introduce SSE and PSE, respectively, and study the latest advances in PSE.

5.2 Symmetric-Key Searchable Encryption

Pioneers of SE schemes are only related to symmetric-key cryptosystems, and SE only refers to symmetric-key searchable encryption (SSE) for a long time.

5.2.1 System and Threat Models

The original target scenario of SSE is straightforward: a user \mathcal{U} outsources her/his data to a cloud server \mathcal{CS} and wants to access some segments of the outsourced data from the cloud server, where the segments are related to some keywords (as we now utilize google for searching). However, \mathcal{U} does not want to reveal the contents of outsourced data and keywords used for searching to others including \mathcal{CS} . As such, the data and keywords should be encrypted before being submitted to \mathcal{CS} . To achieve high efficiency, \mathcal{U} needs to leverage symmetric-key encryption algorithms.

The system model of SSE is shown in Fig. 5.1.

The threat model for SSE mainly considers the cloud server as an adversary. There are two types of threat models to define the misbehaved cloud server. The first one considers the cloud server as an honest-but-curious entity; The second one considers the cloud server as a malicious entity. In both the threat models, the cloud server is assumed to violate the user's privacy by extracting information about the user from the outsourced data and the submitted search queries as much as possible. For a malicious cloud server, it may return incorrect or incomplete data to the user. We stress that using orthogonal techniques, e.g., data integrity verification introduced in Chap. 3 and universal arguments [1], can make an SSE with resistance against honest-but-curious cloud server robust against malicious one. In this chapter, we restrict our attention to the honest-but-curious cloud server as well.

5.2.2 Survey on Symmetric-Key Searchable Encryption

The problem of searching on encrypted data is first defined by Song et al. [2], where the basic requirements of such a technique are first pointed out and the first

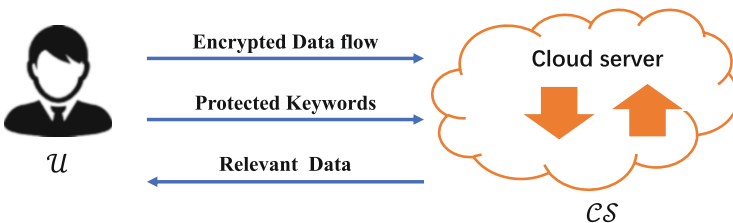


Fig. 5.1 System model of SSE

searchable symmetric-key encryption (SSE) scheme is proposed. In addition to the confidentiality of outsourced files, an SSE scheme should further ensure that:

- Before searching, the cloud server can learn nothing about the outsourced files;
- After searching by some keyword, the cloud server can only determine with some probability whether each file contains the submitted keyword without learning anything else.

Subsequently, the security of SSE is derived from these two requirements and is characterized as the requirement that nothing can be leaked beyond the search pattern and access pattern [3, 4].

To satisfy the above requirements, in an SSE scheme, the user should not submit the ciphertext(s) of the keyword(s) as a search token to the cloud server for searching files. She/he should compute a so-called trapdoor as the search token and submit the trapdoor to the cloud server for searching [5]. This trapdoor should have the following properties:

- The trapdoor can only be computed with the user's secret key;
- Without valid trapdoors, the cloud server can learn nothing from the outsourced files.

Song et al. are inspired by stream cipher and propose the first SSE scheme. However, this scheme requires the cloud server to scan each file in a word-by-word way for a given keyword, and thus supports a very slow search. The search costs are linear in $|M|$, where $|M|$ denotes the length of M .

To improve efficiency significantly, a promising approach is to build up an index that, for each keyword w of interest, lists the files that contain w . Since in SSE, the user herself/himself encrypts all files before outsourcing, she/he can organize the files in an arbitrary way and include additional data structure. It is feasible and affordable to build the index to allow the user to search on encrypted files with high efficiency. Such an index-based approach has become an underlying construction of all SSE schemes.

With the above basic idea of SSE, we introduce subsequent works on SSE from four different lines of research.

5.2.2.1 Basic SSE and Its Development

Due to the low efficiency of Song et al.'s scheme, Goh proposes the first secure index-based SSE scheme [5] to improve efficiency from the costs that are linear in $|M|$ of [2] to those that are linear in $\#M$, where $\#M$ denotes the number of files included in M .

There are two types of methods to construct the index. The first one is to construct the index by files, as shown in Fig. 5.2; The second one is to construct the index by keywords, as shown in Fig. 5.3.

The scheme in [5] generates a sub-index (i.e., bloom filter) for each file and its keywords, which essentially builds a secure index by files. To capture the security

File identifier	Keywords included in the file
M_1	w_1, w_3, \dots
\dots	\dots
M_n	w_1, w_2, \dots

Fig. 5.2 Index constructed by files

Keyword	Files including the keyword
w_1	M_1, M_3, \dots
\dots	\dots
w_k	M_1, M_2, \dots

Fig. 5.3 Index constructed by keywords

of secure-index-based SSE. A security notion of semantic security against adaptive chosen keyword attack (IND-CKA) is proposed to prove that an adversary cannot extract a file’s contents from its index, other than what he has extracted from previous search results or from other channels.

A milestone of SSE is the work of Curtmola et al. [3, 4]. The contributions of this work are twofold: The first formal and complete security definition of SSE is provided; The first SSE scheme with search costs that are sublinear in $\#M$ is proposed.

Specifically, in [3, 4], the security notion of IND-CKA is first analyzed, and its weakness is pointed out. Recall that the security of trapdoors is not explicitly required in IND-CKA, it causes that IND-CKA fails to guarantee the privacy of user queries. In [5] and [6], trapdoors are generated in a deterministic way. As a consequence, an external adversary who eavesdrops the interaction channels between the user and the cloud server is able to determine whether a search is repeated, which enables the adversary to violate the user’s search pattern.

To remedy it, Curtmola et al. [3, 4] first formally define four auxiliary notions: history, access pattern, search pattern, and trace.

- *History*: It refers to the interaction between the user and the cloud server that is determined by a file collection and a sequence of keywords that the user wants to search for and that SSE wants to protect against the adversary.
- *Access pattern*: It refers to the pointers to (encrypted) files that satisfy the search query, which has been introduced before.
- *Search pattern*: It refers to the information that whether a search query is repeated, which also has been introduced before.
- *Trace*: It refers to the trace of a history, which consists of exactly the information SSE has to leak about the history and nothing else. Trace, at least, should include the identifiers of the files containing each keyword in the history, and information describing which trapdoors submitted in the history that correspond to the same underlying keywords.

With the above notions, the security of SSE is formally defined in a simulation-based way. Intuitively, there are two worlds (i.e., environments) in the security game: The one is called real world, in which an SSE scheme is executed by the adversary; The other one is the simulated world (also called ideal world), in which the information that the SSE scheme cannot protect is directly leaked to the adversary. If the adversary cannot distinguish interactions between the simulated world and the real world, it means that the SSE scheme leaks nothing beyond the search pattern and access pattern.

Two versions of the above security are defined. The first one is non-adaptive, and the second one is adaptive. The difference between them is whether the user generates her/his keywords as a function of the outcome of previous searches. Obviously, the adaptive security definition is stronger than its non-adaptive counterpart.

The SSE scheme proposed by Curtmola et al. [3, 4] is based on the secure index constructed by keywords. It achieves adaptive security and hides everything else except the search pattern and access pattern. Furthermore, the scheme reduces the computational costs on the cloud server from those that are linear with the number of all files to those that are linear with the number of *the files containing the querying keyword*.

On the other hand, a limitation to all SSE schemes introduced so far is that they only support the single-keyword search, i.e., the cloud server is able to identify the subset of files matching a certain keyword. However, in reality, users may want to search their files using some combinations of keywords to access the outsourced files effectively.

It is nontrivial to extend the scheme in [3, 4] to support *conjunctive* keyword search (i.e., search on the encrypted data by Boolean combinations of keywords). We note that two straightforward ways can be applied to extend the scheme in [3, 4] to support conjunctive keyword search. The first one is to extend the single-keyword search scheme trivially: Given a combination of some keywords, it can provide the cloud server with a search capability for each individual keyword in the combination. For each keyword, the cloud server first retrieves the set of files matching that keyword, then outputs the intersection of all those sets. The second one is to require the user to define a meta-keyword for each possible combination of keywords. The user then associates the newly defined meta-keywords with her/his files as she/he did for the original keywords. Subsequent searches by any combination of keywords are the same as those by original keywords.

Whereas, both the above schemes are unsatisfactory. In particular, the first scheme is flawed, since a lot of information in addition to the search pattern and access pattern is leaked to the cloud server, and the cloud server can extract which files contain each individual keyword from the outcomes of the search. With this additional information, the cloud server might extract the infer information about the user's files during multiple searches. The second scheme is inefficient: for a file containing k keywords, it requires an additional 2^k meta-keywords to support for all possible combinations of keywords. As a result, the cloud server needs to store an exponential in k blowup in the number of files.

To address the above problems, Golle et al. [7] propose the first secure and efficient SSE scheme supporting the conjunctive keyword search. The key idea is to add an additional keyword field associated with each file. With this keyword files, the user is able to know in advance where (in which keyword field) the match would occur. This enables the user to pre-compute necessary parameters that are sent to the cloud server, and allows the cloud server to respond to the user's query of conjunctive search with relatively high efficiency compared with the above second scheme. The pre-computed parameters are one-time-only, which ensures that the cloud server can extract nothing beyond the access pattern and search pattern from the outsourced files and responses to the user.

The requirements in terms of security and efficiency proposed in [7] have served as basic criteria for the subsequent SSE schemes that support (both specific and general) Boolean queries, e.g., the scheme in [8].

5.2.2.2 SSE for Dynamic Data

In reality, after a user outsources her/his files to the cloud server, she/he might update the outsourced files as needed. Considering the requirements in terms of data confidentiality and search by keywords, a practical SSE scheme needs to support dynamic data operations (e.g., addition and deletion). However, all SSE schemes we have introduced so far either fail to support dynamic operations on the outsourced files, e.g., the scheme in [2], or incur substantial costs that are linear in all files when extending them to support dynamic data, e.g., the scheme in [5].

At a high level, all efficient SSE schemes are based on a secure index. The process of such an index-based SSE scheme is as follows. The user first executes the encryption algorithm which takes as input an index and a sequence of n files and outputs an encrypted index and a sequence of n encrypted files. The user then outsources the encrypted index and files to the cloud server. To search for a keyword w , the user computes a trapdoor using her/his secret key and sends the trapdoor to the cloud server. The cloud server takes as input the trapdoor, the encrypted index, and the encrypted files, identifies the corresponding encrypted files containing w and responds to the user with the encrypted files.

Extending the index-based SSE schemes, e.g., the one proposed in [3, 4], to support dynamic data operation suffers from the following problems. First, it faces an efficiency problem when supporting data additions. Given a new file M' that contains a set of keywords \mathbf{w}' , the user needs to interact with the cloud server to append array entries to the corresponding locations in the index constructed by keywords. We provide a simple example in Fig. 5.4, where we only consider the case of plaintext, the user wants to add a new file M_4 to the outsourced data set, and M_4 contains keywords of w_1 and w_2 . Note that when the outsourced files and index are encrypted, the updation on the index incurs substantial costs in terms of communication and computation. The similar problem also exists in the case that the user wants to delete some outsourced files from the entire data set.

Original Index		Updated Index	
Keyword	Files including the keyword	Keyword	Files including the keyword
w_1	M_1, M_3	w_1	M_1, M_3, M_4
w_2	M_2, M_3	w_2	M_2, M_3, M_4
w_3	M_1, M_2	w_3	M_1, M_2

The user adds a new file M_4 which contains keywords of w_1 and w_2 .

Fig. 5.4 Simple example of data addition in SSE

In [9], a secure and efficient SSE supporting dynamic data operations is proposed, where the search costs on the cloud server side of the scheme are linear in outsourced files containing the submitted keyword, which is optimal. This scheme is constructed on the scheme proposed by Curtmola et al. [3, 4], with the integration of the following key techniques.

- To support efficient file deletion, an additional encrypted data structure called deletion array is introduced. The deletion array enables the cloud server with a trapdoor received from the user to recover pointers to some nodes corresponding to the file to be deleted.
- To support efficient pointer modification, the pointers stored in a node are encrypted with a homomorphic encryption scheme, such that if the cloud server receives a ciphertext of an appropriate value from the user, it can modify the pointer without having to decrypt the node.
- To support efficient file addition, a memory management mechanism (where an additional space is added into the search array) is employed to keep track of the free locations in the search array, which allows the cloud server to add new nodes.

Although the SSE scheme in [9] achieves an optimal search efficiency, the encrypted indexes in the scheme [9] as well as the scheme [3, 4] store files at random disk locations, it fails to support parallel search. As a result, when the user submits a frequent-used keyword w (maybe thousands of files contain w), a significant computational delay is still required.

To address this problem, a parallel and dynamic SSE scheme is proposed by Kamara et al. [10]. The key technique behind the scheme [10] is to build an additional hierarchical tree to merge the indexes. This enables the cloud server with multiple processors to retrieve the target files in a parallel way, which reduces the computational delay significantly.

Dynamic SSE is an extension of SSE. It also should achieve all security criteria of SSE introduced in Sect. 5.2.2.1, i.e., in an SSE scheme, nothing can be leaked to adversaries beyond the search pattern and access pattern. In regards to dynamic SSE, to achieve the security criteria, it should have the following additional security properties.

- Forward security. If a user searches for keywords and subsequently adds a new file containing some of the keywords that are searched before, the cloud server

should not learn that the newly added file has a keyword the user searched for in the past.

- **Backward security.** If a user asks the cloud server to delete an outsourced file, and the cloud server removes the target file from its local storage, the cloud server should not learn that the deleted file has a keyword the user searched for in the past.

We stress that both the forward security and backward security do not just have theoretical value.

In reality, even small leakage can be utilized by adversaries to reveal the user's queries in both static and dynamic databases. Recent works, e.g., [11, 12], have demonstrated how adversaries, who extract a small piece of information during user's searches, launch attacks to violate the users' privacy. This problem could be further exacerbated in dynamic data settings. In dynamic SSE schemes, the adversary might send new files to a target user, and when the user updates her/his outsourced files, the newly received files which are generated by the adversary would be outsourced to the cloud server. By doing so, the adversary is able to control a part of the outsourced files of a target user, know the contents and the corresponding keywords of these files, and violate the user's privacy by obtaining the query information of the user. Such an attack is called file injection attack, and Zhang et al. [13] have demonstrated that it works on dynamic SSE schemes which are not forward-secure.

To achieve forward security, Stefanov et al. [14] propose a practical dynamic SSE scheme which supports searches, insertions, and deletions, and protects forward pattern and update pattern from leakage, and achieves search time of $O(m \log_3 n)$ and update time of $O(k \log_2 n)$, where m denotes the number of files matching the search and k denotes the number of unique keywords contained in the file. However, the scheme in [14] also incurs the efficiency problem. It requires the user to store some data, where the storage costs are linear in the number of files to be deleted. To further improve the efficiency, Bost [15] proposes a dynamic SSE scheme that is based on the private index to achieve high efficiency in terms of search and update. The construction of the scheme in [15] is much simpler than that in [14].

We note that the SSE scheme proposed by Kerschbaum et al. [16] also supports efficient index updation. This scheme does not offer a specific operation for initially outsourced files, and all files are added incrementally, where initially the user only encrypts the entire file set and does not need to generate other auxiliary information, e.g., index. This scheme achieves sublinear search time, and its efficiency is asymptotically optimal. The basic idea to construct such a dynamic SSE scheme is to require the cloud server to gradually construct an index from the access pattern. As such, the cloud server can only generate the index from searches of the user, and keywords in a newly added file would not be leaked to the cloud server after the file is outsourced. However, this scheme is confronted with two problems. The first one is that the search time of a keyword that the user first submits to the cloud server is slow, since the index is not constructed, and the cloud server needs to scan the entire file set to retrieve the target data. The second one is that if the index is generated by

the cloud server, the forward security cannot be guaranteed since which keywords have been submitted before is shared with the cloud server when a new file is added [17].

Prior dynamic SSE scheme with backward security is proposed by Naveed et al. [18], where the scheme protects the sensitive information about the outsourced files (e.g., number of outsourced files, the length of individual files, the files' identifiers and contents) and the operations performed by the user (e.g., access, addition, deletion, and updation) from leakage. Subsequently, several dynamic SSE schemes [19] with backward security are proposed. However, all these schemes incur massive costs in terms of communication and computation. The main challenge to design a backward-security dynamic SSE scheme is to ensure that newly submitted queries should not be tested on deleted files on the cloud server side.

5.2.2.3 Variants of SSE

The SSE schemes we have studied so far are mainly to design specific structures and schemes for keyword search with trade-offs on security, functionality, and efficiency. There is another line of work on SSE to leverage appropriate cryptographic primitives to support as many as query functions in database systems. This type of SSE can be considered as an extension or variant of SSE, and mainly focuses on applying SSE in reality.

The first work that considers the problem of encrypting structured data in such a way that it can be efficiently and privately queried is proposed by Chase et al. [20]. Structured data are widely used in different scenarios, e.g., Bioinformatics, Chemistry, social networks, relational database, semantic web, etc. Chase et al. generalize SSE to the setting of arbitrarily-structured data.

An important line of research focuses on extending SSE to support rich query types. Typical works mainly put effort into the following directions.

Boolean Search As discussed before, the multi-keyword search is a natural demand for users in reality. In the existing SSE scheme [7], only the conjunctive search is supported, and it works only for structured attribute-value type databases. It is desired to extend the scheme to support free text search and to the general Boolean search to provide a truly practical search capability. Several SSE schemes [8, 21–23] with support for general Boolean search have been proposed.

Range Search In reality, the user might search her/his files by a way that the keyword is within a range, e.g., return all records of employees whose salary within a range of \$10,000–15,000. Actually, range searches can be applied to any ordered set of elements. Order-preserving symmetric encryption (OPE) [24, 25] is a cryptographic primitive supporting efficient range queries on encrypted data. Order-revealing encryption (ORE) [26, 27] is a cryptographic primitive for ciphertext comparisons based on the order relationship of plaintexts without revealing the contents of the underlying plaintexts, which can also be leveraged to support efficient range queries on encrypted data. ORE provides a stronger security guarantee than OPE.

Fuzzy Search Fuzzy keyword search is a common type of search method, which greatly enhances system usability by returning the matching files when the user's searching inputs exactly match the predefined keywords or the closest possible matching files based on keyword similarity semantics, when the exact match fails. Several fuzzy SSE schemes have also been proposed [28, 29].

Ranked Search In reality, ranked searches, which enables the user to find the most or least relevant information quickly without unnecessary communication costs, are usually more effective than Boolean search. OPE can also be used to construct an SSE scheme with the ranked search. Several ranked SSE schemes have also been proposed [30, 31].

5.2.2.4 Other Works

In Sect. 5.2.1, we have mentioned that two different threat models are proposed to describe the attacks that the cloud server might launch in SSE. The SSE schemes we have studied so far only consider the cloud server as an honest-but-curious entity, i.e., it will honestly execute the prescribed scheme but may perform other attacks to violate the user's privacy. There is another line of works that focuses on resisting the malicious cloud server.

To thwart the malicious cloud server, we should first enumerate potential attacks that it may perform. Actually, most attacks can be resisted by utilizing the corresponding countermeasures. For example, to resist repudiation attacks, digital signature algorithms can be utilized; to protect the outsourced files against modification, proofs of storage can be leveraged.

However, a common type of attack cannot be addressed by directly integrating existing mechanisms into SSE, i.e., a malicious adversary might modify search results for profits. This motivates researchers to propose *verifiable* SSE.

There are generally two types of verifiable SSE schemes [32]: hash-based ones and accumulator-based ones. These schemes essentially utilize a proof of membership to enable the user to verify the correctness and integrity of the files received from the cloud server. Here, the integrity and correctness refer to the files that correspond with the desired search result, rather than the contents of outsourced files.

In modern cryptography, some cryptographic primitives that are proposed before the problem of search over ciphertext is defined can be utilized to support privacy-preserving search, although the original motivation of these primitives is not to address the problem. However, there are some important differences between these primitives and SSE.

Private Information Retrieval (PIR) In PIR, two entities (the one is a sender and the other one is a receiver) are involved, where the sender stores a database M_1, M_2, \dots, M_n and the receiver holds an index $i \in \{1, \dots, n\}$. PIR enables the receiver to learn M_i from the sender while ensuring that the sender can learn nothing [33]. PIR supports privacy-preserving search, but the data in PIR are always stored

in the plaintext form. The confidentiality of outsourced files cannot be ensured by utilizing PIR.

Oblivious RAM (ORAM) ORAM is a cryptographic primitive that enables a user who only stores locally a constant amount of data to outsource n files to the cloud server and to access the outsourced files while hiding the identities of the files which are being accessed [34, 35]. ORAM can be utilized directly to support all functionalities that the basic SSE scheme can provide, and even achieve a stronger privacy guarantee, since it would leak any information to the cloud server. However, none of the existing ORAM schemes can achieve high efficiency that an SSE scheme has, and is too inefficient to be applied in practice.

5.3 Public-Key Searchable Encryption

Considering the following scenario: Alice utilizes a cloud-based email system and wants to read her email on different devices: laptop, smartphone, etc. The cloud server (which is subject to an email service provider) is supposed to retrieve emails by some keywords received from Alice and sends the retrieved emails to the appropriate device. Note that all emails are generated by other users (e.g., Bob) and are sent to Alice by outsourcing these emails to the cloud server, such a cloud-based storage service provides users an efficient way to send their data to others and a flexible way to access their received data in different devices.

While users (in the remainder of this chapter, the senders and receiver are collectively referred to as “users”) enjoy great benefits from the cloud storage services, critical security concerns in data outsourcing have been raised seriously. One of the most important security issues is data confidentiality. From the perspective of users, contents of outsourced data are very sensitive, and should not be leaked for preserving privacy. Therefore, senders always encrypt the data before outsourcing. This can be achieved by utilizing conventional encryption, but it makes efficient searches over ciphertexts by keyword impossible.

Recall that SSE mainly targets at the scenario that a user encrypts her/his files, outsources the ciphertexts to a cloud server, and subsequently retrieve target files by keywords from the cloud server. The above problem cannot be addressed by employing SSE, which motivates a new cryptographic primitive: public-key searchable encryption (PSE) [36]. PSE is a variant of SE that addresses the problem of search over encrypted data under public-key cryptosystems.

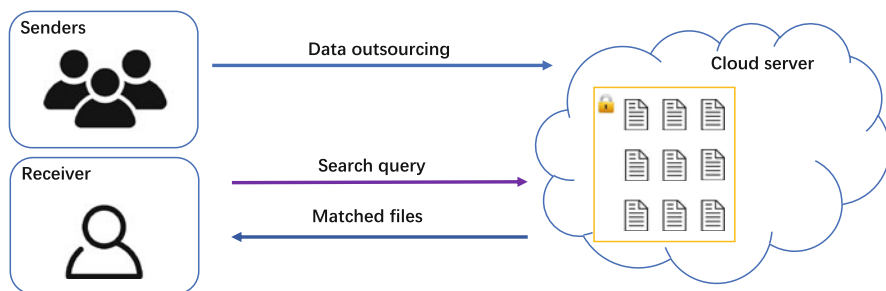


Fig. 5.5 System model of PSE

5.3.1 System model

The system model of PSE is shown in Fig. 5.5, where three entities are involved.

- **Senders:** Senders generate files which contain a small number of keywords and send the files and the keywords to a target receiver securely by encrypting the files and the selected keywords with the receiver's public key. The ciphertexts are outsourced to a cloud server.
- **Receiver:** The receiver is the data (i.e., encrypted files) owner, she/he receives the encrypted files from the cloud server, and decrypts them locally. Furthermore, she/he would search her/his outsourced files by keywords at a later point in time.
- **Cloud server:** The cloud server is subject to a cloud service provider. It receives encrypted files and encrypted keywords. Later, it provides the receiver with an efficient and secure way to search the ciphertexts by keywords and forwards the target ciphertexts to the receiver.

The process of the PSE is as follows: a sender \mathcal{S} first generates a file M that she/he wants to send it to a receiver \mathcal{R} . M contains a small number of keywords $\mathbf{w} = \{w_1, w_2, \dots, w_n\}$. Then \mathcal{S} encrypts M and \mathbf{w} using \mathcal{R} 's public key to obtain the corresponding ciphertexts C and $\{c_1, c_2, \dots, c_n\}$. \mathcal{S} sends C and $\{c_1, c_2, \dots, c_n\}$ to \mathcal{R} by outsourcing them to a cloud server \mathcal{CS} . When \mathcal{R} wishes to search her/his files by a keyword from \mathcal{CS} , she/he first generates a trapdoor on the keyword by using her/his secret key and sends the trapdoor to \mathcal{CS} . Upon receiving the trapdoor, \mathcal{CS} can test whether the ciphertext of a keyword (i.e., each item in \mathbf{w}) matches the trapdoor for data retrieval.

In PSE, the index-based idea [5] can also be utilized to improve the search efficiency. We note that the original secure index scheme cannot be applicable to PSE, since the index should be generated and encrypted by the user who generates and searches files. However, the method [16] that requires the cloud server to gradually construct an index (which is constructed by keywords) from the access pattern can be utilized, which allows the cloud server to efficiently search by the keywords that have been submitted before. In PSE, such an index-based mechanism can serve as an independent building block to be integrated directly into the system

to improve efficiency. Therefore, when introducing PSE, we would not explicitly show how the matches files are retrieved, but only show how the cloud server tests the correspondence between a ciphertext and a trapdoor. Actually, with the knowledge of the correspondence between the ciphertext and the trapdoor, the corresponding files can be easily retrieved.

5.3.2 Threat Model and Security Definition

In reality, threats towards PSE are mainly from two different angles: an adversarial cloud server and a malicious sender.

- Adversarial cloud server. The adversarial cloud server attempts to violate the receiver's privacy by breaking confidentiality.
- Malicious sender. Any adversary can become a sender to send any file to the receiver. By doing so, any adversary may control some of the receiver's outsourced files.

To achieve the security of PSE under the above threat model, new security notions should be defined. Note that the basic security requirement for PSE is that the ciphertext of a keyword does not reveal any information about the underlying keyword unless the corresponding trapdoor is available. Therefore, the first security notion for PSE is to capture the security against an active adversary who can get trapdoors for any keyword he selects. Even under this attack, the adversary should not have the capability of distinguishing an encryption of a keyword w_0 from an encryption of a keyword w_1 for which he did not get the trapdoor. Intuitively, such a security property guarantees that for an adversary, without the corresponding trapdoors, the information extracted from an encryption of a keyword should not be more than that extracted from an encryption of another keyword. This security property is defined as semantic security against an adaptive chosen keyword attack.

5.3.3 Survey on Public-Key Searchable Encryption

The problem of searching on encrypted data under public-key cryptosystems is first defined by Boneh et al. [36], where the security notions and schemes are proposed. Compared with its symmetric-key counterpart (i.e., SSE), PSK has some advantages, e.g., it is more expressive, and can be easily extended to support multi-keyword search, and enriched functionalities, such as disjunctive searches, conjunctive searches, equality searches, range searches, and subset searches. However, compared with SSE, PSE also has some disadvantages. Particularly, PSE generally is less efficient, and most existing PSE schemes still focus on text-formed files, regardless of complex data structures in reality.

The first PSE scheme is proposed by Boneh et al. [36]. This scheme is based on bilinear pairing (i.e., $e : G \times G \rightarrow G_T$) and is essentially transformed from

the identity-based encryption (short for IBE) scheme [37], where the “keyword” in the PSE scheme is considered as the “identity” in the IBE scheme. Subsequent PSE schemes essentially share the common system model.

Specifically, in the Boneh et al.’s scheme, the system parameters are $\{p, P, G, G_T, e, H_1, H_2\}$, where $e : G \times G \rightarrow G_T$ is a bilinear map, and P is the generator of G whose order is p , $H_1 : \{0, 1\}^* \rightarrow G$ and $H_2 : G_T \rightarrow \{0, 1\}^{\lg p}$ are secure hash functions. The receiver \mathcal{R} ’s secret key is α which is randomly chosen from Z_p^* , the corresponding public key is $\{P, Q = \alpha P\}$, where P is a generator of G .

Given a keyword w , the sender \mathcal{S} computes the ciphertext as follows. \mathcal{S} randomly chooses $\chi \in Z_p^*$, computes $\tau = e(H_1(w), \chi Q)$ and $C_w = (\chi P, H_2(\tau))$. Then, \mathcal{S} outsources C_w to the cloud server \mathcal{CS} .

Given the keyword w , \mathcal{R} computes the trapdoor as $td_w = \alpha H_1(w)$ and sends td_w to \mathcal{CS} .

Given a ciphertext $C_w = (A, B) = (\chi P, H_2(\tau))$, if the trapdoor is computed on w and α , the equation $H_2(e(td_w, A)) = B$ holds, which means that the target file containing w can be retrieved via td_w from the entire ciphertext set.

Following the Boneh et al.’s scheme [36], lots of PSE schemes are proposed with different features. In the following sections, we introduce existing PSE schemes from three different angles.

5.3.3.1 Vulnerability of PSE Against Keyword Guessing Attacks and Countermeasures

Note that due to the randomness of χ , for the same keyword, different senders would produce different ciphertexts, which ensures that the adversary, i.e., the adversarial cloud server, cannot know which files contain the same keywords before these files are searched.

Despite the advantages of PSE, it also suffers from critical threats. In reality, keywords are always chosen from a small space, and users usually leverage well-known keywords for the search of files. Since everyone including adversaries can become a sender in PSE, the ciphertext can be generated by the adversary as needed. In other words, a fundamental problem with PSE is that the ciphertext can be computed given only the keyword, but the keyword is the only secret that is contained in the outsourced ciphertexts and that should be well protected, thus enabling off-line keyword guessing attacks (KGA): Given a trapdoor, an adversary (i.e., the adversarial cloud server) enumerates all possible keyword candidates and encrypts them with the receiver’s public key one by one; He tests the ciphertexts on the trapdoor, which enables him to identify the ciphertext which matches the targeted trapdoor and to recover the keyword hidden in the trapdoor to violate the users’ privacy.

We stress that vulnerability of PSE against off-line KGA on keywords is a major hindrance towards the broad adoption of PSE, since searched data (e.g., emails with sensitive keywords) is considered as being highly secret by many individuals and

organizations. Existing works [38, 39] have proven the feasibility of KGA on PSE from both theory and practice.

Existing works on thwarting off-line KGA can be mainly classified into four categories.

The first one is PSE with an authorized tester to resist off-line KGA. Typical works including [40, 41]. In this mechanism, testing matching between a ciphertext and a trapdoor only can be performed by an authority (i.e., authenticated cloud storage server). This protects keywords from KGA performed by outside adversaries. However, such the mechanism cannot resist malicious cloud storage servers, since the authenticated server can test whether a ciphertext matches a trapdoor without any limitation.

The second one is PSE that are constructed on emerging cryptographic primitives to resist off-line KGA. In [42], Sun et al. present a PSE scheme with resistance against off-line KGA by using $i\mathcal{O}$. In this scheme, ciphertexts are generated by a signcryption algorithm, and the cloud server cannot generate a legitimate ciphertext to test whether it matches a received trapdoor. To ensure security, the keys used to unsigncryption are embedded into an obfuscated program generated by the receiver. The obfuscated program is executed by the cloud server for searching files by keywords. Whereas, current $i\mathcal{O}$ constructions require prohibitive costs in terms of storage and computation to obfuscate programs and execute the obfuscated program, which makes the scheme presented in [42] inefficient.

The third one is PSE supporting fuzzy keyword search to resist off-line KGA. Typical works including [43]. In such a mechanism, a keyword is related to an exact keyword search trapdoor and a fuzzy keyword search trapdoor. A fuzzy keyword search trapdoor corresponds to two or more keywords. The cloud server only has the fuzzy keyword search trapdoor to retrieve data. As a result, a malicious cloud server would not learn the exact keyword. Nonetheless, the malicious cloud storage server can still narrow down the space of the target keyword, and thereby the keyword privacy is not well protected in such the mechanism. Furthermore, it requires the receiver to filter out the non-matching ciphertexts received from the cloud server, which incurs heavy communication and computation costs on the receiver side. Readers may recall SSE with the fuzzy search, we stress that the target problem of PSE with fuzzy search is different from that of SSE with fuzzy search. PSE with fuzzy search mainly focuses on resisting off-line KGA, but SSE with fuzzy search mainly focuses on enriching the search functionality to provide users a better search service than basic SSE.

The fourth one is PSE that are constructed on new frameworks to resist off-line KGA. Typical works including [44, 45]. An independent entity is introduced to help protecting keywords against adversaries. For example, in [44], an independent key server that holds a server-side secret is employed to assist users in generating and searching ciphertexts. This mechanism follows the Bellare et al.'s scheme [46] that resists brute-force attacks for secure deduplication, which we have introduced in Sect. 3.3.2. Compared with the above three categories of mechanisms, this mechanism balances the trade-off between efficiency and security and is more practical. Nevertheless, such a server-aided mechanism also bears a quite strong assumption that the key server is reliable, and it well maintains its secret over the

entire lifetime of the storage server. Furthermore, it is also vulnerable to online KGA, where the adversary (e.g., the adversarial cloud server) impersonates a valid sender to access the key server for performing KGA.

5.3.3.2 Constructing PSE on Different Cryptosystems

The PSE schemes we introduce so far are impliedly based on the public-key infrastructure (PKI), in which a fully trusted certificate authority is employed to issue the users' certificates. Thus these schemes inherit the limitations of PKI-based cryptosystems. Specifically, in a PSE scheme, both the senders and cloud server have to manage the receiver's certificate to choose the correct public key for file searching. As such, the senders and cloud server suffer from the certificate management problem, which includes certificate revocation, storage, distribution, and verification. In reality, managing certificates is very costly and cumbersome and removing the certificate management problem could be economic and favorable for commercial cloud storage systems.

To address the certificate management problem in PSE, it can be constructed on certificateless cryptosystems, where a key generation center (KGC) is introduced to assist the receiver in generating her/his private key. The first certificateless PSE scheme is proposed by Peng et al. [47]. Subsequently, some certificateless PSE schemes, such as [48, 49], with enhanced security or other features, are proposed.

Another line of work focuses on designing secure PSE schemes on different mathematics hard problems or different cryptographic assumptions. All the above schemes are constructed on bilinear pairing, and their security relies on the hard problems in cyclic groups that are derived from elliptic curves. However, these hard problems are believed to be easy to be solved, with the emergence of quantum computers [50], and thus existing PSE schemes would be threatened. Recent breakthrough results [51] show that adopting quantum computers in reality would be possible in the near future, and thus poses the post-quantum secure PSE schemes more demanding than ever.

To enabling PSE to be secure in the post-quantum era, several lattice-based PSE schemes are proposed [52–54]. These schemes are constructed on the lattice-based cryptosystem [55], which is considered to be secure against attacks from quantum computers.

5.3.3.3 Essential Relationships Between PSE and Other Cryptographic Primitives

In public-key cryptosystems, in addition to PSE, several cryptographic primitives can be utilized to achieve searches over encrypted files protected under public-key encryption schemes. In this section, we discuss the relationships between PSE and them.

Identity-Based Encryption Recall that the Boneh et al.’s PSE scheme [36] is based on the Boneh–Franklin identity-based encryption (IBE) [37], where the “identity” in the IBE scheme becomes the “keyword” in the PSE scheme. Readers might ask whether any IBE scheme can be transformed into a secure PSE scheme. In the research of IBE, a variant called anonymity IBE is proposed, where an IBE is anonymous if the receiver’s identity cannot be extracted from a ciphertext. Boneh et al. have proven that if an IBE scheme satisfies a specific formal notion of anonymity, then one is able to easily construct a secure PSE scheme from the IBE scheme using a general method proposed in [36].

Functional Encryption Functional encryption (FE) is a cryptographic primitive [56, 57] that supports restricted private keys that allow a receiver (i.e., the key holder) to learn a specific function of encrypted files while learning nothing else about the files. Oblivious, if the specific function is straightforward set to output whether given two ciphertexts correspond to the same plaintext, a secure PSE is produced. However, constructing an FE that meets the requirement is very challenging, and the efficiency of such an FE is low.

Fully Homomorphic Encryption Fully homomorphic encryption (FHE) is also a cryptographic primitive [58] that enables one to evaluate programs over encrypted files without decrypting the files. We also note that when the program is set to be subtraction on two ciphertexts, where the one is outsourced by the sender, and the other one is provided by the receiver, and the program is executed by the cloud server. The cloud server returns each result to the receiver, the later decrypts it and tests whether the decryption is equal to 0, if yes, the underlying file is satisfactory. However, existing FHE constructions require very high costs in terms of computation and storage. Thus, the PSE scheme based on FHE is very inefficient.

5.4 Latest Advances in Public-Key Searchable Encryption

Recall Sect. 5.3.3.1, existing PSE schemes with resistance against KGA are confronted with security and efficiency problems. In this section, we study the latest advances of PSE that resists KGA and introduce a practical scheme called SEPSE [59].

5.4.1 *Public-Key Searchable Encryption Against Keyword Guessing Attacks*

SEPSE follows the fourth PSE mechanism that resists KGA, i.e., constructing PSE on a new framework. The key observation of SEPSE is that the server-aided PSE suffers from

- the single-point-of-failure problem, where the security relies on the reliability of the key server, if the key server is compromised, the scheme is vulnerable to off-line KGA,
- and vulnerability to online KGA, where the adversarial cloud server can impersonate a sender to access the key server such that it obtains enough server-derived keywords to perform off-line KGA.

We note that directly extending the single key server to multiple ones in a threshold way is a natural solution. Whereas, it is still vulnerable to key compromising: an adversary (e.g., the adversarial cloud server) can perpetually attempt to break into the target key server over a long period of time (i.e., the entire lifetime of the cloud server). Once it succeeds, the security of affected keywords cannot be guaranteed. Furthermore, in the above threshold-based solution, each task only requires t of key servers' participation, but if so, the adversary can try different keywords to interact with different t key servers to obtain a significant number of ciphertexts of keywords to perform off-line KGA.

To address the above problem, SEPSE leverages a rate-limiting mechanism, and to further improve the communication efficiency, a blockchain-assisted rate-limiting mechanism is proposed and integrated into SEPSE.

5.4.1.1 System Model

As shown in Fig. 5.6, five entities are involved in SEPSE.

- Senders: Senders generate files which contain a small number of keywords, and send the files as well as the keywords to a target receiver *securely* by encrypting the files and the selected keywords with the receiver's public key. The ciphertexts are outsourced to the storage server.
- Receiver: The receiver is the data (i.e., encrypted data files) owner, she/he receives the encrypted files from the cloud server, and decrypts them locally. Furthermore, she/he would search her/his files stored on the cloud server by keywords at a later point in time.
- Key servers: The key servers are employed to help senders and receivers in generating the keyword to be encrypted, which is able to resist KGA.
- Cloud server: The cloud server is subject to a cloud service provider. It receives encrypted data as well as the encrypted keywords. Later, it provides the receiver with an efficient and secure way to search the ciphertexts by keywords and forwards the target ciphertexts to the receiver.
- Gateway: The gateway is introduced to help users in forwarding the request of keyword generation to key servers and assist the key servers in collecting service charges from the users.

Time in SEPSE is divided into fixed intervals of predetermined length called epochs. SEPSE needs to be set up only once over the entire lifetime, the servers-side secret on each key server needs to be updated only once in an epoch, and the senders

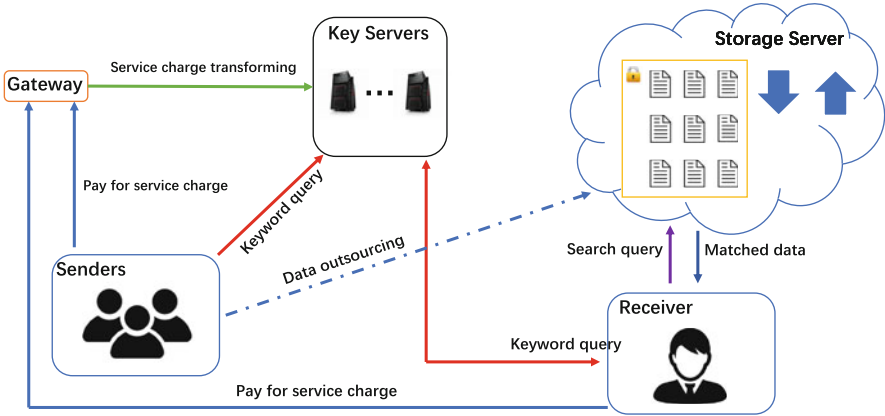


Fig. 5.6 System model of SEPSE

and receiver can interact with the cloud server for file outsourcing and searching multiple times in an epoch.

5.4.1.2 Threat Model

In a servers-aided PSE scheme, threats are mainly from four angles: adversarial cloud server, compromised key server(s), malicious sender, and malicious gateway.

- **Adversarial storage server.** The adversarial cloud server would perform KGA to break the confidentiality of the outsourced keywords.
- **Compromised key server(s).** An adversary may compromise the key servers, where he may break into each key server multiple times, to retrieve the outsourced keywords. Here, we assume that the number of key servers that can be compromised by the adversary within one epoch is less than the threshold, a key server can be compromised until the end of the current epoch, and compromised key servers are allowed to deviate from the scheme arbitrarily as the adversary needed. When a new epoch begins, the key servers that are compromised by the adversary in the previous epoch are “released” by the adversary [60].
- **Malicious sender.** Any adversary can become a sender to query the servers-derived keywords and perform KGA to retrieve the outsourced keywords.
- **Malicious gateway.** The malicious gateway may collude with an online keyword guessing adversary (will be detailed later) to break the security.

To prove the security of SEPSE under the above threat model, SEPSE follows the security notions of semantic security against chosen keyword guessing attacks (SS-CKGA) and indistinguishability against chosen keyword attacks (IND-CKA) [36, 44].

5.4.1.3 Introduction to SEPSE

To design SEPSE, two challenges should be addressed:

1. How to resist KGA without the single-point-of-failure problem. Since the keywords are inherently low-entropy, when the receiver searches data from the storage server by keyword, the storage server can enumerate all possible keywords and test them one by one to retrieve the target keyword. Although the existing scheme [44] can resist such attacks, it faces the single-point-of-failure problem, and its security solely relies on the reliability of the key server.
2. How to periodically renew the secret shares on the key servers. In reality, an active adversary may break into a subset of key servers multiple times over the entire lifetime of the storage server. As such, the periodical renewal of secret share on each key server is an affordable and effective solution to resist the active adversary.

SEPSE addresses the above first challenge by employing multiple key servers that jointly assist users in encrypting keywords to resist KGA. The main idea behind SEPSE is to require these key servers to help users in protecting keyword using a server-side secret, and to let each user obtain a servers-derived keyword. Then, the user encrypts each servers-derived keyword as her/he did in the original PSE scheme [36]. Here, these key servers jointly in a (t, n) -threshold and oblivious way, such that any t of them are able to assist a user in executing the prescribed protocol, but any one of them cannot learn any information about the keyword submitted by the user. Furthermore, the server-side secret is generated and shared among all the key servers in a distributed way, which frees SEPSE from a trusted dealer.

SEPSE addresses the above second challenge by requiring each key server to renew its secret share. This is achieved by introducing a “0-sharing” mechanism: at the end of each epoch, all key servers jointly share a 0, and each of them adds the secret share of 0 to its current secret share. Since a share of 0 is not equal to 0, the above 0-sharing mechanism enables each key server to change its secret share. On the other hand, since the newly shared secret is 0, the above 0-sharing mechanism does not change the server-side secret shared among all key servers.

There is still a subtle security issue. Since anyone can become a sender in PSE, an adversary (e.g., the adversarial cloud server) is able to try different keywords by requesting the corresponding servers-derived keywords. The adversary uses different identities of senders to perform this online attack,¹ and then obtains all servers-derived keywords. Finally, the adversary can perform off-line KGA to recover the keywords. We call such attacks online keyword guessing attacks (online KGA).

To thwart online KGA, SEPSE employs a rate-limiting mechanism: the number of servers-derived keyword requests made by a user in an epoch is limited by key

¹It is easy to detect in reality if the adversary is trying to request servers-derived keywords very frequently within a short period of time in reality.

servers. Specifically, a bound ρ is determined with the security parameter, and key servers stop responding after the bound ρ is reached. To improve communication efficiency on the user side, a gateway is utilized to help users in forwarding the servers-derived keyword queries. The servers-derived keyword request made by a user is first submitted to the gateway and then is transfers to all key servers by the gateway.

SEPSE further considers the case that the number of all key servers n is large but the threshold t is small (compared with n). In this case, submitting the request of the servers-derived keyword to n key servers incurs heavy communication costs for users. We stress that requiring the user to submit the servers-derived keywords to only t key servers is confronted with a malicious gateway or malicious users. Since the malicious gateway may omit the number of servers-derived keyword queries made by an adversary and always allows him to request servers-derived keywords, and a malicious user can select different t key servers for different queries to perform online KGA.

To resist the malicious gateway and malicious users, the key technique used here is the public blockchain, e.g., Ethereum [61], where each servers-derived keyword query made by a user is converted to a transaction on the Ethereum blockchain, and the total number of keyword queries for a user in each epoch can be verified by checking the number that the user creates transactions in the epoch. This yields the blockchain-assisted rate-limiting mechanism which can be proved that breaking the security by performing online KGA as hard as forking the Ethereum blockchain. As a result, integrating this mechanism into SEPSE can reduce the communication overhead significantly while achieving the same security guarantee.

Specifically, a sender \mathcal{S} , a set of key servers $\{\mathcal{KS}_1, \mathcal{KS}_2, \dots, \mathcal{KS}_n\}$, a cloud storage server \mathcal{CS} , a receiver \mathcal{R} , and a manager of the gateway \mathcal{D} are involved in SEPSE.

Setup

- With the security parameter ℓ , system parameters $\{p, P, G, G_T, e, h, H, H_1, H_2, F, \rho, t\}$ are generated, in which $e : G \times G \rightarrow G_T$ is a bilinear map, and P is the generator of G , $h : G \rightarrow Z_p$, $H, H_1 : \{0, 1\}^* \rightarrow G$, and $H_2 : G_T \rightarrow \{0, 1\}^{\lg P}$ are secure hash functions, $F : Z_p \times \{0, 1\}^* \rightarrow \{0, 1\}^*$ is a pseudorandom function, ρ is a maximum number that each sender and the receiver can require the keyword in each epoch, and t is the threshold number.
- For the receiver \mathcal{R} , her/his secret key is α randomly chosen from Z_p^* and the corresponding public key is $Q_{\mathcal{R}} = \alpha P \in G$.
- For $i = \{1, 2, \dots, n\}$, \mathcal{KS}_i randomly picks $a_{i0} \in Z_p^*$ and a polynomial $f_i(x) \in Z_p$ whose degree is $t - 1$, s.t. $f_i(0) = a_{i0}$, where $f_i(x) = a_{i0} + a_{i1}x + \dots + a_{i,t-1}x^{t-1}$.
- \mathcal{KS}_i computes $a_{i0}P$ and $a_{i\epsilon}P$ ($\epsilon = \{1, 2, \dots, t - 1\}$), and publishes them. \mathcal{KS}_i computes $f_i(j)$ and sends it to \mathcal{KS}_j for $j = 1, 2, \dots, n; j \neq i$ via a secure channel.
- \mathcal{KS}_i extracts $f_j(i)$ and checks if by verifying $f_j(i)P = \sum_{\gamma=0}^{t-1} i^\gamma \cdot a_{j\gamma}P$. If the checking passes, it accepts $f_j(i)$.

- \mathcal{KS}_i 's secret share is $s_i = \sum_{\gamma=1}^n f_{\gamma}(i)$, its public share is $Q_i = s_i P$. The secret key shared among $\{\mathcal{KS}_1, \mathcal{KS}_2, \dots, \mathcal{KS}_n\}$ is $s = \sum_{i=1}^n a_i 0$, and the corresponding public key is $Q = \sum_{i=1}^n a_i 0 P$.
- The gateway \mathcal{D} and \mathcal{KS}_i maintain a log file to count the number of keywords requested by \mathcal{S} (denoted by $\rho_{\mathcal{S}}$) and \mathcal{R} (denoted by $\rho_{\mathcal{R}}$). Initially, $\rho_{\mathcal{S}} = 0$ and $\rho_{\mathcal{R}} = 0$.

PEKS Given a keyword w , \mathcal{S} computes the ciphertext as follows.

- \mathcal{S} randomly chooses $r \in \mathbb{Z}_p^*$, computes $w' = rH(w)$.
- \mathcal{S} transfers a service charge and sends w' to \mathcal{D} .
- \mathcal{D} checks whether $\rho_{\mathcal{S}} < \rho$. If yes, it transfers the corresponding service charge to \mathcal{KS}_k for each $k \in \{1, 2, \dots, n\}$, sets $\rho_{\mathcal{S}} ++$, and informs \mathcal{S} ; otherwise, it rejects.
- \mathcal{S} sends w' to \mathcal{KS}_k ($k = 1, 2, \dots, n$).
- After receiving the service charge, \mathcal{KS}_k ($k = 1, 2, \dots, n$) checks whether $\rho_{\mathcal{S}} < \rho$. If yes, it generates a signature σ_k on w' by using the secret share s_k as $\sigma_k = s_k w'$, sets $\rho_{\mathcal{S}} ++$, and sends σ_k to \mathcal{S} ; Otherwise, it aborts.
- \mathcal{S} verifies σ_k by checking

$$e(\sigma_k, P) \stackrel{?}{=} e(w', Q_k).$$

If the checking fails, \mathcal{S} rejects σ_k ; Otherwise, \mathcal{S} stores σ_k locally.

- After receiving t valid signatures (these t signatures are denoted by $\{\sigma_{i_1}, \sigma_{i_2}, \dots, \sigma_{i_t}\}$ and their indexes form a set $T = \{i_1, i_2, \dots, i_t\}$), \mathcal{S} computes

$$\sigma_w = r^{-1} \sum_{k=i_1}^{i_t} \omega_k \sigma_k, \quad (5.1)$$

where $\omega_k = \prod_{\substack{i_1 \leq j \leq i_t \\ j \neq k, j \in T}} \frac{j}{j-k}$. \mathcal{S} verifies the correctness of σ_w by checking

$$e(\sigma_w, P) \stackrel{?}{=} e(H(w), Q).$$

- \mathcal{S} computes $sd_w = F(h(\sigma_w), w)$ as the servers-derived keyword of w , randomly chooses $\chi \in \mathbb{Z}_p^*$, computes $\tau = e(H_1(sd_w), \chi Q_{\mathcal{R}})$ and $C_{sd_w} = (\chi P, H_2(\tau))$. Finally, \mathcal{S} sends C_{sd_w} to \mathcal{CS} .

Trapdoor Given a keyword w , the trapdoor td_w used to retrieve data is generated by \mathcal{R} as follows.

- \mathcal{R} interacts with the key servers to generate the servers-derived keyword of w (i.e., sd_w), this process is the same as the one performed by \mathcal{S} in **PEKS**, where \mathcal{R} plays the role of \mathcal{U} .
- \mathcal{R} computes $td_w = \alpha H_1(sd_w)$.

Finally, \mathcal{R} sends td_w to \mathcal{CS} .

Test \mathcal{CS} takes as input $C_{sd_w} = (A, B) = (\chi P, H_2(\tau))$ and td_w , and checks $H_2(e(td_w, A)) \stackrel{?}{=} B$. If the equation holds, \mathcal{CS} outputs *True*; Otherwise it outputs *False*.

KeyRenew For each key server \mathcal{KS}_i ($i \in [1, n]$), it renews its secret share as follows.

- \mathcal{KS}_i randomly selects a polynomial $g_i(x)$ over Z_p with degree at most $t - 1$ s.t. $g_i(0) = 0$, in which $g_i(x) = b_{i1}x + b_{i2}x^2 + \dots + b_{i,t-1}x^{t-1}$.
- \mathcal{KS}_i computes $b_{i\epsilon}P$, $\epsilon \in \{1, 2, \dots, t - 1\}$ and publishes it. \mathcal{KS}_i sends $g_i(j) \pmod{p}$ to \mathcal{KS}_j for $j = 1, 2, \dots, n$; $j \neq i$ via a secure channel.
- \mathcal{KS}_i extracts $g_j(i)$ and checks $g_j(i)P \stackrel{?}{=} \sum_{\gamma=1}^{t-1} i^\gamma b_{j\gamma}P$. If the checking succeeds, \mathcal{KS}_i accepts $g_j(i)$.
- \mathcal{KS}_i computes a new secret share s'_i as

$$s'_i = s_i + \sum_{j=1}^n g_j(i). \quad (5.2)$$

Note that the secret s has the form $s = \sum_{i=1}^n a_{i,0} = \sum_{i=1}^n f_i(0)$. Assume the renewed secret distributed to all key servers is s' , it has the form $s' = \sum_{i=1}^n f'_i(0)$. Since $f'_i(x) = f_i(x) + g_i(x)$, we have $s' = \sum_{i=1}^n f'_i(0) = \sum_{i=1}^n f_i(0) + g_i(0)$. Because $g_i(0) = 0$, we further have $s' = \sum_{i=1}^n f_i(0) = s$. Therefore, the renewal of secret shares would not change the secret s shared among all key servers. This ensures the proactive security of SEPSE: an adversary who breaks into multiple key servers cannot break the confidentiality of keywords by performing KGA.

Improvement on the Communication Efficiency With the establishment of SEPSE, the total number of key servers n may be large while the threshold t may be small. The rate-limiting mechanism that requires a servers-derived keyword query to be submitted to *all* key servers can be tedious and inefficient. We assume that there are dozens of key servers and only a couple of them need to assist users in generating servers-derived keywords, e.g., $n = 20, t = 2$, it is more advantageous for both users and the gateway to only submit a servers-derived keyword request to t key servers rather than all of them. However, a malicious sender who compromises the gateway can request different servers-derived keywords from different t key servers. If the sender's request is only submitted to t key servers rather than all of them, in the extreme case, he can obtain up to $\lfloor \frac{n}{t} \rfloor \cdot \rho$ servers-derived keywords in an epoch,² which significantly increases the success probability of online KGA. Therefore, it is necessary to achieve the synchronization on the number of servers-derived keyword requests made by a user among all key servers without requiring

² $\lfloor \frac{n}{t} \rfloor$ denotes the largest integer smaller than $\frac{n}{t}$.

the user to submit each request to all key servers. A straightforward method is to require each key server to broadcast each received request to all others. However, it requires key servers to interact with each other to generate one servers-derived keyword and causes a heavy communication burden for them. Actually, it is very challenging to achieve such the synchronization among all key servers without introducing substantial communication costs when the gateway is compromised by an adversary (i.e., a malicious sender). Therefore, a blockchain-assisted rate-limiting mechanism is proposed and integrated into SEPSE to resist online KGA with high communication efficiency.

In **Setup**, the system parameters are the same as those in the basic scheme, with one difference. The sender \mathcal{S} , the receiver \mathcal{R} , and the manager of gateway \mathcal{D} each create their own accounts in Ethereum, where their accounts are denoted by $A_{\mathcal{S}}$, $A_{\mathcal{R}}$, and $A_{\mathcal{D}}$, respectively. Only \mathcal{D} is required to maintain $\rho_{\mathcal{S}}$ and $\rho_{\mathcal{R}}$.

In **PEKS**, given a keyword w , \mathcal{S} generates the servers-derived keyword sd_w as before with the following differences.

- After computing $w' = rH(w)$, \mathcal{S} randomly picks a subset T of set $\{1, \dots, n\}$, in which $|T| = t$ is the threshold. Support the selected key servers are $\{\mathcal{K}S_{i_1}, \mathcal{K}S_{i_2}, \dots, \mathcal{K}S_{i_t}\}$, i.e., $T = \{i_1, i_2, \dots, i_t\}$. For each $k \in T$, \mathcal{S} computes $w_k = \prod_{\substack{i_1 \leq j \leq i_t \\ j \neq k, j \in T}} \frac{j}{j-k}$.
- \mathcal{S} creates a transaction shown in Fig. 5.7, where \mathcal{S} plays the role of \mathcal{U} to transfer the service charge to \mathcal{D} 's account (i.e., the service charge is transferred from $A_{\mathcal{S}}$ to $A_{\mathcal{D}}$), and the selected key servers' indexes are attached as the transaction information.
- \mathcal{D} checks whether $\rho_{\mathcal{S}} < \rho$, if yes, \mathcal{D} transfers the corresponding service charge to $\mathcal{K}S_k$ for each $k \in T$, sets $\rho_{\mathcal{S}} ++$, and informs \mathcal{S} ; Otherwise, \mathcal{D} aborts.

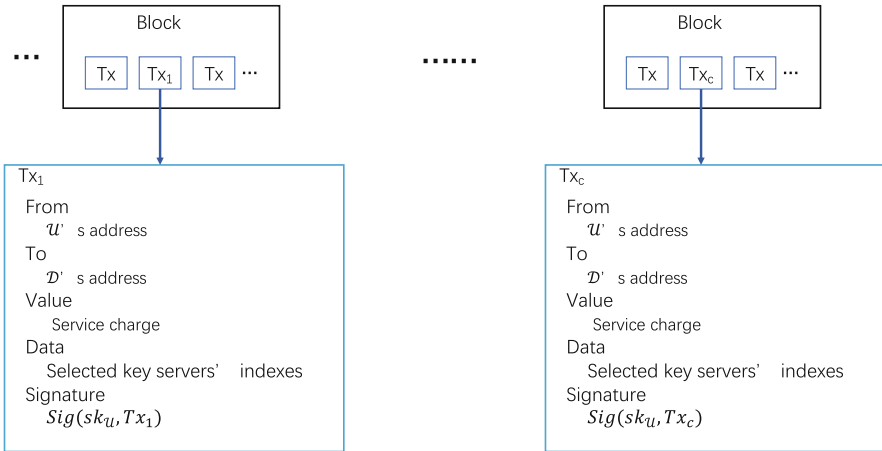


Fig. 5.7 Transactions of transferring service charge in SEPSE

- \mathcal{S} only sends w' to $\mathcal{KS}_{i_1}, \mathcal{KS}_{i_2}, \dots, \mathcal{KS}_{i_t}$.
- After receiving the service charge, \mathcal{KS}_k ($k = i_1, i_2, \dots, i_t$) first obtains the account information of $A_{\mathcal{S}}$ from Ethereum blockchain and obtains $\rho_{\mathcal{S}}$ by extracting the number of transactions that $A_{\mathcal{S}}$ creates (transferring service charge from $A_{\mathcal{S}}$ to $A_{\mathcal{D}}$). \mathcal{KS}_k ($k = i_1, i_2, \dots, i_t$) checks whether $\rho_{\mathcal{S}} < \rho$. If the checking fails, it aborts; otherwise, it generates σ_k on w' as $\sigma_k = s_k w'$, and sends σ_k to \mathcal{S} .

In **Trapdoor**, given a keyword w , \mathcal{R} first generates the servers-derived keyword sd_w with the aid of *only* t key servers. This process is the same as the one performed by \mathcal{S} described above, where \mathcal{R} plays the role of \mathcal{U} (i.e., a service charge is transferred from $A_{\mathcal{R}}$ to $A_{\mathcal{D}}$). With sd_w , \mathcal{R} generates the corresponding trapdoor td_w as the same as that introduced before.

Test and **KeyRenew** are the same as those before, for the sake of brevity, we would not repeat them.

Efficiency Improvement As shown before, the blockchain-assisted rate-limiting mechanism allows users to only communicate with t of key servers rather than all of them without sacrificing the security. This is because integrating each servers-derived keyword query into a transaction on the Ethereum blockchain enables each key server to obtain the total number of queries made by each user in one epoch without interacting with the user. It should be stress again that the blockchain-assisted rate-limiting mechanism is favorable in the case that n is large but t is small. Since creating a transaction in Ethereum takes slight communication and computation costs, if n is *slight more than* t , the communication efficiency on users cannot be improved significantly.

5.4.2 Remark and Further Discussion

Although SEPSE is constructed on the Boneh et al.'s PSE scheme [36], the mechanism that keywords are derived from multiple key servers is also well compatible with other PSE schemes, such as [38, 62], to resist off-line KGA with retention the characteristics of underlying schemes.

The blockchain-assisted rate-limiting mechanism is built on the Ethereum blockchain, in which its security relies on the public verifiability property of Ethereum: the number of transactions created by a user cannot be modified or forged. It is possible to utilize other public blockchains to construct the rate-limiting mechanism, since public verifiability is an inherent property of any secure public blockchain. However, for a public blockchain system, the more participants in it, the stronger the security guarantee it can provide [63]. As Ethereum is one of the most well-established and widely used public blockchains in reality, we recommend using the Ethereum blockchain, which balances the trade-off between security and efficiency.

It is worth to mention that the blockchain-assisted rate-limiting mechanism does not require the key servers, gateway, and users to become a full node in Ethereum, because a light wallet of Ethereum has been issued, which enables one to create transactions without downloading the entire Ethereum blockchain. Moreover, the account information, as well as block content in Ethereum, is being released by multiple sites, platforms, and supernodes of Ethereum, such as Etherscan. It enables the key servers to efficiently extract the account information of senders and the receiver without bearing heavy computation and communication costs.

We also note that the key observation of the success of KGA is that keywords are always selected from small space, and users usually utilize well-known keywords for searches of data. In other words, the number of keywords that a user utilizes would not be too large, and most of the keywords can be predetermined. As such, a user can request her/his commonly used keywords from key servers when the keywords are determined. Furthermore, as discussed before, the same keyword would yield the same servers-derived keyword. Therefore, servers-derived keywords can be reused subsequently. Therefore, the generation of servers-derived keywords can be considered as a one-time operation.

Actually, the idea proposed in DECKS [64] and analogous schemes (e.g., [65]) can also be applied to PSE to resist off-line KGA with freeing it from trusting of a specific group of key servers in a long period of time. We stress that the underlying threat model of DECKS is different from that of SEPSE, where in the later, a specific group of key servers is trusted by users during the entire lifetime of the files to be outsourced and protected.

5.5 Summary and Further Reading

In this chapter, we have introduced the searchable encryption technique for cloud storage systems. We have provided a comprehensive survey on existing SSE schemes and PSE schemes, and analyzed their pros and cons, and also conducted a comparison between them. Finally, we have studied the latest advances of PSE that resists (off-line and online) KGA and discussed their potentials to enhance cloud storage services.

There are some survey papers to introduce the searchable encryption technique from different aspects, e.g., [66–68]. There are also some papers to discuss the relationships between SE and other cryptographic primitives, e.g., [8, 38]. Furthermore, there are also some works that integrate the SE technique into database systems and implements prototypes, e.g., [69, 70].

References

1. Barak B, Goldreich O (2008) Universal arguments and their applications. *SIAM J Comput* 38(5):1661–1694
2. Song D, Wagner D, Perrig A (2000) Practical techniques for searches on encrypted data. In: *IEEE symposium on security and privacy*, pp 44–55
3. Curtmola R, Garay J, Kamara S, Ostrovsky R (2006) Searchable symmetric encryption: improved definitions and efficient constructions. In: *ACM conference on computer and communications security*, pp 79–88
4. Curtmola R, Garay J, Kamara S, Ostrovsky R (2011) Searchable symmetric encryption: improved definitions and efficient constructions. *J Comput Secur* 19(5):895–934
5. Goh E (2003) Secure indexes. *Cryptology ePrint Archive*, Report 2003/216
6. Chang Y, Mitzenmacher M (2005) Privacy preserving keyword searches on remote encrypted data. In: *International conference on applied cryptography and network security*, vol 5, pp 442–455
7. Golle P, Staddon J, Waters B (2004) Secure conjunctive keyword search over encrypted data. In: *International conference on applied cryptography and network security*, pp 31–45
8. Cash D, Jarecki S, Jutla C, Krawczyk H, Roşu M, Steiner M (2013) Highly-scalable searchable symmetric encryption with support for Boolean queries. In: *Annual cryptology conference*, pp 353–373
9. Kamara S, Papamanthou C, Roeder T (2012) Dynamic searchable symmetric encryption. In: *ACM conference on computer and communications security*, pp 965–976
10. Kamara S, Papamanthou C (2013) Parallel and dynamic searchable symmetric encryption. In: *International conference on financial cryptography and data security*, pp 258–274
11. Islam MS, Kuzu M, Kantarcioglu M (2012) Access pattern disclosure on searchable encryption: ramification, attack and mitigation. In: *Network and distributed system security symposium*, pp 1–15
12. Cash D, Grubbs P, Perry J, Ristenpart T (2015) Leakage-abuse attacks against searchable encryption. In: *ACM conference on computer and communications security*, pp 668–679
13. Zhang Y, Katz J, Papamanthou C (2016) All your queries are belong to us: the power of file-injection attacks on searchable encryption. In: *USENIX security symposium*, pp 707–720
14. Stefanov E, Papamanthou C, Shi E (2014) Practical dynamic searchable encryption with small leakage. In: *Network and distributed system security symposium*, pp 1–15
15. Bost R (2016) σ σ σ : forward secure searchable encryption. In: *ACM conference on computer and communications security*, pp 1143–1154
16. Hahn F, Kerschbaum F (2014) Searchable encryption with secure and efficient updates. In: *ACM conference on computer and communications security*, pp 310–320
17. Kim SS, Kim M, Lee D, Park JH, Kim W (2017) Forward secure dynamic searchable symmetric encryption with efficient updates. In: *ACM conference on computer and communications security*, pp 1449–1463
18. Naveed M, Prabhakaran M, Gunter CA (2014) Dynamic searchable encryption via blind storage. In: *IEEE symposium on security and privacy*, pp 639–654
19. Hoang T, Yavuz AA, Guajardo J (2016) Practical and secure dynamic searchable encryption via oblivious access on distributed data structure. In: *Annual conference on computer security applications*, pp 302–313
20. Chase M, Kamara S (2010) Structured encryption and controlled disclosure. In: *International conference on the theory and application of cryptology and information security*, pp 577–594
21. Moataz T, Shikfa A (2013) Boolean symmetric searchable encryption. In: *ACM SIGSAC symposium on information, computer and communications security*, pp 265–276
22. Faber S, Jarecki S, Krawczyk H, Nguyen Q, Rosu M, Steiner M (2015) Rich queries on encrypted data: Beyond exact matches. In: *European symposium on research in computer security*, pp 123–145

23. Sun S, Liu JK, Sakzad A, Steinfeld R, Yuen TH (2016) An efficient non-interactive multi-client searchable encryption with support for Boolean queries. In: European symposium on research in computer security, pp 154–172
24. Boldyreva A, Chenette N, Lee Y, O’Neill A (2009) Order-preserving symmetric encryption. In: International conference on the theory and applications of cryptographic techniques, pp 224–241
25. Boldyreva A, Chenette N, O’Neill A (2011) Order-preserving encryption revisited: improved security analysis and alternative solutions. In: Annual cryptology conference, pp 578–595
26. Chenette N, Lewi K, Weis SA, Wu DJ (2016) Practical order-revealing encryption with limited leakage. In: International conference on fast software encryption, pp 474–493
27. Boneh D, Lewi K, Raykova M, Sahai A, Zhandry M, Zimmerman J (2015) Semantically secure order-revealing encryption: multi-input functional encryption without obfuscation. In: International conference on the theory and applications of cryptographic techniques, pp 563–594
28. Li J, Wang Q, Wang C, Cao N, Ren K, Lou W (2010) Fuzzy keyword search over encrypted data in cloud computing. In: IEEE international conference on computer communications, pp 1–5
29. Wang C, Ren K, Yu S, Urs (2012) Achieving usable and privacy-assured similarity search over outsourced cloud data. In: IEEE international conference on computer communications, pp 451–459
30. Wang C, Cao N, Ren K, Lou W (2011) Enabling secure and efficient ranked keyword search over outsourced cloud data. *IEEE Trans Parallel and Distrib Syst* 23(8):1467–1479
31. Wang C, Cao N, Li J, Ren K, Lou W (2010) Secure ranked keyword search over encrypted cloud data. In: IEEE international conference on distributed computing systems, pp 253–262
32. Bost R, Fouque P, Pointcheval D (2016) Verifiable dynamic symmetric searchable encryption: optimality and forward security. *IACR Cryptol ePrint Archive* 2016:62
33. Kushilevitz E, Ostrovsky R (1997) Replication is not needed: Single database, computationally-private information retrieval. In: IEEE annual symposium on foundations of computer science, pp 364–373
34. Goldreich O, Ostrovsky R (1996) Software protection and simulation on oblivious RAMs. *J ACM* 43(3):431–473
35. Pinkas BB, Reinman T (2010) Oblivious RAM revisited. In: Annual cryptology conference, pp 502–519
36. Boneh D, Crescemzo G, Ostrovsky R, Persiano G (2004) Public key encryption with keyword search. In: International conference on the theory and applications of cryptographic techniques, pp 506–522
37. Boneh D, Franklin M (2001) Identity-based encryption from the Weil pairing. In: Annual cryptology conference, pp 213–229
38. Bellare M, Boldyreva A, O’Neill A (2007) Deterministic and efficiently searchable encryption. In: Annual cryptology conference, pp 535–552
39. Byun J, Rhee H, Park H, Lee D (2006) Off-line keyword guessing attacks on recent keyword search schemes over encrypted data. In: Workshop on secure data management, pp 75–83
40. Chen Y (2014) SPEKS: secure server-designation public key encryption with keyword search against keyword guessing attacks. *Comput J* 58(4):922–933
41. Rhee HS, Park JH, Susilo W, Lee DH (2010) Trapdoor security in a searchable public-key encryption scheme with a designated tester. *J Syst Softw* 83(5):763–771
42. Sun L, Xu C, Zhang M, Chen K, Li H (2018) Secure searchable public key encryption against insider keyword guessing attacks from indistinguishability obfuscation. *Sci China Inf Sci* 61(3):038106
43. Xu P, Jin H, Wu Q, Wang W (2013) Public-key encryption with fuzzy keyword search: a provably secure scheme under keyword guessing attack. *IEEE Trans Comput* 62(11):2266–2277
44. Chen R, Mu Y, Yang G, Guo F, Huang X, Wang X, Wang Y (2016) Server-aided public key encryption with keyword search. *IEEE Trans Inf Forensics Secur* 11(12):2833–2842

45. Chen R, Mu Y, Yang G, Guo F, Wang X (2016) Dual-server public-key encryption with keyword search for secure cloud storage. *IEEE Trans Inf Forensics Secur* 11(4):789–798
46. Bellare M, Keelveedhi S, Ristenpart T (2013) DupLESS: server-aided encryption for deduplicated storage. In: *USENIX security symposium*, pp 179–194
47. Peng Y, Cui J, Peng C, Zuobin Y (2014) Certificateless public key encryption with keyword search. *China Commun* 11(11):100–113
48. He D, Ma M, Zeadally S, Kumar N, Liang K (2017) Certificateless public key authenticated encryption with keyword search for industrial internet of things. *IEEE Trans Ind Inf* 14(8):3618–3627
49. Ma M, He D, Kumar N, Choo KR, Chen J (2017) Certificateless searchable public key encryption scheme for industrial internet of things. *IEEE Trans Indust Inf* 14(2): 759–767
50. Shor PW (1999) Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev* 41(2):303–332
51. Ladd TD, Jelezko F, Laflamme R, Nakamura Y, Monroe C, O’Brien JL (2010) Quantum computers. *Nature* 464(7285):45
52. Zhang X, Xu C, Wang H, Zhang Y, Wang S (2019) FS-PEKS: lattice-based forward secure public-key encryption with keyword search for cloud-assisted industrial internet of things. *IEEE Trans Dependable Secure Comput*. <https://doi.org/10.1109/TDSC.2019.2914117>
53. Brakerski Z, Perlman R (2016) Lattice-based fully dynamic multi-key FHE with short ciphertexts. In: *Annual cryptology conference*, pp 190–213
54. Behnia R, Ozmen MO, Yavuz AA (2018) Lattice-based public key searchable encryption from experimental perspectives. *IEEE Trans Dependable Secure Comput*. <https://doi.org/10.1109/TDSC.2018.2867462>
55. Regev O (2006) Lattice-based cryptography. In: *Annual cryptology conference*, pp 131–141
56. Boneh D, Sahai A, Waters B (2011) Functional encryption: definitions and challenges. In: *Theory of cryptography conference*, pp 253–273
57. Lewko A, Okamoto T, Sahai A, Takashima K, Waters B (2010) Fully secure functional encryption: attribute-based encryption and (hierarchical) inner product encryption. In: *International conference on the theory and applications of cryptographic techniques*, pp 62–91
58. Gentry C (2009) Fully homomorphic encryption using ideal lattices. In: *Proceedings of the forty-first annual ACM symposium on theory of computing*, pp 169–178
59. Zhang Y, Xu C, Ni J, Li H, Shen X (2019) Blockchain-assisted public-key encryption with keyword search against keyword guessing attacks for cloud storage. *IEEE Trans Cloud Comput*. <https://doi.org/10.1109/TCC.2019.2923222>
60. Herzberg A, Jarecki S, Krawczyk H, Yung M (1995) Proactive secret sharing or: how to cope with perpetual leakage. In: *Annual cryptology conference*, pp 339–352
61. Wood G (2014) Ethereum: a secure decentralised generalised transaction ledger. *Ethereum Project Yellow Pap* 151:1–32
62. Abdalla M, Bellare M, Catalano D, Kiltz E, Kohno T, Lange T, Malone-Lee J, Neven G, Paillier P, Shi H (2005) Searchable encryption revisited: consistency properties, relation to anonymous IBE, and extensions. In: *Annual cryptology conference*, vol 3621, pp 205–222
63. Zhang Y, Xu C, Lin X, Shen X (2019) Blockchain-based public integrity verification for cloud storage against procrastinating auditors. *IEEE Trans Cloud Comput*. <https://doi.org/10.1109/TCC.2019.2908400>
64. Zhang Y, Xu C, Cheng N, Shen X (2019) Secure encrypted data deduplication for cloud storage against compromised key servers. In: *IEEE global communications conference*, pp 1–6
65. Zhang Y, Xu C, Li H, Yang K, Cheng N, Shen X (2020) PROTECT: efficient password-based threshold single-sign-on authentication for mobile users against perpetual leakage. *IEEE Trans Mob Comput*. <https://doi.org/10.1109/TMC.2020.2975792>
66. Bösch C, Hartel P, Jonker W, Peter A (2014) A survey of provably secure searchable encryption. *ACM Comput Surv* 47(2):Article 18
67. Li H, Liu D, Dai Y, Luan TH (2015) Engineering searchable encryption of mobile cloud networks: when QOE meets QOP. *IEEE Wireless Commun* 22(4):74–80

68. Ren K, Wang C, Wang Q (2012) Toward secure and effective data utilization in public cloud. *IEEE Netw* 26(6):69–74
69. Popa RA, Redfield C, Zeldovich N, Balakrishnan H (2011) CryptDB: protecting confidentiality with encrypted query processing. In: *ACM symposium on operating systems principles*, pp 85–100
70. Schuster F, Costa M, Fournet C, Gkantsidis C, Peinado M, Mainar-Ruiz G, Russinovich M (2015) VC3: trustworthy data analytics in the cloud using SGX. In: *IEEE symposium on security and privacy*, pp 38–54

Chapter 6

Secure Data Provenance



This chapter introduces the secure data provenance technique. We mainly focus on secure data provenance schemes for cloud storage systems. In the subsequent sections of this chapter, we first introduce the basic paradigms and principles of secure data provenance. Then, we give a comprehensive survey on the secure data provenance schemes. Finally, we review the latest advances in secure data provenance and introduce how to construct a secure data provenance scheme on a public blockchain.

6.1 Introduction to Secure Data Provenance

Digital investigations play an important role in any information system. In regards to cloud storage systems, as users cannot physically control the data once they outsource the data to the cloud server, data investigations, which clearly and securely reflect the statement of outsourced data, are essential for the success of cloud storage.

Data provenance is one of the essential techniques in data investigations. It records the history of the ownership and process of a file during its lifecycle. Secure data provenance is an enhanced technique considering the threats towards the underlying data, which ensures data provenance while protecting the provenance information from various attacks.

In this section, we introduce the secure data provenance technique, especially that is designed for cloud storage systems. We first introduce the data provenance technique and then introduce the *secure* data provenance technique and analyze their relationship.

6.1.1 *Data Provenance vs. Secure Data Provenance*

Data provenance is an important technique in the database. It refers to a record trail that accounts for the origin of a piece of data (in a database, document, file, or repository) together with an explanation or description of how and why it got to the present place [1, 2]. Data provenance can be regarded as the derivation from a particular source to a specific state of an item, where the explanation of such a derivation might take different forms or record different information depending on users' individual interest [3]. Data provenance can be utilized in various cloud-based applications to provide their users a transparent, understandable, and investigation-enabled data management service.

We enumerate some instances of cloud-based services that are enhanced by integrating the data provenance in the following.

Cloud-Based eHealth Systems Recall Sect. 4.5, in a cloud-based eHealth system, all EHRs are generated by doctors and outsourced to the cloud server [4]. The correctness and integrity of outsourced EHRs are being put at risk in reality. This problem is further exacerbated by the fact that patients' EHRs always need to be updated. Therefore, how EHRs are modified and why they should be modified in such a way are very important, which not only provides significant information for subsequent diagnoses, but also makes a great contribution to the judgment and dispute resolution in medical malpractice. With the data provenance technique, EHRs generated in cloud-based eHealth systems can be traced, which enhances the functionality and security of the systems significantly.

Cloud-Based Supply Chain Generally, a cloud-based supply chain is a network between a company and its suppliers to produce and distribute a specific product to customers, where the data generated in the process are outsourced to the cloud server and are shared, accessed, and operated via the cloud storage service [5]. The data provenance technique has an oblivious value for a cloud-based supply chain system, since it enables the company, suppliers, and customers in the system to check and verify the information related to the product and producing, which provides a transparent product tracing interface for the entities in the system.

Cloud-Based Data Sharing Systems The data sharing service is one of the most featured services in cloud storage systems. Typical applications include cloud-based open source software development systems (e.g., Codeanywhere [6]) and cloud-based collaboration systems (e.g., Google Doc [7]). In such an application scenario, a file (e.g., software code, system bug report, and work document) is generated by a data owner and is outsourced to the cloud server. Subsequently, some authorized users can access and edit the file. The data provenance technique enables any change of the file's statement to be recorded, which not only allows any authorized user to keep track of what happened to the outsourced file and to learn how the file is generated, but also supports post investigations on the outsourced file.



Fig. 6.1 Data provenance

Cloud-Based Internet of Things Cloud-based Internet of Things (IoT) has been widely applied in our daily life. For example, cloud-based smart home systems are based on IoT and provide residents with a safe, comfortable, and convenient living environment [8, 9]. In such a system, several IoT devices are deployed, and data are generated by these devices and outsourced to the cloud server. Devices and apps might be chained together in long sequences of trigger-action policies to the point that from an occurrence of some events (e.g., the door is unlocked, or smoke rises up). However, due to the existence of malicious apps and adversaries, a log file recording the statements of IoT devices during a long period of time should be maintained to detect the misbehaviors and to adjust the policies. The log file is essentially formed by the provenance information about the IoT devices.

At a high level, data provenance for cloud storage systems keeps track of what happens to a target outsourced data throughout its lifecycle (from creation to destruction or deletion), such as its ownership and custodial history as well as how it has been accessed and modified by the authorized users, as shown in Fig. 6.1, where each record provides sufficient information about how the statement of the target data is changed between two successive stages.

The data provenance technique also serves as a key component in digital investigations. For example, in a digital investigation, digital evidences must be strictly secured and clearly documented about its ownership transfer as well as how it was handled during its lifecycle. It is not unusual that the defendant challenges the authenticity of a digital evidence during the trial. It has been proved very useful to solve crimes by using the data provenance technique. A good example of it is the BTK killer case [10]. Whenever a Microsoft office document is created, Microsoft office automatically embeds an author name into the document. In the BTK killer case, the killer is traced and identified by policemen, due to a document that is created by the killer and where the killer's name is embedded.

In reality, the most common types of digital evidence are hard disk images, and the defendant may question the hard disk image that investigators are working on and presented in the courtroom is not the same one acquired from the hard disk found at the crime scene.

In the past decades, many security mechanisms have been developed to ensure the security and privacy of sensitive or confidential information, as well as achieve accountability and auditability through data access logging or audit trails [11], such as logging activities on data creation, modification, and access. Much of the focus has been on protecting digitally stored information from unauthorized

use or modification. However, despite extensive research on information security and privacy, little attention has been paid to securing provenance information and providing assurance that a data file is trustworthy. It is worth mentioning that as the current best of practice, log files are also protected from tampering and illegal access. For instance, in the banking industry, any activities, such as bank transfers, can only be recorded by creating a new log, and past logs cannot be modified or deleted for security reasons. Nevertheless, another important question still needs to be answered about whether the provenance information can be trusted to make sure that the corresponding data file is a trusted one after a series of user activities on the file which have been detailed in the provenance information.

Unlike the traditional file access auditing where file access activities are logged, provenance information contains the ownership history of data files as well as activities that occurred on these files by their owners or users. Furthermore, such information is organized in chronological order during the lifecycles of the data files, allowing to track accesses and activities of data files. As a result, it not only improves accountability and reliability [12], but also meets the requirements of emerging applications, such as maintaining the digital chain of custody in a digital forensics investigation [13, 14], as well as regulatory compliance requirements and industry standards, such as the Health Insurance Portability and Accountability Act (HIPAA) [15].

Actually, provenance information is not useful if it cannot be trusted. It is inadvisable to trust provenance information without proper protection. Recall the BTK killer case, although the identity of file creator would be automatically embedded into the file, this identity can be arbitrarily modified and removed by the creator. In regarding to the cloud storage service, this problem is further exacerbated by the fact that data files and the corresponding provenance records are outsourced to the cloud storage which is essentially untrusted from the perspective of users, since the data files and the provenance information would not be physically owned by data owners and are transmitted over an insecure network [16, 17]. Hence, it is crucial to ensure provenance information security.

The secure data provenance technique [18] focuses on ensuring the security of provenance information of a target file. Here, the difference between the data provenance technique and the secure data provenance technique is clear. The former mainly investigates how to generate provenance information for a target system to provide sufficient information about the target file. In contrast, the latter mainly investigates how to secure the generated provenance information for a target system to protect the provenance information from forgery, modification, substitution, and deletion.

6.1.2 *System and Threat Models*

In the secure provenance for cloud storage systems, the system model is different from that of other data protection schemes for cloud storage systems, which will be elaborated in the following.

Note that different from traditional data storage systems where the data and the corresponding provenance information are stored locally on the user side, in cloud storage systems, data and the corresponding provenance information are outsourced to a cloud server which is a separate administrative entity and might be compromised in reality. As a consequence, both the data and the corresponding provenance information are being put at risk.

Recall that a secure data provenance scheme should be secure against provenance forgery, modification, substitution, and deletion, the first challenge to design a secure data provenance scheme is to address the “authentication” problem. Specifically, if a proper authentication mechanism is not employed, it is impossible to ensure the security of provenance information, since any adversary who colludes with the cloud server can generate a provenance record to substitute existing ones arbitrarily. Therefore, delegating and authenticating the qualification of a user to generate the provenance record should not be performed by the cloud server, due to the security reasons.

The second challenge to design a secure data provenance scheme is to address the “privacy” problem. Particularly, each provenance record explicitly describes the change of statement for a target file. In other words, each provenance record originally contains the identity of the user who performs operations on the file. However, this information is sensitive in practice, since if the cloud server can learn who performs operations on the target file, it may extract more privacy about the user from the learned information. A straightforward way to address this problem is to utilize an anonymous mechanism, where each user randomly chooses an element from Z_p^* as her/his pseudonym such that the cloud server can learn nothing about the user’s identity from the pseudonym. However, the primary goal of data provenance is to support data investigations in cloud storage systems. If an adversary launch attacks to break the security of the data provenance scheme, such a misbehavior cannot be tracked, and the identity of the adversary cannot be identified, due to the anonymous mechanism, which deviates from the primary goal of secure data provenance schemes. As such, a concept of “conditional privacy preservation” is proposed to meet the privacy requirement in secure data provenance for cloud storage systems. Conditional privacy preservation refers to a requirement that only a trusted authority (who is qualified to perform the data investigation) can reveal the real identities recorded in provenance records, while anyone else cannot.

At a high level, from the above analysis, we can conclude that a secure data provenance scheme should achieve the following primary goals.

- Security of provenance records. A genuine provenance record is outsourced to the cloud server and can reflect the corresponding statement of the target data. Any adversary cannot forge, tamper with, and delete a valid provenance record.

- **Conditional privacy preservation.** To preserve users' privacy, any adversary cannot extract the identity information (i.e., who generates the provenance record) from a genuine provenance record. In addition, when the provenance records are used in a data investigation where a trusted authority participates in, the real identities of users who generate the provenance records can be easily retrieved.

To address the above problem, existing secure provenance schemes [19, 20] employ an independent identity manager to secure the outsourced provenance information. In such a paradigm, any operation on the data performed by a user would correspond with a provenance record and is required to be authorized by the identity manager. In other words, any user who wants to generate a provenance record should first be authenticated by the identity manager, and the validity of the provenance record should be verified by the identity manager. The identity manager is responsible for all provenance records.

Generally, there are four entities in a secure data provenance scheme: identity manager, cloud server, users, and auditor, as shown in Fig. 6.2.

- **Identity manager.** The identity manager is a powerful entity and located at the top of the cloud storage system. It manages the users and authorizes them to perform operations on the outsourced file and to generate provenance records. It

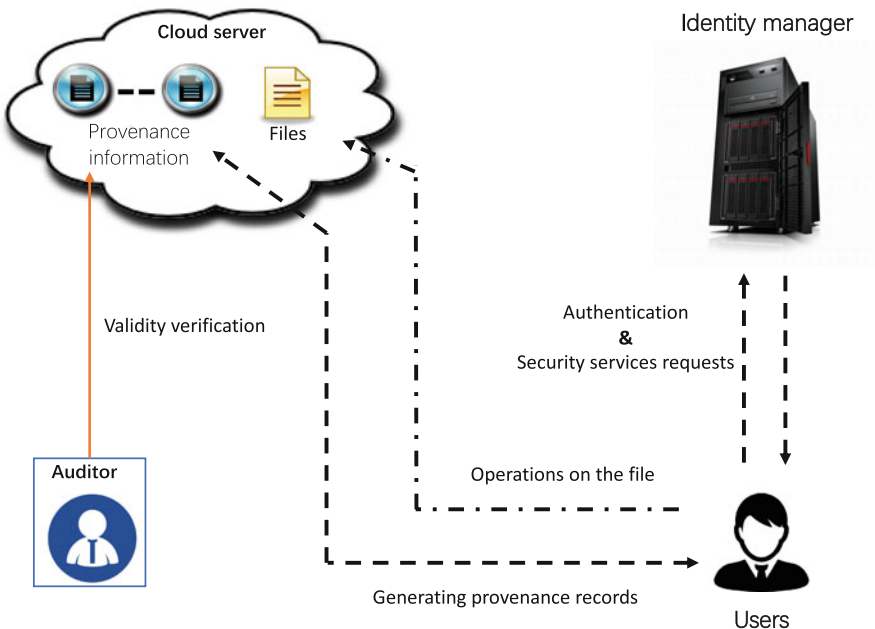


Fig. 6.2 System model of secure data provenance

also assists users in protecting their identity from leakage, and helps the auditor retrieving the real identity of users who generate the provenance records.

- **Cloud server.** The cloud server is subject to the cloud storage service provider and provides storage services for users. It also provides users with interfaces to enable them to perform various operations on outsourced data and records the corresponding provenance information (i.e., how the data is processed and who performs the operation on the data).
- **Users.** In a secure data provenance scheme, we consider the multi-user and single-file case. Specifically, a group of users exists, and these users register with both the identity manager and the cloud server, these users are delegated to process a specific file M .
- **Auditor.** The auditor is subject to an authority. In a data investigation, the auditor checks the validity of the provenance records.

In a secure data provenance scheme, we consider threats from two different angles. The one targets at breaking the security of provenance records, and the other one targets at violating users' privacy.

Generally, both the auditor and identity manager are honest and trustable. The data the auditor uses for investigations are publicly verifiable, and their validity can be easily verified by any participant. Hence, if the auditor misbehaves, anyone is able to detect it easily. The identity manager is employed to secure the provenance information. It is responsible for the security of provenance records.

On the other hand, both the users and the cloud server might perform attacks to break the security of the data provenance scheme. Both the users and the cloud server attempt to forge and delete valid provenance records for profits; the cloud server may also violate the users' privacy by extracting the real identities of users from the outsourced provenance records.

6.2 Survey on Secure Data Provenance

Data provenance provides sufficient information about target data that what happens to the data from creation to destruction, e.g., its ownership, custodial history, access pattern, and modification records. As we are moving into the age of big data where digital data are explosively generated nowadays, and most data are managed via the Internet with the aid of cloud storage systems, data provenance becomes pretty important to information forensics and digital investigations than ever. Once a dispute arises in outsourced data, provenances serve as the most vital evidence for post investigations.

Data provenance has been investigated extensively in the past, especially for the special data field, such as e-science data and geospatial data [1, 3, 21–24].

While most works on data provenance have focused on constructing systems with rich functionalities and high efficiency, they are also confronted with critical security issues. Lynch [18] first points out the need for trust and provenance in information

retrieval. Hasan et al. [25] first define the problem of secure provenance and argued that it is of vital importance in practice, in which the main challenges in trustworthy provenance are identified, a preliminary adversarial model is defined, and the potential security and privacy issues related to securing provenance information from the considered adversary are analyzed. Specifically, in the Hasan et al.'s work, the secure provenance problem is defined as the tasks of guaranteeing the integrity, confidentiality, and availability of provenance records. The security of secure data provenance requires that

- unauthorized parties cannot obtain any information from the provenance records (confidentiality);
- adversaries cannot forge valid provenance records (security of provenance records);
- authorized auditors can verify the validity of provenance records in terms of integrity and correctness, without knowing the contents of individual records;
- each user is provided with an option that her/his privacy can be preserved by masking her/his name in the corresponding provenance record.

However, Hasan et al.'s work focuses on traditional storage systems, and its threat model and security requirements cannot well suitable for cloud storage systems.

The first secure data provenance scheme for cloud storage systems is proposed by Lu et al. [19]. In this work, the authors demonstrate that the secure data provenance technique serves as the essential bread and butter of data investigations in cloud storage systems, and introduce the formal definition and security notions of secure data provenance for cloud storage systems. The system model of secure data provenance for cloud storage systems is also presented in this work, where the identity manager is first employed to secure the provenance records. To ensure the security of provenance records in cloud storage systems, Lu et al. construct a secure data provenance scheme on a group signature scheme. Group signature is a specific type of signature, where a group of signers and a group manager are involved in the system. It has the following properties:

- Only signers in the group can sign a message;
- Anyone can verify the validity of a group signature but cannot learn which signer in the group signs it;
- If necessary, the signature can be “opened” with the aid of the group manager such that the real identity of the signer in the group who generates the signature can be retrieved.

It is obvious that the inherent properties of the group signature can be utilized to construct secure data provenance schemes in cloud storage systems. The group signature in Lu et al.'s scheme is based on bilinear pairing such that the size of the signature is reduced significantly compared with its RSA counterparts.

On the other hand, the Lu et al.'s work only supports the static-user case, where the qualified users who can perform operations on a target file should be determined in the phase of system initialization. However, the support of dynamic user management is an inherent requirement for any practical cloud storage system,

where the users who can perform operations on the target file can be dynamically adjusted to meet different requirements in reality.

To address this problem, Chow et al. [20] propose a secure data provenance scheme for cloud storage systems supporting dynamic user management, where the users who are qualified to perform operations on a target data form a group, and the group member can be added and revoked. This makes the secure data provenance scheme flexible. The key techniques behind the Chow et al.'s scheme are revocable group signature and dynamic broadcast encryption. Broadcast encryption is a multi-recipient encryption scheme that enables the encryptor to decide who is allowed to decrypt the ciphertext. Dynamic broadcast encryption is an enhanced version of broadcast encryption where the encryptor is able to re-encrypt or update the ciphertext outsourced to the cloud storage such that the decryption set of outsourced ciphertexts can be dynamically adjusted. With the integration of revocable group signature and dynamic broadcast encryption, the secure data provenance scheme proposed by Chow et al. ensures the security of provenance records and supports dynamic user management simultaneously.

Following the Lu et al.'s scheme [19] and the Chow et al.'s scheme [20], several secure data provenance schemes for cloud storage systems are proposed. These schemes essentially share the same system and threat models, where the identity manager is employed to secure provenance records. However, such a paradigm is confronted with a strong assumption that the identity manager is honest and reliable. Once the identity manager is compromised, the security of these schemes is broken. Particularly, if the cloud server or a misbehaved user colludes with the identity manager, the outsourced provenance records can be modified without detection. In reality, compromising the identity manager is feasible for adversaries, since an adversary can perpetually incentivize the identity manager to deviate from the prescribed scheme over a long period of time.

6.3 Blockchain: A Panacea for Secure Data Provenance

It seems that resistance against compromised identity manager cannot be addressed without introducing any trusted entity, since any provenance record should be authorized by the identity manager before it is published. However, with the booming development of the blockchain technology, it has demonstrated that adopting blockchains in cloud storage systems can enhance data security significantly [16, 17, 26, 27]. We note that public blockchains can serve as a panacea for secure data provenance.

Recall that the most prominent manifestation of public blockchains is on-chain currencies, e.g., Bitcoin [28] and Ethereum [29], which provides a secure way to conduct transactions without a central authority (i.e., bank). Interestingly, the immutability of public blockchains can itself be seen as a secure data provenance problem, where the underlying ledger of a public blockchain essentially maintains the provenance information about the underlying currency, i.e., it keeps track of the

ownership of each underlying value token. Therefore, utilizing a public blockchain to construct a secure data provenance scheme for cloud storage systems is very promising.

In this section, we study a blockchain-based secure data provenance scheme for cloud storage systems, called ESP [30].

6.3.1 Blockchain-Based Secure Data Provenance

The primary motivation of ESP [30] is to address the single-point-of-failure problem of existing secure data provenance schemes [19, 20]. In these schemes, the security of outsourced provenance records relies on the security and reliability of the identity manager.

As discussed before, a secure data provenance scheme goes one step beyond a data provenance scheme, that is, there is an underlying data provenance model behind a secure data provenance scheme. Therefore, designing a data provenance model is a preparatory work for proposing a secure data provenance scheme.

With the analysis of cloud storage systems, a model of data provenance is formalized, in which the lifecycle of outsourced files is formally formulated. Based on the data provenance model and with the integration of the public blockchain, an efficient and secure data provenance scheme, i.e., ESP, for cloud storage systems is presented. Furthermore, a concept of window of latching (short for WoL) is proposed to measure the practicality of ESP as well as other secure data provenance schemes.

6.3.1.1 A Model of Data Provenance

Lifecycle of an Outsourced File and Its Users In defining a data provenance model, an outsourced file's lifecycle is first considered. Actually, the lifecycle can be viewed as a sequence of stages from the file creation to modification, destruction, and ownership transfer.

Figure 6.3 illustrates the file lifecycle considered in this work. After a file is created, it may go through many stages due to the file modification or ownership transfer. Finally, a file may be destructed or securely deleted, becoming unavailable to its users. Thus, an individual state of a file can be uniquely identified by its content and owner and can be represented as $St_i = H(M_i, O_i)$, where St_i denotes a state where a file has been at, M_i denotes the content of the file at the state of St_i , O_i is the owner of the file at the state of St_i , and H is a secure hash function.

With the file lifecycle, the roles of the users involved in the process of the file lifecycle should also be analyzed.

During the lifetime of an outsourced file, users can play different roles in it, and can be generally classified into four types: *creator*, *owner*, *editor*, and *viewer*.

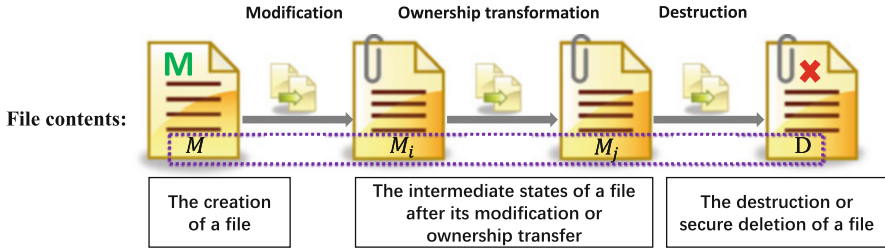


Fig. 6.3 File lifecycle

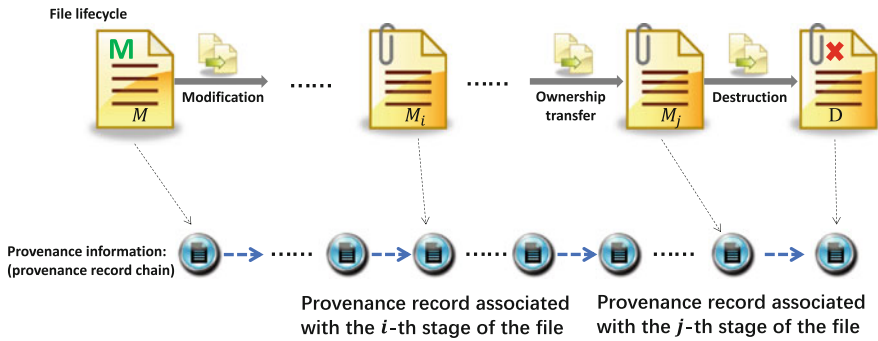


Fig. 6.4 Provenance model

- **Creator:** a user is the creator of a file, she/he is the first one to outsource the file to the cloud server.
- **Owner:** a user is the owner of the outsourced file. By taking ownership of the file, the user can assign other users access rights or permissions to the file, including editing (or modifying) and viewing (or reading) the file, and transferring ownership of the file. By default, the creator of an outsourced file is also its owner, and file ownership can be transferred to another user by its current owner.
- **Editor:** a user has the capability of editing (or modifying) the outsourced file.
- **Viewer:** a user can only view (or read) the outsourced file.

Files can have many editors and viewers, but only one creator during their lifetime and one owner at a time. In addition to the four aforementioned types of users, there also exists an auditor who can verify the validity and trustworthiness of any provenance information but without any knowledge of the user’s identity who generates each individual provenance record, which follows existing secure data provenance schemes [19, 20].

Provenance Model As shown in Fig. 6.4, in the data provenance, provenance information is organized into a chain in chronological order, where each chain item represents a provenance record which details how an outsourced file was processed

at every stage of its lifecycle. Each provenance record is also associated with a specific file stage, and a legitimate user (e.g., editors) may perform many operations on outsourced files. A typical provenance record consists of a specially formatted data block that contains information related to how an outsourced file is processed at a time as well as its ownership information, which usually can be classified into two types:

Essential Provenance Data (EPD) Information related to activities performed on the outsourced file.

Nonessential Provenance Data (NPD) Security overhead which has been generated by security mechanisms that are used in a data provenance system to secure provenance records.

Measure the Practicality of Secure Provenance With the provenance model, we note that although an outsourced file can be processed by multiple users during its lifecycle, these users would process the file one by one. In other words, when a user processes the target file, the file is “locked” for other users, even if these users are qualified to process the file.

As such, a concept of window of latching (WoL) is introduced to evaluate the practicality of secure provenance schemes. Actually, window of latching (WoL) means the time interval between two successive provenance records that are accepted and published, i.e., it is a period of time when a file is unavailable to other users. The shorter WoL, the more practical the secure provenance scheme.

6.3.1.2 System and Threat Models

The system model of ESP follows existing schemes [19, 20] as shown in Fig. 6.2, where a cloud server CS , a group of users $\mathcal{U} = \{u_1, u_2, \dots\}$, an identity manager \mathcal{IM} , and a third-party auditor \mathcal{A} are involved.

However, different from existing schemes [19, 20], ESP considers a stronger threat model than them: the identity manager is not fully trusted by others. With the threat model, ESP mainly considers the following security and privacy threats against provenance records:

- Unauthorized disclosure. Unauthorized users have access to provenance records.
- Provenance record forgery attack. A malicious user may collude with others to forge a valid provenance record, which could hinder the examination of provenance records.
- Provenance record removal attack. A malicious user colludes with others to remove one or several existing provenance records that have been generated due to the operations performed on outsourced files.
- Modification attack. Similar to the two above threats, a malicious user may collude with others to tamper with provenance records by modifying them or changing their sequences.

- Repudiation attack. A malicious user may deny that he performed an operation on an outsourced file.
- Privacy violation. Privacy violation refers to the attack that the identity of a user who generates a provenance record is leaked out. Recall that in secure provenance schemes, only conditional privacy preservation should be ensured, where the identity manager has the ability to reveal the real identity recorded in a provenance record, while anyone else cannot.

6.3.1.3 Overview of ESP

ESP consists of three parts: system setup, secure provenance generation, and secure provenance verification.

In the first part, system parameters are generated. The identity manager assigns a human-memorizable password to each user and maintains a list that records the assigned passwords and the corresponding identities. With the list, the identity manager is able to authenticate each user securely and efficiently. This authentication between the identity manager and each user enables the former to control who can access the outsourced files and can generate provenance records. In ESP, this “access control” is controlled by the identity manager.

When a user wants to process a target outsourced file, she/he needs to be authorized by the identity manager at first. Then the identity manager assists the user in generating a provenance record. This allows the user to prove herself/himself to the cloud server that she/he is qualified to process the file. The key technique behind ESP is to integrate each provenance record into a transaction on a public blockchain, e.g., Ethereum, where the user transfers a service charge to the authenticated server, which makes the security of a provenance record to be related to the security of a transaction on Ethereum. As a consequence, the advantage that an adversary breaks the security of a provenance record is essentially equal to the one that he breaks the security of the underlying blockchain system.

Note that a straightforward way to integrate provenance records into transactions of Ethereum is to set the hash value of each provenance record as the data value of a transaction on Ethereum. However, this causes heavy computational costs for the auditor, since when the auditor verifies the validity of provenance records, in addition to verifying the correspondence between the hash values and the provenance records, the auditor also needs to verify the validity of all provenance records one by one, which essentially requires the auditor to verify multiple signatures and is very costly in practice.

To improve efficiency significantly, in ESP, all provenance records are chained together as a whole with the aid of the Ethereum blockchain, as illustrated in Fig. 6.5, where the provenance record chain is indicated by dashed gray lines. Assume that there currently are n provenance records, $\{P_1, P_2, \dots, P_n\}$, in which each of them stands for a state of the underlying file at the corresponding stage during its lifecycle as modeled in Sect. 6.3.1.1. They are chained together as follows: from the second provenance record P_2 , each record contains a data field that points

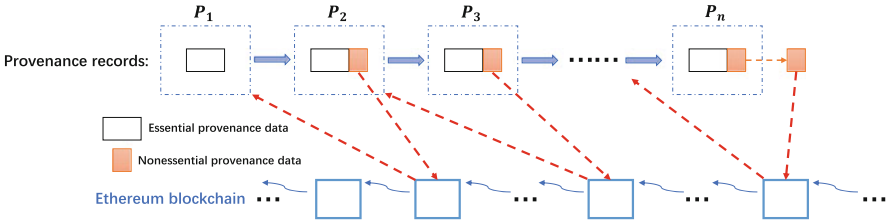


Fig. 6.5 Ethereum-aided provenance record chain

to a block on the Ethereum blockchain, this block relates to the last provenance record. Each record is appended to the last one until it reaches the last one of the current provenance record, P_n , and each record would be signed by the identity manager. The signature of the last record becomes the tail of the provenance record chain. In this case, if any existing record is modified or removed, the provenance record chain is broken. The computational costs to verify provenance records in ESP mainly depend on the hashing operation along with one signature verification for the last element or the tail of the provenance record chain, which is very efficient. As a result, the verification is very fast.

Note that due to the employment of the public blockchain, as long as the identity manager remains inaccessible to adversaries, ESP guarantees both the security and privacy preservation. If both the identity manager and cloud storage server are compromised, ESP retains the security assurance on the provenance records in existing schemes (i.e., resistance against forgery, removal, modification, and repudiation on provenance records).

6.3.1.4 Construction of ESP

System Setup With the security parameter ℓ , the system parameters $\{p, G, G_T, P, e, E(\cdot), h, H\}$ are determined, where G is an additive group whose generator is P , $e : G \times G \rightarrow G_T$, G and G_T have the same prime order p , $E(\cdot)$ is a secure symmetric encryption algorithm, $h : \{0, 1\}^* \rightarrow Z_p^*$, and $H : \{0, 1\}^* \rightarrow G$. \mathcal{IM} randomly chooses $s \in Z_p^*$, and computes $P_{pub} = sP$ and $k = h(s)$. \mathcal{IM} 's secret keys are (s, k) , the corresponding public key is P_{pub} . For each $\mathcal{U}_i \in \mathcal{U}$ with an identifier ID_i , she/he registers with \mathcal{IM} , where a human-memorizable password pwd_i is generated. After the registration, \mathcal{IM} stores (ID_i, pwd_i) locally for subsequent authentications.

Secure Provenance Generation

Once a user \mathcal{U}_i processes a file outsourced to \mathcal{CS} and generates a provenance record P_j , she/he will request \mathcal{IM} to generate provenance a record on the file process.

Phase 1 With the identifier ID_i and password pwd_i \mathcal{U}_i makes mutual authentication with \mathcal{IM} to establish a secure channel. Specifically, \mathcal{U}_i randomly selects

$r_1, a \in Z_p^*$, obtains the current timestamp ct , and computes C_1, C_2 , where $C_1 = r_1 P$, $C_2 = E_k(ID_i || pwd_i || aP || ct)$, $k = r_1 P_{pub}$. Then, \mathcal{U}_i sends (C_1, C_2) to \mathcal{IM} . After receiving (C_1, C_2) , \mathcal{IM} computes $k = sC_1 = sr_1 P = r_1 P_{pub}$, extracts $ID_i || pwd_i || aP || ct$ from C_2 with k , and verifies the validity of the timestamp ct to resist the replay attack. \mathcal{IM} authenticates \mathcal{U}_i by checking the correctness of (ID_i, pwd_i) . Then, \mathcal{IM} randomly chooses $b \in Z_p^*$, computes $sk = b(aP)$ as the session key, and computes \mathcal{U}_i 's pseudonym $PID_j = E_k(ID_i || ct || b)$, C_3 , and C_4 , where $C_3 = bP$, $C_4 = E_{sk}(ID_i || aP || bP || ct || PID_j)$. Finally, \mathcal{IM} sends (C_3, C_4) to \mathcal{U}_i . With (C_3, C_4) , \mathcal{U}_i computes the session key $sk = aC_3 = abP$, extracts $ID_i || aP || bP || ct || PID_j$ from C_4 with sk , and authenticates \mathcal{IM} and confirms the correctness of sk by verifying the correctness of $ID_i || ct || aP || bP$. Since the session key sk is shared between \mathcal{U}_i and \mathcal{IM} , a secure channel between them is established for secure provenance.

Phase 2 Different roles of \mathcal{U}_i require different execution between \mathcal{U}_i and \mathcal{IM} .

Creator \mathcal{U}_i creates a new file

If \mathcal{U}_i creates a new file M , i.e., the provenance record P_j is $P_1 = h(M_1 || ID_i)$, where M_1 denotes the content of the file at the first state, she/he requests a secure provenance from \mathcal{IM} as follows.

\mathcal{U}_i sends P_1 to \mathcal{IM} via the secure channel. \mathcal{IM} extracts PID_1 from local storage (i.e., $j = 1$), and signs P_1 and PID_1 as $\sigma_{T_1} = sH(P_1 || PID_1)$, and sends σ_{T_1} to \mathcal{U}_i . \mathcal{U}_i verifies the validity of σ_{T_1} by checking $e(\sigma_{T_1}, P) \stackrel{?}{=} e(H(P_1 || PID_1), P_{pub})$. If the verification fails, she/he rejects σ_{T_1} . Otherwise, \mathcal{U}_i creates a transaction T_{x_1} shown in Fig. 6.6, where \mathcal{U}_i transfers service charge to the \mathcal{IM} 's account, and the data field of the transaction is set to $h(h(P_1 || PID_1) || \sigma_{T_1})$. After the transaction T_{x_1} is recorded into the Ethereum blockchain, \mathcal{U}_i sends $(P_1 || PID_1 || Bl_1, \sigma_{T_1})$ to \mathcal{CS} , and publishes it as the first provenance record.

Editor/viewer \mathcal{U}_i edits/views an existing file

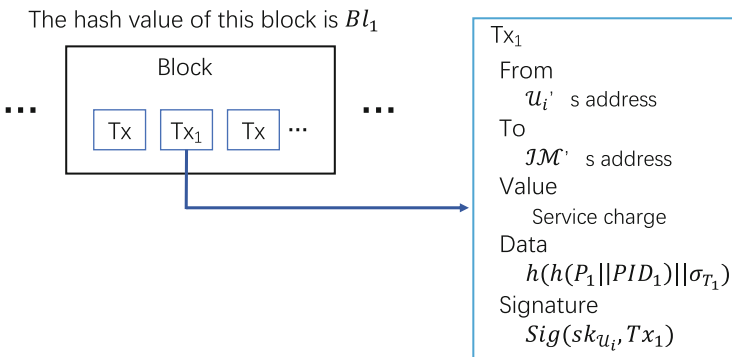


Fig. 6.6 The transaction created by the creator

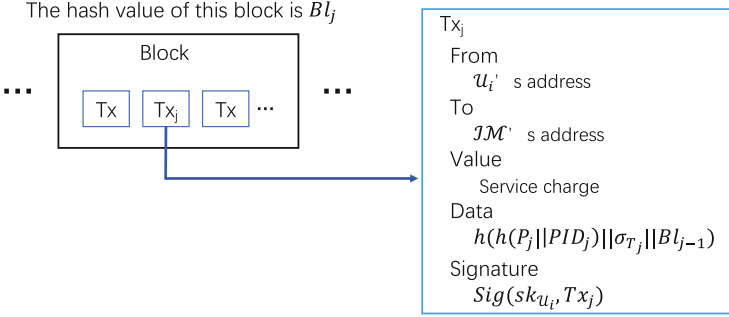


Fig. 6.7 The transaction created by the editor/viewer

If \mathcal{U}_i edits/views an existing file, without loss of generality, we assume the underlying file is M whose state at the first stage is M_1 , i.e., the provenance record P_j with $j \geq 2$ is $P_j = h(M_j||ID_i)$, where M_j denotes the content of the file at the j -th stage. \mathcal{U}_i interacts with \mathcal{IM} as follows.

\mathcal{U}_i sends $(P_j, Bl_{j-1}, \sigma_{T_{j-1}})$ to \mathcal{IM} via the secure channel, where Bl_{j-1} denotes the hash value of the block that contains the transaction whose data field is $h(h(P_{j-1}||PID_{j-1})||\sigma_{T_{j-1}})$. \mathcal{IM} checks the validity of Bl_{j-1} , if the checking fails, it rejects Bl_{j-1} . \mathcal{IM} extracts PID_j from local storage, computes $\Theta(P_j) = H(P_j||PID_j||Bl_{j-1})$, signs $\Theta(P_j)$ as $\sigma_{T_j} = s \cdot \Theta(P_j)$, and sends σ_{T_j} to \mathcal{U}_i . \mathcal{U}_i verifies σ_{T_j} by checking whether $e(\sigma_{T_j}, P) \stackrel{?}{=} e(\Theta(P_j), P_{pub})$, creates a transaction Tx_j shown in Fig. 6.7, where \mathcal{U}_i transfers service charge to the \mathcal{IM} 's account, and the data field of the transaction is set to $h(h(P_j||PID_j)||\sigma_{T_j}||Bl_{j-1})$. After the block containing the transaction Tx_j is chained to the Ethereum blockchain, \mathcal{U}_i sends $(P_j||PID_j||Bl_j, \sigma_{T_j}, Bl_{j-1})$ to \mathcal{CS} , and publishes it as the provenance record.

Finally, the provenance records of the outsourced file M becomes

$$(P_1||PID_1||Bl_1, P_2||PID_2||Bl_2, \dots, P_j||PID_j||Bl_j, \sigma_{T_j}),$$

where $P_1||PID_1||Bl_1, P_2||PID_2||Bl_2, \dots, P_j||PID_j||Bl_j$ are essential provenance data (EPD), and σ_{T_j} is nonessential provenance data (NPD).

Secure Provenance Verification

Given the provenance records

$$(P_1||PID_1||Bl_1, P_2||PID_2||Bl_2, \dots, P_j||PID_j||Bl_j, \sigma_{T_j}),$$

at this time, the corresponding blockchain has the form shown in Fig. 6.8.

The auditor \mathcal{A} checks the correctness of these provenance records as follows.

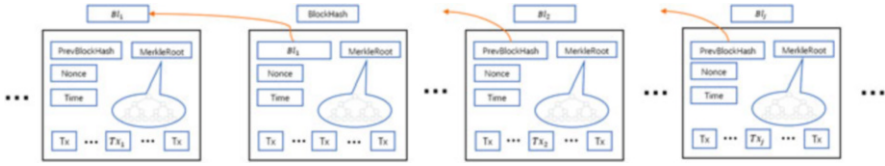


Fig. 6.8 The transactions on the blockchain

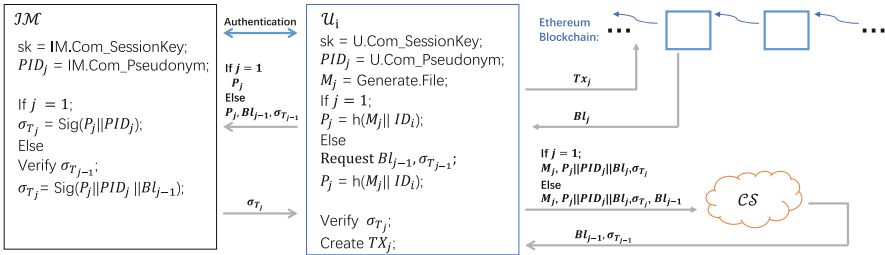


Fig. 6.9 Implementation of ESP

\mathcal{A} verifies whether the number of provenance records is equal to that of corresponding transactions recorded into the blockchain; it locates the last block BL_j on the Ethereum blockchain, and verifies the validity of the last recorded provenance record $P_j || PID_j || BL_j$; It computes $\Theta(P_j) = H(P_j || BL_{j-1})$ and checks whether $e(\sigma_{T_j}, P) \stackrel{?}{=} e(\Theta(P_j), P_{pub})$; it extracts the data information from blockchain according to BL_1, \dots, BL_j ; it verifies the integrity of provenance by checking whether the provenance records match the extracted data. If all these processes succeed, the provenance records can be accepted.

6.3.2 Implementation Based on Ethereum

We introduce the implementation of ESP, as shown in Fig. 6.9. The implementation uses JAVA language, and the experiments are conducted on a laptop with a Windows 7 system, an Intel Core 2 i5 CPU and 8 GB DDR3 of RAM. The security level is chosen to 80 bits, and the hash function h is selected to SHA3-256. The implementation is described below. For clarity, we prefix calls with IM when they are made by \mathcal{IM} and with U when they are made by \mathcal{U} .

Identity Manager (\mathcal{IM}). \mathcal{IM} executes `IM.Com_SessionKey` to compute the session key between the user and \mathcal{IM} and executes `IM.Com_Pseudonym` to compute a pseudonym for the user. \mathcal{IM} generates a signature on the provenance record and the pseudonym for the file creator (which is implemented by `Sig(Pj||PIDj)`); \mathcal{IM} needs to verify the validity of the last provenance record (which is implemented by

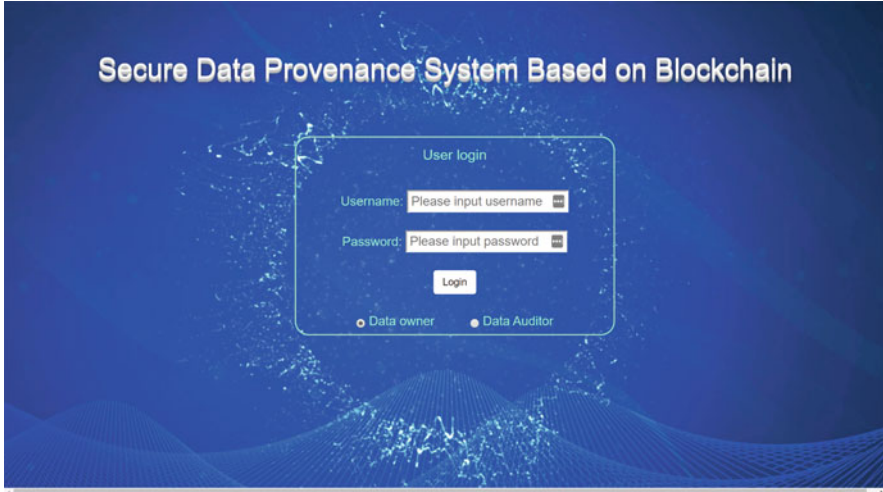


Fig. 6.10 Log-in interface

Verify $\sigma_{T_{j-1}}$) and generates a signature on the provenance record, the pseudonym, and the hash value of the block which records this provenance record on the blockchain. In the implementation of the system, this process is completed by a log-in system, as shown in Fig. 6.10.

User \mathcal{U} first computes the session key (by executing `U.Com_SessionKey`) and interacts with \mathcal{IM} to obtain a pseudonym (implemented by `U.Com_Pseudonym`). Then \mathcal{U} generates/edits the target file (by `Generate.File`). These interfaces are shown in Figs. 6.11 and 6.12.

If \mathcal{U} is the creator (i.e., $j = 1$), she/he computes a provenance record P_j and sends it to \mathcal{IM} . Then she/he verifies the signature received from \mathcal{IM} and creates a transaction in Ethereum where the provenance record is integrated into the transaction. Finally, she/he sends the provenance record to the cloud server. The transaction information on the blockchain is shown in Fig. 6.13. After the file is created, the system has an output as shown in Fig. 6.14.

If the user is the editor (i.e., $j > 1$), she/he first queries the cloud server for the necessary information about the last provenance record (i.e., Bl_{j-1} and σ_{j-1}). Then she/he computes the provenance record and sends it as well as the necessary information received from the cloud server to \mathcal{IM} . Finally, she/he verifies the validity of the signature generated by \mathcal{IM} and conducts a transaction as the creator does.

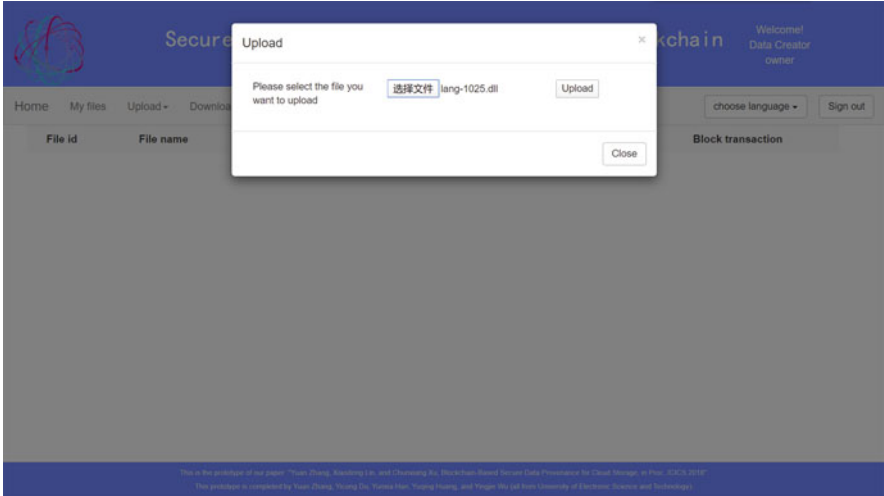


Fig. 6.11 Creation of file

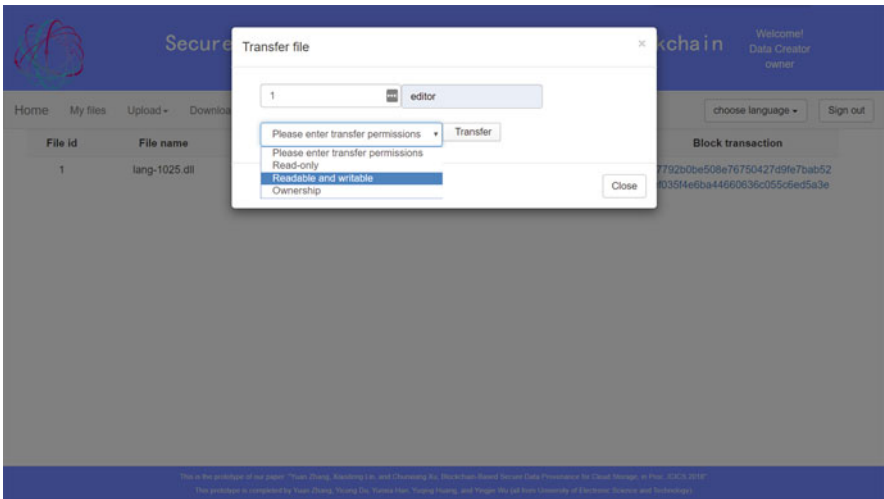


Fig. 6.12 Editing the file

6.3.3 Data Provenance and Beyond: Further Discussion

In the above description, ESP does not rely on smart contracts. The information about each provenance record is set to be the data value of a general transaction on the Ethereum blockchain. By doing so, each provenance record is integrated into a transaction on the blockchain. However, a malicious user could collude with the identity manager to create a new provenance record and a corresponding transaction

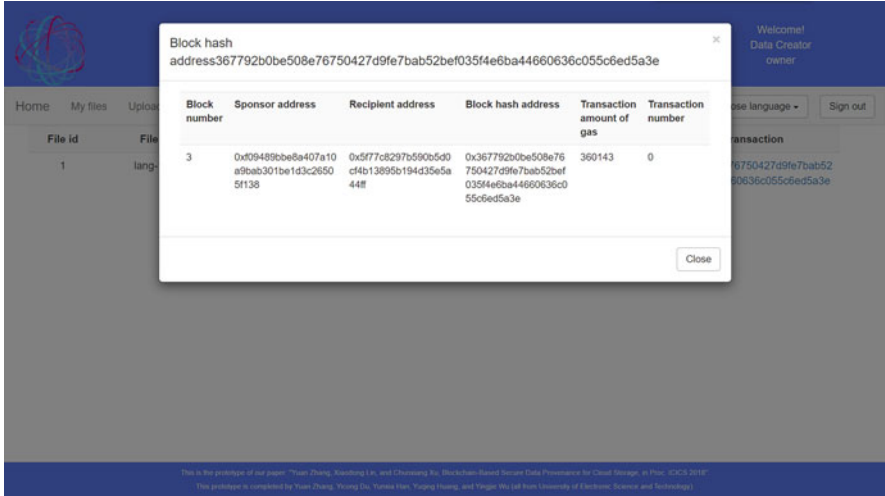


Fig. 6.13 Transaction information on the blockchain

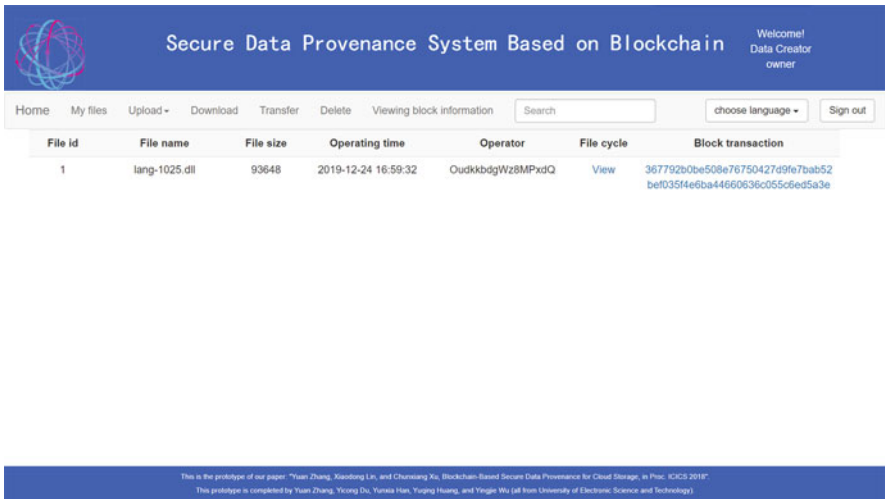


Fig. 6.14 The output after the file creation

on the blockchain and substitute an existing provenance record with the newly generated one, which essentially deletes the existing provenance record. To resist such a substitution attack, when the auditor verifies the validity of provenance records, it first checks whether the number of the corresponding transactions matches that of provenance records. If they do not match, it means that one or more provenance records have been generated but are hidden.

Another method to resist the substitution attack is to employ a smart contract. The functionality of the contract is to store the information about each provenance records to the contract storage. When the auditor verifies the validity of provenance records, it extracts all data from the contract storage and only utilizes those data that are uploaded by authorized users (the identities of these users can be known to the auditor when it performs data investigations).

We stress that both the above methods require that the accounts used to create the transactions are specially crafted and dedicated, which ensures that the number of corresponding transactions on the blockchain can be easily extracted.

Furthermore, ESP is based on Ethereum, its practicality is mainly affected by two factors, the one is WoL, and the other one is the monetary cost to publish a provenance record.

In Ethereum, a block and its transactions are considered confirmed if at least 12 consecutive blocks are mined following it. The average time that a block is mined is 15 s, and hence a transaction takes averagely 15 s (more details would be provided in the next chapter) to be chained to the Ethereum blockchain. As such, publishing a new provenance record takes an average of 3.25 min in ESP, and the time interval between two successive provenance records only requires around 3.25 min. Another user may have to wait at least 3.25 min to work on the same file, which means that ESP's WoL is around 3.25 min.

Another factor that affects the practicality of ESP is the costs to publish a provenance record. Here, we only consider ESP that is based on general transactions, rather than the smart contract. As of Jan. 2020, publishing a provenance record requires a user to pay an average of 1 US cent, which is acceptable to users with respect to the value of the file that ESP protects.

We also note that the secure data provenance scheme can also be constructed on other public blockchain systems, e.g., Ouroboros [31] and Thunderella [32], to reduce WoL and the monetary cost. These blockchain systems are based on proof-of-stake and thereby are more efficient than those based on proof-of-work, which would reduce WoL and the monetary cost significantly. However, generally for a blockchain system, the more participants are in it, the stronger security guarantee could be achieved. Therefore, constructing ESP on Ethereum would balance the trade-off between security and efficiency.

6.4 Summary and Further Reading

In this chapter, we have introduced the secure data provenance technique for cloud storage systems. We have introduced both the data provenance technique and its enhanced version, i.e., the secure data provenance technique, and provided a comparison between them. We have given a comprehensive survey on the secure data provenance technique. Finally, we have studied the latest secure data provenance scheme that is constructed on a public blockchain to resist the compromised identity

manager and discussed the potentials to construct secure data provenance schemes on blockchains.

There are some survey papers to introduce the data provenance technique from different aspects, e.g., [22, 33]. There is also a paper to discuss how to construct a secure provenance scheme on blockchain [34].

References

1. Simmhan YL, Plale B, Gannon D (2005) A survey of data provenance in e-science. *ACM SIGMOD Rec* 34(3):31–36
2. Gupta A (2009) Data provenance. Springer, Boston, pp 608–608. https://doi.org/10.1007/978-0-387-39940-9_1305
3. Moreau L, Groth P, Miles S, Vazquez-Salceda J, Ibbotson J, Jiang S, Munroe S, Rana O, Schreiber A, Tan V, Varga L (2008) The provenance of electronic data. *Commun ACM* 51(4):52–58
4. Zhang Y, Xu C, Li H, Yang K, Zhou J, Lin X (2018) HealthDep: an efficient and secure deduplication scheme for cloud-assisted eHealth systems. *IEEE Trans Ind Inf* 14(9):4101–4112
5. Wu Y, Cegielski CG, Hazen BT, Hall DJ (2013) Cloud computing in support of supply chain information system infrastructure: understanding when to go to the cloud. *J Supply Chain Manag* 49(3):25–41
6. Codeanywhere. <https://codeanywhere.com>
7. Google doc. http://www.google.cn/intl/zh-cn_all/docs/about/
8. Xue J, Xu C, Zhang Y (2018) Private blockchain-based secure access control for smart home systems. *KSII Trans Inter Inf Syst* 12(12):6057–6078
9. Zhang Y, Xu C, Li H, Yang K, Cheng N, Shen X (2020) PROTECT: efficient password-based threshold single-sign-on authentication for mobile users against perpetual leakage. *IEEE Trans Mob Comput*. <https://doi.org/10.1109/TMC.2020.2975792>
10. Btk killer. https://en.wikipedia.org/wiki/Dennis_Rader
11. Audit trails. <http://csrc.nist.gov/publications/nistbul/it197-03.txt>
12. Madden BA, Adams IF, Storer MW, Miller EL, Long DD, Kroeger TM (2011) Provenance based rebuild: using data provenance to improve reliability. UCSC, Technical Report
13. Ashcroft J, Daniels DJ, Hart SV (2011) Forensic examination of digital evidence: a guide for law enforcement. <https://www.ncjrs.gov/txtfiles1/nij/199408.txt>
14. Lin X, Chen T, Zhu T, Yang K, Wei F (2018) Automated forensic analysis of mobile applications on android devices. *Digital Investigation* 26:59–66
15. Health insurance portability and accountability act. https://en.wikipedia.org/wiki/Health_Insurance_Portability_and_Account-ability_Act
16. Zhang Y, Xu C, Lin X, Shen X (2019) Blockchain-based public integrity verification for cloud storage against procrastinating auditors. *IEEE Trans Cloud Comput*. <https://doi.org/10.1109/TCC.2019.2908400>
17. Zhang Y, Xu C, Ni J, Li H, Shen X (2019) Blockchain-assisted public-key encryption with keyword search against keyword guessing attacks for cloud storage. *IEEE Trans Cloud Comput*. <https://doi.org/10.1109/TCC.2019.2923222>
18. Lynch CA (2001) When documents deceive: trust and provenance as new factors for information retrieval in a tangled web. *J Assoc Inf Sci Technol* 52(1):12
19. Lu R, Lin X, Liang X, Shen X (2010) Secure provenance: the essential of bread and butter of data forensics in cloud computing. In: *ACM symposium on information, computer and communications security*, pp 282–292
20. Chow S, Chu C, Huang X, Zhou J, Deng RH (2012) Dynamic secure cloud storage with provenance. In: *Cryptography and security*, pp 442–464

21. Margo DW, Smogor R (2010) Using provenance to extract semantic file attributes. In: Conference on theory and practice of provenance
22. Di L, Yue P, Ramapriyan HK, King RL (2013) Geoscience data provenance: an overview. *IEEE Trans Geosci. Remote Sensing* 51(11):5065–5072
23. He L, Yue P, Di L, Zhang M, Hu L (2014) Adding geospatial data provenance into SDI—a service-oriented approach. *IEEE J Selected Topics Appl Earth Observations and Remote Sensing* 8(2):926–936
24. Interlandi M, Ekmekji A, Shah , MA Gulzar, Tetali SD, Kim M, Millstein T, Condie T (2017) Adding data provenance support to apache spark. *VLDB J* 27(5):1–21
25. Hasan R, Sion R, Winslett M (2007) Introducing secure provenance: problems and challenges. In: ACM workshop on storage security and survivability, pp 13–18
26. Zhang Y, Xu C, Li H, Yang H, She X (2019) Chronos: secure and accurate time-stamping scheme for digital files via blockchain. In: IEEE international conference on communications, pp 1–6
27. Zhang Y, Xu C, Cheng N, Li H, Yang H, Shen X (2019) Chronos+: an accurate blockchain-based time-stamping scheme for cloud storage. *IEEE Trans Serv Comput* 13(2):216–229. <https://doi.org/10.1109/TSC.2019.2947476>
28. Nakamoto S, Bitcoin: a peer-to-peer electronic cash system. <https://bitcoin.org/bitcoin.pdf>
29. Wood G (2014) Ethereum: a secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper 151:1–32
30. Zhang Y, Lin X, Xu C (2018) Blockchain-based secure data provenance for cloud storage. In: International conference on information and communications security, pp 3–19
31. Kiayias A, Russell A, David B, Oliynykov R (2017) Ouroboros: a provably secure proof-of-stake blockchain protocol. In: Annual cryptology conference, pp 357–388
32. Pass R, Shi E (2018) Thunderella: blockchains with optimistic instant confirmation. In: International conference on the theory and applications of cryptographic techniques, pp 3–33
33. Wang J, Crawl D, Purawat S, Nguyen M, Altintas I (2015) Big data provenance: challenges, state of the art and opportunities. In: IEEE international conference on big data, pp 2509–2516
34. Ramachandran A, Kantarcioglu M (2018) Smartprovenance: a distributed, blockchain based dataprovenance system. In: ACM conference on data and application security and privacy, pp 35–42.

Chapter 7

Secure Data Time-Stamping



This chapter introduces the secure time-stamping technique and discusses how to construct a secure time-stamping scheme for cloud storage systems. This chapter first introduces the traditional secure time-stamping technique that is mainly applied in traditional data storage systems (where the data are stored locally on the user side). Then, a comprehensive survey on secure time-stamping schemes is provided. Finally, the relationship between the secure time-stamping technique and blockchains is discussed, and the secure time-stamping for cloud storage is studied.

7.1 Introduction to Secure Data Time-Stamping

In data investigations, it is not uncommon to certify when a file was created [1–4]. For instance, there is a need to determine the first inventor for a patentable idea in intellectual property systems to resolve disputes. This is a universal problem in the physical world, where time-stamping mechanisms have been widely applied in our daily life. Typical applications include the paper-based sign-in system that registers someone’s attendance at a meeting or at an office. This is always completed by using an “attendance book.”

The dated names are entered one after another in the attendance book, with no pages left blank. The sequentially numbered, sewn-in pages of the attendance book make it hard to tamper with the record without leaving telltale signs. If the attendance book is then stamped on a regular basis by an authority, the security of the recorded names and recorded dates is further enhanced. If someone challenges the items recorded to the attendance book, the attendance book serves as the key evidence to prove the trustworthiness of the recorded items. However, in the above mechanism, an adversary might back-date or forward-date items. Particularly, the adversary might erase the name of an existing item with a target date and substitute the name with another one, which essentially back-date the item; the adversary also

might add a name into a blank space that corresponds to a future date, this essentially forward-date the item. An affordable way to resist the back-date attack is to use a tailor-made pen to record each item such that the recorded items cannot be erased; an effective way to resist the forward-date attack is to require each one, who wants to add a new item into the attendance book, to check that all spaces related to future dates are kept empty.

Now we consider the problem of time-stamping files in digital worlds [5]. The tools used to timestamp items in the physical world also have counterparts in modern cryptography of the digital world. Specifically, a secure signature algorithm serves as the “unerasable pen” in the digital world and a time-stamping service provider (TSP) is introduced to serve users, where TSP plays the role of the attendance book in the digital world. However, different from time-stamping physical world items, time-stamping digital files suffers from new threats towards security and privacy.

In this section, we introduce the traditional time-stamping schemes for digital files.

7.1.1 What Kinds of Data Would Benefit from Secure Time-Stamping?

We note that the secure time-stamping technique is not a panacea, which means that not every digital file needs to be protected by secure time-stamped. For example, for a traditional backup system [6], i.e., a data owner periodically backups her/his files using a portable storage device. After the files are stored in the portable storage device, the data owner leaves the device along. In this case, if the user wants to learn when the backup is performed, a timestamp indicating when the files are stored to the device can be generated by herself/himself. Note that the timestamp here would not suffer from back-dating and forward-dating from external adversaries, and thereby the secure time-stamping technique is not required.

Generally, files that need to be protected by the time-stamping technique are tamper-unpredictable and should be archived timely. It means that when a file is created, the file creator (i.e., file owner) cannot establish the necessary knowledge and motivation to tamper with the timestamp of the file. Meanwhile, the file should be outsourced and time-stamped timely to archive data for post investigations. Typically, we introduce two application scenarios in the following, where the secure time-stamping technique is required.

First, in an intellectual property system [7–9], a file when an inventor first applied for a patent should be maintained during the lifetime of the patent to determine the patent term. More crucially, this file also serves as a key evidence to indicate who is the first inventor of the patent when a dispute arises. However, it is usual that the defendant argues about the timeliness of the digital evidence during the trial. The defendant might dispute the creation time of the file, which is admitted by investigators and utilized for the judgment, is not the actual one in reality. As

a result, the file is confronted with back-dating and forward-dating: an adversary might back-date the file to forge that he is the first inventor of the patent but actually is not; an adversary, who is the first inventor of the patent, might forward-date the file to lengthen the patent term. In such a scenario, the file should be protected by the secure time-stamping technique to resist back-date and forward-date, which ensures its trustworthiness.

Second, considering a big company, it includes multiple departments and deals with a large number of files at a time. To efficiently manage data, there is an inherent need to archive files in real time. However, after the file creation, at a point in time, the file is incriminating and serves as a key evidence to incriminate the company itself in a commercial crime. Here, the company manager or other concerned people have a strong motivation to back-date and forward-date the file to cover up the crime. Such files are tamper-unpredictable: when the file is created, the file owner does not have the motivation to tamper with its timestamp, but at a later point in time, he has. In such a scenario, the secure time-stamping also has a clear value to support post investigations.

In addition to the above (traditional) application scenarios, there is also a need to certify the creation time of files in emerging application scenarios, such as electronic healthcare systems (eHealth) [10] and non-repudiation systems [11].

7.1.2 System and Threat Models

In a traditional secure time-stamping scheme [5, 12, 13], there are two entities, i.e., a set of users and a time-stamping service provider (TSP), as shown in Fig. 7.1.

A user first generates a file M and requests the time-stamping service from TSP on M . TSP generates a timestamp on M , sends the timestamp to the user, and stores the timestamp locally. Subsequently, if the time when M is generated needs to be certified, one is able to check the content of the timestamp and verify the validity of the timestamp with the aid of TSP.

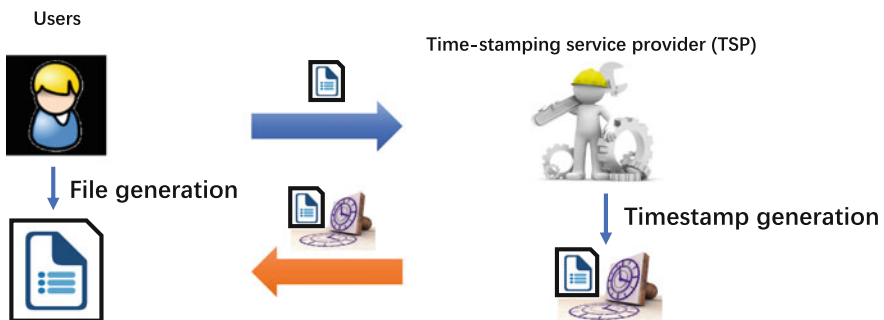


Fig. 7.1 System model of traditional secure time-stamping

In the threat model, malicious users, external adversaries, and compromised TSP are considered, where both the security of generated timestamps and the users' privacy face critical threats. As discussed before, a malicious user would attempt to back-date or forward-date a target file for profits, which essentially breaks the security of generated timestamps. To this end, the malicious user might collude with TSP to launch various attacks. Moreover, an external adversary might attempt to learn the contents of files to be time-stamped to violate the user's privacy. For example, in an intellectual property system, the contents of the patent file are very sensitive before the patent is time-stamped. Therefore, if the adversary is able to learn the contents of the file before the timestamp of the file is generated, he can violate the user's privacy and even steal the user's thunder. We note that it is easy to establish a secure channel between a valid user and TSP, and thereby resistance against eavesdropping performed by external adversaries can be achieved. In addition, any adversary can become a valid user in the secure time-stamping system, in the following, we mainly consider malicious users in the threat model, i.e., if a time-stamping scheme can resist malicious users, it is trivial to resist external adversaries with a straightforward extension.

7.2 Survey on Secure Time-Stamping

The time-stamping technique plays an important role in protecting digital files, especially in digital investigations. In time-stamping schemes, a file is time-stamped once it is created such that it can be time-sensitive. The security of the time-stamping ensures that anyone (including the file owner) cannot back-date and forward-date the file.

The problem of time-stamping digital file is first defined by Haber et al. [5], where the first time-stamping scheme was proposed. In this scheme, a trusted service provider (TSP) is introduced to provide the time-stamping service for its users.

Actually, employing TSP to securely timestamp files for users is not straightforward, where challenges in terms of security, privacy, and efficiency should be well addressed. We elaborate on the secure time-stamping scheme [5], focusing on the challenges addressed by the scheme.

For a time-stamping scheme, the first problem is that how to "timestamp" the file M . In digital words, this can be directly done by such a way: the user sends M to TSP, and TSP generates a message that "TSP received M at the time of t "; it signs this message using a private key and sends the message as well as the signature to the user. Here, the message and the corresponding signature serve as the timestamp of M , which indicates the generation time of M and authorized by TSP.

The first challenge that should be addressed in the above mechanism is to reduce communication costs. In the above mechanism, to timestamp a file M , the user needs to send M to TSP. As a consequence, the communication costs to timestamp M linearly increase with the size of M . Note that only sending a part of M to TSP cannot ensure the security, since it cannot provide true non-repudiation. To

address this problem, the cryptographic hash function can be utilized: when the user generates M , she/he computes the hash value of M , i.e., $h(M)$, and sends $h(M)$ to TSP. The timestamp generated by TSP becomes a message that “TSP received $h(M)$ at the time of t ” and the corresponding signature. Since the hash function is collision-resistant, adversaries cannot find M' such that $M' \neq M$ but $h(M') = h(M)$. In addition, the size of outputs of the hash function is a constant with a small size (e.g., 256 bits). Therefore, with the employment of the cryptographic hash function, the communication costs between the user and TSP are reduced significantly. Actually, due to preimage resistance of the cryptographic hash function, this mechanism also provides privacy preservation of file's contents against leakage, even if a malicious user colludes with TSP, he cannot learn the content of the file to be time-stamped.

However, the above mechanism still suffers from the collusion between a file owner and TSP. Specifically, after the file M is time-stamped, the file owner may incentivize TSP to help him in re-time-stamping M , i.e., TSP generates another message that “TSP received $h(M)$ at the time of t^* ,” where $t^* \neq t$. By doing so, M can be back-dated or forward-dated as needed, which invalidates the secure time-stamping scheme. The scheme of [5] resists the collusion between the malicious file owner and TSP by a technique that multiple time-stamping requests from multiple users are handled in one system. The key observation behind the scheme in [5] is that the sequence of users requesting timestamps and the files they submit to TSP cannot be known in advance. With this observation, Haber et al. utilized a hashchain to link a sequence of files from different users via the cryptographic hash function. In particular, we assume that there are n different users $\mathcal{U} = \{\mathcal{U}_1, \mathcal{U}_2, \dots, \mathcal{U}_n\}$, each of them has a file, i.e., for $\mathcal{U}_i (i \in [1, n])$, her/his file is M_i , and they request the time-stamping service from TSP one by one (in ascending order of index). TSP timestamps these files as follows.

For the first time-stamping request (i.e., the request from \mathcal{U}_1), the data TS_{M_1} that corresponds to M_1 and is submitted to TSP consists of two parts: the first one is $h(M_1)$ which is the hash value of M_1 and the second one is $ID_{\mathcal{U}_1}$ which is the identity of \mathcal{U}_1 . \mathcal{U}_1 sends $TS_{M_1} = \{ID_{\mathcal{U}_1}, h(M_1)\}$ to TSP. TSP has a pair of keys (sk_{TSP}, pk_{TSP}) , where sk_{TSP} is the private key to sign data and pk_{TSP} is the public key to verify signatures. On receiving TS_{M_1} , TSP generates a timestamp $\tau_1 = \{1, t_1, ID_{\mathcal{U}_1}, h(M_1)\}$ and signs τ_1 to get the signature $\sigma_{\tau_1} = Sig(sk_{TSP}, \tau_1)$, where t_1 is the physical time when TSP receives TS_{M_1} . TSP sends $\{\tau_1, \sigma_{\tau_1}\}$ to \mathcal{U}_1 as the timestamp of M_1 . TSP also stores $\{\tau_1, \sigma_{\tau_1}\}$ locally.

For the following time-stamping requests (i.e., the requests from $\mathcal{U}_i, (i \in [2, n])$), the data TS_{M_i} that corresponds to M_i and is submitted to TSP is similar to TS_{M_1} , that is, $TS_{M_i} = \{ID_{\mathcal{U}_i}, h(M_i)\}$. \mathcal{U}_i sends TS_{M_i} to TSP. Upon receiving TS_{M_i} , TSP generates a timestamp $\tau_i = \{i, t_i, ID_{\mathcal{U}_i}, h(M_i), L_i\}$ and signs τ_i to get the signature $\sigma_{\tau_i} = Sig(sk_{TSP}, \tau_i)$, where t_i is the physical time when TSP receives TS_{M_i} and $L_i = (t_{i-1}, ID_{\mathcal{U}_{i-1}}, h(M_{i-1}), H(\tau_{i-1}))$, and H is the cryptographic hash function that is different from h . TSP sends $\{\tau_i, \sigma_{\tau_i}\}$ to \mathcal{U}_i as the timestamp of M_i . TSP also stores $\{\tau_i, \sigma_{\tau_i}\}$ locally.

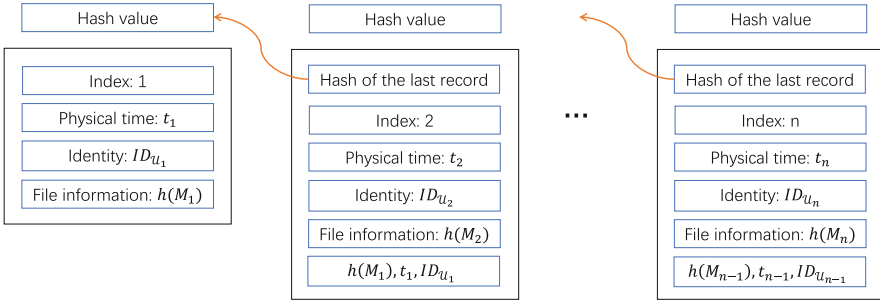


Fig. 7.2 Linked timestamps

After receiving $\{\tau, \sigma_{\tau_i}\}$ for $i = 1, 2, \dots, n$, \mathcal{U}_i checks the validity of σ_{τ_i} and t_i . If the checking succeeds, \mathcal{U}_i accepts the timestamp.

Note that on the TSP side, all the data corresponding to the files that are from n users and have been time-stamped, i.e., $TS_{M_1}, TS_{M_2}, \dots, TS_{M_n}$, have been linked as shown in Fig. 7.2, such that if any one of them is changed, all the following data are invalidated.

When a time-stamped file M_i is later challenged by an auditor \mathcal{A} for a data investigation, \mathcal{A} first verifies that $\{\tau_i, \sigma_{\tau_i}\}$ is valid, i.e., the contents of τ_i are correct and σ_{τ_i} is a valid signature under pk_{TSP} . Then, \mathcal{A} checks the trustworthiness of the timestamp by the aid of the user \mathcal{U}_{i+1} : whether the timestamp of M_{i+1} includes the corresponding information about the timestamp of M_i . If the checking passes but \mathcal{A} still suspects the validity of the timestamp of M_i , \mathcal{A} is able to call up user \mathcal{U}_{i+2} and check the next timestamp in the sequence. This can continue for as long as \mathcal{A} wishes. Likewise, \mathcal{A} is able to check the linked timestamps backward from the user \mathcal{U}_{i-1} .

Following the work proposed by Haber et al. [5], some schemes [12–15] were proposed. However, these schemes bear strong assumptions that <i> multiple users should be involved in the system, and <ii> TSP is essentially reliable and honest. Actually, in the case described above where a big company employs the time-stamping technique for data archiving and support of data investigations, it is always desired to utilize a “customized” and “personal” time-stamping service to protect the files for the big company. As a consequence, the time-stamping schemes [12–15] are unsatisfactory. Furthermore, in the above schemes, once TSP is compromised, the generated timestamps can be arbitrarily modified, and the security of these schemes is broken. Actually, in a multi-user system, the frequency that different users request the time-stamping service from TSP is variant. If last time-stamped j files are generated by one user, once the user compromises TSP, these files can be easily back-dated and forward-dated. As such, TSP becomes a single point of failure in these systems.

With the digital data being explosively generated in recent years, individuals and commercial users are increasingly outsourcing their data to cloud servers [16–20]. Some cloud storage systems also need to certify when outsourced files are

generated. For example, in cloud-based intellectual property systems, a file when an inventor first applied for a patent should be maintained during the lifetime of the patent to determine the patent term. More crucially, this file also serves as a key evidence to indicate who is the first inventor of the patent when a dispute arises. Another example is the cloud-based eHealth system [10]. In such a system, patients' EHRs are outsourced to a cloud server. The timeliness of patients' EHRs is very sensitive, it not only serves as a key reference for subsequent diagnoses, but also serves as a key evidence for postinvestigations when medical malpractices arise. However, in both of the above cases, since all files are outsourced to a cloud server that cannot be fully trusted by users and investigators, it is more challenging to securely timestamp the files in cloud storage systems than that in traditional storage systems.

Considering cloud storage systems, the above time-stamping schemes surely can securely timestamp outsourced files. Whereas, in addition to the above two problems, there are also two problems that should be addressed:

- In cloud storage systems, users are provided a transparent and efficient storage management service. Different users are not required to know the existence of each other. Moreover, the overhead of using cloud storage services should be minimized as much as possible, which means that a user should not be responsible for the security and trustworthiness of others' files, and users should not provide any service to each other. However, the above time-stamping schemes require at least one user to assist another one in ensuring the security and trustworthiness of the corresponding timestamp.
- With the deployment of TSP, users using the cloud storage service have to change their interaction pattern: since TSP is a trusted entity that is independent of the cloud service provider, the users have to interact with both TSP and cloud server to secure their files in data outsourcing. As a result, users need to bear not only an additional communication overhead but also extra costs to employ TSP.

Data outsourcing makes the assurance of files' timeliness more challenging than ever, and hence requires a new time-stamping mechanism for outsourced files. In Sect. 7.3.2, we study a case to show how to securely and efficiently timestamp outsourced files in cloud storage systems.

7.3 Secure Time-Stamping and Blockchain

Recently, blockchains have been envisioned as a powerful tool to enhance data security in both traditional storage systems and cloud storage systems. As a matter of fact, the relationship between the time-stamping technique and the blockchain is very strong. In this section, we first review this relationship and introduce how a blockchain system is derived from a time-stamping scheme. Then, we study how a secure time-stamping scheme is constructed on a blockchain system.

7.3.1 *Distributed Cryptocurrencies from Secure Time-Stamping*

We have briefly introduced the blockchain in Sect. 2.5. From the perspective of technique, a blockchain consists of two core components, the one is the underlying data structure, and the other one is the underlying consensus algorithm.

The blockchain is a technique derived from Bitcoin [21] which is the first distributed cryptocurrency system and is the most prominent manifestation of the blockchain. In any currency system (no matter whether it is designed for the digital world), the transactions in it should be time-sensitive to ensure security. Without a time-stamping mechanism, a currency system cannot resist an adversary who double spends coins (which we consider as the underlying value token).

In our physical world, all transactions are authorized by a central authority, i.e., a bank, and the ledger which accounts for the ownership of coins is maintained by the authority. With the authorization, the bank also timestamps each transaction with an accurate physical authorization time. Here, the bank plays the role of TSP in secure time-stamping schemes described before. The early attempts to build digital currencies, such as [22, 23], also follow the paradigm of the currency in the physical world. They also employ a central authority to manage transactions and maintain the ledger. Since the authority is trust with respect to the security, the secure time-stamping schemes presented in [5, 12] are not required, and the authority can straightforward generate and record all timestamps of transactions. However, centralized digital currencies have an oblivious disadvantage that the central authority knows “everything.” It can learn any transaction detail from the authorized transactions and maintained ledger if it wants, and the users’ privacy cannot be protected.

To completely eliminate the central authority, the transaction must be authorized in a distributed way, and the ledger also must be distributed. However, resistance against the adversary who double spends coins is much more challenging than ever. Since copying digital data is very easy, the adversary can issue two transactions in parallel, transferring the same coin to different recipients. Interestingly, resistance against the double-spending attack and assurance of the immutability of conducted transactions can themselves be seen as a time-stamping problem. With this observation, Bitcoin is constructed on a time-stamping server.

Specifically, Bitcoin eliminates the central bank is very pragmatic: in a sense, everyone is the bank, and every participant keeps a copy of the record (i.e., transactions and the ledger) which would traditionally be maintained by the central bank. Each participant maintains the record in the way as TSP does in secure time-stamping schemes [5]. Bitcoin utilizes a paradigm of “coin-based” transaction to conduct transactions, where the transaction is described in the way that a coin with a unique identifier is transferred from the payer’s address to the payee’s address. By doing so, each coin is spent in order of time: all transactions related to a bitcoin form a chain on the Bitcoin blockchain such that the bitcoin cannot be transferred to different payees simultaneously. Furthermore, from the perspective of technique,

a blockchain is composed of multiple data elements, in which each data element is called a block. All blocks form a chain, where the security is guaranteed by utilizing a cryptographic hash function. Each block typically contains a hash pointer as a link to a previous block, a timestamp, and transaction data. This essentially timestamps all blocks in the way proposed by Haber et al. [5].

The above “time-stamping” paradigm has been applied in most existing public blockchain systems, e.g., Ethereum [24], Ouroboros [25], including proof-of-work-based ones and proof-of-stake-based ones. Both Bitcoin (the first public blockchain system) and Ethereum (the most widely used public blockchain system) have considered verifying the validity of the timestamp in each block as a part of verifying the validity of the block. However, both of them do not require each miner to verify that the timestamp is an accurate value that is the current physical time when the miner receives the corresponding block, since this requires a precise time synchronization among all miners, which would introduce considerable costs in terms of communication and computation. Instead, briefly speaking, they require the miner to check whether the timestamp of the received block is no earlier than that of its antecedent block.

With the above discussion, we can say that the blockchain, e.g., Bitcoin, Ethereum, in a sense, is derived from the secure time-stamping scheme [5] and leverages the hashchain of [5] as the underlying data structure.

7.3.2 Secure Time-Stamping from Blockchain

In the previous section, we have discussed the relationship between the blockchain and the secure time-stamping technique and also introduced how a blockchain is derived from a secure time-stamping scheme. However, as we analyzed before, existing time-stamping schemes also suffer from some problems such that they are unsatisfactory for protecting outsourced files in cloud storage systems. Actually, on the contrary, a secure time-stamping scheme can be derived from a secure blockchain system. In this section, we study a system, called Chronos⁺ [26, 27], to show how to construct a secure time-stamping scheme on the blockchain.

7.3.2.1 Overview of Chronos⁺

We first provide an overview of Chronos⁺, focusing on the challenges addressed by it.

Recall that traditional secure time-stamping schemes [5, 12] are confronted with the single-point-of-failure problem and are cumbersome for cloud storage systems. All these problems can be addressed by employing the blockchain.

There are two lines of work to construct blockchain-based secure time-stamping schemes.

On the one hand, Coleman [28] proposes the scheme of universal hash time. The key idea behind it is to enable all files from different users to form an authenticated data structure (e.g., hashchain and Merkle hash tree) with the aid of a public blockchain (e.g., Bitcoin). The timeliness of these files in the data structure is reflected in the chronological order. By doing so, one can determine that a file was generated no earlier than the previous one and no later than the subsequent one. The same idea has also been proposed by Landerreche et al. [29], where the cryptographic time-stamping through sequential work is presented.

However, this line of the work, including [28–30] still unsatisfactory due to the following reasons. First, the timestamp of a file depends on other files in the system, which cannot accurately reveal the physical time when the file was created. Second, these schemes are designed for traditional storage systems, and if they are applied in the cloud storage systems, they also face the same problems as traditional time-stamping schemes do.

On the other hand, a secure time-stamping scheme can be derived from the public blockchain (e.g., the scheme presented by Gipp et al. [31]) and current blockchain-based storage schemes (e.g., Blockstack [32] Catena [33], and ESP [34]). Specifically, with the employment of the blockchain, the cloud server can provide users with both the storage and secure time-stamping services. This scheme is shown in Fig. 7.3, and is detailed as follows.

There are three different entities, users, a log server, and an authenticated auditor. The log server is subject to the cloud service provider and provides both the data outsourcing service and secure time-stamping service. The authenticated auditor outputs the physical creation time of a given file. When a file is created by a user, it is outsourced to the log server. Then, the file is integrated into a transaction on the Bitcoin blockchain, where the log server uses the “OP-RETURN” outputs to store the digest of the file in Bitcoin. After the transaction is recorded into a block on the blockchain, the file has been time-stamped. The block’s timestamp that indicates the physical time when the block was appended to the blockchain can serve as the file’s

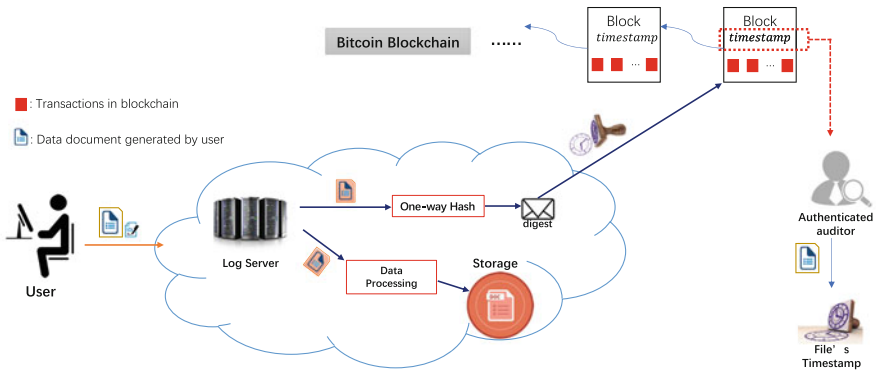


Fig. 7.3 A plain blockchain-based time-stamping scheme for cloud storage systems

timestamp to show that the file was generated earlier than this point in time. As such, when the auditor checks the timestamp of a file, it only extracts the timestamp of the corresponding block from the Bitcoin blockchain and outputs this physical time as the creation time of the file. Since the blockchain is resistant to modification, anyone cannot back-date/forward-date the time-stamped file. Furthermore, in the scheme, different users are independent, and they are not aware of each other when they outsource files to the log server.

However, in the above scheme, the accuracy of the timestamp cannot be guaranteed, which causes an unacceptable time error. The time error is mainly caused by the following reasons.

- The block's timestamp on the Bitcoin blockchain faces up to 2-h errors.
- A transaction needs to wait to be lumped into a block with a considerably long delay (about 1 h and more in extreme cases).

Therefore, the first challenge to design a secure and practical blockchain-based time-stamping scheme is to improve the accuracy of timestamps of outsourced files, which requires an alternative method to extract files' timestamps. The key technique behind Chronos⁺ is to adopt the blockchains' property of chain growth to build a "clock" outputting the time that each block was appended to the blockchain. In particular, chain growth formalizes blockchains' property that a blockchain's height will steadily increase in respect of both short and long terms. This allows us to accurately derive the physical time when a block was chained to the blockchain from the block's height on the blockchain. Such the height-derived timestamp removes the time errors in the block's timestamp. On the other hand, to significantly reduce the considerably long delay caused by uploading the transaction to the Bitcoin blockchain, Chronos⁺ is built on Ethereum, because the handling capacity of Ethereum is much stronger than that of Bitcoin. This reduces the delay of uploading transactions dramatically.

The above schemes essentially share the same paradigm: the file's timestamp is only related to the time when the block containing the information about the file is chained to the blockchain. Whereas, such a timestamp only indicates that the file was created earlier than the time when the block was appended. This would be insufficient for protecting time-sensitive data in reality. Consider such an adversary: a malicious user, who colludes with the time-stamping service provider (i.e., the log server), can launch an attack of "stealing the thunder" by changing the ownership of a target file. Such an adversary is called a malicious competitor. More details, the malicious competitor is a valid user in the system. He targets at a specific group of users in the system to "steal the thunder." For instance, in a cloud-based intellectual property system, when a user uploads a patent to the log server, a malicious competitor can intercept and tamper with the patent to change the ownership of the patent. This problem is further exacerbated by the fact that the malicious competitor can incentivize the log server to perform the above attack. Due to the existence of the malicious competitor, it is very important to certify the earliest creation time of the outsourced file. A trivial method to prove the earliest creation time of a file is to let the user embed the creation time in the file before sending it

to the log server. Nevertheless, this requires a precise time synchronization among all users, and all log servers (data investigations are always performed on multiple systems), and would introduce heavy communication and computation costs.

Chronos⁺ resolves this deadlock by utilizing (ι, φ) -chain quality of blockchains. Instead of requiring the user to embed physical time in the file to prove the earliest creation time, the user only embeds a time-dependent random seed in the file for this purpose. The time-dependent random seed is unpredictable, unforgeable, and publicly verifiable, which proves that the seed was generated no earlier than a point in time. Note that the hash values of φ -successive blocks on the blockchain can actually serve as such a time-dependent random seed. With the integration of the hash values of φ -successive blocks that are latest confirmed on the Ethereum blockchain, Chronos⁺ allows users to prove that the file was generated no earlier than the physical time that the last block of φ -successive ones was appended to the blockchain. Again, this physical time is also derived from the block's height to ensure its accuracy. Consequently, the file's timestamp in Chronos⁺ is a time interval denoted by $[ts_1, ts_2]$, where ts_1 is extracted from the height of the last block in φ -successive ones that are latest confirmed on the Ethereum blockchain when the file was created, and ts_2 is extracted from the height of the block containing the information about the file.

We stress that the scheme introduced so far is still vulnerable to the malicious competitor. We assume that the malicious competitor targets at a specific user for stealing her/his file. To this end, the malicious competitor intercepts the file sent from the user, compromises the user's network, changes the ownership of the file (e.g., changing the author information), and sends the modified file to the log server. As a result, the malicious competitor can steal the user's thunder. Notice that such an attack cannot be resisted by encrypting the file on the user side before outsourcing, since the contents of files (e.g., patents) protected by time-stamping schemes should always be publicly verifiable, and the malicious competitor may incentivize the log server to perform the attack.

Chronos⁺ addresses this issue by leveraging an elaborate mechanism of “unlock on delivery.” Particularly, the user encrypts the file (using a symmetric-key encryption algorithm) and sends the ciphertext to the log server. The log server timestamps the ciphertext as described before. After the timestamp is generated, the user sends the encryption/decryption key to the log server. The log server then decrypts the ciphertext and stores the file as well as the encryption/decryption key and the timestamp locally. The “stealing the thunder” attack no longer works, since the malicious competitor cannot change the ownership of the file without the encryption/decryption key, even if he colludes with the log server.

There is still a subtle security problem: A malicious user may collude with the log server to perform equivocation attacks to modify an existing timestamp of a specific file that generated by himself, as shown in Fig. 7.4. Specifically, after a file M is time-stamped, the log server has maintained the ciphertext of M , the corresponding decryption key, and the timestamp of M that includes the information about the block containing M on the Ethereum blockchain. However, the file owner can incentivize the log server to “re-timestamp” the file and substitute the

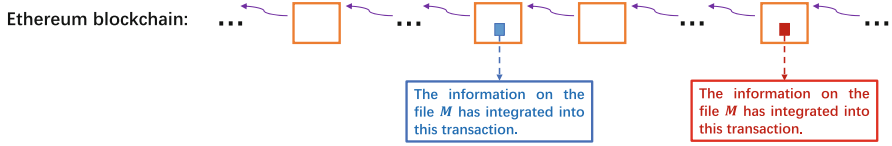


Fig. 7.4 Equivocation about the timestamp of the file M

newly generated timestamp for the existing one. As such, the malicious user can equivocate on the timestamp of the target file. It is cumbersome to detect such an attack in practice, since it requires the auditor to maintain the entire blockchain and scan it to ensure the non-equivocation. In Catena [33], resistance against the equivocation attack is achieved by adopting a non-membership proof which is constructed on the Bitcoin’s UTXO mechanism. However, such a strategy cannot work in Chronos⁺, since Ethereum does not employ the UTXO mechanism as the underlying transaction management mechanism.

To ensure the non-equivocation, Chronos⁺ utilizes a new non-membership proof on the Ethereum blockchain. The accounts used to create the transactions in Chronos⁺ are specially crafted and dedicated, which enables the authenticated auditor to verify whether the number of transactions created by the account matches the number of files that have been time-stamped. This number can be easily extracted from the “nonce” field of the account in Ethereum. Such a mechanism has been utilized in the secure data provenance scheme introduced in Sect. 6.3.1.3.

7.3.2.2 Construction of Chronos⁺

Chronos⁺ consists of four algorithms: **Setup**, **Outsource**, **TimeStamp**, and **CheckStamp**. There are three entities in Chronos⁺: a user \mathcal{U} , a log server \mathcal{LS} , and an authenticated auditor \mathcal{A} . The process of Chronos⁺ is illustrated in Fig. 7.5, and described in the following. For the sake of brevity, we would not provide the details of some algorithms, e.g., encrypting/decrypting a file and computing a signature.

Setup With the security parameter, the system parameters $\{h, A_{\mathcal{LS}}, A_{\mathcal{U}}, E/D, Sig\}$ are determined, where h is a cryptographic hash function, $A_{\mathcal{LS}}$ is the \mathcal{LS} ’s account and $A_{\mathcal{U}}$ is the \mathcal{U} ’s account on the Ethereum blockchain, E/D is a symmetric encryption/decryption algorithm (e.g., AES), and Sig is a secure digital signature algorithm. \mathcal{U} randomly chooses $sk_{\mathcal{U}}$ as the secret key and computes $pk_{\mathcal{U}}$ as the corresponding public key. \mathcal{U} also chooses $k_{\mathcal{U}} \in \mathbb{Z}_p$ as an encryption/decryption key for E/D .

Outsource \mathcal{U} generates a new file M and outsources it to \mathcal{LS} as follows.

- \mathcal{U} generates M and encrypts it using $k_{\mathcal{U}}$ as:

$$C = E(k_{\mathcal{U}}, M).$$

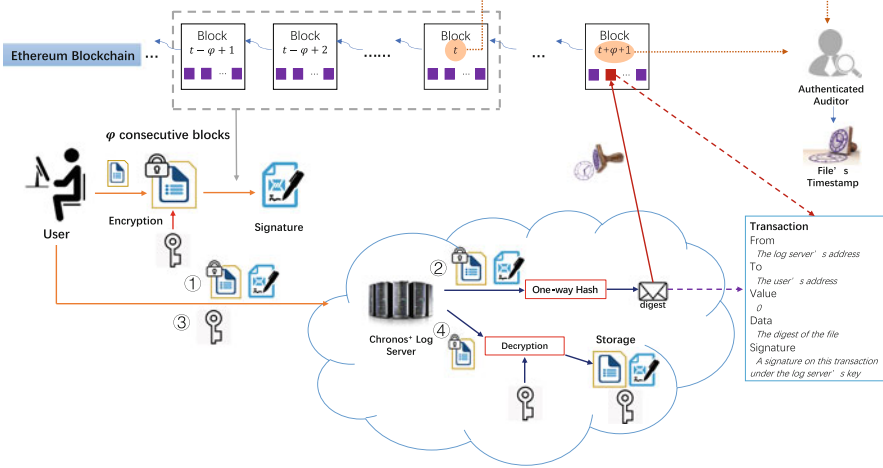


Fig. 7.5 Chronos⁺

- Based on the current time, \mathcal{U} acquires the hash values of φ -consecutive blocks that are latest confirmed on the Ethereum blockchain. The hash values of these blocks are denoted by $Bl_{t-\varphi+1}$, $Bl_{t-\varphi+2}$, ..., Bl_t , respectively, where t is the height of the block that is latest confirmed on the blockchain, and it is recommended to choose $\varphi \geq 12$ for Ethereum.
- \mathcal{U} generates a signature $\sigma = \text{Sig}(sk_{\mathcal{U}}, C || Bl_{t-\varphi+1} || Bl_{t-\varphi+2} || \dots || Bl_t)$.
- \mathcal{U} sends $\hat{C} = \{C, Bl_{t-\varphi+1}, Bl_{t-\varphi+2}, \dots, Bl_t, \sigma\}$ to \mathcal{LS} .
- Upon receiving \hat{C} , \mathcal{LS} verifies whether the block corresponding to Bl_t is the latest one that is confirmed on the blockchain. (In practice, the block corresponding to Bl_t may be not the latest one due to the delay caused by communication. However, this delay would not be long, and the block can be accepted if it is one of the latest ones that is confirmed on the blockchain. For the sake of brevity, we do not consider the delay in this section.) If the verification fails, \mathcal{LS} rejects \hat{C} ; otherwise, \mathcal{LS} verifies the validity of σ . If σ is valid, \mathcal{LS} accepts \hat{C} ; otherwise, \mathcal{LS} rejects \hat{C} .

TimeStamp \mathcal{LS} timestamps \hat{C} as follows.

- \mathcal{LS} computes a digest of \hat{C} as

$$\delta = h(Bl_{t-\varphi+1} || Bl_{t-\varphi+2} || \dots || Bl_t || C || \sigma).$$

- \mathcal{LS} generates a transaction shown in Fig. 7.5, where \mathcal{LS} transfers 0 Ether from its account $A_{\mathcal{LS}}$ to \mathcal{U} 's account $A_{\mathcal{U}}$ and sets δ on the data value of the transaction. \mathcal{LS} then uploads the transaction to the Ethereum blockchain. Ideally, this transaction would be recorded in the block with the height $t + \varphi + 1$.

- Once the transaction is accepted and confirmed by the Ethereum blockchain, \mathcal{U} sends $k_{\mathcal{U}}$ to \mathcal{LS} .
- \mathcal{LS} decrypts C by computing

$$M = D(k_{\mathcal{U}}, C).$$

If the decryption fails, \mathcal{LS} aborts; otherwise, \mathcal{LS} locally stores

$$\{Bl_{t-\varphi+1}, \dots, Bl_t, t + \varphi + 1, M, k_{\mathcal{U}}, \sigma\}.$$

CheckStamp Given $\{Bl_{t-\varphi+1}, \dots, Bl_t, t + \varphi + 1, M, k_{\mathcal{U}}, \sigma\}$, \mathcal{A} is able to check the creation time of M as follows:

- \mathcal{A} acquires the information of $A_{\mathcal{LS}}$ and $A_{\mathcal{U}}$ from the Ethereum blockchain, extracts the number of transactions from $A_{\mathcal{LS}}$ to $A_{\mathcal{U}}$ based on the nonce value of $A_{\mathcal{LS}}$, and checks whether the number of transactions matches the number of files generated by \mathcal{U} and time-stamped by $A_{\mathcal{LS}}$. If the checking fails, \mathcal{A} aborts.
- \mathcal{A} computes $C = E(k_{\mathcal{U}}, M)$ and verifies the validity of σ . If the verification fails, \mathcal{A} aborts.
- Based on the block height $t + \varphi + 1$, \mathcal{A} locates the block and extracts δ' from the corresponding transaction and verifies the following equation:

$$h(Bl_{t-\varphi+1} || Bl_{t-\varphi+2} || \dots || Bl_t || C || \sigma) = \delta'.$$

If the verification fails, \mathcal{A} aborts.

- \mathcal{A} computes

$$ts_1 = \tau + \rho \cdot t, \tag{7.1}$$

$$ts_2 = \tau + \rho \cdot (t + \varphi + 1), \tag{7.2}$$

where τ is 2015-07-30, 03:26:13 PM +UTC (i.e., the time the genesis block of Ethereum was created) and ρ is the average time that a new block is mined from the day of 2015-07-30 to the day the block is appended to the blockchain in Ethereum. We will provide the details on setting ρ later.

- \mathcal{A} outputs $[ts_1, ts_2]$ as a timestamp of M , which indicates that M was generated during $[ts_1, ts_2]$.

Support for Batch Time-Stamping In practice, a log server needs to serve multiple users concurrently, i.e., it needs to handle multiple time-stamping requests on different files from different users at a time. The individual time-stamping of these files for the log server and users could be tedious, inefficient, and inaccurate. Given n time-stamping tasks on n distinct files from n different file owners, it is more advantageous for the log server to batch these tasks together and timestamp these files at one time. Actually, Chronos⁺ can be extended to support batch time-stamping, which is described in the following.

We assume that there are n users $\{\mathcal{U}_1, \mathcal{U}_2, \dots, \mathcal{U}_n\}$.

Setup The system parameters $\{h, A_{\mathcal{L}\mathcal{S}_{send}}, A_{\mathcal{L}\mathcal{S}_{receive}}, E/D, Sig\}$ are determined with the security parameter, where $A_{\mathcal{L}\mathcal{S}_{send}}$ and $A_{\mathcal{L}\mathcal{S}_{receive}}$ are two accounts of $\mathcal{L}\mathcal{S}$ on the Ethereum blockchain, and other parameters are the same as the ones described before. For $i = 1, 2, \dots, n$, \mathcal{U}_i generates $\{sk_{\mathcal{U}_i}, pk_{\mathcal{U}_i}\}$ and $k_{\mathcal{U}_i}$ as the same as the basic scheme.

Outsource For $i = 1, 2, \dots, n$, \mathcal{U}_i generates a new file and outsources it to $\mathcal{L}\mathcal{S}$ as follows.

- \mathcal{U}_i generates a new file M_i and encrypts it as

$$C_i = E(k_{\mathcal{U}_i}, M_i).$$

- Based on the current time, \mathcal{U}_i acquires $Bl_{t-\varphi+1}, Bl_{t-\varphi+2}, \dots, Bl_t$ from the Ethereum blockchain, generates a signature $\sigma_i = Sig(sk_{\mathcal{U}_i}, C_i || Bl_{t-\varphi+1} || Bl_{t-\varphi+2} || \dots || Bl_t)$, and sends $\hat{C}_i = \{C_i, Bl_{t-\varphi+1}, Bl_{t-\varphi+2}, \dots, Bl_t, \sigma_i\}$ to $\mathcal{L}\mathcal{S}$.
- Upon receiving \hat{C}_i , $\mathcal{L}\mathcal{S}$ checks the validity of $Bl_{t-\varphi+1}, Bl_{t-\varphi+2}, \dots, Bl_t$. In the batch time-stamping, n files from n users who choose the same $Bl_{t-\varphi+1}, Bl_{t-\varphi+2}, \dots, Bl_t$ are time-stamped simultaneously. Then $\mathcal{L}\mathcal{S}$ verifies the validity of σ_i . If σ_i is of the correct form, $\mathcal{L}\mathcal{S}$ accepts \hat{C}_i .

TimeStamp On receiving $\hat{C}_1, \hat{C}_2, \dots, \hat{C}_n$, $\mathcal{L}\mathcal{S}$ timestamps them as follows.

- $\mathcal{L}\mathcal{S}$ computes

$$\delta = h(Bl_{t-\varphi+1} || \dots || Bl_t) || h(C_1 || \sigma_1) || \dots || h(C_n || \sigma_n).$$

- $\mathcal{L}\mathcal{S}$ generates a transaction Tx_1 shown in Fig. 7.6, where 0 Ether is transferred from $A_{\mathcal{L}\mathcal{S}_{send}}$ to $A_{\mathcal{L}\mathcal{S}_{receive}}$, and δ is set to the data value of Tx_1 .

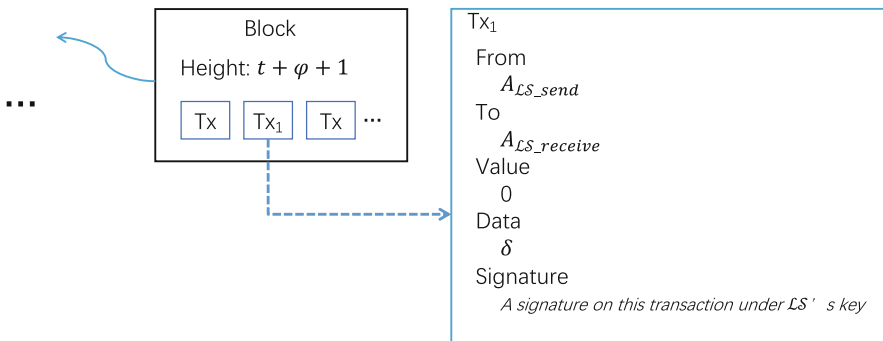


Fig. 7.6 Transaction in the batch time-stamping scheme

Table 7.1 Data stored on \mathcal{LS}

$Bl_{t-\varphi+1}, Bl_{t-\varphi+2}, \dots, Bl_t$	$t + \varphi + 1$	
F_1	$k_{\mathcal{U}_1}$	σ_1
\dots	\dots	\dots
F_n	$k_{\mathcal{U}_n}$	σ_n

- \mathcal{LS} uploads the transaction to the Ethereum blockchain. Ideally, this transaction would be recorded in the block whose height is $t + \varphi + 1$.
- Once the transaction is accepted and confirmed by the blockchain, \mathcal{LS} sends the information of the corresponding block to all users.
- For $i = 1, 2, \dots, n$, \mathcal{U}_i verifies that $h(C_i || \sigma_i)$ has been recorded into the Ethereum blockchain. If the verification passes, \mathcal{U}_i sends $k_{\mathcal{U}_i}$ to \mathcal{LS} .
- \mathcal{LS} decrypts C_i by computing $M_i = D(k_{\mathcal{U}_i}, C_i)$, and locally stores the data which is shown in Table 7.1.

CheckStamp This algorithm is the same as that in the basic scheme, it is not repeated here for the sake of brevity.

7.3.2.3 Accuracy of Height-Derived Timestamps

We denote the average time block mining on the Ethereum blockchain by BlockTime. Figure 7.7 shows BlockTime (The data are collected from Etherscan,

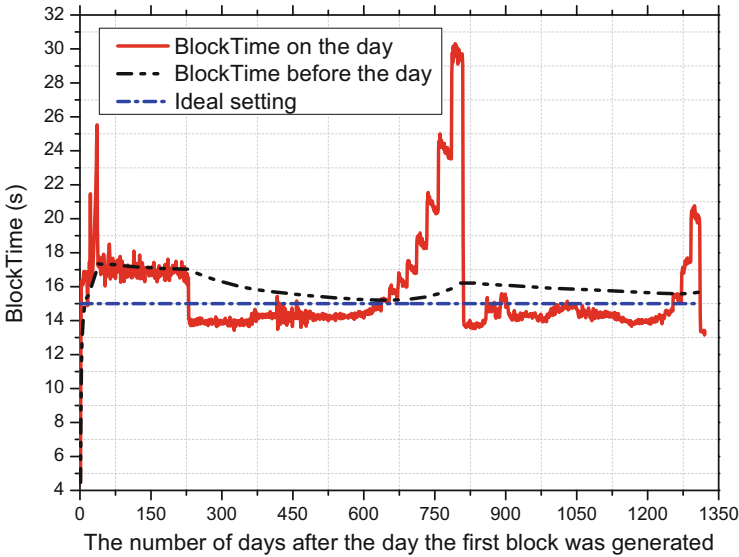


Fig. 7.7 BlockTime of Ethereum

<https://etherscan.io>), where the blue dash line indicates BlockTime of system setting (i.e., 15s), the red line indicates BlockTime on the x -th day after the day of τ (i.e., the genesis block was created, 2015-07-30 +UTC), and the black dot line indicates BlockTime from the day of τ to the x -th day after τ (x is the corresponding value of the X-axis). As shown in Fig 7.7, BlockTime in Ethereum is larger than the pre-set one, due to the network delay, the fluctuation of network hashing power, and so on. Therefore, if we set $\rho = 15$ in Eqs. (7.1) and (7.2), the height-derived timestamp is still not accurate. However, although BlockTime is not equal to 15s, the chain growth property is not broken, since BlockTime in Ethereum still falls into a small range of time with respect to both short or long term, even if BlockTime is fluctuating on a single day. We assume that ρ_x is the average BlockTime from τ to the x -th day after the day of τ . In Eqs. (7.1) and (7.2), ρ should be set to ρ_x when the block was appended to the blockchain on the x -th day after 2015-07-30 +UTC. We can compute ρ_x as

$$\rho_x = \frac{\sum_{j=1}^x \hat{\rho}_j}{x + 1}, \quad (7.3)$$

where $\hat{\rho}_x$ denotes the BlockTime on the x -th day after the day of τ (shown by the red line in Fig. 7.7). We stress that BlockTime on each day is very important for the Ethereum. Multiple supernodes and full nodes have maintained and released Ethereum's BlockTime on each day in real time. Therefore, $\hat{\rho}_x$ can be easily derived from the Ethereum blockchain. To ensure the accuracy of height-derived timestamp, ρ_x should be periodically adjusted due to the fluctuation of BlockTime in Ethereum. For example, when $x = 1146$, i.e., the day is 2018-09-16, $\rho_x = 15.72$ s. When $x = 1322$, i.e., the day is 2019-03-13, $\rho_x = 15.65704769$ s.

Another factor that affects the accuracy of files' timestamps is the range of the time interval, i.e., the range of $[ts_1, ts_2]$ denoted by R_{TS} . In Chronos⁺, for the x -th day after the day of τ , $R_{TS} = \varphi \hat{\rho}_x$. For the recommended $\varphi = 12$, R_{TS} in Chronos⁺ is around 3 min. Although R_{TS} varies with BlockTime, it would not be too large in practice, due to chain growth property of the blockchain.

7.3.2.4 Simulation on Ethereum

Actually, the criterion to evaluate the practicality of the blockchain-based secure data provenance scheme, i.e., window of latching (WoL, defined in Sect. 6.3.1.1), can be modified to obtain a criterion that evaluates the practicality of a blockchain-based secure time-stamping scheme. There is also a counterpart of WoL in the latter case, called window of time-stamping (short for WoT). WoT is a time interval from the time that a request on a file's timestamp is made to the time that the timestamp is securely recorded. Obviously, the longer WoT, the larger the time errors in the timestamp and the longer the latency period that a user has to bear for time-stamping a file. Hence, the shorter WoT is, the more practical a time-stamping scheme is.

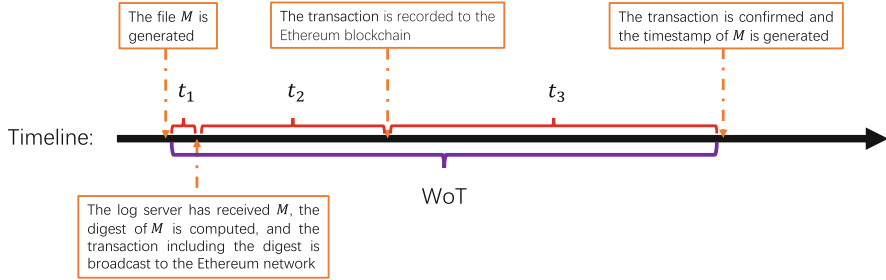


Fig. 7.8 Illustration of WoT

Regarding to Chronos⁺, its WoT consists of three time intervals, as shown in Fig. 7.8. The first one is the time to process the file M , which is denoted by t_{i1} ; the second one is the time to record the transaction to the blockchain (denoted by t_{i2}); the third one is the time to confirm the transaction (denoted by t_{i3}).

t_{i1} consists of two parts: the one is that the user encrypts M and generates the signature; the other one is that the log server verifies the validity of the signature, computes the digest of M , generates the transaction, and decrypts the ciphertext. In reality, when Chronos⁺ utilizes the BLS signature as the underlying signature algorithm, the AES as the underlying encryption/decryption algorithm, and both the user and the log server equip with a not-so-powerful device (e.g., a MacBook Pro with macOS, an Intel Core i7 CPU, and 16 GB DDR3 of RAM), t_{i1} is less than 30 ms. t_{i1} would be further reduced in reality, since the cloud server equips with the device that is much more powerful than that the experiment is tested on. Anyway, this delay is very trivial in reality and would not become the main factor in impacting WoT.

t_{i2} and t_{i3} refer to the latency that the log server ensures that the file is successfully time-stamped. Specifically, t_{i2} is the latency that the transaction created by the log server can be recorded to the Ethereum blockchain, and t_{i3} is the latency that the transaction is confirmed after it is recorded to the Ethereum blockchain. We have conducted experiments to evaluate t_{i1} , t_{i2} , and WoT of Chronos⁺. Specifically, we use MyEtherWallet as the wallet App (<https://www.myetherwallet.com/>) to conduct transactions on the Ethereum blockchain. The time-stamped message is the identity of the paper of [27], i.e., “TSCSI-2019-03-0116 Chronos+.” We have created 15 transactions to evaluate the “time the transaction to complete” (i.e., t_2), “time to confirm the transaction” (i.e., t_3), and WoT. In Ethereum, the time the transaction to complete depends on two factors: how much transaction fees set by the transaction creator (i.e., the log server in Chronos⁺) and how congested the Ethereum network is. In reality, the wallet App provides users three options of transaction fees, i.e., “Slow/Economy,” “Standard/Regular,” and “Fast.” In the simulation, we choose the option of “Standard/Regular” by default. The transaction information is shown in Table 7.2.

Table 7.2 Transaction information

Index	Block height	Transaction hash
1	8401901	0xfea484efdd3a0b741780c69f1206451ea1e0f1fe5391b58e2ccd5b0503498084
2	8402086	0x3a834dc5506704b453eede3d728d95ccd243f33f73aebc74af8646cc7d356dde
3	8402117	0x5d8f12bb32cadd8be52a57664a928e4903a23c28a3a684db994c4b894e75ad57
4	8402233	0xe00e13b3821328256265986bf1c6119cb5b030b1f87dad7359bb4f6916636e50
5	8402252	0x75e6a03512d978812463758fd89495f6f70f487fb2ee8488858929d28ff572d7
6	8402260	0x40a0ec8cf3642a0cbdae6fb3c158fd4499fb411cde557f78ca3eff6824e74980
7	8402401	0xd7feb9c8451deb67196648d2e24eb1fab34d7c1dbba30236100c7ccc000154cc
8	8402414	0x4759e671c792f680c44f98879ac78e75687f15aa5f67e2479f987d8bb6b12869
9	8402421	0x8e814b284023ffb1f1811cff62442e57c19de6757ef55f804960133457b74539
10	8402502	0xe8daceec22ba60523462a8698751dfed7e687c301fa5b0dce30a9913e2eb98be
11	8403493	0x5a3c2de954f290834832edba7c0b3c9ed21e0b2ba5d0c689494c1f7a98f20ded
12	8403501	0xb879e919a2aefb2d3ff1f7fdea0596db310c4652e0d5ac1b55936954036cc4da
13	8403512	0x9662700301afaf64a443e70f117d26cf67c5533ba232f99bb946f5996d759a96
14	8403525	0x8c63ff9b76900c246eca66a4f1fb1b17116f03dd228ff6d17c4bd24e272dfa55
15	8404338	0xffff6b5674ff59a6ae5f6567cdf7cd36d1ab76941dc042c2bec5799bb8ba73ee9

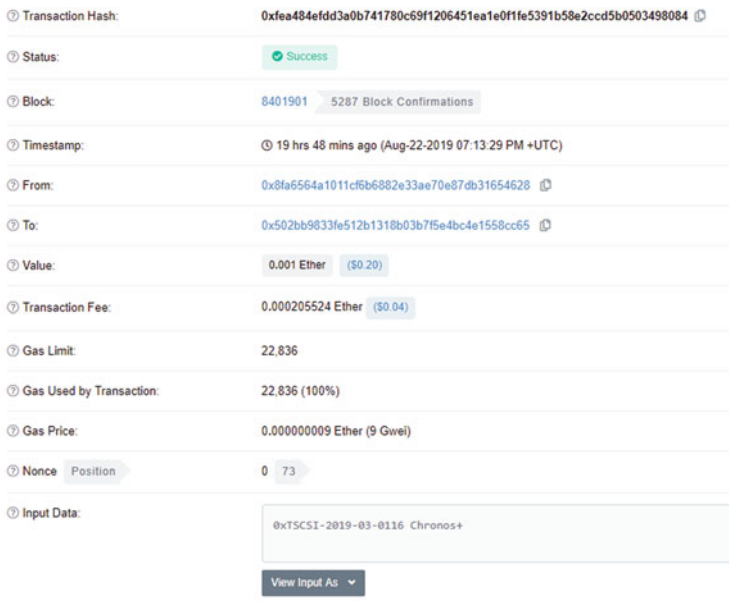


Fig. 7.9 Transaction details

We take the first transaction as an example, as shown in Fig. 7.9, if we select “view Input As UTF-8,” we can see that the data value of this transaction is “TSCSI-2019-03-0116 Chronos+.” Transactions’ details can be searched by the transaction hash value from Etherscan (<https://etherscan.io/blocks>).

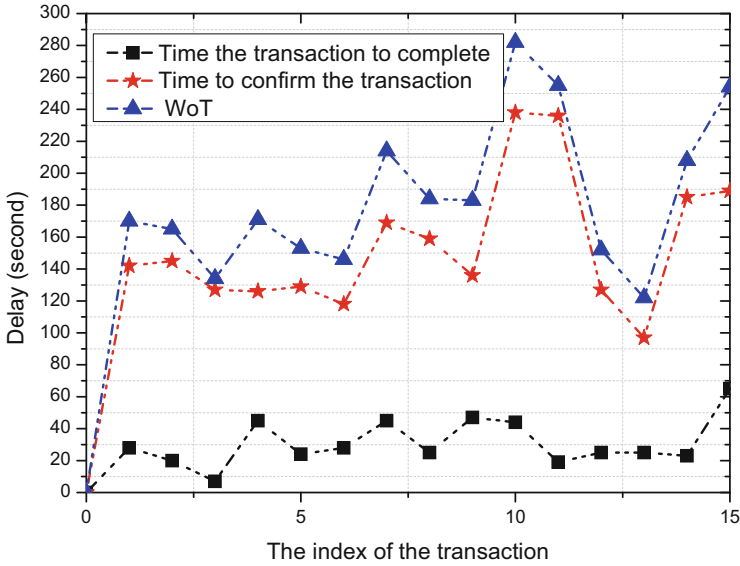


Fig. 7.10 Evaluation results

The evaluation results are depicted in Fig. 7.10. According to the evaluation results, we can observe that although WoT is fluctuant, it would not be very high. Generally, it would not exceed 300s, i.e., 5 min, due to the difficulty adjustment mechanism of Ethereum.

The transaction fees in the experiments are very low, it only takes within 0.1 US dollar to conduct a transaction (in most of the cases, the “Standard/Regular” transaction fees would not be more than 0.05 dollar).

The monetary costs and WoT of Chronos⁺ can be reduced by utilizing other blockchain systems (e.g., Ouroboros [25] and Thunderella [35] which are based on the proof-of-stake), since their transaction fees are less than that of Ethereum and their throughput capacity is stronger than Ethereum. However, the costs that an adversary breaks the security of these blockchains are much lower than those that the adversary breaks the security of Ethereum, because the market capitalization of Ethereum is much larger than them. There are too many factors affecting the security of blockchains, analyzing these factors is out of the scope of this monograph. Generally, the larger market capitalization a public blockchain has, the more participants including miners and users are in it (vice versa), and the higher costs an adversary breaks its security.

7.4 Summary and Further Reading

In this chapter, we have introduced the secure time-stamping technique. We have reviewed the traditional secure time-stamping schemes and introduced its basic idea and advantages. Then we have analyzed why these schemes are unsatisfactory when they protect outsourced files in cloud storage systems. We also have discussed the relationship between the secure time-stamping technique and blockchains and described how the Bitcoin blockchain is derived from the secure time-stamping technique. Finally, we have studied a blockchain-based secure time-stamping scheme for cloud storage systems.

There are some applications that provide secure time-stamping services, such as Tierion and Factom (<https://tierion.com/>; <https://www.factom.com/>). Readers may refer to their homepages for more information. However, these applications still suffer from some problems that exist in other time-stamping schemes and have been pointed out in this chapter. Furthermore, the data investigation technique plays a very important role in cloud storage systems, both the secure data provenance technique and the secure time-stamping technique are parts of it. In this monograph, we only introduce these two techniques. A comprehensive survey on data investigations from different aspects can be found in [36–39].

References

1. Schneier B, Kelsey J (1998) Cryptographic support for secure logs on untrusted machines. In: USENIX security symposium, vol 98, pp 53–62
2. Schneier B, Kelsey J (1999) Secure audit logs to support computer forensics. *ACM Trans Inf Syst Secur* 2(2):159–176
3. Crosby SA, Wallach DS (2009) Efficient data structures for tamper-evident logging. In: USENIX security symposium, pp 317–334
4. Pavlou KE, Snodgrass RT (2008) Forensic analysis of database tampering. *ACM Trans Database Syst* 33(4):1–47
5. Haber S, Stornetta, WS (1990) How to time-stamp a digital document. In: Annual cryptology conference, pp 437–455
6. Tolia N, Harkes J, Kozuch M, Satyanarayanan M (2004) Integrating portable and distributed storage. In: USENIX conference on file and storage technologies, vol 4, pp 227–238
7. Yang D, Yang D (2003) Intellectual property and doing business in China. Pergamon, Amsterdam
8. Dinwoodie GB (2001) The architecture of the international intellectual property system. *Chi.-Kent L Rev* 77:993
9. Adelsbach A, Pfitzmann B, Sadeghi A (1999) Proving ownership of digital content. In: International workshop on information hiding, pp 117–133
10. Zhang Y, Xu C, Li H, Yang K, Zhou J, Lin X (2018) HealthDep: an efficient and secure deduplication scheme for cloud-assisted eHealth systems. *IEEE Trans Industrial Inf* 14(9):4101–4112
11. Zhou J, Lam K (1999) Securing digital signatures for non-repudiation. *Comput Commun* 22(8):710–716
12. Bayer D, Haber S, Stornetta WS (1992) Improving the efficiency and reliability of digital time-stamping. In: Sequences II: methods in communication, security, and computer science, pp 329–334

13. Buldas A, Lipmaa H, Schoenmakers B (2000) Optimally efficient accountable time-stamping. In: International workshop on public key cryptography, pp 293–305
14. Buldas A, Laud P, Lipmaa H, Vilemson J (1998) Time-stamping with binary linking schemes. In: Annual cryptology conference, pp 486–501
15. Lipmaa H (1999) Secure and efficient time-stamping systems. Tartu University Press, Tartu
16. Zhang Y, Xu C, Lin X, Shen X (2019) Blockchain-based public integrity verification for cloud storage against procrastinating auditors. *IEEE Trans Cloud Comput*. <https://doi.org/10.1109/TCC.2019.2908400>
17. Zhang Y, Xu C, Ni J, Li H, Shen X (2019) Blockchain-assisted public-key encryption with keyword search against keyword guessing attacks for cloud storage. *IEEE Trans Cloud Comput*. <https://doi.org/10.1109/TCC.2019.2923222>
18. Zhang Y, Xu C, Liang X, Li H, Mu Y, Zhang X (2017) Efficient public verification of data integrity for cloud storage systems from indistinguishability obfuscation. *IEEE Trans Inf Forensics Secur* 12(3):676–688
19. Zhang Y, Xu C, Yu S, Li H, Zhang X (2015) SCLPV: secure certificateless public verification for cloud-based cyber-physical-social systems against malicious auditors. *IEEE Trans Comput Soc Syst* 2(4):159–170
20. Zhang Y, Xu C, Li H, Yang K, Cheng N, Shen X (2020) PROTECT: efficient password-based threshold single-sign-on authentication for mobile users against perpetual leakage. *IEEE Trans Mob Comput*. <https://doi.org/10.1109/TMC.2020.2975792>
21. Nakamoto S, Bitcoin: a peer-to-peer electronic cash system. <https://bitcoin.org/bitcoin.pdf>
22. Chaum D (1983) Blind signatures for untraceable payments. In: Annual cryptology conference, pp 199–203
23. Law L, Sabett S, Solinas J (1996) How to make a mint: the cryptography of anonymous electronic cash. *Am UL Rev* 46:1131
24. Wood G (2014) Ethereum: a secure decentralised generalised transaction ledger. Ethereum Project Yellow Pap 151:1–32
25. Kiayias A, Russell A, David B, Oliynykov R (2017) Ouroboros: a provably secure proof-of-stake blockchain protocol. In: Annual cryptology conference, pp 357–388
26. Zhang Y, Xu C, Li H, Yang H, She X (2019) Chronos: secure and accurate time-stamping scheme for digital files via blockchain. In: IEEE international conference on communications, pp 1–6
27. Zhang Y, Xu C, Cheng N, Li H, Yang H, Shen X (2019) Chronos+: an accurate blockchain-based time-stamping scheme for cloud storage. *IEEE Trans Serv Comput*. <https://doi.org/10.1109/TSC.2019.2947476>
28. Coleman J, Universal hash time. <https://www.youtube.com/watch?v=phXohYF0xGo>
29. Landerreche E, Schaffner C, Stevens M (2018) Cryptographic timestamping through sequential work. CWI, Amsterdam. Technical Report
30. Cao S, Zhang G, Liu P, Zhang X, Neri F (2019) Cloud-assisted secure eHealth systems for tamper-proofing EHR via blockchain. *Inf Sci* 485:427–440
31. Gipp B, Meuschke N, Gernandt A (2015) Decentralized trusted timestamping using the crypto currency bitcoin. arXiv preprint:1502.04015
32. Ali M, Nelson J, Shea R, Freedman MJ (2016) Blockstack: a global naming and storage system secured by blockchains. In: USENIX annual technical conference, pp 181–194
33. Tomescu A, Devadas S (2017) Catena: efficient non-equivocation via bitcoin. In: IEEE symposium on security and privacy, pp 393–409
34. Zhang Y, Lin X, Xu C (2018) Blockchain-based secure data provenance for cloud storage. In: International conference on information and communications security, pp 3–19
35. Pass R, Shi E (2018) Thunderella: blockchains with optimistic instant confirmation. In: International conference on the theory and applications of cryptographic techniques, pp 3–33
36. Quick D, Choo KR (2014) Google drive: forensic analysis of data remnants. *J Netw Comput Appl* 40:179–193

37. Martini B, Choo KR (2014) Cloud forensic technical challenges and solutions: a snapshot. *IEEE Cloud Comput* 1(4):20–25
38. Quick D, Martini B, Choo KR (2013) Cloud storage forensics. Syngress. https://books.google.com.hk/books?hl=zh-CN&lr=&id=_Q4rAQAQBAJ&oi=fnd&pg=PP1&dq=cloud+storage+forensics&ots=2ci2TBOmRu&sig=xV6lF9WMu7gW72xiHYa2iW76os4&redir_esc=y#v=onepage&q=cloud%20storage%20forensics&f=false
39. Pietro RD, Lombardi F (2018) Virtualization technologies and cloud security: advantages, issues, and perspectives. In: *From database to cyber security*. Springer, Berlin, pp 166–185

Chapter 8

Summary and Future Research Directions



In this chapter, we summarize the monograph and discuss several potential research directions for future work.

8.1 Summary

In this monograph, we have explored data security in cloud storage systems. Based on the surveys, analyses, and discussion provided throughout this monograph, we present the following highlights.

- We have introduced the cloud storage architecture and general applications. We have also pointed out general threats towards data security and general data security requirements in cloud storage systems. Furthermore, we have introduced some emerging cloud-based applications associated with challenging security issues, e.g., cloud-based eHealth systems, cloud-based data sharing systems, and cloud-based IoT.
- We have introduced some fundamental theorems, cryptographic primitives, building blocks, and techniques, which serve as the preliminary knowledge for understanding the remainder of this monograph.
- Data integrity is the fundamental factor affecting the reliability of cloud storage systems. We have analyzed the potential attacks towards the integrity of outsourced data and have provided a comprehensive survey on proofs of storage schemes to show how to ensure data integrity in cloud storage systems. Moreover, we also have introduced the latest advances in the proof-of-storage technique from two aspects. First, we have introduced how to utilize an emerging and powerful cryptographic primitive, i.e., indistinguishability obfuscation ($i\mathcal{O}$), to construct a proof-of-storage scheme to improve efficiency significantly. Second, we have analyzed existing public proof-of-storage schemes and pointed out that they are vulnerable to a dishonest auditor who might collude with the

cloud server to deceive users by generating biased challenging messages or/and procrastinating on the scheduled data verification. We have also introduced how to thwart the dishonest auditor by adopting the public blockchain (e.g., Ethereum) and studied a concrete, i.e., a blockchain-based public proof-of-storage scheme against malicious and procrastinating auditors.

- Secure deduplication is an important technique in cloud storage systems. It enables the cloud service provider to reduce storage costs dramatically by performing deduplication among all its users. We have introduced secure deduplication techniques and provided a comprehensive survey on this topic. In addition, we have also discussed potentials that cloud-based eHealth systems benefit from the secure deduplication technique and analyzed new threats and challenges towards data security in cloud-based eHealth systems when employing the secure deduplication technique. With the integration of the trust execution environment, the observation derived from actual EHRs, and the latest advances in secure deduplication, we have introduced how to construct a secure and efficient deduplication scheme for cloud-based eHealth systems.
- Searchable encryption (SE) is a cryptographic primitive that enables users to retrieve target data from their entire outsourced data set by keywords without leakage of the data contents. SE can be mainly categorized into two types: symmetric-key searchable encryption (SSE) and public-key searchable encryption (PSE). First, we have introduced SSE and provided a comprehensive survey on it. Then, we have also introduced PSE and made a comprehensive survey of it. We have analyzed that vulnerability of PSE against off-line keyword guessing attacks (KGA) on keywords becomes a major hindrance towards the broad adoption of PSE but existing PSE schemes with resistance KGA have their limitations and are unsatisfactory and have introduced a multi-server-aided PSE that thwarts KGA in a secure way. In addition, we also have introduced a blockchain-based rate-limiting mechanism and demonstrated that the multi-server-aided PSE with the integration of this rate-limiting mechanism is able to resist online KGA in an efficient way.
- Secure data provenance is the essential of bread and butter of data investigations in cloud storage systems. It securely keeps track of what happens to outsourced data. We have introduced the data provenance and secure data provenance and provided a comparison between them, which has highlighted the difference between them. Then, we have given a comprehensive survey on the secure data provenance and analyzed the pros and cons of existing schemes. Furthermore, we have demonstrated that the blockchain is a panacea for secure data provenance and utilizing the blockchain can enhance the security of secure data provenance significantly. Finally, we have introduced how to construct a secure data provenance scheme on the public blockchain to resist the malicious identity manager.
- Cryptographic time-stamping is the most important way to certify when a file was created in digital investigations. We have analyzed the secure time-stamping technique and concluded the inherent characteristics of data that would benefit from the secure time-stamping technique. We have made a comprehensive survey of the secure time-stamping technique, pointed out that the security of existing

schemes relies on the reliability of the time-stamping service provider, and elaborated on why existing schemes are inefficient to protect outsourced data in cloud storage systems. Then, we have discussed the relationship between the secure time-stamping technique and the public blockchain. Finally, we have introduced how to build a blockchain-based secure and accurate time-stamping scheme for cloud storage systems.

8.2 Future Work

In this monograph, we have introduced data security issues and techniques in cloud storage systems from five different research topics. However, there are still some problems in the above research topics, which remains open research issues that should be further investigated.

8.2.1 *Secure Data Integrity Verification from Smart Contract*

We notice that the public data integrity verification scheme that resists malicious and procrastinating auditors are constructed on the public blockchain (e.g., Ethereum). Actually, the vulnerability of public verification schemes against malicious and procrastinating auditors can be addressed by an alternative paradigm: Instead of employing the third-party auditor, the data verification algorithm originally executed by the auditor and a deposit mechanism are encapsulated in a smart contract on the Ethereum blockchain.

Specifically, after the user successfully outsources the data to the cloud server, the smart contract is also deployed to the Ethereum blockchain. The functionality of the smart contract is as follows: The cloud server initially deposits some Ethers to the smart contract; at each time when data integrity should be verified, the cloud server generates the challenging message based on the latest confirmed φ -successive blocks, computes the corresponding proof, and triggers the smart contract with taking the latest confirmed φ -successive blocks and the proof as inputs; the smart contract verifies the validity of the proof, if the verification succeeds, it refund some Ethers (less than the deposited ones, and this number depends on the system setting) to the cloud server; otherwise, the smart contract sends the deposited Ethers to the user as the punishment and compensation.

Note that if the verification performed by the smart contract fails, it means that the outsourced data might be corrupted. Since this paradigm does not employ the third-party auditor, it surely frees from the malicious and procrastinating auditor. Furthermore, by doing so, the public verification scheme frees from the costs to employ the auditor. The security of this paradigm relies on the security of Ethereum (if the smart contract does not have any bug). Therefore, as long as Ethereum remains secure, the cloud server cannot hide the data corruption.

However, there are still two challenges to be addressed. First, in Ethereum smart contracts, it is challenging to verify the validity of the “latest confirmed φ -successive blocks,” i.e., whether these blocks are truly the latest confirmed ones. Second, the Ethereum smart contract is in its nascent stages, and it only supports a few cryptographic operations. However, in the smart contract described above, some complex cryptographic operations, such as group operation in an additive group, multiplication in an additive group, bilinear pairing, need to be executed. It is plausible that the above challenges can be addressed by the next generation smart contract or emerging public blockchain systems, which remains an open problem.

8.2.2 Combination of Encrypted Deduplication and Symmetric-key Searchable Encryption

Today’s cloud storage service is very different from that in the nascent stage. Currently, a commercial cloud storage service provider always attempts to provide users with multiple functionalities in one system. For example, the service provider would provide users with an assurance of data confidentiality. Meanwhile, it also provides users with an efficient data outsourcing service by performing data deduplication and efficient data access by supporting the keyword search. This requires the cloud service provider to integrate the encrypted data deduplication and the symmetric-key searchable encryption (SSE) into one system.

However, there are also some challenges to be addressed. First, the security and privacy model for such a system should be re-defined, rather than integrating each one of the two techniques into one straightforward. We notice that to ensure the functionality of encrypted deduplication, it is inevitable to allow the cloud server to know whether two users have the same file and which file is owned by a specific group of users. On the other hand, the security of SSE requires that it would leak nothing about the outsourced files before any keyword search request is submitted and it would leak nothing beyond the search pattern and access pattern after one or more keyword search requests are submitted. It seems that straightforward integrating MLE and SSE into one system fails to meet the security of SSE. Worse still, it might introduce new threats towards the data security and users’ privacy, since the adversary (i.e., the adversarial cloud server) might extract more information about outsourced data than that from each separate one.

Moreover, in the scheme, the keywords of each file should be individually encrypted by each user, and different users surely generate different secure indexes for the same file, which also causes the key management problem to the user (since the files and keywords are encrypted using different keys) and data management to the cloud server (the data stored on the cloud server in a very complex form).

Therefore, how to design a cloud storage system that supports encrypted deduplication and keyword search simultaneously is still an open problem and needs to be further explored.

8.2.3 Secure Provenance Under Complex Models

The blockchain-based secure data provenance scheme, i.e., ESP, mainly focuses on how to resist the malicious identity manager. The underlying data provenance model is somewhat simple. It only supports that multiple users process a target file one by one. However, this model is not suitable for systems that a file is edited by multiple users concurrently, e.g., Google Docs. Applying the blockchain-based secure data provenance scheme in these systems, whereas when some users concurrently edit a file, it could lead to a fork on the provenance record chain.

Therefore, it is desired to construct a blockchain-based secure data provenance scheme on a complex data provenance model while remaining all characteristics of ESP (i.e., resistance against malicious identity manager and efficient provenance verification), which supports secure data investigations in the cloud storage systems with different paradigms of data management.

8.2.4 Securely Time-stamping Operations in the Digital World

The secure time-stamping schemes we introduced in this monograph only target at protecting digital data in the digital world. However, in cloud storage systems, in addition to the digital data, some operations performed by users should also be time-stamped for post investigations.

In Sect. 3.4.2, we have studied the blockchain-based public data integrity verification scheme against procrastinating auditors. This scheme is actually a prior work on time-stamping operations in the digital world and surely has significant value for not only the research topic of data integrity verification, but also for the research topic of secure time-stamping. In this prior work, the key challenge to transfer a secure time-stamping scheme for digital files to that for operations is to securely transfer an operation to a file in the digital world. In the above scheme, this is achieved by two techniques: the one is the blockchain-based time-dependent random seed, and the other one is the digital signature.

Nevertheless, since copying digital files and re-generating signatures are trivial for the adversary who originally generates the signatures, it is still unclear that the above mechanism can be utilized in other application scenarios to timestamp users' operations. Constructing a general framework of time-stamping operations in the digital world, defining its security model, formulating its threat model, and providing a concrete are very challenging but important in the research area of data investigations, which also is worth to be investigated.