Efficient, Revocable and Auditable Access over Encrypted Cloud Data

by

NAVEEN KUMAR
201021001

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY
in
INFORMATION AND COMMUNICATION TECHNOLOGY
to

DHIRUBHAI AMBANI INSTITUTE OF INFORMATION AND COMMUNICATION TECHNOLOGY

May, 2017

Declaration

I hereby declare that

i) the thesis comprises of my original work towards the degree of Doctor of Philosophy in Information and Communication Technology at Dhirubhai Ambani Institute of Information and Communication Technology and has not been submitted elsewhere for a degree,

ii) due acknowledgment has been made in the text to all the reference material used.

Naveen Kumar

Certificate

This is to certify that the thesis work entitled "Efficient, Revocable and Auditable Access over Encrypted Cloud Data" has been carried out by Naveen Kumar for the degree of Doctor of Philosophy in Information and Communication Technology at *Dhirubhai Ambani Institute of Information and Communication Technology* under my supervision.

Prof. Anish Mathuria
Thesis Supervisor

Acknowledgments

I would like to thank all the people who contributed directly or indirectly to the work described in this thesis. First and foremost, I thank my academic supervisor, Professor Anish Mathuria, for accepting me in his group. During my tenure, he contributed by giving me intellectual freedom in my work, supporting my attendance at various conferences, engaging me in new ideas, and demanding a high quality of work. Also, I would like to thank my committee members Professor Ashok Amin, Professor Manik Lal Das and, Professor Sourish Dasgupta for their interest in my work. I specially thank to Prof. Manik Lal Das who have shaped my thinking about this research direction. Additionally, I would like to thank our collaborators Professor Kanta Matsuura, and Dr. Takahiro Matsuda at University of Tokyo for their valuable suggestions in the initial phase of my research work. I would also like to thank Professor R. Nagaraj, Professor Suman Mitra, Professor Sanjay Chaudhary and Professor Mehul Raval for their financial and mental support through various means during my tenure.

Every result described in this thesis was accomplished with the help and support of fellow lab-mates and collaborators.

I would also like to say a heartfelt thank to my family for always believing in me and encouraging me in whatever way they could during this challenging period.

Naveen Kumar
May 2017

Contents

Abstract **viii**

List of Principal Symbols and Acronyms **viii**

List of Tables **ix**

List of Figures **x**

1 Motivation and overview **1**

1.1 Introduction 1

1.1.1 Revocation 3

1.1.2 Data privacy 3

1.2 Reference architecture 4

1.2.1 Data consistency and serializability 6

1.2.2 Types of consistency 7

1.3 Security requirements 8

1.3.1 Application: Cloud-based e-Health 9

1.3.2 Other requirements 11

1.4 Our contributions 12

1.4.1 Read access control 12

1.4.2 Write access control 16

1.4.3 Privacy enabled access control 19

1.5 Organization of the thesis 20

2 Key management for read access control **21**

2.1 Access control matrix representation 21

2.2 KMH: definitions and properties 24
2.2.1 User-based hierarchies 24
2.2.2 Resource-based hierarchies 31
2.2.3 Comparison of static hierarches 33
2.3 Dynamic access control 35
2.3.1 Algorithms for user hierarchy 35
2.3.2 Algorithms for resource hierarchy 37
2.3.3 Comparison of dynamic hierarchies 39
2.4 Experimental evaluation 40
2.5 Re-keying 46
2.5.1 Wang et al. protocol 46
2.5.2 Improving efficiency 49
2.6 Summary 51

3 Key management for time-bound read access control 53
3.1 Introduction 53
3.1.1 Classification 55
3.2 Previous schemes 56
3.2.1 Ateniese et al. scheme 56
3.2.2 Atallah et al. base scheme 59
3.2.3 Atallah et al. improved scheme 60
3.2.4 Improved scheme with shortcut edges 65
3.2.5 Crampton scheme 68
3.3 A new construction with reduced public storage cost 70
3.3.1 Extension to multiple resources 73
3.3.2 Security analysis 75
3.4 Comparison with related schemes 77
3.5 Dynamic operations 79
3.6 Summary 81

4 Enhancing write integrity and data freshness 82
4.1 Introduction 82

4.2 Preliminaries 85
4.2.1 A resource record 85
4.2.2 Chain hash 85
4.2.3 Auditing 86
4.3 Audit-based protocols 88
4.3.1 Protocol for enforcing time-limited access 89
4.3.2 Protocol to defend against revoked access 92
4.3.3 Experimental evaluation 94
4.4 Freshness guarantee 96
4.4.1 Existing notions 96
4.4.2 Improved notion: $< v, d, t >$-staleness 98
4.5 Related work 100
4.5.1 Write-serializability 100
4.5.2 Freshness 102
4.5.3 Revocation 102
4.6 Summary 103

5 A novel PHRMS scheme supporting unobservability and forward secrecy 105

5.1 Introduction 105
5.2 System overview 108
5.2.1 Requirements and assumptions 110
5.2.2 Unobservability 110
5.3 PHR encryption and access control 111
5.3.1 Key management 112
5.3.2 Comparison 114
5.4 Protocols for publishing PHR documents 115
5.4.1 Publishing a medical prescription 115
5.4.2 Publishing a Lab report 117
5.5 Security analysis 120
5.6 Summary 123

6 Conclusions 124

References 127

Appendix A Some algorithms 138

Appendix B Formal security proof for new SBHKAS 140

Appendix C Formal security analysis of Lab report publishing protocol 147

Abstract

Cloud data outsourcing services can potentially help reduce the IT budget of organizations. However, they pose significant risks to the security and privacy of the data as the data is outsourced to untrusted third-party servers. In this thesis, we propose security mechanisms for cloud data access control using symmetric key primitives.

The contributions of this thesis are summarized below.

- We critically analyze the two types of key management hierarchy used for access control in outsourced data: user-based and resource-based. We show that both types of hierarchy have comparable public storage requirements. This result disproves a common belief that resource-based hierarchies require significantly more storage than user-based hierarchies. We also show that resource-based hierarchies are more efficient in terms of computation and communication cost as compared to user-based hierarchies with respect to dynamic operations. The performance evaluation of dynamic operations is shown experimentally.

- We design a subscription-based hierarchical key assignment scheme with single key storage per user. Our construction is based on indirect key derivation with dependent keys. It reduces the public storage requirement of existing schemes, while also reducing the secret storage cost at the central authority. The scheme is formally analyzed using the provable security notion of key non-recovery. To our knowledge, this would be the first hierarchical key assignment scheme using dependent keys with a rigorous security proof.

- A weakness of existing write access control schemes is that a write authorized user can modify the files written by him even after the write privilege is revoked. We propose audit-based protocols so that if any unauthorized writes are performed they can be detected by the data owner. The protocols are implemented on Microsoft Azure platform and it is shown that the suggested mechanisms are viable in practice.

  It is important to ensure that the read operation returns the latest updated version of the requested file. The service provider may misbehave by sending an old version of a file instead of the current version. If the read operation returns stale data, the reader may be mislead. We propose an audit-based mechanism that provides a strong freshness guarantee ensuring that the file returned by the read operation is fresh at least until the time when the file was sent by the service provider.

- A cloud-based personal health record (PHR) management system allows a user to store, share and update her outsourced PHR data, access online medical services, at anytime and from anywhere. Unlinkability is an essential privacy requirement for such system which ensures that PHR data cannot be linked to its owner. However, a cloud service provider can still observe the linkage between them as it can observe the traffic. We propose a symmetric key based PHR management system that provides a stronger privacy guarantee called unobservability. Unobservability implies unlinkability between the communicating parties against a malicious service provider, whereas the converse is not true.

List of Tables

1.1 Comparison of Wang et al. scheme and proposed modified scheme 17

2.1 Notations used 23
2.2 Secret keys with each user 26
2.3 Comparison of minimal vertex resource hierarchy with user-based hierarchies 35
2.4 Comparison of computation and communication cost 40
2.5 Grant and revoke cost in user hierarchy 44
2.6 Efficiency comparison 51

3.1 Notations and abbreviations used 55
3.2 Comparison of shortcut schemes 67
3.3 Key assignment to the subscription hierarchy given in Figure 3.11(c) 71
3.4 Comparison of single key $SBHKAS$ 79

4.1 Notations and abbreviations used 85
4.2 Outsourced resource record structure proposed by Popa et al. . . . 85
4.3 Comparison of existing schemes against the considered security properties 103

5.1 Notations and abbreviations used 108
5.2 An example PHR 112
5.3 Encryption keys in the example PHR 113
5.4 Coarse level comparison of PHRMS schemes 114

List of Figures

1.1 A typical cloud architecture 5

1.2 A non-serializable schedule 7

1.3 A reference architecture for PHRMS 10

1.4 (a) User u subscribes for $Jan - May$ and user u_1 subscribes for $Apr - Jun$ (b) Subscription withdrawn by user u for Apr and May months. 15

2.1 An example access control matrix as (i) ACLs (ii) CPLs. 22

2.2 Example hierarchy structures based on (i) ACLs (ii) CPLs. 22

2.3 A user graph over a set $\{A, B, C, D\}$ of four users. 24

2.4 (i) Example ACLs with read authorization, (ii) A user tree, and (iii) Minimal vertex user tree 26

2.5 (i) Example ACLs with read authorization, and (ii) A minimal vertex user hierarchy. 29

2.6 A resource graph over a set $\{a, b, c, d\}$ of four resources. 31

2.7 (i) Example CPLs, and (ii) A minimal vertex resource hierarchy. . . 32

2.8 Modified example of minimal vertex user hierarchy (i) before, and (ii) after granting read access 37

2.9 Modified example of minimal vertex user hierarchy after revoking user D 37

2.10 (i) An example minimal vertex resource hierarchy, and (ii) Granting read access for resource c to user C. 38

2.11 After revoking read access of resource d from user B. 39

2.12 Permission operation cost 41

2.13 Average elapsed time of one grant/revoke operation 42
2.14 Elapsed time performance of data owner machine for evaluating user threads 43
2.15 Elapsed time of one grant read subscription operation 44
2.16 Average elapsed time of one user revoke operation 45
2.17 Data outsourcing architecture 50

3.1 An example subscription hierarchy 54
3.2 An example of user hierarchy 56
3.3 (a) User hierarchy, (b) Subscription hierarchy 57
3.4 A complete system hierarchy corresponding to Figure 3.3 57
3.5 Base data structure with $|T| = z$ 60
3.6 Data Structure with $|T| = 16$ 61
3.7 $D(v), L(v), R(v)$ for node v with ($|T| = 16$) 62
3.8 Base data structure with one-dimensional sub-hierarchies 67
3.9 A nodes structure for an example subscription hierarchy (with $z = 4$) 69
3.10 (a) Division of nodes, and (b) Key derivation 69
3.11 (a) An example subscription hierarchy structure, and (b) Key derivation structure corresponding to subscription hierarchy in Figure (a) 70
3.12 (a) User hierarchy, (b) Subscription hierarchy structure, and (c) System subscription hierarchy 75

4.1 Write operation latency 95
4.2 Data owner performance for computing MACs over chain hashes . 95
4.3 Implementation cost for Protocol 2 96
4.4 Attestation protocol 102

5.1 The PHRMS system 108
5.2 A view of cloud data store 111
5.3 Publishing a medical prescription 116
5.4 Publishing a lab report 117
5.5 CLab handling two user requests 119

B.1 Type of hierarchies in Case 2 142
B.2 Type of hierarchies in Case 3 145

CHAPTER 1

Motivation and overview

In the present digital age, information and communications technology plays a significant role in day-to-day life. Information can be gathered, stored in a computer or communication system, and shared by a large number of users. In recent years, data outsourcing in the cloud has emerged as an attractive solution for small scale IT organizations having a huge amount of data and large global user base. An organization can outsource its data to a third party cloud service provider that stores the data and allows the authorized users to access it. However, since the cloud service provider is an independent entity, it cannot be trusted by the data owner.

In this chapter, we discuss the security, privacy and efficiency issues related to the read and write access to outsourced data, and outline our contributions.

1.1 Introduction

The loss of control over its data presents a serious risk to the business operations of any organization. An important security goal is to ensure that no unauthorized user can access any sensitive corporate data. This goal is known as "access control". There are two basic types of access permission: *read* and *write*. A read authorized user can only read the file's content, whereas a write authorized user can write a new file or modify the content of an existing file. There are non-cryptographic techniques to implement access control mechanisms such as file access control in an operating system. However, they will not work in data outsourcing scenario as the cloud service provider is untrusted. We therefore consider the use of cryptographic access control to protect against unauthorized read and detect unauthorized writes by any unauthorized user including the cloud service provider.

The data files are encrypted by the data owner before outsourcing them to the cloud. For read access control, each data file for which certain users are authorized is encrypted with a distinct secret key. The keys are then distributed to the authorized users who can now retrieve the outsourced encrypted data file directly from the cloud server and decrypt it.

As with read access control, write access control ensures that only write authorized users can write the outsourced data files. A read authorized user can access a resource (or file) only if it has the corresponding decryption key. As we assume that the service provider is untrusted, it may allow unauthorized write access, which can significantly affect the data owner's business operations.

Two basic types of cryptographic systems can be used for enforcing access control: asymmetric key-based and symmetric key-based. Asymmetric cryptosystems use different keys for encryption and decryption operation, whereas symmetric systems use the same key for both the operations. There exist schemes ([1, 2, 3]) which implement access control using Attribute-Based Encryption (ABE) [4]. We note that all such schemes require a trusted third party to manage the system attributes. These schemes require a large number of keys to be stored at each user, i.e., one corresponding to each attribute he/she is authorized for. A symmetric key-based cryptosystem uses comparatively smaller key size and the cost of encryption/ decryption operations is less than in the ABE.

In this thesis, we explore symmetric key-based cryptographic access control techniques or approaches as they are more efficient than those based on advanced cryptographic primitives. We address enforcement of dynamic changes to access rights as it is an important aspect of access control. An example of dynamic operation is extending or revoking read (or write) access permission for a user. Also, privacy related issues in data outsourcing are explored.

In what follows, we give a short introduction to access right revocation and data privacy in data outsourcing scenario. Section 1.2 will discuss the considered reference architecture and related security requirements will be discussed in Section 1.3. Section 1.4 will give an overview of our contributions in this thesis.

1.1.1 Revocation

An important security requirement related to access control is “access right revocation”. The goal of revocation is to ensure that a user whose access right is revoked cannot access any update of the revoked resource (or data file) which is published after revocation. Fu et al. proposed *lazy revocation* [5] to handle access right revocation, i.e., a revoked file is re-encrypted only when the file is modified for the first time after being revoked. It is challenging to enforce revocation in the cloud outsourcing scenario because it requires cooperation between the data owner and the service provider. The revocation mechanism must be efficient, i.e., it should not impact a large number of users or resources.

1.1.2 Data privacy

Data privacy deals with the ability of an individual to determine what information can be shared with others [6]. It is a concern related to disclosure of personal information. Anonymity is a privacy property. User anonymity is the state of a user being not identifiable within a set of users, the anonymity set [7]. It ensures that a user involved in some transaction be non-linkable with the transaction. Similarly, anonymous message refers to a message that does not reveal its originator’s identity.

The use of encryption does not hide the linkage between a sender and the message that is sent, or the linkage between the two communicating parties. An adversary can see the corresponding packet header and view the identity information of communicating parties. The linkage information may cause a breach of privacy. For example, if it becomes known that a patient is communicating with a specialty doctor such as a psychiatrist, this will reveal the type of illness.

Sometimes it is desired that a user can establish a long-term relationship to his/her data, without revealing his/her association to the data item. This is especially needed in a privacy enabled data outsourcing application, such as, cloud-based e-Health management system where the user’s private medical data is stored with an untrusted cloud service provider. A mechanism of establishing a long-term relationship between a user and his/her data is by using pseudonyms [8]. It associates a unique identifier with a data item used by the authorized users to access the data. A unique identifier can be an account number, nickname, etc. Such identifiers cannot be trivially linked to a user, thus ensuring message anonymity. However, such pseudonymization process requires a trusted third party which stores the linking information and helps the document receivers to link it to its owner. Although the pseudonymization process provides unlinkability, it does not provide unobservability which is a stronger privacy guarantee. Unobservability of an item of interest (the resource) means its undetectability against all subjects (may be an outsider such as the cloud service provider) uninvolved in it [7]. It is to be noted that unlinkability does not imply unobservability, whereas the converse is true [7]. This is true because using network traffic analysis one can identify who communicates with whom.

1.2 Reference architecture

A typical data outsourcing architecture as shown in Figure 1.1, consists of three entities, viz., a data owner, a cloud service provider (CSP), and the end users. We consider the user-read-write setting (in contrast to the owner-write-user-read setting in [9, 10, 11]), where a user can have both read and write access permissions to the outsourced files. An example of the user-read-write setting is a blog on social networking site where a user can read every blog and write her own. Similarly, an example of the owner-write-user-read setting is an online e-Newspaper, where a publisher writes the news content and online users are the readers.

There are two types of write privileges: write-with-read [12] and write-with-or-without-read [13]. In the first type, a user can have write access only if she has read access to the file, i.e., a user can read and modify the existing file content. Two examples of this kind are Google docs where many users can update a document at the same time, and blog writing on social networking sites. In the second type, a user can have write access to a file without having read access, i.e., such user

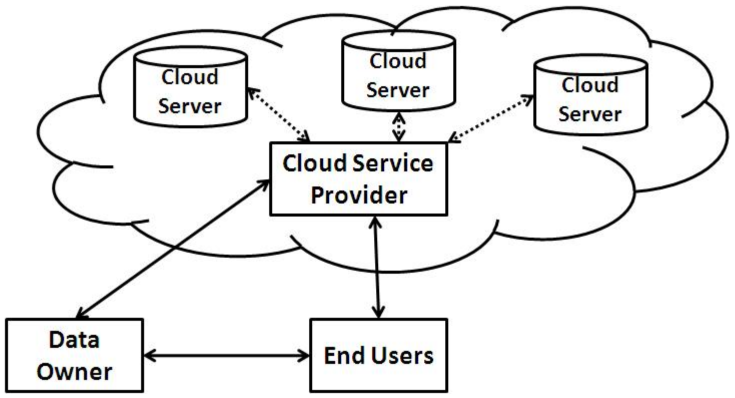


Figure 1.1: A typical cloud architecture

can only write a new file to the server. An example of this type is the dropbox-like behavior of directories where one can only drop (or store) the data files and cannot read any file from the directory.

The data owner is assumed to have read and write access to all the outsourced files. It is assumed that only the data owner can delete these files; the other authorized users can only read and write them. The CSP mediates access to the outsourced data based on the access control policy defined by the data owner. The end users first register with the data owner and are issued some access credentials. Upon a read request, the CSP authenticates the user and responds with the requested resource. The end users send their access requests directly to the CSP, instead through the data owner. This reduces the computational load on the data owner. We assume that the CSP is always on-line, whereas the data owner may be off-line.

For the purpose of security analysis, CSP behavior has been classified into four types as discussed by Arapinis et al. [14]: honest, honest-but-curious, malicious-but-cautious and malicious. For read access scenario, a standard assumption about the CSP is that it is "honest-but-curious", i.e., it does not launch any active attack. However, it may launch passive attacks such as data eavesdropping. The assumption that the CSP is honest-but-cautious is not appropriate for write access control, since the write operation is executed under the supervision of the CSP and an unauthorized modification to a file can ruin the data owner's business operations. Therefore for write access, we consider "malicious-but-cautious" CSP that launches no attack that leads to any provable trace, i.e., one cannot prove that the misbehavior is done by the CSP. We do not consider the last type, i.e., malicious since it can have negative effects on the CSP. Such CSP may launch active attacks and may collude with the unauthorized users in order to access the outsourced content. It may maliciously provide wrong information or deny any authorized user's request. If it is found that the CSP has engaged in some activity that violates the business agreement with the data owner, the data owner may blacklist the CSP.

In a distributed cloud system, the data is cloned at multiple servers to allow fast data access from closest server and data recovery in emergency situations such as a server crash. At the time of write access, a file is modified or updated by a write authorized user on the closest server. The CSP then propagates the updates to other servers. The outsourced data can be accessed by multiple users concurrently. The cloud is responsible to ensure that users see a coherent view of the stored data files.

1.2.1 Data consistency and serializability

As defined in traditional database management systems, a transaction is a group of one or more read and write operations over one or more data items. It is desirable that concurrent transactions do not change a data item in such a way that the resulting view of the data is incorrect. For example, consider two transactions, T1 and T2, containing two operations read and then write to some data item x (denoted, read(x) and write(x), respectively). A schedule is shown in Figure 1.2 and is written as

$$T1 : read(x) \rightarrow T2 : read(x) \rightarrow T1 : write(x) \rightarrow T2 : write(x)$$

A schedule is called serializable if it is equivalent to some serial execution of the transactions. It is the commonly accepted criterion for database correctness. The above schedule is not serializable. This is because it is not equivalent to either T1→T2 or T2→T1. The reason is that both of the write operations (T1:write(x) and T2:write(x)) have updated over the same value in the read operation. The first update of T1 is lost as it is overwritten by the second update of T2. Following two types of sequence of operations can result in non-serialized (also known as conflict serialized) schedule: read-write and write-write.

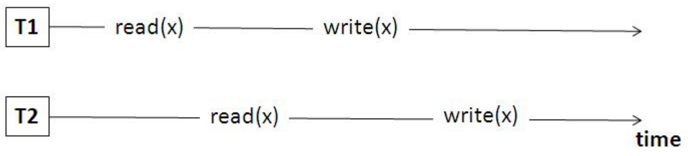


Figure 1.2: A non-serializable schedule

Data consistency ensures that the execution of concurrent transactions will be equivalent to some serial schedule of the transactions (also known as write-serializability [15]). For example, a concurrent execution of the two transactions T1 and T2 will be same (or consistent) as either a schedule where T1 executed before T2 or where T2 executed before T1.

In a data outsourcing scenario with the untrusted service provider and concurrent transactions, it is an essential requirement to enforce write-serializability, in the absence of which the service provider can maliciously delete an important update from the storage. For example in the case of two transactions with read-write operations sequence, the service provider can execute just one out of the two. Recent works ([15, 12, 16]) use timestamps or multi-versioning to enforce write-serializability in data outsourcing transactions. Timestamps capture the time when a transaction happened. Versioning ([17]) assigns an incremental sequence number to each write operation on a data item.

1.2.2 Types of consistency

In the literature [16, 18] there are three notions of data consistency: strong consistency, weak eventual consistency, and strong eventual consistency.

Strong consistency ensures that updates are visible to all readers at the same time, i.e., the update is not visible until the updated value is replicated at all nodes. It requires that all accesses to any copy of that data are blocked (or locked) until that time. Therefore, it guarantees that a read will always return the most recent copy. The strong consistency model is required for real-time systems such as ebanking. It is noted here that implementing strong consistency negatively affects the availability requirement.

A weak eventual consistency model guarantees that the system will eventually become consistent and have the most up-to-date version of data for all copies. A read returns the version it finds first, whether or not it is the most recent version. It has better data availability as compare to strong consistency. An example includes logging data for weather forecasting applications where one or two hours stale data may not have any adverse effect.

A strong eventual consistency model guarantees that any two nodes that have received updates over the same data item will be in the same state. An example of the use of strong eventual consistency is online bidding application.

1.3 Security requirements

We list below some security requirements that are commonly considered for access control on outsourced data [15, 19, 12]. Our security model does not consider malicious clients.

Data confidentiality The data confidentiality is a set of rules that prevents access to the data from unauthorized users. In data outsourcing scenario, it ensures that no unauthorized user including the CSP can access the outsourced data. This requirement is important when an organization outsources it's secret data to an untrusted CSP.

Data integrity The data integrity property is the assurance that the data has not been modified by unauthorized users. In data outsourcing scenario, it ensures that no unauthorized user (including the CSP) can update or modify the outsourced content even if they collude. An update here means writing a new version of a data file and by modifying a file we mean altering the content of the file.

Write-serializability In database management system, serializability ensures that a schedule for executing concurrent transactions is equivalent to one that executes the transactions serially in some order. While data versioning, a copy of the original file is kept and modified file is kept as a new version. Therefore, a sequence of versions is stored for each data file. In data outsourcing scenario, a malicious CSP may hide an unauthorized user's update from the data owner while making it visible to the other authorized readers. This will mislead the readers including the data owner. The "write-serializability" property ensures that all users will see all versions of a resource in the same order as they are updated by various users [15]. It ensures strong eventual consistency, i.e., each update is written on the latest update.

Freshness The freshness property implies that the viewed data is recent and not replayed by an adversary. It ensures that a read operation will always fetch the latest updated version of the requested resource. The stale data may mislead the readers in many time-sensitive applications. For example, in a stock market application, a small delay in updating the bid price can cause a huge loss to a bidder. Similarly, a small delay in getting the current status of seat allocation in a railway reservation system may prevent a user from getting a confirmed seat reservation. The freshness requirement should be relaxed since staleness cannot be fully avoided in a distributed setting due to the delay in data replication process.

Access right revocation It implies removal of access right permissions for an object from a previously authorized user. A user whose access rights are revoked should not be able to access any new or modified file which is published or modified after revocation.

1.3.1 Application: Cloud-based e-Health

E-Health effectively uses information and communication technology to provide health-related services to end users. A primary objective of an e-Health system is to efficiently manage its users' e-Health data, providing faster access to health services. Personal health record (PHR) has emerged as a prominent e-Health model that is controlled and managed entirely by the individual. PHR is a collection of private health related information of an individual [20]. A record management system for personal health records is called personal health record management system (PHRMS). A cloud-based PHRMS allows a user with resource constrained device to store, share and update her outsourced PHR data, access different medical services online, anytime and from anywhere. Security and privacy of the PHR data are the major concerns in cloud-based e-Health [21].

PHRMS architecture

As shown in Figure 1.3, a typical PHRMS has two types of user: primary user and the secondary user. A primary user is an entity such as a PHR owner or a doctor. A secondary user is an entity such as surveyor or researcher. The PHR owner will create, maintain and allow authorized access to her PHR. A doctor can generate medical prescriptions and progressive notes as and when requested by the PHR owner. The PHR service provider performs the following functions: users registration, store and maintain each user's PHR information, and process service requests of authorized users.

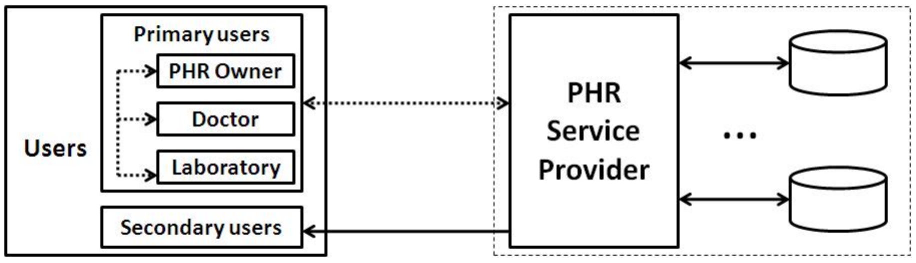


Figure 1.3: A reference architecture for PHRMS

We list below two important privacy requirements for a PHRMS:

Data unlinkability Unlinkability of two or more items of interest (IOIs, e.g., subjects, messages, actions, ...) from an attacker's perspective means that within the system, the attacker cannot sufficiently distinguish whether these IOIs are related or not [7]. In the considered scenario, it ensures that no unauthorized user including the cloud service provider can link a user's identity with his PHR. The service provider can link a document with its owner while observing the real-time communication between the entities if the same pseudonym is used more than once [7]. The use of distinct pseudonym per document is difficult to achieve in healthcare systems where documents are stored on third party servers. Therefore, a secure communication healthcare architecture is needed where data access patterns are unobservable by an adversary who can sniff the communication traffic. The dotted line in Figure 1.3 represents a link where at least one end is connected with a primary user. Such links are vulnerable to privacy threats and are required to be unobservable.

Forward secrecy This property ensures that a user cannot access any future version of a resource using expired credentials. For example, access to future e-Health record information by the doctor using expired credentials must be restricted (forward secrecy), since the patient may change her doctor at any time. It is different then revocation in a way that access rights are not forcedly revoked, however they get expired as the consulting session between a patient and her doctor ends. If forward secrecy is not provided, an unauthorized doctor can see the patient's future consultation information, such as to whom she is consulting and what prescriptions she is getting. In general, a patient may not want to disclose their PHR information to any person including a doctor without her consent.

1.3.2 Other requirements

Below we briefly describe some of the other security requirements that are not in the scope of this thesis but presented here for the sake of better understanding our system.

Proof of storage It is important for a data owner to verify its outsourced data against any undesired modification intentionally or unintentionally. Ideally, it requires downloading the whole data locally by the data owner and verify it whenever needed which is an inefficient process. The proof of storage ensures that a data owner or an auditor (on behalf of the data owner) can verify the integrity of its outsourced data without downloading the whole data locally [22].

The data owner constructs queries for a set of randomly selected resources for which the service provider sends back a response. The response messages are then verified by the data owner. Additionally, the time it takes to receive a response can be used as a proof of geographic location where the resource is stored [23].

Assured data deletion This property ensures that after a file delete operation takes place the file will be permanently inaccessible. If all copies of the file are not removed from the cloud, then a malicious party may be able to access the data. Assured deletion also ensures that the data owner does not have to pay for unwanted storage [24].

Searchable encryption This enables secure search queries over encrypted sensitive data without disclosing any information about the data or search query [25]. The objective is to preserve user privacy involved in the search operation. For efficiency reasons, keyword-based search techniques are generally used. The integrity of the query results needs to be verified by the user [26].

1.4 Our contributions

In this section, we outline the contributions of this thesis.

1.4.1 Read access control

Analysis of key management hierarchies To reduce the amount of secret storage, *key management hierarchies* are generally used. A key management hierarchy can be represented as a directed acyclic graph, where a key is assigned to each node in the graph. Each file is associated with some node and is encrypted with the key of that node. The edges of the graph represent the direction of key derivation. The relationship among the keys is such that using a key one can efficiently compute any descendant node's key. Also, it is computationally infeasible to derive a key corresponding to a non-descendant node.

There are two existing types of key management hierarchy for data outsourcing: user-based [13, 12] and resource-based [27]. In a user-based hierarchy, each node represents a set of users having access to that node's key. On the other hand, each node of a resource-based hierarchy represents a subset of resources such that a user having access to the node's key can access each resource associated with the node.

When considering dynamic operations such as extending read access and user revocation, we will show in Chapter 2 that the resource-based hierarchy has a significant advantage over user-based hierarchy. To perform the extend read access operation on a user-based hierarchy, it is necessary to re-encrypt the resource. In the resource-based hierarchy, no such resource re-encryption is needed. Similarly, to perform user revocation, the user hierarchy is required to be modified, whereas in the case of resource hierarchy the hierarchy structure is kept intact. Therefore, to effectively deploy these common dynamic operations, we recommend the use of resource-based hierarchies for key management. We have implemented both types of hierarchy to demonstrate our analytical results experimentally.

Let R be the set of resources. Then there are $2^{|R|}$ subsets of R, where $|R|$ denotes the cardinality of R. Consider a resource hierarchy that contains a unique node for each possible subset of R. Clearly, the number of internal nodes will be equal to $2^{|R|}$. Let U be the set of users. Now consider a user hierarchy that contains a node for each possible subset of U. Clearly, the number of internal nodes will be equal to $2^{|U|}$. If we assume $|U| < |R|$, then the resource hierarchy will take more space than the user hierarchy. However, as we shall show in Chapter 2, both types of hierarchy do not require more than $O(|R|)$ space (a part of this work is published in [28]).

New key assignment scheme A hierarchical key assignment scheme (HKAS) [29, 30] is a method for assigning encryption keys and private information to each node in the hierarchy in such a way that using a node's key it is feasible to derive the keys of its descendants, whereas it is infeasible to derive the key of any other node.

There are specialized kinds of applications which require subscription-based (or time-bound) access to the data. For example, in a digital pay-TV system, the service provider organizes the channels into several possible subscription packages that can be accessed by the authorized users for a fixed period of time.

A subscription-based $HKAS$ (or $SBHKAS$) assigns keys in a subscription hierarchy so that a user subscribed to a node in the hierarchy can efficiently access (only) its authorized subscription keys. Recently proposed SBHKASs [31, 32, 33, 34, 35, 36] require a single secret key per user and large amount of public storage. A trusted Central Authority (CA) will generate, assign and maintain secret keys in the system. In our architecture, the data owner works as a CA who will generate secret keys and then distribute them among the authorized users. The secret keys are never disclosed to the untrusted cloud service provider. We have designed an alternative scheme using indirect key derivation with dependent keys [37]. The proposed scheme reduces the secret storage at CA and public storage (as compared to the most promising scheme by Crampton [36]). It reduces the secret storage at CA to one key and improves the public storage by a factor of $\approx 3/8$ with same key derivation cost. For example, a weekly subscription hierarchy for 10 years will contain 260261 nodes [1]. Then the public storage cost of Crampton's scheme will be 2.7 lakh edges, whereas the proposed scheme has a storage cost of 1.7 lakh edges.

We have carried out a formal security proof of the proposed SBHKAS using modern notion of security known as "key recovery". This would be the first provable security style proof for any dependent SBHKAS in the literature. In the existing schemes (Atallah et al. [38, 30] and D'Arco et al. [39]) with independent keys, the information related to a node and its successor is only in the public information. In the schemes with dependent keys, the related information is also found in the secret keys. Atallah et al. constructed the proof by breaking the dependency in the public information, using which a node can derive its successor's key. They used an assumption based on the security of pseudo random functions and then using an adversary against their scheme, they constructed a probabilistic polyno-

[1] The number of nodes in a subscription hierarchy with n leaf nodes is $n(n+1)/2$. Therefore, the number of nodes in a subscription hierarchy with 520 leaf nodes (the number of weeks in 10 years) is $(520 \times 521)/2$, i.e., 260261

mial time algorithm to break the security of pseudo random functions. In case of dependent scheme, we need to use different assumption (i.e., *preimage resistance*) because the information is scattered and present even in the secret keys of siblings of all the nodes encountered in the path from root node to target node. So, we have to replace this information with random strings.

Access right revocation The goal of revocation is to prevent a user from accessing a resource in future. As an example of a scheme that fails to provide security against a revoked user, we consider a SBHKAS proposed by Vimercati et al. [40].

Consider a hierarchy shown in Figure 1.4 (a) where user u is subscribed for time interval from Jan to May. Suppose a user u withdraws his subscription for the months of Apr and May (before the start of Apr). The data owner will first update u's subscription with $Jan - Mar$. Since the node $Jan - Mar$ is present in the subscription hierarchy, data owner will compute and publish public link from u to $Jan - Mar$. The updated subscription hierarchy is shown in Figure 1.4 (b).

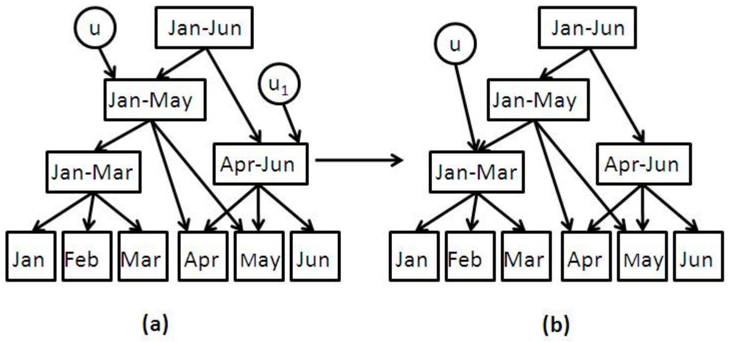


Figure 1.4: (a) User u subscribes for $Jan - May$ and user u_1 subscribes for $Apr - Jun$ (b) Subscription withdrawn by user u for Apr and May months.

In the example, a user after withdrawal of his subscription can still have unauthorized access to revoked data files. Consider the hierarchy given in Figure 1.4(a), a user u with key K_u is initially subscribed for $Jan - May$ using which he can compute encryption keys of leaf nodes with a subscription from Jan to May months. After the withdrawal of subscription for Apr and May, u can still decrypt data files for subscription nodes Apr and May using his old subscription keys and public information. This is because the subscription keys of nodes from Jan to May are not changed by subscription withdrawal procedure defined by them. An alternate link to the new subscription node is given to the user during subscription withdrawal procedure as shown in Figure 1.4 (b). The rest of the hierarchy is not updated or re-keyed. Therefore, a user in possession of old subscription keys can still have an unauthorized access to old subscription nodes' data files.

The re-keying operation [29, 30] assigns new keys to the (affected) set of data files and re-encrypts them with new keys. Although the *re-keying* operation will restrict a user from accessing their revoked resources, it requires re-encryption of all the descendant resources with fresh keys and re-distribution of these new keys among all existing authorized users. This operation is very costly when working with outsourced data. It requires the re-encryption operation over outsourced data, by the data owner.

Wang et al. [9] proposed a revocation scheme for outsourced data that avoids re-encryption. A detailed description of their scheme is given in Section 2.5.1. In their scheme re-encryption is not at all required for revocation, whereas in the case of lazy-revocation it is delayed. Each user is assigned an authorization certificate by the data owner. When an access authorization is changed in Wang et al. scheme, a significant amount of computation and communication overhead is needed at data owner to compute each existing certificate again and re-distribute these among the users. The certificate needs to be updated because it contains a common index number which needs to be updated on each change to the access control policy. Therefore, the system does not scale with a large number of users in which changes in users' access rights are frequent.

We have proposed modifications to Wang et al. scheme to avoid the bottleneck at the data owner [41]. Our proposal is detailed in Section 2.5.2. Table 1.1 shows the advantages over Wang et al. scheme.

1.4.2 Write access control

The goal of write access control is to prevent write access by unauthorized users. A file updated by a user is committed by the CSP without the involvement of the data owner. As the CSP is a third party, it can allow a malicious write oper-

Table 1.1: Comparison of Wang et al. scheme and proposed modified scheme

	Wang et al. scheme	Proposed scheme
Effect of a user revocation on other users access	each user requires new certificate for further access	Independent
Subscription extension/revocation cost at data owner	$O(\|U\|)$	$O(1)$

ation without the data owner noticing the unauthorized write. For example, the dishonest CSP can allow a revoked user to modify an existing file. We therefore assume that the CSP is "malicious-but-cautious" type.

In what follows, we discuss a scenario where a user in collusion with CSP can perform an unauthorized file modification, i.e., violating data integrity property. Existing work on write access control [42, 43, 15, 12] do not address such scenario.

Scenario: Suppose that an authorized user Alice has just performed a write operation on some file F. Further, suppose that no another user has performed a write operation on F after Alice's write. In this situation, the CSP may misbehave by omitting a previous version written by Alice. That is, it allows Alice to modify the contents of F without creating a new version. Modifying a file here means altering the content of the file. This is a matter of concern even if the user is an authorized writer. For example, in an e-Health system, if a medical prescription is published by a Doctor, the patient will immediately startup following the prescribed medication. The patient will then not allow any modification in the stored e-prescription without his knowledge. However, it may happen that the doctor later realizes that the published prescription has a wrong medicine which may lead to some legal action or embarrassment for him. In this event, the doctor can collude with the service provider to modify the prescription stored at the server in order to avoid any legal consequences. Now nobody including the patient can frame any charges against the doctor once the outsourced prescription is modified.

Similarly, in the case of user's access right revocation, i.e., the revoked user can collude with the CSP and modify their latest committed version (s) of a resource whose access right is now revoked. Such unauthorized write becomes possible when the record is not updated by the data owner after its access right is revoked. We assume that expired write authorization credentials of the resource are available with its writer.

In the outsourcing scenario, the write operation is executed by an authorized user directly at the untrusted CSP. Therefore, the data owner requires a posterior procedure to verify the write operation. Auditing is a well-known posterior misbehavior detection mechanism. It will detect the misbehavior at some later time and take an appropriate action to avoid it in future. In the outsourcing scenario, the auditing process is executed by the data owner at regular intervals of time.

To address the above discussed threat, we propose an audit-based protocol (in Chapter 4.3) so that a write authorized user who colludes with the CSP is not able to modify even their own written data files, after a fixed amount of time. If the cloud allows a user to modify the file, it will be detected during the audit process by the data owner and charges can be framed against the misbehaving party. We also propose a protocol so that a revoked user who colludes with the CSP is not able to modify their written data files. We have implemented the above protocols on Microsoft Azure platform. The implementation shows that the suggested mechanisms are viable in practice.

Data freshness If the CSP deliberately sends a stale version of some resource to a reader, it must be detectable by the data owner. Also, it is desirable to reduce the read staleness (or improving freshness guarantee) as much as possible.

We examine the staleness problem and improve on the freshness guarantee as compared to the existing works. The scheme in [15] provides version-based staleness, i.e., a read returns one of the kth latest version. The parameter k is fixed and depends on the time required by a data record to reach a reader. The works in [44, 12] use timestamp stored securely with each record version that assures the time when it was written. Bailis et al. [45] combine the above two (version and timestamp based staleness) to define the notion of $< k, t >$-staleness. It ensures that a read request which begins t time units after the write commit operation returns one of the last k updated versions. All the above discussed notions for staleness will not guarantee that the file is fresh as the last update. For example, in a bidding application, if a reader does not read the last updated file it may have an adverse effect on the reader's business operations. Our proposed mechanism described in Chapter 4 ensures that a file retrieved from the cloud server is fresh at least until the time when the file is dispatched by the CSP to the reader. In case the stale data is dispatched by the CSP during a read request, it will be detected by the data owner (a part of this work is published in [46]).

1.4.3 Privacy enabled access control

Pseudonymisation is a well-known unlinkability technique used in existing PHRMSs [8]. The basic idea is to identify field of a user or a file is replaced by a unique identifier called a pseudonym. The pseudonyms are stored in encrypted form. Only a user capable of decrypting the pseudonym (the process is called de-pseudonymisation) can find a link between the document and the patient. In PHR systems, a doctor needs to decrypt the patient's PHR; therefore, the processes of de-pseudonymisation and pseudonymisation must be separated. This separation is generally implemented using asymmetric keys and requires a trusted third party for de-pseudonymisation process. This will allow a separation but with a significant computation cost [47].

Pseudonymisation-based techniques can be used for achieving unlinkable relationships between the resources and their IDs in a system where all communicating entities are trusted. However, an adversary who can observe or control the network communication may find the linkability between them. In data outsourcing scenario, the untrusted cloud service provider who controls the traffic may observe the communication and break the unlinkability. Therefore, we require a stronger privacy property, namely unobservability which ensures that even after observing the network traffic one cannot find the linkage between the communicating parties. Formally, unobservability of an item of interest (the resource) means its undetectability against all subjects (may be an outsider such as CSP) uninvolved in it [7].

In our proposed cloud-based PHRMS scheme described in Chapter 5, we achieve unobservability property, which implies unlinkability. We use an existing concept called mix node [48] to implement unobservable communication (or unobservability) between a user and its PHR information. The mix node is used to create an anonymous channel between the communicating parties (a part of this work is published in [49]).

We conducted a formal analysis using Proverif [50] to verify that the presence of mix node among the communicating parties makes the communication unobservable. Two protocols are designed, viz., medical prescription (or progress notes) publishing protocol and laboratory report publishing protocol. The protocols are formally analyzed against an adversary who can observe the communication.

We show that our scheme enjoys the forward secrecy property so that a current consulting doctor is prevented from accessing any future document in a patient's PHR, using expired access authorization secrets. To the best of our knowledge, forward security property in PHRMS is not previously addressed in the literature.

1.5 Organization of the thesis

The rest of the thesis is organized as follows. Chapter 2 gives a detailed evaluation of user and resource-based hierarchies. Chapter 3 gives the construction for the proposed subscription-based hierarchical key assignment scheme. Chapter 4 discusses the issues related to unauthorized write access and freshness property, and gives our proposed solutions. In chapter 5, we propose a privacy enabled cloud-based personal health record management system. Finally, we conclude the thesis in Chapter 6.

Chapter 2

Key management for read access control

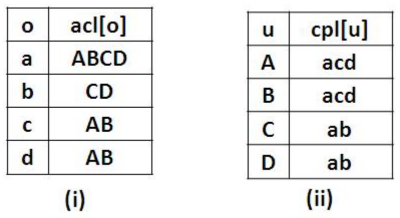
Key management is an essential component of cryptographic read access control. Managing a large number of secret keys is a challenge for the organization that outsourced its data. An important objective of key management is to reduce the secret key storage with each authorized user. To this end, this chapter discusses an important tool: key management hierarchy. We critically evaluate two existing types of key management hierarchy for data outsourcing in cloud.

In a cryptographic system, revocation can be achieved through the so-called re-keying operation. This operation assigns new keys to all the affected nodes in the hierarchy. The respective outsourced resources are re-encrypted with the newly assigned keys. Wang et al. proposed a scheme for handling access right revocation without re-keying. We identify an inefficient feature of their scheme that makes it unscalable and propose a modification that can handle a large number of users.

2.1 Access control matrix representation

An authorization policy defines who can access what resource. Access authorizations are generally defined using an Access Control Matrix (ACM). We assume each user has read authorization for some resource. An ACM can be represented in two ways, either as a collection of Access Control Lists (ACLs) or CaPability Lists (CPLs). An ACL corresponding to a resource is the set of users who are authorized to read the resource. On the other hand, a CPL is the set of resources for which a given user has read authorization. Both are dual of each other.

For example, consider a system with four users A, B, C, D and four resources



o	acl[o]
a	ABCD
b	CD
c	AB
d	AB

(i)

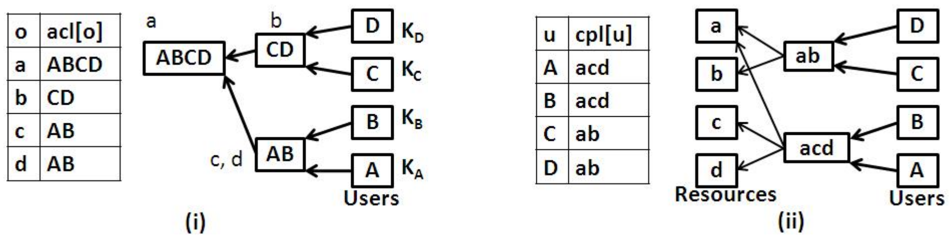
u	cpl[u]
A	acd
B	acd
C	ab
D	ab

(ii)

Figure 2.1: An example access control matrix as (i) ACLs (ii) CPLs.

a, b, c, d. An example of ACM is shown in Figure 2.1. In table (i), each row represents an ACL. $acl[o]$ represents an ACL corresponding to resource o, i.e., the set of users who are authorized to read o. The entry $acl[a] = \{A, B, C, D\}$ means that the resource a can be read by the users A, B, C and D. Similarly, in table (ii), each row represents a CPL. $cpl[u]$ represents a CPL corresponding to user u, i.e., the set of resources for which u has read authorization. The entry $cpl[A] = \{a, c, d\}$ means that the user A can access the resources a, c and d.

In general, a resource can be accessed by a group of users. A subset of these users may be authorized to access another resource. For example, resource a can be accessed by users A, B, C and D. The subsets $\{C, D\}$ and $\{A, B\}$ are authorized to access resources b and c, d, respectively. The relationships between user subsets can be represented using a hierarchy structure as shown in Figure 2.2 (i). In the hierarchy, each node is labeled by a subset of users, hence the name user-based hierarchy (or user hierarchy). For example, user B (in HKAS) can access the descendant nodes AB and $ABCD$, and hence can access the associated resources, i.e., c, d and a, respectively.



o	acl[o]
a	ABCD
b	CD
c	AB
d	AB

u	cpl[u]
A	acd
B	acd
C	ab
D	ab

Figure 2.2: Example hierarchy structures based on (i) ACLs (ii) CPLs.

Consider the hierarchy shown in Figure 2.2 (ii), where the nodes other than the individual user nodes represent resource groupings. This type of hierarchy is called a resource-based hierarchy. In the figure, user A can access all the resources

a, b, c, d, whereas user D can only access a and b.

To enforce the restriction that a user can read only its (descendant) authorized resources, the nodes are assigned keys using a hierarchical key assignment scheme (HKAS) given in [51]. A HKAS is a pair of algorithms (*Gen*, *Der*). The algorithm *Gen* generates and assigns keys to the nodes in the hierarchy. A resource is encrypted with its associated node's key. The *Der* algorithm derives a node's key using an ancestor node's key and public values.

In the following section, we review two types of key management hierarchy (KMH) previously proposed in the literature: user-based and resource-based. We critically compare the two hierarchy types with respect to their static structure. Section 2.3 gives the procedures for dynamic operations such as granting and revoking read access. It compares the two hierarchy types with respect to dynamic characteristics. In Section 2.4, operations for both hierarchy types are experimentally evaluated and compared. In Section 2.5, we analyze an access right revocation scheme that avoids re-keying (traditionally used to avoid a revoked access) and give a modified efficient scheme for the same. For the sake of readability, the notations used in this chapter are listed in Table 2.1.

Table 2.1: Notations used

Notation	Description
$a, b, c, ...$	Resources
$A, B, C, ...$	End users
K_i	Random key assigned to node i
$e(i, j)$	A directed edge from node i to node j
$r_{i,j}$	A public token associated with an edge $e(i, j)$
$E()$	Symmetric encryption function
$\mathcal{E}$ and $\mathcal{D}$	A symmetric encryption and decryption operation
$\mathcal{C}$	A communication between data owner and CSP
$acl[o]$	A set of read authorized users for the resource o
$[X]$	It represents a node corresponding to set X of users/resources

2.2 KMH: definitions and properties

In a key management hierarchy, each user is assigned a fixed number of keys using which she can derive the rest of the authorized keys. The design goals of a key management hierarchy are to minimize the cost of secret key storage per user, system public storage, and key derivation time.

2.2.1 User-based hierarchies

In this section, we review user-based key management hierarchies ([52, 13, 27, 12]) for enforcing data access control. Following Blundo et al. [52], a user graph is defined as follows, where each node represents a group of users and v_0 is the root node. In the definition, notation $v.acl$ represents a set of users that can access the node v's key.

Definition 1 (User graph)**.** *A user graph over a given set of users U, denoted G_U, is a graph (V_U, E_U) rooted at node v_0, where V_U is the power set of U and $E_U = \{e(v_i, v_j) | v_i.acl \subset v_j.acl\}$.*

It follows from Definition 1 that v_0 is a root node, there is a node corresponding to each subset of users and there is a directed path from each node v_i to node v_j with $v_i.acl \subset v_j.acl$. Also, there is an edge from the root node to each single user node. Figure 2.3 shows Hasse diagram of a user graph with four users $\{A, B, C, D\}$. For simplicity, the edges that are implied by other edges are not shown in the figure.

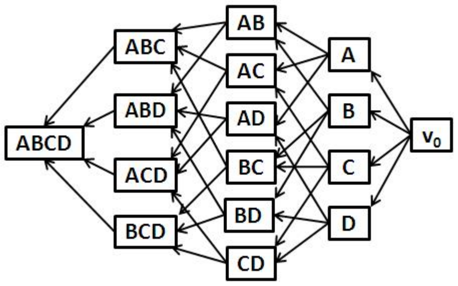


Figure 2.3: A user graph over a set $\{A, B, C, D\}$ of four users.

In a user graph, each user stores only one secret key corresponding to its respective node in the graph. For example, knowledge of key assigned to node

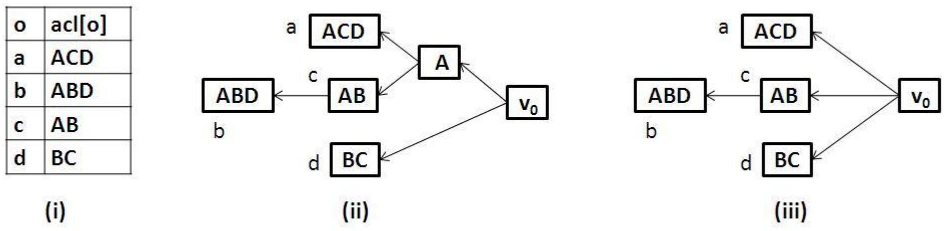
A is sufficient to derive the keys assigned to nodes AB, AC, AD, ABC, ABD and $ABCD$, respectively. Note that a user graph is a worst case graph over a set of users, i.e., it contains a node for every possible grouping of users in the given user set and an edge between every related pair of nodes. It contains one hop distance to reach any descendant node in the graph but with a significant increase in the number of edges (or the public storage). It requires $O(n^n)$ edges in the worst case even when excluding those implied by transitive property, where n is the number of nodes in the hierarchy.

A connected graph with at most single directed edge between two nodes is a tree. A user tree is a subgraph of a user graph with at most single directed edge between two nodes. It contains all nodes whose keys are used for encrypting resources; these nodes are called material nodes (denoted as $\mathcal{M}$). Formally, for a set of ACLs over a set of resources R, $\mathcal{M} = \{[acl[o]] : o \in R\}$. Following [52], a user tree can be defined as follows.

Definition 2 (User tree)**.** *Let G_U be a user graph over a set of users U, with root node v_0 and a set of material nodes M. A subgraph $T = (V, E)$ of G_U with $\mathcal{M} \bigcup \{v_0\} \subseteq V \subseteq V_U$ and $E = \{e(v_i, v_j) | v_i, v_j \in V, v_i.acl \subset v_j.acl\}$ that satisfies the property of being a tree rooted at v_0 is called a user tree.*

There can be more than one user tree exists for a given set of ACLs. An example with four users $U = \{A, B, C, D\}$ and four resources $R = \{a, b, c, d\}$ is shown in Figure 2.4. Figure 2.4 (i) represents example ACLs, and figure (ii) represents one possible user tree corresponding to the given ACLs. Each node in the user tree represents a user grouping, i.e., a set of users that can access the node's key and the associated resources. For example, node ACD represents a group of users A, C and D that can access the key K_{ACD} and hence the associated resource a. We can see in the figure that there is a node for each read authorization set $acl[o]$ for resource o. For example, there are nodes $acl[a] = ACD$, $acl[b] = ABD$, $acl[c] = AB$ and $acl[d] = BC$, in the figure.

Although there is a node for each $acl[o]$ in figure (ii), for each node there is no guarantee that it's respective ACL exists. For example, there is no ACL for node A. To reduce the public storage, such nodes may be deleted from the tree,



o	acl[o]
a	ACD
b	ABD
c	AB
d	BC

Figure 2.4: (i) Example *ACL*s with read authorization, (ii) A user tree, and (iii) Minimal vertex user tree

resulting in a minimal vertex user tree. In what follows, we first define the notion of a minimal graph to reduce the system public storage cost and then use it to define a minimal vertex user tree.

Definition 3 (Minimal user graph)**.** *Let G_U be a user graph over a set of users U, with root node v_0, and let M be a set of material nodes. A minimal user graph is a subgraph (V, E) of G_U for which $|V| + |E|$ is minimum over all $\mathcal{M} \bigcup \{v_0\} \subseteq V \subseteq V_U$ and $E = \{e(v_i, v_j) | v_i, v_j \in V, v_i.acl \subset v_j.acl\}$.*

A minimal vertex user tree can be defined as follows.

Definition 4 (Minimal vertex user tree)**.** *Let $\mathcal{A}$ be a set of ACLs over a set of users U and set of resources R. A minimal vertex user tree $T_m = (V_m, E_m)$ is a subgraph of $G_U = (V_U, E_U)$, rooted at node v_0 with $v_0.acl = \phi$, where $V_m = \mathcal{M} \bigcup \{v_o\}$ and $E_m = \{e(v_i, v_j) | v_i, v_j \in V_m, v_i.acl \subset v_j.acl\}$.*

A minimal vertex user tree contains exactly the material vertices $\mathcal{M}$ and the root node v_0. An example minimal vertex user tree is shown in Figure 2.4 (iii). The secret storage with each user in the tree is as follows:

Table 2.2: Secret keys with each user

User	Secret keys
A	K_{ACD}, K_{AB}
B	K_{AB}, K_{BC}
C	K_{ACD}, K_{BC}
D	K_{ACD}, K_{ABD}

It is seen from Table 2.2 that a user may require more than one secret key storage in a minimal vertex user tree. The maximum number of keys that a user may required to store will be equal to the number of leaf nodes in the tree.

Claim 1. *A minimal vertex user tree is a minimal user graph.*

Proof. A minimal vertex user tree contains exactly one node for each ACL. Since each node's key is used to encrypt at least one resource, the number of nodes cannot be reduced. If the number of nodes is n, then the minimum number of edges required to retain connectivity is exactly $n-1$. Therefore, a minimal vertex user tree is always a minimal graph.

□

In comparison to user graph, minimal vertex user tree help reduce the public storage, while increasing the secret storage at each user. In contrast to user trees, a user hierarchy needs to store a single secret key per user and consists of a node for each user. Moreover, a node can have more than one incoming edge. Following [13, 27, 12], a user hierarchy can be defined as follows.

Definition 5 (User hierarchy)**.** *Let $\mathcal{A}$ be a set of ACLs over a set U of users and set R of resources. A user hierarchy denoted $UH = (V, E)$ for given $\mathcal{A}$ is a subgraph of $G_U = (V_U, E_U)$ where $\mathcal{M} \bigcup U \subseteq V \subseteq V_U$ and $E = \{e(v_i, v_j) | v_i, v_j \in V, v_i.acl \subset v_j.acl\}$.*

In general, the public storage is defined as the total number of nodes and edges present in the hierarchy as there is a public value for each node and for each edge ([38]). Although the user hierarchy contains a minimum number of nodes, the total number of edges can be further reduced to some extent in the hierarchy by adding additional nodes. For example, suppose $v1.acl = BCDEX$ and $v2.acl = ABCDEF$, then a common subset of the two given ACLs is $BCDE$. Adding $BCDE$ node into the hierarchy may reduce the number of existing edges. If another node $v3.acl = ABCDFY$ exists, then it may happen that instead of node $BCDE$, node BCD (common to $v1$, $v2$ and $v3$) reduces more number of edges. Hence, there exists many such possibilities.

Definition 6 (Minimal user hierarchy problem)**.** *To find a user hierarchy $UH = (V, E)$ for which $|V| + |E|$ is minimum over all $\mathcal{M} \bigcup U \subseteq V \subseteq V_U$ and $E = \{e(v_i, v_j) | v_i, v_j \in V, v_i.acl \subset v_j.acl\}$ is called a minimal user hierarchy problem.*

Our objective is to find a user hierarchy which optimizes $|V| + |E|$. We call this problem as minimal hierarchy problem (MHP). In what follows, we show that the MHP is a hard problem.

The Steiner tree problem (STP [53]) on weighted graphs asks for a tree of minimum weight that contains all leaf nodes, but may also include additional nodes. Therefore, when edge weight is fixed to 1, the problem is same as minimizing the number of edges and the non-leaf nodes in the graph. It is known that the Steiner tree problem is NP-hard and remains so even in very restricted planar cases [54]. A variation of STP is directed STP whose goal is to find a minimum cost tree in a directed graph $G = (V, E)$ that connects all leaf nodes $X \in V$ to a given root $r \in V$ [55].

A generalization of directed STP is directed STP with multiple roots (or q-Root Steiner Tree, i.e., q-RST problem [56]). The q-RST problem is that given a directed graph $G = (V, E)$, two subsets of its nodes, set of root nodes R of size q and T, the goal is to find a minimum cost subgraph of G that contains a path from each node of R to each node of T. The rest of the nodes in set $V \setminus (R \cup T)$ can be added to form a minimum cost subgraph. This optimization problem is known to be NP-hard [56, 55].

Now, consider the q-RST problem with given directed graph $G_U = (V_U, E_U)$ containing unit weight edges, two subsets of its nodes, R of size q as user nodes and T the leaf nodes represent the ACLs. The goal is to find a minimum cost subgraph of G_U that contains a path from each node $v1$ of R to each node $v2$ in T, where $v1.acl \subseteq v2.acl$ and $v2 \neq \Phi$, i.e., there is at least one target node corresponding to the given root node. This problem is equivalent to the MHP. Therefore, if there exists an algorithm to solve MHP, the algorithm can be used to solve the q-RST problem. Below we show that MHP and q-RST problems are equivalent.

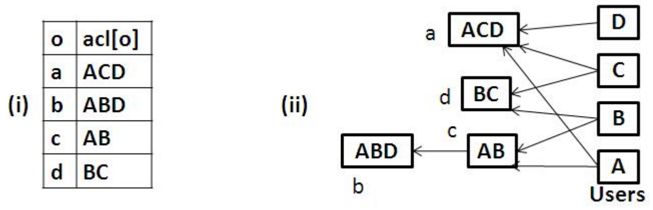
Theorem 2.2.1. *MHP and q-RST problems are equivalent.*

Proof. To show the equivalence between them, consider an arbitrary instance graph of MHP with unit directed edges, set R of size q containing user nodes as root nodes and T the leaf nodes represent the ACLs. Now, we will show how the MHP instance can be converted into general weighted graph as in q-RST problem. Consider each chain $C = < x_1, x_2, ..., x_i >$ of nodes in the graph such that each node except x_i in C has only one outgoing edge. Then replace C with one edge chain $C' = < x_1, x_i >$ and weight of the edge is $i - 1$, i.e., the sum of edge weights in C. The updated graph has now become an instance of q-RST problem which says that the q-RST problem is no harder than the MHP problem. This implies that the two problems are equivalent.

As an approximation to the MHP problem, we define a minimal vertex user hierarchy containing a minimum number of nodes and respective edges. A minimal vertex user hierarchy is defined as follows.

Definition 7 (Minimal vertex user hierarchy)**.** *A minimal vertex user hierarchy $UH_m = (V_m, E_m)$ for a given $UH = (V, E)$ is a subgraph of UH with $V_m = \mathcal{M} \bigcup U$.*

Consider the set of ACLs shown in Figure 2.5 (i). A minimal vertex user hierarchy implementing the given ACLs is shown in Figure 2.5 (ii).



(i)

o	acl[o]
a	ACD
b	ABD
c	AB
d	BC

Figure 2.5: (i) Example *ACL*s with read authorization, and (ii) A minimal vertex user hierarchy.

In a minimal vertex user hierarchy, each user requires only one secret key, as in the case of user graph. However, a user hierarchy will take a number of edges, i.e., the public storage, as compared to the corresponding user tree (see in Figure 2.4 (ii)). This is because there is a node for each system user in the user hierarchy.

Although the MHP problem is NP-hard, constructing a minimal vertex user hierarchy for a given ACM can be done in polynomial time. A procedure for constructing a minimal vertex user hierarchy for a given ACM is shown in Algorithm 1. In the algorithm, the notation [x] represents a node corresponding to set x of users. A node n is called a out-neighbor of node m if there is a directed edge from m to n.

Algorithm 1 *Create_Hier*(ACM, U)

Input: An ACM containing a set of ACLs and a set of users U.
Output: Create a user hierarchy corresponding to the given ACM.

```
1: for (each user $u \in U$) do
2:    Create a node $[u]$ for $u$
3: end for
4: for (each ACL $\in$ ACM) do
5:    Create a node $X$ for ACL
6:    for (each user $u \in ACL$) do
7:       Create and initialize set $S = \{u\}$
8:       Set “Update” = $True$
9:       while ( “Update” = $True$) do
10:         Set “Update” = $False$
11:         for (each out-neighbor $n$ of $[S]$) do
12:            if ($n.acl \subset$ ACL) then
13:               $S \leftarrow n.acl$                       /* Update set $S$ */
14:               Set “Update” = $True$
15:               Break                              /* Exit from inner for loop */
16:            end if
17:         end for
18:      end while
19:      for (each out-neighbor $n1$ of $[S]$) do
20:         if (ACL$\subset$ $n1.acl$) then
21:            Create an edge from node $X$ to $n1$ /* Update outgoing edges from $X$ */
22:            Delete edge from node $[S]$ to $n1$
23:         end if
24:      end for
25:      Create an edge from node $[S]$ to $X$       /* Update incoming edge on $X$ */
26:   end for
27: end for
```

The Algorithm 1 works as follows. A node is created for each user in set U (Steps 1-3). For each ACL in the given ACM, a corresponding node X is created (Step 5) and inserted into the hierarchy (Steps 6 to 26). For each user u in the given ACL, a node S after which X can be inserted (satisfying the access control relationships) is searched (Steps 7 to 18). Then, outgoing edges from node X corresponding to S and u are updated (Steps 19 to 24). Incoming edge to X is then updated (Step 25). At the end of this algorithm, a user hierarchy is created corresponding to the given ACLs in the ACM. For a given set of resources R, the Algorithm 1 will take a running time cost of $O(|R|^2)$ in the worst case, considering

$|U| << |R|$. It is due to the statement numbers 4 and 11 in the algorithm each of which iterates $O(|R|)$ times. Statement number 6 and 9 will iterate $O(|U|)$ times each.

2.2.2 Resource-based hierarchies

In this section, we discuss a key derivation structure called *resource hierarchy* (introduced in [27]), where nodes are defined based on the resource groupings (i.e., CPLs), instead of the user groupings (i.e., ACLs).

We first define a resource graph in a similar fashion to a user graph. In the definition, $v.cpl$ for a node v is a set of resources that will be accessed using node v's key.

Definition 8 (Resource graph). *A resource graph over a given set of resources R, denoted G_R, is a graph (V_R, E_R), where V_R is the power set of R and $E_R = \{e(v_i, v_j) | v_j.cpl \subset v_i.cpl\}$.*

Definition 8 ensures that a resource graph over a set of resources R contains every element from the power set of R. An example resource graph with four resources $\{a, b, c, d\}$ is shown in Figure 2.6. In the graph, there is a directed path from each node v_i to node v_j such that $v_j.cpl \subset v_i.cpl$. For example, the node *abc* with capability list $\{a, b, c\}$ has a path to each node with subset capability list such as *ab*, *ac*, *bc*, *a*, *b* and *c*.

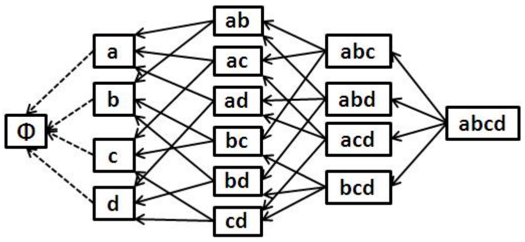


Figure 2.6: A resource graph over a set $\{a, b, c, d\}$ of four resources.

In contrast to a user graph, nodes in a resource graph are created by grouping resources from set R. It contains $2^{|R|}$ number of nodes. Since $|R| >> |U|$, resource graphs are less practical than user graphs.

A resource hierarchy is a sub-graph of the resource graph and can be viewed as a dual of user hierarchy. We redefine $\mathcal{M}$ that only contains a node used to encrypt data file. As user hierarchy, a resource hierarchy is defined as follows.

Definition 9 (Resource hierarchy)**.** *Let $\mathcal{A}$ be a set of CPLs over a set of users U and set of resources R. A resource hierarchy denoted $RH = (V, E)$ for given $\mathcal{A}$ is a subgraph of $G_R = (V_R, E_R)$ where $\mathcal{M} \bigcup U \subseteq V \subseteq V_R$ and $E = \{e(v_i, v_j) | v_i, v_j \in V, v_i.acl \subset v_j.acl\}$.*

Now with the same spirit as of minimal vertex user hierarchy, a minimal vertex resource hierarchy is now defined as follows.

Definition 10 (Minimal vertex resource hierarchy)**.** *Let $\mathcal{A}$ be a set of CPLs over a set U of users and set R of resources. A resource hierarchy denoted $RH = (V, E)$ for given $\mathcal{A}$ is a subgraph of $G_R = (V_R, E_R)$ with $V = U \bigcup R$ and $E = \{e(v_i, v_j) | v_i \in U, v_j \in R$ with $v_j.cpl \subseteq v_i.cpl\}$*

The above definition ensures that a minimal vertex resource hierarchy includes root nodes representing the users and leaf nodes representing the resources. Since each resource is encrypted with its dedicated leaf node's key, no intermediate node is needed between user and resource nodes. There is a direct edge from every user node u to a node corresponding to a resource r if $r \in cpl[u]$. An example minimal vertex resource hierarchy is shown in Figure 2.7, where (i) represents an example set of CPLs and (ii) gives a corresponding minimal vertex resource hierarchy. In the example hierarchy, there is a direct edge from node A to the set of nodes $\{[a], [b], [c]\}$ since $cpl[A] = \{a, b, c\}$ as shown in figure (i). Similarly, there are edges from node B to the set of nodes $\{[b], [c], [d]\}$, node C to the set of nodes $\{[a], [d]\}$ and node D to the set of nodes $\{[a], [b]\}$.

(i)

u	cpl[u]
A	abc
B	bcd
C	ad
D	ab

(ii)

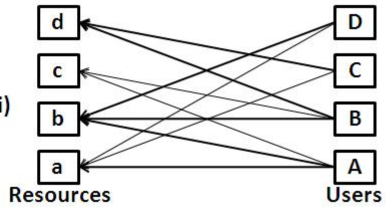


Figure 2.7: (i) Example CPLs, and (ii) A minimal vertex resource hierarchy.

Unlike the minimal vertex user hierarchies, each leaf node in a minimal vertex resource hierarchy represents a resource node, i.e., a resource is encrypted with a leaf node's key. There is a direct (key derivation path) edge from each user node u to all of her authorized resource nodes, i.e., resources in her capability list ($cpl[u]$).

2.2.3 Comparison of static hierarches

A hierarchy with a fixed structure is called a static hierarchy. In this section, we compare minimal vertex user and resource hierarchies in a static situation. An ACM is said to be in the worst case if all of its ACLs or CPLs are distinct. We will compare the number of nodes and edges that are required to construct a minimal vertex hierarchy for a worst case ACM. In Section 2.3, we give algorithms for dynamic operations that guarantee the minimal vertex hierarchy construction.

Let $|U|$ and $|R|$ denote the number of users and resources, respectively. We assume that $|U| << |R|$ but $|R| < 2^{|U|}$. For example, consider that we need to create an electronic health record management system for India, and assume that 1 crore patients receive care every year. Suppose a central database is created to store the patient records. For 100 years and assuming 20 documents per patient per year, it requires $\sim 10^{10}$ data files to be stored. However, for a set of only 50 users, $2^{|U|} = 2^{50} \sim 10^{15}$ which is a significant number, as compared to the total number of resources in an organization.

Cost of user hierarchies In a user hierarchy, consider a set of ACLs in worst case, i.e., each resource o has a distinct $acl[o]$. As there is a node for each $acl[o]$, the maximum number of nodes is $|R|$. In case $|U|$ is small and $2^{|U|} < |R|$ then maximum number of nodes will be $2^{|U|}$. Therefore, the total number of nodes in the hierarchy will be $min(2^{|U|}, |R|)$. In total, $min(2^{|U|}, |R|)$ or $O(|R|)$ nodes are needed assuming $|R| < 2^{|U|}$.

For finding the number of edges required for a given number of nodes, consider user nodes as level 0 nodes, directly connected nodes of the level 0 nodes as level 1 nodes, and so on. In the worst case, the level 0 contains ${}^{|U|}C_1$ nodes, level 1 contains ${}^{|U|}C_2$ nodes, and so on (similar to user graph). Also, the number of incoming edges at each node in level 1 is 1 and in level 2 is 2 and so on. Therefore, the total number of incoming edges at level 1 is $1 \times^{|U|} C_1$, at level 2 is $2 \times^{|U|} C_2$ and so on. Now the total number of edges can be written as follows.

$$1 \times^{|U|} C_1 + 2 \times^{|U|} C_2 + ... + (|U| - 1) \times^{|U|} C_{|U|-1} + (|U|) \times^{|U|} C_{|U|} \tag{2.1}$$

$$= \frac{|U|}{0!} + \frac{|U|(|U|-1)}{1!} + ... + \frac{|U|(|U|-1)}{1!} + \frac{|U|}{0!} \tag{2.2}$$

$$= 2(\frac{|U|}{0!} + \frac{|U|(|U|-1)}{1!} + ... + \frac{|U|(|U|-1)...(|U|-(|U|/2))}{(|U|/2)!}) \tag{2.3}$$

In total, it comes out as $2(\sum_{i=0}^{|U|/2} \frac{|U|!}{(|U|-i-1)!i!})$, i.e., $O(|U|^{|U|/2})$ due to the last term in equation 3. Also, the number of levels gives the key derivation steps (or time), i.e., $O(|U|)$ (in worst case).

When considering number of edges in worst case minimal vertex user hierarchy, all the ACLs are distinct of $O|R|$ number of users each and there is no node whose corresponding ACL is subset of other (i.e., all nodes are at same level). It creates two level hierarchy: user nodes in one level and other nodes in second level. Now, the total number of edges will be $O(|U||R|)$.

Cost of minimal vertex resource hierarchy In the worst case minimal vertex resource hierarchy, each user has a direct edge to each of its authorization resource node. In total, $|U| + |R|$ nodes and $|U||R|$ edges are needed in the worst case. Also, the key derivation cost will be $O(1)$ due to a direct edge from a user to an authorized resource node.

Table 2.3 compares the minimal vertex resource hierarchy with existing user-based hierarchies (user graph, user tree and minimal vertex user hierarchy) in the worst case. We can see from the table that, the maximum number of nodes and edges in both minimal vertex user and resource hierarchies are $|U| + |R|$ and $O(|U||R|)$, respectively. The key derivation cost in minimal vertex resource hierarchy is only one edge whereas in minimal vertex user hierarchy is $|U| - 1$ edges in the worst case. This is more in minimal vertex user hierarchy because it may form a longest chain of $O(|U|)$ nodes.

Table 2.3: Comparison of minimal vertex resource hierarchy with user-based hierarchies

Hierarchy $\rightarrow$	User-based hierarchies			Minimal vertex
	User	User	Minimal vertex	resource
$\downarrow$ Attributes	graph	tree	user hierarchy	hierarchy
# of keys/users	Single	Multiple	Single	Single
# of nodes	$2^{\|U\|}+1$	$min(\|R\|, 2^{\|U\|})+1$	$\|U\|+\|R\|$	$\|U\|+\|R\|$
# of edges	$O(\|U\|^{\|U\|/2})$	$O(min(\|R\|, 2^{\|U\|}))$	$O(\|U\|\|R\|)$	$O(\|U\|\|R\|)$
key derivation cost	$\|U\|$	$\|U\|$	$\|U\|-1$	1

2.3 Dynamic access control

Data access authorizations change with time as employees join and leave the organization or the department within the organization. A scheme with dynamic access control would allow granting or revoking access authorizations. In the following, we evaluate the user and resource-based hierarchies in terms of computational and communication costs of the common dynamic operations.

2.3.1 Algorithms for user hierarchy

Grant/revoke read privilege In user hierarchy, if access authorization is granted (or revoked) for a resource o to a user u then $acl[o]$ will be updated to $acl[o]' = acl[o] \cup \{u\}$ (or $acl[o]' = acl[o] \setminus \{u\}$). Now, since $acl[o] \neq acl[o]'$ (both represent different nodes in the hierarchy), resource o will be now encrypted with the key $K_{[acl[o]']}$ corresponding to $acl[o]'$. To avoid storing multiple copies of the resource encrypted with different keys ($K_{[acl[o]']}$ and $K_{[acl[o]]}$) for security reasons, data owner must delete the old copy from the server. Since granting read access is a frequent operation, associated re-encryption operation to the outsourced resource by the data owner should be avoided, if possible.

Algorithm 2 *Grant_Revoke_Read_Access*(UH, o, u)

```
Input: A minimal vertex user hierarchy $UH$, a resource $o$ and a user $u$.
Output: Read access for $o$ is granted or revoked to $u$.
1: Find node $v$ with $v.acl = acl[o]$ in $UH$
2: In case of
   "Grant operation": $acl[o] \leftarrow acl[o] \bigcup \{u\}$    /* update the $ACL$ of resource $o$ */
   "Revoke operation": $acl[o] \leftarrow acl[o] \setminus \{u\}$
3: Find node $v_{new}$ with $v_{new}.acl = acl[o]$
4: if ($v_{new}$ does not exist in the UH) then
5:     Create node $v_{new}$ with $v_{new}.acl = acl[o]$
6:     Insert $v_{new}$ into UH
7: end if
8: Download the encrypted version $o'$ of $o$ from the server
9: $o \leftarrow D_{K_v}(o')$
10: $o'' \leftarrow E_{K_{v_{new}}}(o)$
11: Outsource $o''$ to the server
   /* Delete $v$ if $v.acl$ is not in the ACL list*/
12: if (does not exist $p \in R$ with $acl[p] = v.acl$) then
13:     Delete $v$ and associated edges from $UH$        /* deleting redundant node */
14: end if
15: Publish updated $UH$ to the cloud server
```

Consider Algorithm 2 for granting read access. Running time of the algorithm with respect to the hierarchy manipulation, i.e., excluding encryption, decryption or communication cost is $O(U + R)$. It is due to the statement number 6 in the algorithm that requires cost $O(U)$ in updating incoming edges to new node v_{new} and $O(R)$ in updating outgoing edges. In the following, $\mathcal{E}$ represents the cost of one symmetric encryption operation, $\mathcal{D}$ the cost of one symmetric decryption operation and $\mathcal{C}$ the cost of one communication between the data owner and the CSP.

In Algorithm 2, granting read access for a resource to a user requires the following steps: (1) downloading the resource from the server ($1\mathcal{C}$), (2) decrypting it using the old key ($1\mathcal{D}$), (3) encrypting it with the new key ($1\mathcal{E}$), and (4) storing it back to the server ($1\mathcal{C}$) (i.e., $total\ cost = 1\mathcal{E} + 1\mathcal{D} + 2\mathcal{C}$). For example, consider the user hierarchy shown in Figure 2.8 (i), granting read access for resource c to users C leads to the modified hierarchy shown in Figure 2.8 (ii). In the modified hierarchy, a new node ABC is inserted and the resource c is encrypted with K_{ABC}.

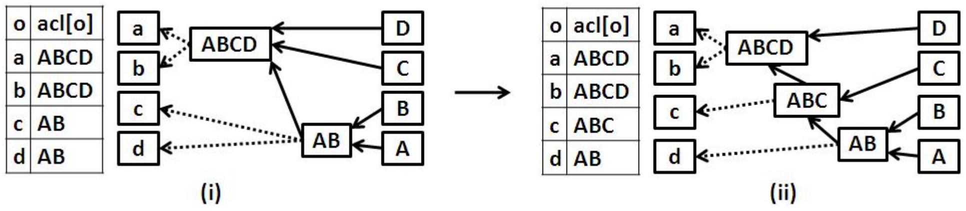


Figure 2.8: Modified example of minimal vertex user hierarchy (i) before, and (ii) after granting read access

User revocation Since each node in a user hierarchy represents a user grouping, a user revoke operation requires a modification to the hierarchy. Revoking a user requires that each node previously accessible to the revoked user be deleted and replaced by a new node (without revoked user label). For example, consider the minimum vertex user hierarchy given in Figure 2.8 (i). To revoke D we delete the node $ABCD$ and replace it with the new node ABC (by deleting label D). Now, resources a and b are re-encrypted with the new key (K_{ABC}) so that user D will not be able to access the revoked resources. The updated hierarchy is shown in Figure 2.9.

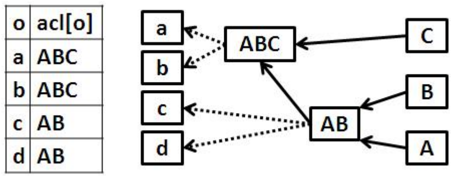


Figure 2.9: Modified example of minimal vertex user hierarchy after revoking user D

2.3.2 Algorithms for resource hierarchy

Grant read access To grant read access for a resource o to a user u, the data owner executes Algorithm 3.

Algorithm 3 $Grant\_readAccess(RH, o, u)$

```
Input: A minimal vertex resource hierarchy $RH$, a resource $o$, and a user $u$.
Output: Grant read access of resource $o$ to user $u$.
1: $u.cpl = u.cpl \bigcup \{o\}$                    /* updating user $u$'s CPL */
2: Create an edge from $[u]$ to $[o]$ by computing a public edge token $r_{[u],[o]}$
3: Publish $r_{[u],[o]}$ (and $E(o, K_{[o]})$, if new resource) at the cloud server to update $RH$
```

In the algorithm, $[x]$ represents a node corresponding to set x of users or resources. $K_{[o]}$ is the key used to encrypt resource o. Consider the example hierarchy in Figure 2.10 (i). Initially, user C has read access to the resources a and b. Suppose, read access for resource c is to be granted to the user C. Using Algorithm 3, user C's capability list $C.cpl = \{a, b\}$ is updated by inserting resource c, i.e., $C.cpl = \{a, b, c\}$ (Step 1). An edge is created from node $[u]$ to $[c]$ (Step 2). All updated public information (i.e., $r_{[u],[o]}$ and $E(o, K_{[o]})$ (if o is new resource)) will be now published at the server (Step 3). The modified CPL and the hierarchy are shown in Figure 2.10 (ii).

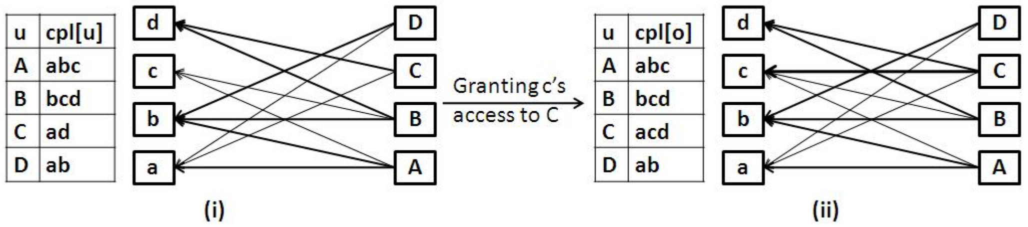
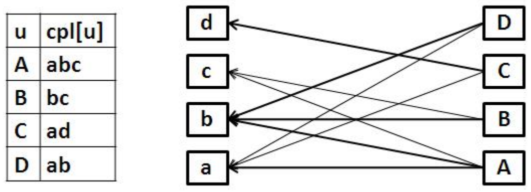


Figure 2.10: (i) An example minimal vertex resource hierarchy, and (ii) Granting read access for resource c to user C.

Revoke read access To revoke read authorization for a resource o to a user u (assuming both exist), the data owner executes Algorithm 4. For example, consider the hierarchy in Figure 2.10 (ii), where user B has initially read access for the resources b, c and d. Suppose, read access of resource d is revoked from user B, the algorithm works as follows. Old capability list of user B, i.e., bcd is updated to bc (Step 1). A new key $K'_{[d]}$ is assigned to node d (Step 2). Encrypted resource d is downloaded from the server, decrypted using old key $K_{[d]}$ and then encrypted with new key $K'_{[d]}$ (Steps $3-5$). Edge $r_{B,[d]}$ is deleted (Step 6). Now, for each user node v with $o \subset v.cpl$, compute public token for edge $e(v, o)$ and update it with the stored one (Steps $7-9$). The updated resource hierarchy information is then sent to the server along with encrypted resource $K'_{[d]}$ (Step 10). The updated *CPL* and resource hierarchy are shown in Figure 2.11.

Algorithm 4 $Revoke\_readAccess(RH, o, u)$

```
Input: A resource hierarchy $RH$, resource $o$, and user $u$.
Output: Revoke read access of resource $o$ from $u$.
 1: $u.cpl \leftarrow u.cpl \setminus \{o\}$                    /* updating user $u$'s CPL */
 2: Update node $[o]$'s key to $K'_{[o]}$
 3: Download the encrypted version $o'$ of $o$ from the server
 4: $o \leftarrow D_{K_{[o]}}(o)$
 5: $o'' \leftarrow E_{K'_{[o]}}(o)$
 6: Delete edge from $[u]$ to $[o]$ by deleting public edge token $r_{[u],[o]}$
 7: for (each user $v$ with $o \subset v.cpl$) do
 8:     Compute $r_{[v],[o]}$ and use it to replace the old public edge token in RH
 9: end for
10: Publish $o''$ and updated RH information at the cloud server
```



u	cpl[u]
A	abc
B	bc
C	ad
D	ab

Figure 2.11: After revoking read access of resource *d* from user *B*.

User revocation To revoke a user u, the data owner executes the following. For each outgoing edge $e(u, o)$ from u to some resource o, the data owner calls the procedure $Revoke\_readAccess(RH, u, o)$ (Algorithm 4).

2.3.3 Comparison of dynamic hierarchies

Table 2.4 compares the minimal vertex UH and RH. It compares the two with respect to the number of encryption ($\mathcal{E}$) or decryption ($\mathcal{D}$) operations needed by the data owner, communications ($\mathcal{C}$) needed with the CSP to grant one read access, revoke one read access, and whether revoking a user requires modification to the hierarchy structure. An attractive property of the minimal vertex RH is that it does not require any encryption or decryption operation while granting read access to

Table 2.4: Comparison of computation and communication cost

Hierarchy type ↓	Cost of a grant read access	Cost of a revoke read access	Modification of hierarchy due to user revocation
Minimal vertex UH	$1\mathcal{E}+1\mathcal{D}+2\mathcal{C}$	$\|R\|(1\mathcal{E}+1\mathcal{D})+2\mathcal{C}$	Yes
Minimal vertex RH	$1\mathcal{C}$	$\|R\|(1\mathcal{E}+1\mathcal{D})+2\mathcal{C}$	No

a user. It requires single communication between the data owner and CSP to update the outsourced hierarchy structure while granting read access to a user. Also, it does not require any modification to the hierarchy structure when a user is revoked, unlike the user-based hierarchies. Revoking a user′s read access right takes similar cost in both the hierarchy types.

2.4 Experimental evaluation

We have implemented the minimal vertex UH and RH for read access control on a local area network. The goal of the experiment is to evaluate the cost of dynamic operations from the perspective of the user and the data owner. We evaluate the time of user′s grant and revoke access right operations, and elapsed time performance of the data owner machine. The elapsed time is the time difference between a start and finishing time for a set of operations.

Setup: For testing purposes, we use two machines: a file server and a data owner. Each machine consists of an Intel core 2 quad $Q8400$ processor 2.66 GHz with 3 GB RAM and 7200 RPM, 16 MB Cache, SATA 3.0 Gb/s hard drive. Both systems running windows XP are connected with a 1 Gbps Ethernet link. We choose $AES-128$ as the cipher for file encryption and employ $SHA-1$ as the hash function (available in java.security package). We implement grant and revoke read methods in Java with JDK 1.7. The test includes a file server that stores 1000 files. The file size varies from 1 MB to 2 MB. The hierarchy is implemented using Hashmap in Java by storing it as an adjacency list. For the test, we fix the number of users to 30 and number of resources to 50. Considering fewer resources will not affect our experimental results since the cost of a grant or revoke operation depend only on the corresponding resource whose access right is updated. After fixing these, we create different initial hierarchies. We define the size of initial hierarchy in terms of the number M of consecutive grant access right operations. Each grant operation randomly selects a user and a resource from the set of 1000 files.

Minimal vertex RH: Grant and revoke read operations cost

We first evaluate the cost of one grant and revoke operations at the data owner. An initial minimal vertex RH is created for a fixed value of M. This defines an initial ACM. Then the grant and revoke permissions are initiated in sequence at the data owner machine for which it updates the respective CPLs and the hierarchy structure. We define a thread containing one grant and one revoke operation that will execute simultaneously to maintain the same size of the initial hierarchy. The thread is executed 100 times. The average cost of each operation in the thread is then computed separately immediately after the corresponding hierarchy is published.

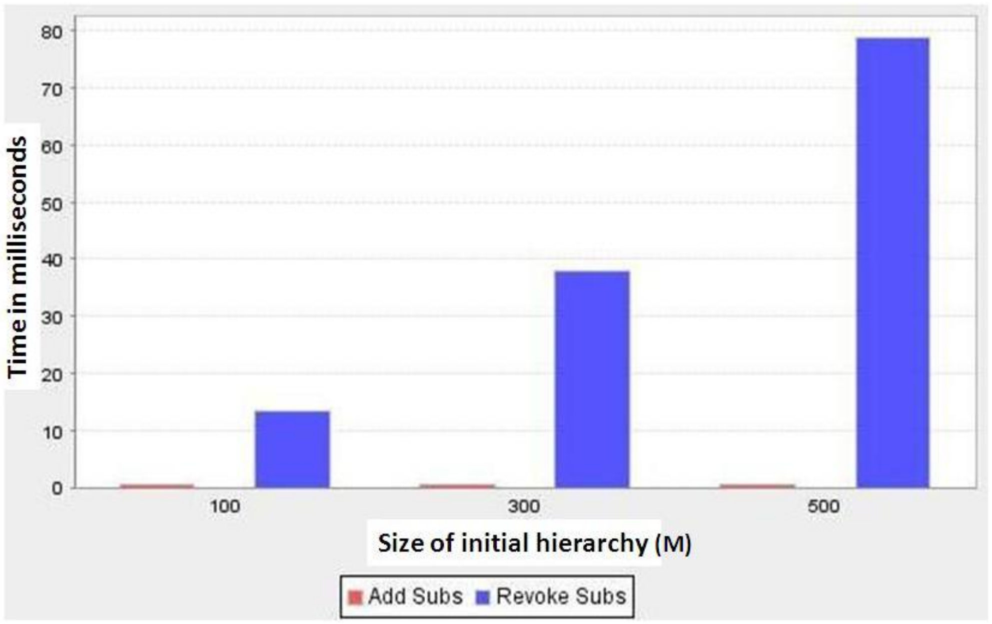


Figure 2.12: Permission operation cost

Figure 2.12 shows the cost of one grant and one revoke operation for different sizes of initial hierarchy, i.e., $M = 100, 300, 500$ and taking an average over 100 operations. In the figure, the cost of one grant permission operation is $0.477, 0.454, 0.440$ milliseconds for $M = 100, 300, 500$, respectively. It shows that the cost of revoke operation are 13.2, 38.0, 78.8 milliseconds with on average 3, 7, 11 file re-encryptions per revoke operation, respectively. From the figure, we conclude that the cost of one grant operation is approximately same with different size of initial hierarchy. This is due to the fact that each grant operation adds to at most one node into the hierarchy and updating of corresponding edges. However, the cost of one revoke operation increases almost linearly with the size of initial hierarchy. As the size of hierarchy increases by randomly applying grant permission operations with the same number of users and resources, the user's subscription (subscribed resources) will increase. This will lead to an increase in the number of re-encryption operations at the time of revoke operation and hence the revocation cost.

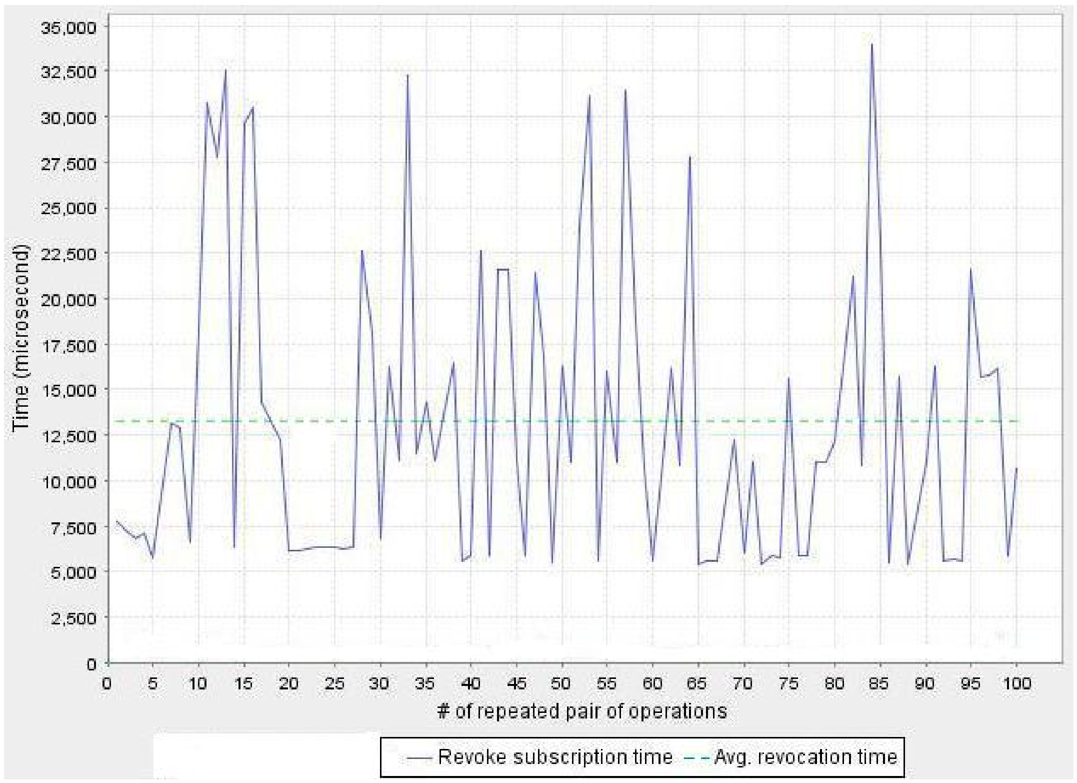


Figure 2.13: Average elapsed time of one grant/revoke operation

Figure 2.13 shows the computation for average cost of revoke operation when considering $M = 100$. We take an average over 100 operations. It requires 296 total file re-encryptions and on average 3 re-encryptions per revoke operation.

The average cost of revoke operation is 132.47 milliseconds.

Performance of data owner machine In the above evaluation, we considered only one user. Now, we consider a number of users involve in grant or revoke operations. For each operation, the data owner will update the ACM and corresponding hierarchy. To evaluate the data owner's elapsed time performance for handling a number of user threads, we simulate T simultaneous threads at the data owner. Due to random inputs for each operation, we perform the test 100 times and then the average cost of one batch of T threads is computed. We perform the tests for T = 10, 50, 100, 200, 300, 500, 700, 1000, 1500, 2000. M is fixed to 100. Figure 2.14 shows the results. From the figure, we conclude that there is almost linear relation between the elapsed time and the number of threads T.

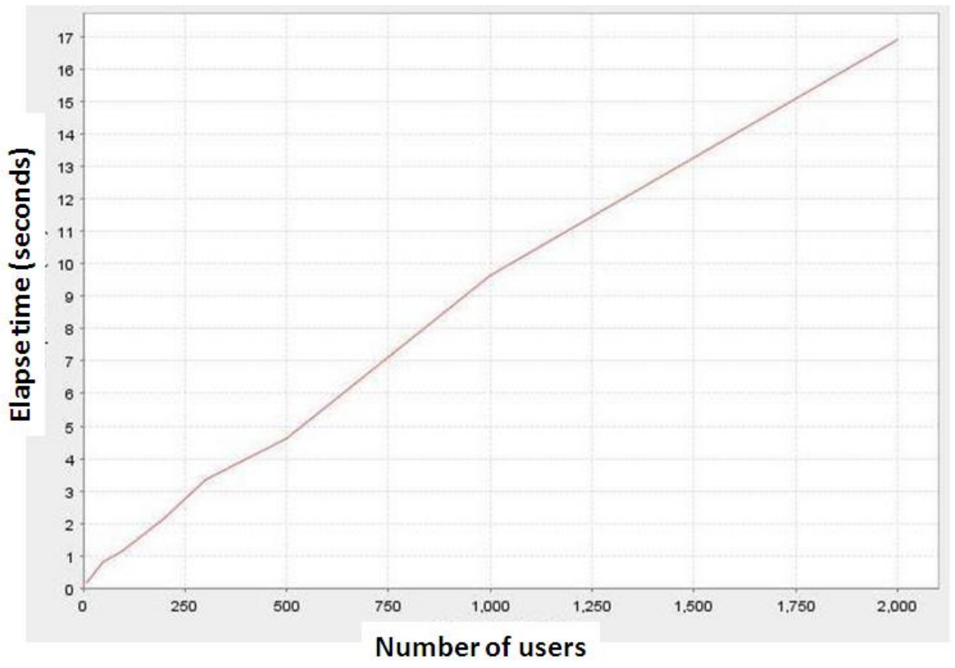


Figure 2.14: Elapsed time performance of data owner machine for evaluating user threads

Minimal vertex UH: Grant and revoke read operations cost

Similar to the minimal vertex RH, the minimal vertex UH is created by fixing M and the corresponding ACM is stored. The grant and revoke operations are initiated in the same way as in the minimal vertex RH. The evaluation cost is

Table 2.5: Grant and revoke cost in user hierarchy

Size of initial hierarchy →		200	500	1000
Operation	Average file size			
Grant	100 KB	12.961ms	13.923ms	18.530ms
	1 MB	146.174ms	160.385ms	186.561ms
Revoke	100 KB	13.511ms	14.223ms	17.935ms
	1 MB	133.274ms	151.843ms	172.015ms

shown in Table 2.5. For a given file size, our results show that the grant and revoke access right operations have a similar cost. This is because each operation requires one re-encryption of an outsourced resource and an addition of at most one node in the hierarchy.

Minimal vertex UH and RH: Comparing grant read operation cost

Considering the experimental setup described above, we evaluate the cost of one grant read permission for a user. Figure 2.15 compares the two hierarchies against

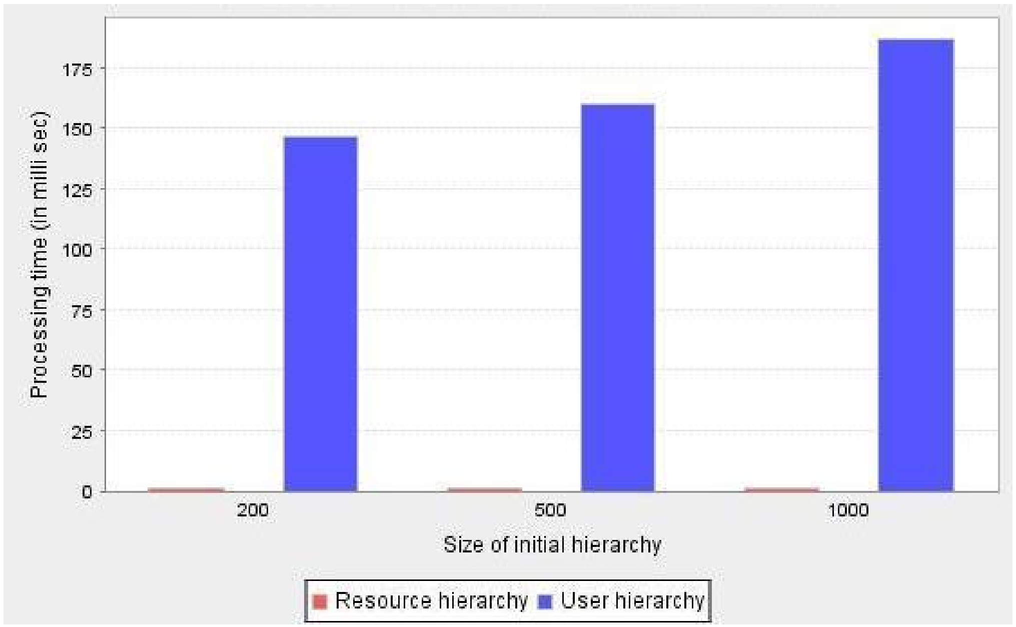


Figure 2.15: Elapsed time of one grant read subscription operation

grant operation cost. We fixed the initial hierarchy parameter $M = 200, 500$ and 1000. The average file size is 1 MB. This grant operation is executed 100 times. The average cost of one operation is then computed. The figure shows that in minimal vertex UH the cost of one grant operation is significantly large in comparison to minimal vertex RH. It is due to file encryption and decryption operations needed in the minimal vertex UH when user subscription is granted. These operations are not required in the minimal vertex RH.

Minimal vertex UH and RH: Comparing user revoke operation cost

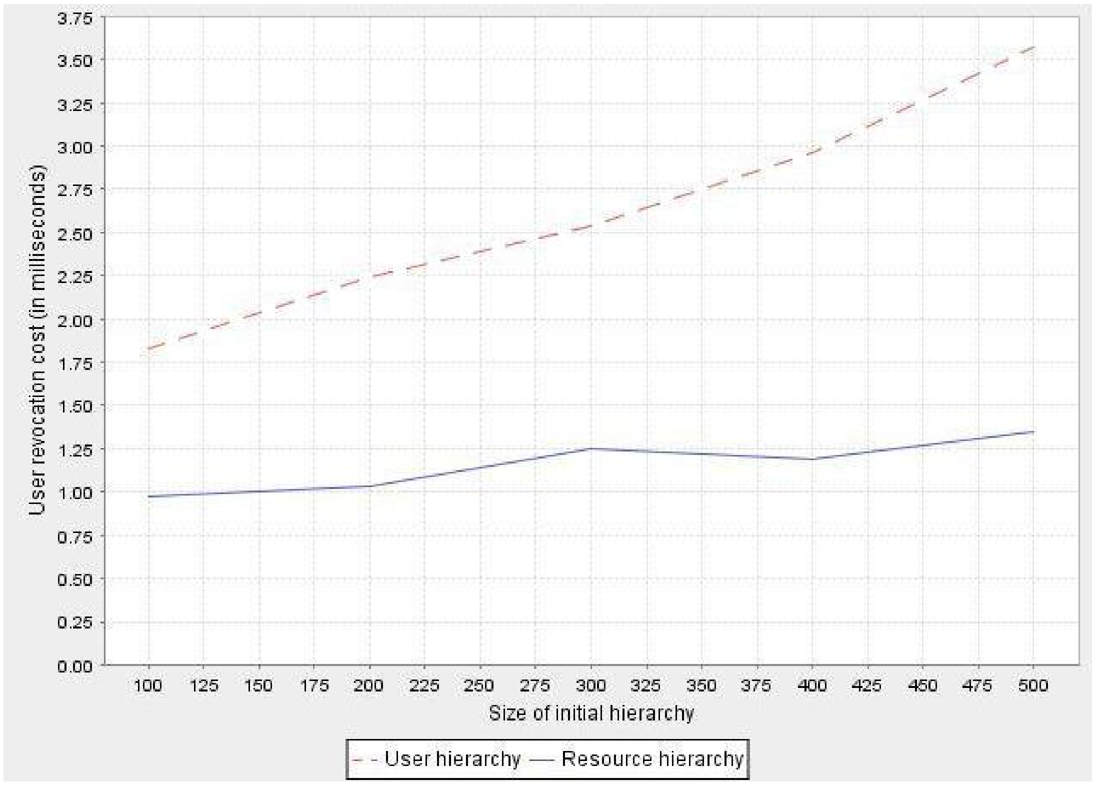


Figure 2.16: Average elapsed time of one user revoke operation

Figure 2.16 compares minimal vertex user and resource hierarchies with respect to a user revoke operation cost. In the experiment we only consider the hierarchy modification cost due to user revoke operation, i.e., the cost of resource encryption and decryption is omitted for simplicity. It is to be noted here that the average cost of encryption and decryption operations required per user revocation is same in both the hierarchies. The graph shows that the hierarchy modification cost significantly increases in minimal vertex UH with the increase in initial hierarchy size. This is due to the increase in a number of nodes to be modified with the increase in the size of user's ACL. In the minimal vertex RH, the increase is constant as only number of edges from user node to the authorized resource node need to be deleted from the hierarchy.

2.5 Re-keying

Although the *re-keying* operation restricts a user from accessing their revoked resources, it requires re-encryption of a set of descendant resources with fresh keys and re-distribution of these new keys among all authorized users. This operation becomes more expensive when working with data outsourcing scenario where the re-encryption operation is performed over outsourced data, by the data owner. Therefore we require some efficient solution for handling access right revocation that avoids this significant re-encryption cost.

2.5.1 Wang et al. protocol

A cryptographic scheme is proposed by Wang et al. [9] to handle access right revocation problem in data outsourcing scenario. The important feature of their scheme is that it does not require the inefficient re-keying operation. The scheme uses over-encryption [1] to avoid unauthorized access through eavesdropping by revoked users (with knowledge of old keys). In the scheme, the secret key used for over-encryption is generated using pseudorandom bit sequence generator $P()$ that takes a secret seed value as input. Each authorized user obtains from the data owner a certificate containing the seed value along with other necessary information. A user U's certificate is of the following form:

$$\{E_{k_{dc}}(U, ACMindex, seed, \{dbi_{read}\}, MAC)\},$$

[1]Vimercati et al. [10] consider a symmetric key based model that uses "over-encryption" where the data is encrypted twice: once by the data owner for confidentiality protection, and then by the CSP with a user-oriented key that makes it accessible only to a specific user. Without re-keying, the over-encryption restricts the revoked users with old authorization keys from accessing any sniffed data while communicated from CSP to the current authorized users.

where $\{dbi_{read}\}$ is a set of resource indexes for which U is authorized for read access. $ACMindex$ is the latest updated index number of the Access Control Matrix (ACM) with the data owner. It is incremented by one each time a user's access right is changed, by the data owner. The $ACMindex$ value is used for latest certificate validation, by CSP. MAC preserves the integrity of the certificate and is computed over rest of the values in encryption. $E()$ is symmetric encryption function. Data owner shares a pair-wise secret key k_{dc} and k_{du} with CSP and user (U) respectively, for secure message communication between them. The certificate credentials are encrypted by data owner at the time of certificate creation using symmetric key k_{dc}.

The data access procedure in Wang et al. [9] scheme is illustrated below. It is assumed that the data owner (DO) shares a pair-wise secret key k_{du} with the user (U), for secure message communication between them.

1. At the registration time, the user U sends the following message to DO,

$$U \rightarrow DO : \{U, DO, E_{k_{du}}(U, DO, ri, \{indexes\}, MAC)\}$$

   where ri is the request index number that is used to protect against replay attack; it is increased by 1 each time the user sends a message to the data owner. $\{indexes\}$ is the set of data block indexes that U wants to register (or require future access). MAC is used for message integrity protection and is computed over all other information in the encryption (i.e., U, DO, ri and $\{indexes\}$).

2. Upon receiving the registration request, DO decrypts it, verifies the integrity and freshness of the message. Then, DO replies with the following message.

$$DO \rightarrow U : \{DO, U, E_{k_{du}}(DO, U, ri, ACMindex, seed, K', cert, MAC)\},$$

   where secret *seed* is used in pseudorandom function $P()$ for generating secure bit string by U and CSP, for secure message communication. This secure bit string is used for over-encryption by CSP. U's subscription key K' is used to derive the authorized encryption keys in the key management hierarchy. The *cert* is the certificate as described before.

3. U sends a resource access request to the CSP containing *request id* (i.e., index to be accessed) and the *cert* as follows.

$$U \rightarrow CSP : \{U, CSP, request\,id, cert\}$$

4. Upon receiving the access request, CSP will verify the user's *cert* for updated $ACMindex$. Then it verifies the *request id* in the *cert* and retrieves the respective cipher blocks. The CSP will then over-encrypt the encrypted data blocks using the secret key generated using $P()$ and *seed*, if needed. Then it sends the over-encrypted data blocks back to the user.

5. Upon receiving message 4, U uses his *seed* value, $P()$ and K' to decrypt the received data blocks.

Analysis If the CSP is malicious then re-keying is necessary to disallow a revoked user from accessing her revoked resources. However, if CSP is honest-but-curious then the re-keying can be avoided as in the above scheme.

In the above procedure, access right revocation is handled using over-encryption and $ACMindex$ value. Over-encryption defends against eavesdropping by revoked users, whereas updated $ACMindex$ value in certificate defends against unauthorized authentication at CSP. However, in the case of malicious CSP, it can maliciously give access of the revoked resources to the users without being detected by the data owner. The CSP can give the revoked resource's access to a user without over-encrypting it. It becomes possible because the revoked resources are not re-encrypted and the revoked resource's encryption key is possibly stored with the user.

The $ACMindex$ value in the Wang et al. protocol is updated after each change in user's access right. Note that the increment in $ACMindex$ value requires a significant amount of computation and communication overhead at data owner to compute each existing certificate again and re-distribute them among the users.

Therefore, the system is not well scaled when a number of users in the system are large and changes in users' access rights are frequent. In what follows, we propose a modification to the above protocol that makes it scalable.

2.5.2 Improving efficiency

Here, we suggest modifications to the work in [9] to reduce computational overhead at the data owner and communication overhead due to user's access right revocation. This is an important requirement when using data outsourcing in the cloud.

Instead of using common incremental $ACMindex$ value in the system, we introduce a separate user certificate index ($UCIndex$) value for each user. Data owner will store latest $UCIndex$ value for each user in a ACM matrix along with user's access authorization details. At the time of service registration (message 1 in Figure 2.17) by the data owner, cloud server will create an empty $UCIndex\ table$ ($UCIT$). $UCIT$ is used by the cloud server at the time of user's data access request to check whether the user contains the latest certificate. $UCIT$ contains two fields: user id and user's latest $UCIndex$ value. At the time of user registration the data owner assigns a $UCIndex$ value in the ACM matrix along with other user authorization details. Then, it sends user's update request with $UCIndex$ value to the cloud server. Receiving which, the cloud server will update $UCIT$ with respective user's $UCIndex$ value and sent back an acknowledgment (Ack). Upon receiving Ack, the data owner creates user's certificate along with assigned $UCIndex$ value and sends it to the user.

Whenever a user's access right is revoked, her $UCIndex$ value is incremented by one without affecting other users' $UCIndex$ values. As a user's $UCIndex$ value is incremented, the data owner immediately send user's $UCIndex$ update request to cloud server as shown in message $2(a)$ in Figure 2.17. The cloud server contains $UCIT$ with the latest value of $UCIndex$ associated with each user in the system. Upon receiving the update request from the data owner, if new user request or received user's $UCIndex$ value is greater than the stored value then the cloud server updates the received user's $UCIndex$ value in $UCIT$ by storing received $UCIndex$

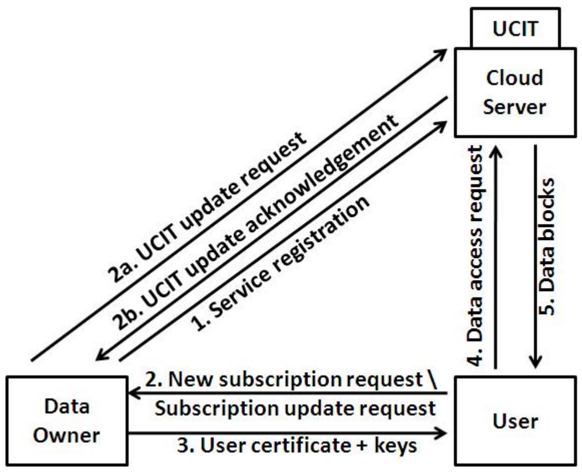

UCIT
Cloud Server
2a. UCIT update request
2b. UCIT update acknowledgement
1. Service registration
4. Data access request
5. Data blocks
Data Owner
2. New subscription request \ Subscription update request
3. User certificate + keys
User

Figure 2.17: Data outsourcing architecture

value. If $UCIT$ is updated, then the cloud server sends an Ack back to the data owner (message $2(b)$). Otherwise, it informs the data owner that greater/equal user's $UCIndex$ value is already stored or incorrect message is received.

To ensure data security, the data owner waits for an acknowledgment (Ack) for every user's $UCIndex$ update request from the cloud server (message $2(b)$). If the acknowledgment is not received in a fixed period of time, it resends the update request to the cloud server. In the case of partial subscription withdrawal by the user, a new certificate should be sent to the user only after receiving acknowledgment for the user's $UCIndex$ update request. In the case of full subscription revocation (or user revocation), user's record corresponding to the revoked subscription is deleted from the $UCIT$. In the case of extension of a user's subscription, same $UCIndex$ value can be used in updated certificate sent to the user.

For every resource request from the user (message 4), cloud server first validates the user's certificate by decrypting it and authenticates the user's access request. If succeed, it will check the user's $UCIndex$ value in $UCIT$ and in user's certificate. If the two values are equal, user's data access request is accepted and the requested data blocks are sent back. Otherwise, it sends an error message. In the case of the request is rejected due to invalid $UCIndex$ match, the user will request the data owner for the latest certificate for further data access. If the user is unauthorized to access any subset of requested data blocks, cloud server sends the data block *id*'s to the user.

Table 5.4, shows worst case comparison between Wang et al. [9] scheme and

Table 2.6: Efficiency comparison

	Wang et al. scheme	My modified scheme
Effect of a user revocation on other users access	Each user requires new certificate for further access	Independent
Subscription extension/ revocation cost at DO	$O(\|N_U\|)$	$O(1)$

our proposed modified scheme. $|N_U|$ represents a number of users in the existing system. In our modified scheme, the effect of a user's access right revocation is independent of other users data access procedure. In Wang et al. scheme, each user requires a newly updated certificate from data owner for further data access. Also, the computational ($O(|N_U|)$ encryption and decryption) overhead at data owner is reduced to 1 encryption and decryption in our modified scheme. Our scheme requires at most one certificate to be created corresponding to the partially revoked user, instead of updating all the user certificates in the system.

2.6 Summary

In this chapter, we have critically analyzed two types of hierarchy used for read access control in data outsourcing scenario. To reduce the public storage, the notion of minimal vertex hierarchy is introduced that makes it practical in use. Finding a minimal vertex hierarchy is an approximation to the problem of finding minimum cost hierarchy for a given hierarchy and a subset of nodes in the hierarchy. The problem of finding minimum cost hierarchy can be easily transformed into well-known q-RST problem [7] which is NP-hard [8]. We proved that finding minimum cost hierarchy and q-RST problems are equivalent. Therefore, if there exists an algorithm to solve minimum cost hierarchy problem, the algorithm can be used to solve the q-RST problem.

We showed that the resource-based hierarchies are more efficient in terms of computation and communication costs as compared to user-based hierarchies with respect to the dynamic operations such as extending read access and user revocation operations. For better understanding, both of the hierarchy types are implemented in local cloud. We experimentally evaluated the performance of dynamic operations in both of the hierarchies to demonstrate our analytical results. For the sake of confidence, the dynamic operations are performed over different size of initial hierarchies and individual results are averaged.

Access right revocation problem is revisited. We reviewed Wang et al. [9] scheme and found that in the scheme a user's access right revocation leads to immediate updation of all other users' authorization certificates. We proposed a modification to their scheme so that any user's access right revocation will be independent of other users', without sacrificing other desirable properties of the scheme.

In the next chapter, we study the use of resource hierarchy for time-limited read access control where access to a file is given for a fixed amount of time. After the time expires, the user authorization is automatically revoked.

CHAPTER 3
Key management for time-bound read access control

Time-limited read access control allows read access to a file only for a pre-specified interval of time. In this chapter, we describe a novel and efficient key assignment scheme for time-limited access with constant secret key storage per user. The scheme is formally analyzed using a modern notion of security, i.e., "key recovery".

3.1 Introduction

There is a set of applications which require time-limited access control over outsourced data such as e-newspaper, e-magazine, digital libraries and PayTV system. A user can subscribe to a service for a fixed time duration. For example, a user can subscribe to an e-newspaper for the months of March and April.

The minimum possible subscription period is referred to as a time slot. In the above example, a time slot is of one month duration. Each time slot has one or more resources associated to it. A continuous sequence of time slots is called an *interval*. A user can subscribe to the service for one or more time intervals.

A cryptographic way to enforce time-limited access control is by assigning a secret key to each time slot in the system. Each key is then used to encrypt the resources, associated with the respective time slot. A user subscribed for a subscription interval must be securely given the respective set of encryption keys. Whenever needed, a user can retrieve any authorized encrypted resource and decrypt it using its known (stored) keys.

To minimize these numbers of keys associated with a user's subscription, a subscription hierarchy can be used where each node represents a subscription-interval (or time-interval).

Figure 3.1 shows a subscription hierarchy with three distinct time slots (requires three leaf nodes). We can see from the figure that each subscription interval is represented by a unique node in the hierarchy. A key is assigned to each node in the subscription hierarchy. Data files are encrypted with leaf node encryption keys. In the figure, a user subscribed for time slots 1 and 2 is given a single secret information K_{1-2} through which it can compute encryption keys K_{1-1} and K_{2-2}. In the following, K_{i-i} is sometimes written as K_i.

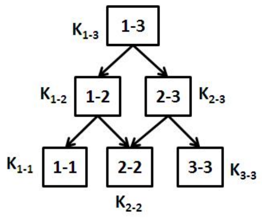


Figure 3.1: An example subscription hierarchy

Similar to $HKAS$, subscription-based $HKAS$ (or $SBHKAS$) assigns keys in a subscription hierarchy so that a user subscribed to a node (time-interval) in the hierarchy can efficiently access only its authorized subscription keys. We define a $SBHKAS$ as follows.

- *Gen :* returns a labeled set of encryption keys (K_{t_a,t_b} where (t_a, t_b) is a time-interval from time slot t_a to t_b) and system public information (Pub).
- *Der :* takes K_{t_a,t_b}, target time slot t and Pub as input, and return encryption key $K_{(t,t)}$ whenever $t_a \leq t \leq t_b$.

In designing a $SBHKAS$, our aim is to minimize secret key storage with each user, system public storage cost and key derivation cost in the subscription hierarchy. It is important to reduce the public storage in cloud scenario since the user has to pay-as-use basis, i.e., the user has to pay more with the increase in storage. Another requirement is that the scheme must be dynamic and can incorporate new time slots with the passage of time.

In this chapter, we focus on the $SBHKAS$ constructions that require single key storage per user per subscription. We propose a simple and efficient hash-based construction for $SBHKAS$ that uses indirect key derivation with dependent keys. With single key storage per user per subscription, our scheme requires less public storage and only one key private storage with Central Authority (CA). Our scheme requires at most $log\, z$ hash operations as key derivation cost in a subscription hierarchy as in [36] where z is the number of time slots. The security of our scheme relies on the one-way property of cryptographic hash functions.

In the remainder of this section, we first discuss the difference between subscription-based access control system ($SBACS$) with a single resource and with multiple resources by giving an example. In the following section, we review selected existing $SBHKAS$s. Section 3.3 describes the proposed $SBHKAS$. We discuss how the new construction can be extended to multiple resource system. Section 3.3.2, gives a formal security proof for the new construction. Section 3.4 provides a comparative analysis of the proposed scheme with similar existing schemes. Section 3.5 discusses the related dynamic operations. Section 3.6 summarizes this chapter.

For the sake of readability, notations used in this chapter are shown in Table 3.1.

3.1.1 Classification

We divide $SBACS$ into two types: $SBACS$ with a single resource and $SBACS$ with multiple resources. An example of $SBACS$ with the single resource is an e-newspaper system with only one type of newspaper. It requires a single sub-

Table 3.1: Notations and abbreviations used

Notation	Description
$h()$	Cryptographic hash function
z	Total number of time slots
t_i	i^{th} time slot
(t_i, t_j)	Time interval from time slot t_i to t_j
K_{t_i,t_j}	Encryption key associated with time interval (t_i, t_j)
$\oplus$	Bit-wise xor operator
$Enc()$	Symmetric encryption function

scription hierarchy. A newspaper is associated with each time slot (say, a day) in the subscription hierarchy. Each copy of the newspaper is encrypted with its associated leaf node's encryption key. Whereas, an example of $SBACS$ with multiple resources is an e-newspaper system with more than one type of newspapers. The user can subscribe for any allowed subset of newspapers types. The hierarchy shown in Figure 3.2 shows 4 groups with different resources R_1, R_2, R_3 and R_4. A user with access to a resource group in the hierarchy is also eligible to access descendant resource groups. Hence, the hierarchy can be viewed as packages; $\{R_1, R_2, R_3, R_4\}$, $\{R_2, R_4\}$, $\{R_3, R_4\}$ and $\{R_4\}$. Each node in the hierarchy is associated with an instance of common subscription hierarchy. A user subscribed for package $\{R_2, R_4\}$ and time interval (t_a, t_b) is given a secret information of a node in instance subscription hierarchy associated with resource R_2. Using given secret information, the user can access any resource of type R_2 and R_4 associated with any time slot t, $t_a \leq t \leq t_b$. Since every user in the system is assigned to any one of the group (node) in the hierarchy, we can also call such hierarchy as a *user hierarchy*.

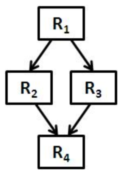


Figure 3.2: An example of user hierarchy

3.2 Previous schemes

3.2.1 Ateniese et al. scheme

Ateniese et al. [35] propose two basic constructions for $SBHKAS$ with multiple resources. Here we describe their symmetric encryption-based construction. This was simple but worst case construction, i.e., every possible subscription information has a direct edge to its all associated encryption keys.

The total system time T is divided into z distinct and equal time slots t_1, t_2, ..., t_z. Let P be the number of possible time intervals in the system. For example, consider a user hierarchy and a subscription hierarchy with $z = 2$ shown in Figure 3.3. Here $|P| = 3$, i.e., t_1, t_2 and t_{1-2}. Their corresponding complete system hierarchy is shown in Figure 3.4. In the figure, every edge has associated with a public value used for key derivation. As per the system hierarchy, the user b who is authorized for subscription t_{1-2} (represented as node bt_{1-2}) can access any resource associated with node bt_1 (i.e., user b with subscription t_1), bt_2, dt_1 and dt_2.

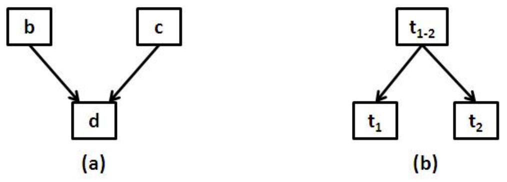


Figure 3.3: (a) User hierarchy, (b) Subscription hierarchy

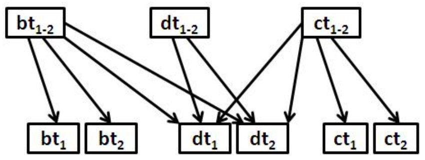


Figure 3.4: A complete system hierarchy corresponding to Figure 3.3

In Ateniese et al. scheme, Algorithm $Gen()$ (Algorithm 5) is used to generate and assign keys to the time intervals. This algorithm returns subscription information set s, assigned set of keys K' to the nodes in the system and a set of public information Pub.

Algorithm 5 Gen($1^k, UH, P$)

```
Input: Security parameter $1^k$, user hierarchy $UH$ and set of time intervals $P$.
Output: Returns set $(s, K', Pub)$.
1: Perform a graph transformation to obtain the two-level partially ordered hierarchy $G_{PT} = (V_{PT}, E_{PT})$, where $V_{PT} = V_P \bigcup V_T$.
2: For each node $u_\lambda \in V_P$, let $s_{u,\lambda} \in \{0,1\}^k$.
3: For each class $u_t \in V_T$, randomly choose a secret value $k_{u,t} \in \{0,1\}^k$.
   /* Let $s$ and $K'$ be the sequences of private information and keys, respectively, computed in the previous two steps. */
4: For any pair of classes $(u_\lambda, v_t) \in V_P$ x $V_T$ such that $(u_\lambda, v_t) \in E_{PT}$, compute the public information $p_{(u,\lambda),(v,t)} = Enc_{s_{u,\lambda}}(k_{v,t})$.
   /* Let $Pub$ be the sequence of public information computed in the previous step. */
5: **return** $(s, K', pub)$
```

In Step 1 of procedure $Gen()$ (Algorithm 5), a hierarchy G_{PT} is generated with set of nodes $V_{PT} = V_P \bigcup V_T$ and set of edges E_{PT} where set V_P contains all the nodes corresponding to set P and set V_T contains nodes corresponding to set T(= $\{t_1, t_2, ..., t_z\}$). Step 2 and 3 assigns random keys to the nodes in set P and T respectively. For every edge in set E_{PT}, a public edge value is computed used for key derivation purpose (Step 4).

A user at node u in the user hierarchy with subscription information $s_{u,\lambda}$ can derive encryption key of any node $v \preceq u$ for a time slot t with $t \in \lambda$. Key derivation procedure is shown in Algorithm 6. It takes as input the system hierarchy G_{PT}, user's node u, target node v, user's authorized time interval λ, user's subscription information $s_{u,\lambda}$, target time slot t and Pub. In Step 1, it finds a public edge information from Pub corresponding to v and t. Then decrypt key $k_{v,t}$ using the public edge information and user's subscription information $s_{u,\lambda}$ (Step 2) and return. Since, every subscription node in the upper level of system hierarchy is associated with a direct edge (value) to every possible encryption (key) node at lower level, key derivation cost will be 1 decryption operation.

Algorithm 6 Der($G_{PT}, u, v, \lambda, s_{u,\lambda}, t, Pub$)

```
Input: A hierarchy $G_{PT}$, source node $u$, target node $v$, authorized time interval $\lambda$,
user's subscription information $s_{u,\lambda}$, target time slot $t$ and $Pub$.
Output: Returns target encryption key $k_{v,t}$.
1: Extract the public value $p_{(u,\lambda),(v,t)}$ from $Pub$.
2: Compute key $k_{v,t} = D_{su,\lambda}(p_{(u,\lambda),(v,t)})$.
3: return $(k_{v,t})$
```

A node in the user hierarchy can subscribe for any of the time intervals in P. Therefore, every node in the user hierarchy must have a $|P|$ number of associated nodes in the top level of the two-level system hierarchy. If we have a m number of nodes in a user hierarchy, the top level of system hierarchy should have mp nodes where p is $O(z^2)$. The lower level of system hierarchy will have z (=$|T|$) nodes corresponding to each node in user hierarchy, i.e., mz nodes in total. Now, since a node in the top level of system hierarchy associated with root node in user hierarchy and with lifetime subscription should be associated with all $|T|$ edges. Therefore, a total number of public edge values will be at most $(mp)(mz)$, i.e., $O(m^2z^3)$.

3.2.2 Atallah et al. base scheme

Atallah et al. [35] proposed a family of SBHKAS with single user hierarchy. Their constructions were based on symmetric encryption. We will discuss their all variations step-by-step. Let m and z denote the number of nodes in user hierarchy and a number of time slots in the system respectively. Initially, we will start with the construction with respect to a subscription hierarchy only, since the remaining part of the construction is similar to all constructions. In the latter part of the description, we merge user hierarchy with subscription hierarchy and give a complete description of the system. Let $T = \{t_1, ..., t_z\}$ denote the set of all time slots and $\{k_{t_1}, ..., k_{t_z}\}$ denote the corresponding encryption keys. First, we will describe their base scheme and later we give their improved scheme.

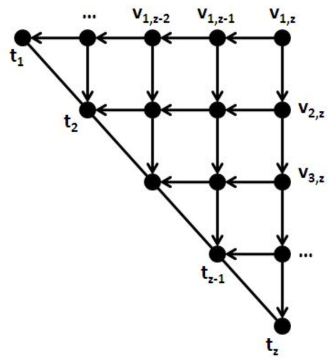


Figure 3.5: Base data structure with $|T| = z$

Base scheme

Consider a subscription hierarchy shown in Figure 3.5. It is built over a set T of time slots with $|T| = z$. Assign a random key $K_{i,j}$ and a random label $l_{i,j}$ to each node $v_{i,j}$ in the hierarchy. For each directed edge $((i,j),(k,p))$ from node $v_{i,j}$ to $v_{k,p}$ in the hierarchy compute a public value $y_{(i,j),(k,p)} = K_{k,p} \oplus h(K_{i,j}, l_{k,p})$.

A user will require one secret key per subscription (For example, $K_{i,j}$ for subscription (t_i, t_j)). The distance between any two nodes in the subscription hierarchy is at most $(z-1)$ edges, therefore key derivation cost is $O(z)$. The number of nodes in the hierarchy is $(1/2)z(z+1)$, i.e., $O(z^2)$. It requires $z(z-1)$ (i.e., $O(z^2)$) public edge values for single subscription hierarchy.

3.2.3 Atallah et al. improved scheme

The improved scheme reduces the public storage cost over the base scheme. A data structure is built over set T using procedure $DataStructBuild(v, T)$ (Algorithm 7) where v is the root node with set of time slots T.

Algorithm 7 *DataStructBuild*(v, T)

Input: A set of system time slots T and node v.

Output: Build a respective data structure.

```
1: If $|T| = 2$, then return.
2: Partition $T$ into $\sqrt{|T|}$ sets of $\sqrt{|T|}$ contiguous time slots each, call these
   $T_1, ..., T_{\sqrt{|T|}}$, i.e.,if $T = t_1, ..., t_{\sqrt{|T|}}$, then $T_i = t_{i\sqrt{|T|}+1}, ..., t_{i\sqrt{|T|}+\sqrt{|T|}}$. Create a
   node $v_i$ for each $T_i$, and make $v_i$ a child of $v$.
3: Generate a problem $Coarse(T)$, derived from $T$ by treating each $T_i$ as a black
   box (i.e., merging the constituents of $T_i$ into a single item). Note that the size
   of set $Coarse(T)$ is $\sqrt{|T|}$.
4: Store at node $v$ an instance of the basic scheme for $Coarse(T)$, denoted $D(v)$.
   $D(v)$ can only process an interval if it is the union of a contiguous subset of
   $Coarse(T)$ (i.e., it cannot handle intervals whose endpoints are inside the $T_i$'s,
   as it cannot see inside a $T_i$).
5: Also store at node $v$ two solutions of one-dimensional problems on $T$:
   One is for intervals all of which start at the right boundary of $T$ and end inside
   $T$ (we call this the right-anchored problem and denote the one-dimensional
   structure for it by $R(v)$);
   another is for intervals all of which start at the left boundary of $T$ and end in-
   side $T$ (we call this the left-anchored problem and denote the one-dimensional
   structure for it by $L(v)$).
   Note that having $R(v)$ and $L(v)$ enables the handling of an interval that lies
   within $T$ and also has its left or right endpoint at a boundary of $T$.
6: Recursively apply the scheme to each child of $T$; that is, call
   $DataStructBuild(v_i, T_i)$ in turn for each $i = 1, 2, ..., \sqrt{|T|}$ .
7: return
```

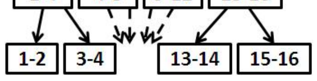


Figure 3.6: Data Structure with $|T| = 16$

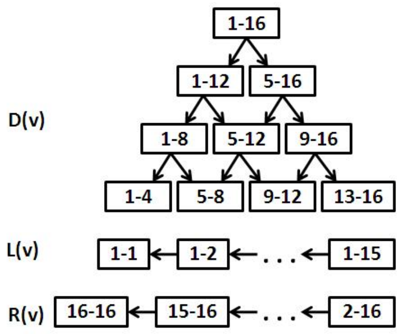


Figure 3.7: $D(v), L(v), R(v)$ for node v with ($|T| = 16$)

Suppose $|T| = z$, then the height of the new data structure is $loglog\, z$. For example, consider node v with $T = \{t_1, ..., t_{16}\}$. The output of procedure *DataStruct Build*(v, T) is shown in Figure 3.6. The children of v are $v_1, ..., v_4$. The Figure 3.7 shows the $D(v)$, $L(v)$ and $R(v)$ structures corresponding to node v.

Let a user request CA to subscribe for a time interval I. It will call procedure *Assignkeys*(I, v, T) (Algorithm 8) where v is the root node with a set of time slots T in the data structure build through Algorithm 7. We can see in the procedure that, the consequence of reduced data structure is that it requires assigning at most 3 keys to the user per subscription.

Algorithm 8 *Assignkeys*(I, v, T)

```
Input: Node $v$, set $T$ and requested subscription interval $I$.
Output: Return secret information corresponding to $I$.
1: If $v$ is a leaf, then return a key for each of the (at most two) time intervals in $I$.
   Else, continue with the next step.
2: Let $v_1, ..., v_{\sqrt{|T|}}$ be the children of $v$, and let $T_1, ..., T_{\sqrt{|T|}}$ be the respective sets
   of times associated with these children. We distinguish two cases:
3: (a). $I$ overlaps with only one set $T_i$. Then we return the keys from the recursive
   call $AssignKeys(I, v_i, T_i)$.
   (b). $I$ overlaps with all of $T_k, T_{k+1}, ..., T_{k+l}$, where $l = 1$. These $l + 1$ intervals
   are handled in 3 different ways:
   Those completely contained in $I$ are collectively processed using the $D(v)$
   structure, resulting in one key.
   If $T_k$ overlaps with $I$ but is not contained in $I$, then it is right-anchored and is
   processed using $R(v_k)$, resulting in one key.
   If $T_{k+l}$ overlaps with $I$, but is not contained in $I$, then it is left-anchored and is
   processed using $L(v_{k+l})$, resulting in one key.
4: return Those (at most) 3 keys are returned.
```

For example, consider data structure given in Figure 3.6. If a user wants to subscribe for an interval from t_1 to t_5, CA will give two keys to the user: K_{t_1,t_4} and K_{t_5}. K_{t_1,t_4} is given from $D(v_1)$ structure and K_{t_5} is given from $L(v_{21})$ structure. v_{21} is left child of v_2 with time slots t_5 and t_6. Similarly, a user need a subscription from t_4 to t_{13} is given three keys K_{t_4}, K_{t_5,t_12} and $K_{t_{13}}$. K_{t_4} is given using $R(v_1)$, K_{t_5,t_12} is given using $D(v)$ and $K_{t_{13}}$ is given using $L(v_4)$.

To derive an authorized key for some time slot t_i, a user with secret information S_{TU} will call procedure *DeriveKey*(t_i, S_{TU}, Pub) (Algorithm 9). The algorithm will first parse the user's subscription information S_{TU} into three intervals R, D and L. The target time slot t_i is in either of them. If $t_i \in R$ (or D, L) then find the node v at level l in data structure R (or D, L) such that $R(v)$ (or $D(v)$, $L(v)$) permits access to t_i. Now, using key k_v (the key corresponding to node v) and Pub, it derives and return the key k_{t_i} corresponding to given time slot t_i. For example, consider data structure given in Figure 3.6. A user with $S_{TU} = \{K_{t_4}, K_{t_5,t_{12}}, K_{t_{13}}\}$ is authorized for time slot t_5. It can first derive K_{t_5,t_8} using $D(v)$ and $K_{t_5,t_{12}}$. Then, derive K_{t_5} using K_{t_5,t_8} in data structure given in Figure 3.6.

Algorithm 9 *DeriveKey*(t_i, S_{TU}, Pub)

Input: A user's subscription information S_{TU}, authorized target time slot t_i and Pub.
Output: Return key K_{t_i}.

```
1: Parse $S_{TU}$ as $k_1(l, R, t_{start}, t_a)$,$k_2(l-1, D, t_{a+1}, t_b)$, $k_3(l, L, t_{b+1}, t_{end})$
2: If $t_i \in \{t_{start}, ..., t_a\}$, find the node $v$ at level $l$ such that $R(v)$ permits access
   to $t_i$. Use $k_1$ and $Pub$ to derive the key corresponding to $t_i$ and return that
   enabling key.
3: Similarly, if $t_i \in \{t_{b+1}, ..., t_{end}\}$, locate node $v$ at level $l$ such that $L(v)$ permits
   access to $t_i$. Use $k_3$ and $Pub$ to derive an enabling key for $t_i$ and return that
   key.
4: Finally, if $t_i \in \{t_{a+1}, t_b\}$, locate $v$ at level $l-1$ such that $D(v)$ permits access to
   $t_i$; use $k_2$ and $Pub$ to derive an enabling key for $t_i$ and return it.
5: return
```

Consider a $D(v)$ structure with root node v in the data structure with $|T| = z$. Since height of $D(v)$ structure is $O(\sqrt{z})$, key derivation cost of deriving a leaf node key in $D(v)$ structure is $O(\sqrt{z})$. Since height of subscription hierarchy is now $loglog\, z$, to derive a leaf node in a subscription hierarchy using $D(v)$ structure will be at most $\sqrt{z} + loglog\, z$, i.e., $O(\sqrt{z})$.

Consider $L(v)$ and $R(v)$ of the root node v in the data structure. Length of structures $L(v)$ and $R(v)$ is $z-1$ each. Therefore, key derivation cost for deriving a leaf node key in a subscription hierarchy will be at most $z + loglog\, z$, i.e., $O(z)$.

Suppose v is the root node in the data structure with $|T| = z$, produced by Algorithm 7. Let R_z is the number of edges in $R(v)$, L_z is the number of edges in $L(v)$, D_z is the number of edges in $D(v)$ and C_z is the number of child (outgoing edges) nodes of in v. The following recurrence defines the public storage requirement corresponding to edges in the system is defined by the following recurrence.

$$\begin{aligned} S(z) &= R_z + L_z + D_z + C_z + z^{1/2}S(z^{1/2}) && (3.1)\\ &= z + z + (z^{1/2})^2 + z^{1/2} + z^{1/2}S(z^{1/2}) && (3.2)\\ &= 3z + z^{1/2} + z^{1/2}S(z^{1/2}) && (3.3)\\ &= 3z + z^{1/2} + z^{1/2}(3z^{1/2} + z^{1/4} + z^{1/4}S(z^{1/4})) && (3.4)\\ &= (3z + z^{1/2}) + (3z + z^{3/4}) + z^{3/4}S(z^{1/4}) && (3.5) \end{aligned}$$

Since, height of the data structure is *loglog* z, we write

$$\begin{aligned} S(z) &\leq 4z\,loglog\,z && (3.6)\\ &= O(z\,loglog\,z) && (3.7) \end{aligned}$$

3.2.4 Improved scheme with shortcut edges

Here, we first merge complete system together. Consider a user hierarchy (Directed acyclic graph) UH in the system. Let us consider the procedure $Gen()$ given in Appendix A. It assigns secret keys, public labels and public edge values to the subscription hierarchy corresponding to each node in the user hierarchy.

The Algorithm 23 first creates a tree data structure (using Algorithm 7) as a subscription hierarchy and assigns public labels to each node in user and subscription hierarchy. Then for each node u in the user hierarchy, it picks secret keys for its copy of subscription hierarchy and generates public information according to those keys. Next, it connects the data structures corresponding to different nodes in the user hierarchy. That is, for each time interval t, if node u_1 is a parent of node u_2, we compute public edge value that permits derivation of k_{u_2}, t from $k_{u_1,t}$. Then assign an edge between each $t \in T$ and an associated node in $D(v)$, $R(v)$ and $L(v)$ structures in G. An associated node with the time slot t means a node with key k_t. Finally, it creates three sets: set K consists of keys for every $t \in T$, set Sec consists of remaining keys in the system and public set Pub containing all public information associated with the system.

In Algorithm 23, Step 7 assigns a direct edge between each time slot t and the corresponding node in $D(u)$, $R(u)$ and $L(u)$ structures for each $u \in V$. Therefore, three additional edges (one for each $D(u)$, $R(u)$ and $L(u)$) are required from a node u at each level in the subscription tree structure. Since, height of subscription tree structure is $loglog\, z$ for a system with $|T| = z$, it requires additional $(3z)loglog\, z$ edges.

A user with subscription key S_{v,T_U} can derive key $K_{w,t}$ whenever $w \preceq v$ in user hierarchy UH and $t \in T_U$. Key derivation procedure is written in Algorithm 10. In Step 1, validity of the request is verified. If destination node w is not descendant of source node v or the user is not authorized for target time slot t then it returns $null$. Step 2, derives kv, t in the subscription hierarchy using Pub. Step 3, returns target key kw, t using kv, t, public labels corresponding to user hierarchy and Pub.

Algorithm 10 Derive($v, w, T_U, t, S_{v,T_U}, Pub$)

Input: A source node v, destination node w, user's authorized time interval T_U, target time slot t, user's secret information S_{v,T_U} and system public information Pub.

Output: If user is authorized, destination key $K_{w,t}$ is returned.

```
1: If $t \notin T_U$ or $w \notin Desc(v, G)$, return $Null$.
2: Execute $DeriveKey(t, S_{v,T_U}, Pub)$ to compute an enabling key for $t$; call it $k_{v,t}$.
3: Use $k_{v,t}$ along with its (level-type) label and $Pub$ to derive key $k_{w,t}$.
4: return
```

Now, we apply shortcuts [51] to each one-dimensional structure $L(v)$ and $R(v)$ in tree data structure. Also, for each two-dimensional $D(v)$ structure, apply shortcuts to each horizontal and vertical one-dimensional sub-hierarchies in the structure. All possible one-dimensional sub-hierarchies in a two-dimensional structure with n leaf nodes are shown in Figure 3.8.

Assume that any two nodes in an one-dimensional sub-hierarchy are at a distance of at most s hops (edges). We call such scheme $s - HS$ (s Hop Scheme) [35]. A procedure for creating $3 - HS$ is written in Algorithm 22 [51] (see Appendix A). The cost of the algorithm will be $O(z)$ for z node one-dimensional hierarchy. Comparison of shortcut schemes for one-dimensional structures with n nodes is

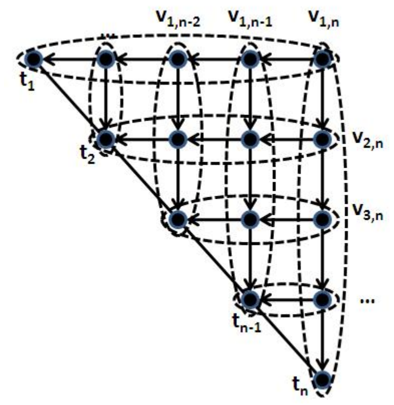


Figure 3.8: Base data structure with one-dimensional sub-hierarchies

given in table 3.2 [35]. Public edges information in the table shows additional public edge information added by the algorithm.

Table 3.2: Comparison of shortcut schemes

	Public (edge) information	Private information	Key derivation cost
2-HS	$O(nlog\,n)$	1	2-op
3-HS	$O(nloglog\,n)$	1	3-op
4-HS	$O(nlog^3\,n)$	1	4-op
log n-HS	$O(n)$	1	$O(log\,n)$-op

Consider $s = log\,n$. For key derivation of a leaf node, it requires at most $log\,n$ operations in horizontal direction and $log\,n$ operations in vertical direction, i.e., at most $2log\,n$ operations.

Assume a $log\,z - HS$ shortcut scheme is used in the system. It means, every two nodes in $D(v)$, $R(v)$ and $L(v)$ corresponding to each node in tree data structure are at a distance of at most $log\,z$ edges. Longest data structure are with root node v, i.e., $R(v)$ and $L(v)$, with z nodes in the chain. Applying shortcut edges using $log\,z - HS$, key derivation cost will be at most $log\,z$. Similarly, the $D(v)$ structure with height $\sqrt{z}$ requires $2log\,(\sqrt{z})$ operations ($log\,(\sqrt{z})$ operations in each horizontal and vertical direction) for leaf node key derivation (in $D(v)$ structure). Then one additional operation is required to derive target leaf node key in subscription tree hierarchy. Hence, total key derivation cost in subscription hierarchy will be at most $log\,z\, +\, 1$.

Total public edge storage S_{sz} associated with a subscription hierarchy will be addition of 3 sets of public edge values associated with: base data structure build using Algorithm 7 ($S(z)$), directed edges in Step 5 of Algorithm 23 and additional shortcut edges $S_{ed}(z)$ (using $log\, z - HS$). Edges associated with base data structure build using Algorithm 7 are computed above as $S(z)$ in inequality (3.6). Hence, total public storage with respect to single subscription hierarchy will be as follows.

$$S_{sz} \leq S(z) + (3z)loglogz + S_{ed}(z) \tag{3.8}$$

$$\leq 4zloglogz + (3z)loglogz + S_{ed}(z) \tag{3.9}$$

$$= O(zloglogz) + S_{ed}(z) \tag{3.10}$$

Additional shortcut edges ($S_{ed}(z)$) associated with single subscription hierarchy with z leaf nodes can be computed by following inequality,

$$S_{ed}(z) \leq \sqrt{z}(S_{ed}(\sqrt{z}) + L_z + R_z + D_{\sqrt{z}}$$

where, L_z and R_z are the cost of shortcut edges associated with $L(v)$ and $R_(v)$ data structures with z nodes respectively, $D_{\sqrt{z}}$ is the cost of shortcut edges associated with $D(v)$ data structure with $\sqrt{z}$ nodes.

$$S_{ed}(z) \leq \sqrt{z}(S_{ed}(\sqrt{z}) + z + z + (\sqrt{z})^2 \tag{3.11}$$

$$\leq \sqrt{z}(S_{ed}(\sqrt{z}) + 3z \tag{3.12}$$

$$= O(zloglogz) \tag{3.13}$$

3.2.5 Crampton scheme

Crampton [36, 58] proposed a SBHKAS with single key storage per user. The nodes in the subscription hierarchy are first divided into 2 triangles (i.e., Ts) and one diamond (i.e., D) structures called blocks. Then edges are inserted between nodes in diamond structure to the nodes in triangle structures for key derivation.

Consider a subscription hierarchy with $z = 4$ as shown in Figure 3.9. A corresponding hierarchy structure named T_{2m} with $m = 2$ (initially, $2m = z$) is shown in Figure 3.10. T_{2m} contains two copies of T_m covering leaf nodes and one copy of D_m containing root node. There are no edges between the nodes in D_m.

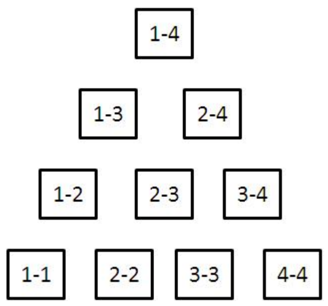


Figure 3.9: A nodes structure for an example subscription hierarchy (with $z = 4$)

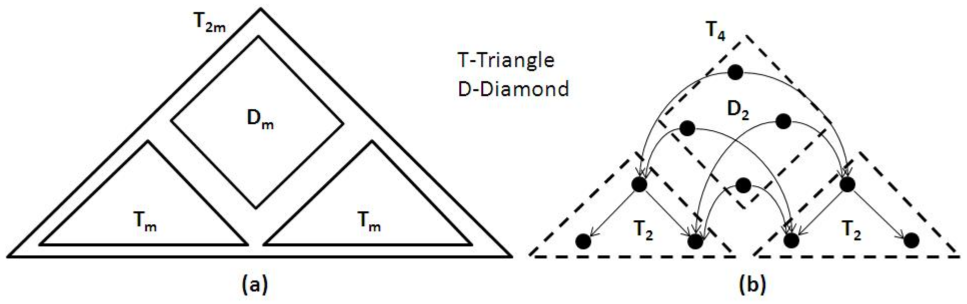


Figure 3.10: (a) Division of nodes, and (b) Key derivation

For key derivation, each node $(i, j) \in D_m$ has two outgoing edges to the nodes (i, m) and $(m + 1, j)$ respectively where each end node belongs to a different triangle structure. For example, the outgoing edges from node $(1, 4)$ in D_2 are $((1, 4), (1, 2))$ and $((1, 4), (3, 4))$.

Key derivation in each of the T_m is implemented recursively, i.e., similar to the procedure used for T_{2m}. Now, since edges are going from D_m to T_m, they are able to reduce key derivation cost to $log\, z$ operations. $z(z - 1)$ number of public edge values are needed corresponding to a subscription hierarchy without using shortcut edges.

3.3 A new construction with reduced public storage cost

In this section, we propose a new construction for $SBHKAS$ with single key storage per user per subscription. The scheme uses indirect key derivation with dependent keys. It requires less public storage ($z(z-1) - \lceil (1/8)z(3z+2) \rceil$) and only one key private storage at CA. It has a key derivation cost of at most $log\, z$ hash operations.

The proposed $SBHKAS$ uses xor and hash functions as primitive operations. The CA generates system public information and secret information for each existing subscription interval so that a user with his secret information can derive any authorized subscription node's key in the hierarchy.

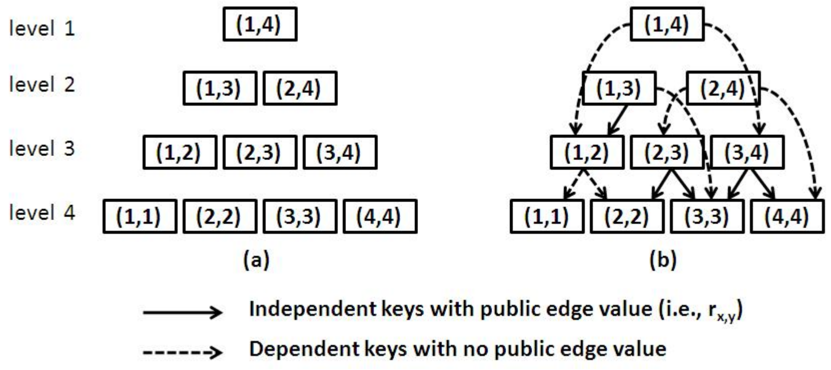

level 1
level 2
level 3
level 4
(1,4)
(1,3)
(2,4)
(1,2)
(2,3)
(3,4)
(1,1)
(2,2)
(3,3)
(4,4)
(a)
(b)
Independent keys with public edge value (i.e., r_{x,y})
Dependent keys with no public edge value

Figure 3.11: (a) An example subscription hierarchy structure, and (b) Key derivation structure corresponding to subscription hierarchy in Figure (a)

Key assignment

Assume z time slots $t_1, ..., t_z$. CA generates a subscription hierarchy structure with z leaf nodes, one for each time slot t_i, with $1 \leq i \leq z$.

Step 1. Arrange nodes into levels in such a way that at each level l in the subscription hierarchy structure, there are $l+1$ nodes with subscription interval of size $z-l$ each. Therefore, at level $l=0$ (root level) there is only one $(0+1)$ node with subscription interval size z, i.e., node with time interval (t_1, t_z) represented as $(1, z)$. An example subscription hierarchy structure for $z=4$ time slots is shown in Figure 3.11(a). The keys are assigned to the nodes as follows.

Step 2. CA chooses a public cryptographic one-way hash function $h()$ for key generation. $h()$ is also used by the subscribers for key derivation purpose. A secret key K_s is generated and stored by the CA. CA assigns a random public label $l_{(a,b)}$ to each node (t_a, t_b) in the subscription hierarchy. In order to assign keys to the nodes in subscription hierarchy, CA will call procedure *Key_Assignment()*, defined in Algorithm 11. Step 1 in Algorithm 11, moves to each level from top to bottom in the subscription hierarchy. Step 2 initializes counter i to one, to point left most node in a level. Step 3 moves to each node from left to right in a level. If no key is assigned to the selected node, it compute and assign key to the node using $h()$ and K_s (Step $4-6$). Steps $7, 8$ computes two child nodes named $left$ and $right$ child nodes respectively of the selected node. In Steps $9-13$, if key to $left$ child node is $Null$, then compute and assign dependent key $K_{(i,left)}$ as in Step 10. Else, compute public edge value $r_{(i,j),(i,left)}$ for key derivation between nodes (Step 12). Steps $14-18$ gives similar treatment to $right$ child. Step 19 will increment i by one, to move next node in same level.

Table 3.3: Key assignment to the subscription hierarchy given in Figure 3.11(c)

node	Key	left child	right child	public edge values	
				left child	right child
$(1,4)$	$h(K_s, l_{(1,4)})$	$(1,2)$	$(3,4)$	$-$	$-$
$(1,3)$	$h(K_s, l_{(1,3)})$	$(1,2)$	$(3,3)$	$r_{(1,3),(1,2)}$	$-$
$(2,4)$	$h(K_s, l_{(2,4)})$	$(2,3)$	$(4,4)$	$-$	$-$
$(1,2)$	$h(K_{(1,4)}, l_{(1,2)})$	$(1,1)$	$(2,2)$	$-$	$-$
$(2,3)$	$h(K_{(2,4)}, l_{(2,3)})$	$(2,2)$	$(3,3)$	$r_{(2,3),(2,2)}$	$r_{(2,3),(3,3)}$
$(3,4)$	$h(K_{(1,4)}, l_{(3,4)})$	$(3,3)$	$(4,4)$	$r_{(3,4),(3,3)}$	$r_{(3,4),(4,4)}$
$(1,1)$	$h(K_{(1,2)}, l_{(1,1)})$	$-$	$-$	$-$	$-$
$(2,2)$	$h(K_{(1,2)}, l_{(2,2)})$	$-$	$-$	$-$	$-$
$(3,3)$	$h(K_{(1,3)}, l_{(3,3)})$	$-$	$-$	$-$	$-$
$(4,4)$	$h(K_{(2,4)}, l_{(4,4)})$	$-$	$-$	$-$	$-$

Figure $3.11(c)$ shows output of *Key_Assignment()* (Algorithm 11) for the input hierarchy shown in Figure 3.11(a). In Figure $3.11(c)$, a smooth directed edge denote a public edge value between two end nodes and dotted directed edge denote a dependent key generation as shown in Figure $3.11(b)$. A dotted directed edge from node u to v shows that the key of node v is computed using key of node

Algorithm 11 *Key_Assignment*$(SH, z, K_s, h())$

Input: A subscription hierarchy SH, number of time slots z, secret key K_s associated with SH and $h()$.
Output: Assign keys to the nodes in SH.

```
1:  for $l = 0$ to $(z-2)$ do
2:     $i = 1$
3:     for $j = (z-l)$ to $z$ do
4:        if ($K_{(i,j)} = Null$) then
5:           $K_{(i,j)} = h(K_s, l_{(i,j)})$
6:        end if
7:        $left = \lfloor (i+j)/2 \rfloor$
8:        $right = left + 1$
9:        if $K_{(i,left)} = Null$ then
10:          $K_{(i,left)} = h(K_{(i,j)}, l_{(i,left)})$
11:       else
12:          $r_{(i,j),(i,left)} = h(K_{(i,j)}, l_{(i,left)}) \oplus K_{(i,left)}$
13:       end if
14:       if $K_{(right,j)} = Null$ then
15:          $K_{(right,j)} = h(K_{(i,j)}, l_{(right,j)})$
16:       else
17:          $r_{(i,j),(right,j)} = h(K_{(i,j)}, l_{(right,j)}) \oplus K_{(right,j)}$
18:       end if
19:       $i = i + 1$
20:    end for
21: end for
22: return
```

u. Table 3.3 shows output of *Key_Assignment()* (Algorithm 11) with respect to the subscription hierarchy shown in Figure 3.11(c).

Key derivation

A user with a subscription key can derive any authorized encryption key within its subscription using procedure *Key_Derivation*$()$, defined in Algorithm 12. Suppose that there is a user with secret information $K_{(t_a,t_b)}$ corresponding to a subscription interval (t_a, t_b). To derive an encryption key $K_{(t,t)}$ with $t_a \leq t \leq t_b$, the user will do the following.

Step 1 Let $(t_a < t_b)$, user will find two nodes with subscription (t_a, t_{mid}) and (t_{mid+1}, t_b) where $t_{mid} = \lfloor (t_a + t_b)/2 \rfloor$.

(a) Let $t \in (t_a, t_{mid})$. To compute key $K_{(t_a,t_{mid})}$, if public edge value $r_{(t_a,t_b),(t_a,t_{mid})}$

exists then user will compute $K_{(t_a,t_{mid})} = h(K_{(t_a,t_b)}, l_{(t_a,t_{mid})}) \oplus r_{(t_a,t_b),(t_a,t_{mid})}$ where $h()$ and $l_{(t_a,t_{mid})}$ are public. Otherwise, user will compute $K_{(t_a,t_{mid})} = h(K_{(t_a,t_b)}, l_{(t_a,t_{mid})})$.

(b) Let $t \in (t_{mid+1}, t_b)$. To compute key $K_{(t_{mid+1},t_b)}$, if public edge value $r_{(t_a,t_b),(t_{mid+1},t_b)}$ exists then user will compute $K_{(t_{mid+1},t_b)} = h(K_{(t_a,t_b)}, l_{(t_{mid+1},t_b)}) \oplus r_{(t_a,t_b),(t_{mid+1},t_b)}$. Otherwise, he will compute $K_{(t_{mid+1},t_b)} = h(K_{(t_a,t_b)}, l_{(t_{mid+1},t_b)})$.

Step 2 The user will repeat Step 1 using fresh computed key $K_{(t_x,t_y)}$ ((t_x, t_y) is either (t_a, t_{mid}) or (t_{mid+1}, t_b)) with $t_x \leq t \leq t_y$ until getting target encryption key $K_{(t,t)}$.

Since there is a path from each subscription node to its all descendant leaf nodes in the subscription hierarchy one can derive any authorized leaf node encryption key using the above steps.

3.3.1 Extension to multiple resources

In this section, we show how to extend our construction to a multiple resource system. Following the construction in [35], we use two separate hierarchies: a user hierarchy and a subscription hierarchy. Assume that there is instances of subscription hierarchy structure and each instance is associated with a node in the user hierarchy. Hence, each node (C_i) in a user hierarchy is associated with a similar instance (SH_i) of subscription hierarchy structure (SH).

System hierarchy integration. Now, we combine all instances of common subscription hierarchy structure associated with nodes in user hierarchy into a system subscription hierarchy. For an edge in user hierarchy, there are a z number of new edges in system subscription hierarchy between leaf nodes of their respective instances of common subscription hierarchy. An example system subscription hierarchy is given in Figure 3.12(c). It considers the user hierarchy of Figure 3.12(a). For simplicity, we have taken a common subscription hierarchy structure with $z = 2$ which contains three possible nodes with subscription $(1-2)$, $(1-1)$ and $(2-2)$ as shown in Figure 3.12(b). There are four instances of common subscription hierarchy structure corresponding to four nodes in the resource hierarchy.

Algorithm 12 *Key_Derivation*$(K_{(i,j)}, i, j, t, Pub)$

```
Input: Given a subscription key $K_{(i,j)}$, subscription start time slot $i$, subscription expiry time slot $j$, target time slot $t$ with $i \le t \le j$ and public information Pub.
Output: it returns target node encryption key $K_{(t,t)}$.
1:  if $(t < i)$ or $(t > j)$ or $(j < i)$ then
2:      return $Null$
3:  end if
4:  while $j > i$ do
5:      $m = \lfloor (i+j)/2 \rfloor$
6:      if $(t \le m)$ then
7:          if $r_{(i,j),(i,m)} \in Pub$ then
8:              $K_{(i,m)} = h(K_{(i,j)}, l_{(i,m)}) \oplus r_{(i,j),(i,m)}$
9:          else
10:             $K_{(i,m)} = h(K_{(i,j)}, l_{(i,m)})$
11:         end if
12:         $j = m$
13:     else
14:         if $r_{(i,j),(m+1,j)} \in Pub$ then
15:             $K_{(m+1,j)} = h(K_{(i,j)}, l_{(m+1,j)}) \oplus r_{(i,j),(m+1,j)}$
16:         else
17:             $K_{(m+1,j)} = h(K_{(i,j)}, l_{(m+1,j)})$
18:         end if
19:         $i = m + 1$
20:     end if
21: end while
22: return $K_{(i,j)}$
```

For each edge in resource hierarchy, there are 2 (since, $z = 2$) edges in system subscription hierarchy. For an edge from node C_1 to C_2 in the resource hierarchy, there are 2 edges which connects leaf nodes of the respective instances of common subscription hierarchy (i.e., SH_1 and SH_2) as shown in Figure 3.12(c).

Key derivation. As an example, consider the system subscription hierarchy shown in Figure 3.12(c). Suppose a user with subscription key $K_{2,(1,2)}$ wants to derive encryption key $K_{4,2}$. It will first compute $K_{2,(2,2)}$ in the instance subscription hierarchy SH_2. Then, it traverses in resource hierarchy towards the target node and computes encryption key $K_{4,(2,2)}$ using $K_{2,(2,2)}$ and Pub.

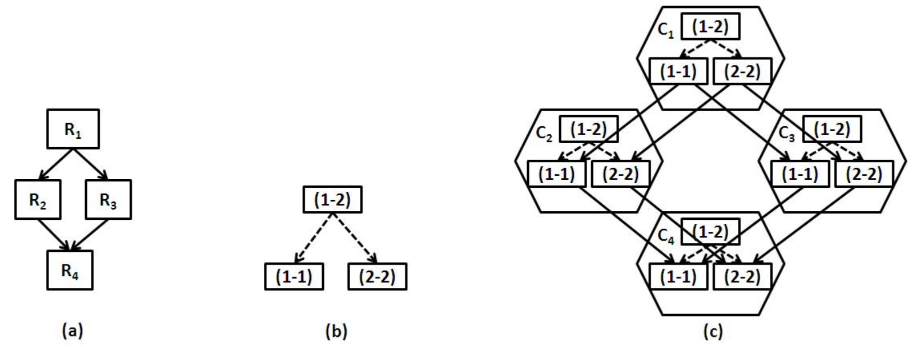


Figure 3.12: (a) User hierarchy, (b) Subscription hierarchy structure, and (c) System subscription hierarchy

3.3.2 Security analysis

In this section, we outline a security proof of the proposed SBHKAS scheme using the modern notion of security, i.e., "key recovery" formally defined as follows.

Definition 11 (Key Recovery (KR)). *A SBHKAS is secure w.r.t. KR if no polynomial time adversary A has a non-negligible advantage (in security parameter τ which defines the cardinality of each secret key used) against the challenger in the following game:*

- *Setup: The challenger sets up the subscription hierarchy, assigns key $K_{(t_a,t_b)}$ to each node with subscription interval (t_a, t_b) in the subscription hierarchy and gives resulting public information (Pub) to the adversary A.*
- *Challenge: Select a challenge time interval (t_a, t_b) with $t_1 \leq t_a \leq t_b \leq t_z$ and send all keys $K_{(t_c,t_d)}$ to the adversary A for each t with $t_c \leq t \leq t_d$ and $t_a \nleq t \nleq t_b$.*
- *Break: The adversary outputs his best guess $K'_{(t_a,t_b)}$ to the key $K_{(t_a,t_b)}$ associated with challenged subscription interval (t_a, t_b).*

The adversary's advantage in attacking the game KR is defined as:

$$Adv_A^{KR} = Pr[K'_{(t_a,t_b)} == K_{(t_a,t_b)}]$$

We say that given $SBHKAS$ is secure w.r.t. KR iff,

$$Adv_A^{KR} < \epsilon_{KR}$$

where ϵ_{KR} is negligible function of security parameter τ.

To carry out the proof, we assume "preimage resistance" property of secure keyed hash functions [59], formally defined below.

Definition 12 (Preimage resistance (h_{pre})). *Let $H = L \times K \rightarrow Y$ be a hash-function family and* A *be an adversary. Then preimage resistance advantage of* A *with respect to a hash function $h \in H$ is defined as follows:*

$$Adv_h^{h_{pre}}(A) = Pr[L \overset{\$}{\leftarrow} \{0,1\}^\tau; K \overset{\$}{\leftarrow} \{0,1\}^\tau; Y \leftarrow h(L,K); K' \leftarrow A(L,Y) : h(L,K') = Y]$$

We write $M \overset{\$}{\leftarrow} S$ for the experiment of choosing a random element from the distribution S and calling it M. τ defines the cardinality of M, i.e., number of times the experiment executes. $A()$ is the adversary's algorithm used by A, which outputs K' as its best guess for an unknown K, given L, $Y(= h(L,K))$ and $h()$ such that $h(L,K') = Y$.

A function $h()$ is said to be preimage resistance secure if the following is true.

$$Adv_h^{h_{pre}}(A) < \epsilon_{pre}$$

$$or,\ \ Pr[A(h(),L,Y) == K] < \epsilon_{pre}$$

where, ϵ_{pre} is negligible function in security parameter τ.

Theorem 3.3.1. *Proposed scheme is secure w.r.t. key recovery against static adversary provided h_{pre} assumption hold.*

Proof sketch. To prove the security of proposed scheme w.r.t. key recovery, we want to model all the information available to the adversary in advance. To achieve this, the game described in Definition 11 is played between the adversary and the challenger. Security of the scheme is bounded to arbitrary but fixed value in terms of security parameter.

Let $G = (V,E)$ be some subscription hierarchy with z time slots (i.e., t_1,...,t_z) where V is the set of nodes and E is the set of edges in the hierarchy. Let a subscription interval (t_a, t_b) in the hierarchy and let, A be a static polynomial time adversary attacking the node u_t with time interval (t, t) with $t_a \leq t \leq t_b$. Now, based on Definition [key recovery], A is having secret information corresponding to all unauthorized access nodes with subscription interval (t_c, t_d) such that there does not exist a time slot t with $t_a \leq t \leq t_b$ and $t_c \leq t \leq t_d$.

We consider three cases (given below) where $|(t_a, t_b)|$ defines the number of time slots in subscription interval (t_a, t_b). The first two cases are special cases and the third case is general case.

Case 1: $|(t_a, t_b)| = z$, i.e., $t_a = t_1$ and $t_b = t_z$.

Case $2(a)$: $|(t_a, t_b)| = z - 1$ with $t_a = t_2$ and $t_b = t_z$.

Case $2(b)$: $|(t_a, t_b)| = z - 1$ with $t_a = t_1$ and $t_b = t_{z-1}$.

Case 3: General case with $|(t_a, t_b)| < (z - 1)$.

We show that all possible cases can be identified as one of the three cases. Later, we prove that KR adversary A has a negligible advantage in finding the key of any node in the hierarchy as described one of the cases above and hence the overall maximum advantage of A is negligible. The detailed proof is given in Appendix B.

3.4 Comparison with related schemes

In this section, we compare the proposed scheme with other similar existing schemes in the literature.

In our scheme, each node in the subscription hierarchy excluding leaf nodes has at most two outgoing edges and each edge has one associated public edge value. Hence, there are at most $z(z - 1)$ public edge values. The nodes in lower half levels of the subscription hierarchy are having dependent keys and hence each such node has one incoming edge which does not have associated public edge value. The number of nodes (X) in lower half levels is computed below,

$$= \# \, nodes \; in \; full \; hierarchy - \# \, nodes \; in \; upper \; half \; levels \tag{3.14}$$

$$= (1/2)z(z+1) - (1/2)(z/2)((z/2)+1) \tag{3.15}$$

$$= (1/8)z(3z+2) \tag{3.16}$$

Hence, out of $z(z-1)$ edges, up to $(1/8)z(3z+2)$ (nodes in lower half levels of subscription hierarchy) nodes have dependent incoming edges without public edge value. Hence, a total of $z(z-1) - (1/8)z(3z+2)$ public edge values are required.

In our scheme, each key in the subscription hierarchy is computed with a single key (e.g., K_s). Hence, only one key is required with CA to derive all other keys. In [36, 58], since keys are independently assigned to the nodes in the subscription hierarchy, there are many nodes whose parent does not exist (are root nodes). A key for each root node must be stored at CA. We can see in their hierarchy that at least upper half of the levels do not have parents (hence are root nodes. Therefore, $(1/2)(z/2)((z/2)+1) = (z/8)(z+2)$ keys must be stored at CA. In [35], since using root node key, CA can derive every key in the hierarchy, it requires only one key to the store. In [34], there are two levels. The upper level will have a node corresponding to each subscription interval with more than one time slots. In another way, it includes all nodes in the considered subscription hierarchy other than leaf nodes (nodes in upper $z-1$ levels) i.e., $(z/2)(z-1)$.

Table 3.4 compares the cost associated with a subscription hierarchy in the existing $SBHKAS$. Ateniese et al. [34] scheme require one decryption operation for key derivation with an expense of huge ($O(z^3)$) public storage. Atallah et al. base scheme [35] requires at most z hash operations as a key derivation. When using $log\, z - Hop$ shortcut edge scheme, Atallah et al. improved scheme reduces key derivation cost up to $2\, log\, z$ with an expense of additional ($> z^2$) public edge values. Crampton [36, 58] was able to reduce key derivation cost up to $log\, z$ without using any additional (shortcut) public edge value by using independent keys. In our scheme, we use dependent keys and are able to further reduce public storage by a factor of $(1/8)z(3z+2)$. Key derivation cost in our scheme is similar (at most $log\, z$ steps) to [36, 60], as a user can jump half of the existing levels towards target leaf node in every step.

Table 3.4: Comparison of single key $SBHKAS$

Scheme	Public edge values	Secret storage at CA	Key derivation cost
Ateniese et al. scheme [34]	$z(z-1)(z+4)/6$	$z(z-1)/2$	1 decryption
Atallah et al. base scheme [35]	$z(z-1)$	1	z
Atallah et al. improved scheme [35] with $log\, z - Hop$ scheme	$> z(z-1) + z^2$	1	$2log\, z$
Crampton scheme [36]	$z(z-1)$	$> z/8(z+2)$	$log\, z$
Our proposed Scheme	$z(z-1)- \lceil(1/8)z(3z+2)\rceil$	1	$log\, z$

Note that allowing more number of outgoing edges to a node in subscription hierarchy will reduce key derivation cost. There is a trade-off between outgoing edges to a node and key derivation cost. In our scheme, if we allow $log\, z$ outgoing edges to a node (with the same spirit as in [60]), key derivation cost will be reduced to $log\, log\, z$ as in [60]. Public storage cost in our construction will be still less with a factor of $(1/8)z(3z+2)$ since these number of nodes are still require an incoming edge with dependent key derivation, i.e., without any public edge value.

3.5 Dynamic operations

In this section, we discuss two operations: add new subscription and subscription withdrawal.

Add new subscription

Adding new subscription is simple and straight forward. When a new user is subscribed for a subscription interval or an existing user is extending his subscription in the system, CA will give the respective subscription node's key from the subscription hierarchy to the user. Using received new subscription key, the user can derive all the encryption keys in his subscription interval.

Subscription withdrawal

Suppose a user with initial subscription interval (t_a, t_b) wants to withdraw his subscription for time interval (t, t_b) where $t_a \leq t \leq t_b$. Let node v represent (t_a, t_b) and node w represents (t_a, t) in the subscription hierarchy. In case $t_a = t$, w is $Null$ represents full subscription withdrawal. After a user withdraws his subscription, CA will call procedure $Subscription\_Withdrawal(v, w)$ (given in Algorithm 13). It re-assign keys to a subset of nodes in the subscription hierarchy such that revoked user will not able to access revoked data blocks.

Algorithm 13 $Subscription\_Withdrawal(SH, v, w)$

Input: Given subscription hierarchy SH, user's old subscription node v and new subscription node w (after subscription withdrawal).
Output: Keys are assigned to a subset of nodes in SH.

```
1: $l_v$ = compute and assign new random label
2: $update\_parents(SH, v)$
3: $update\_children(SH, v, w)$
4: **for** each updated key $K_u$ associated with leaf node $u \in lower(v)$ **do**
5:     **for** each resource $r$ associated with $u$ **do**
6:         Encrypt resource $r$ with key $K_u$
7:     **end for**
8: **end for**
```

Algorithm 14 $update\_parents(SH, v)$

Input: A subscription hierarchy SH and a subscription node v.
Output: v's incoming relationship with its parents will be updated.

```
1: **for** each immediate predecessor node $v'$ of $v$ **do**
2:     **if** $r_{v',v} \in Pub$ **then**
3:         $r_{v',v} = h(K_{v'}, l_v) \oplus K_v$
4:     **else**
5:         $K_v = h(K_{v'}, l_v)$
6:     **end if**
7: **end for**
```

Algorithm 15 *update_children*(SH, v, w)

Input: Given subscription hierarchy SH, old subscription node v and new subscription node w (after subscription withdrawal).
Output: v's incoming relationship with its parents will be updated.

```
1: $v_1 = leftchild(v)$
2: $v_2 = rightchild(v)$
3: if ($r_{v,v_1} \in Pub$) and ($v_1 \notin lower(w)$) then
4:     $l_{v_1}$ = compute and assign new label
5:     $update\_parents(SH, v_1)$
6:     $update\_children(SH, v_1, w)$
7: else if ($r_{v,v_1} \in Pub$) and ($v_1 \in lower(w)$) then
8:     $r_{v,v_1} = h(K_v, l_{v_1}) \oplus K_{v_1}$
9: else
10:    $K_{v_1} = h(K_v, l_{v_1})$
11:    $update\_parents(SH, v_1)$
12:    $update\_children(SH, v_1, w)$
13: end if
14: Repeat if clause by replacing $v_1$ with $v_2$
```

3.6 Summary

A subscription hierarchy is used as a key management hierarchy for time-limited access control systems. SBHKAS assigns keys to a subscription hierarchy. Existing *SBHKAS*s will have a trade-off between private storage requirement by a user, system public storage requirement, and key derivation cost. We have proposed a SBHKAS using dependent keys with only one secret key per user. The comparison with similar existing schemes shows that the proposed SBHKAS further reduces the secret storage cost at data owner and system public storage without increasing other costs such as secret storage per user and key derivation cost. A formal security proof of the proposed scheme is given against a static key recovery security adversary assuming that the pre-image resistance property of a secure hash function holds. A limitation of our proof is that it does not consider dynamic operations such as re-keying operation.

CHAPTER 4

Enhancing write integrity and data freshness

Controlling write access to outsourced data is more challenging as compared to read access because of two major reasons. First, even a single unauthorized write operation may heavily ruin the data owner's business. Second, multiple users may try to update a data item at the same time that may lead to an inconsistent view of the data among readers. Existing works on secure write access with a malicious service provider do not consider security against users who have written the outsourced data files. Also, they consider weak freshness guarantee, i.e., the recipient only knows that the current message is recent than the previous. It suited to applications which do not require strong freshness guarantee such as Twitter updates. In this chapter, we address and analyze the above security issues so that even a user who has written an outsourced data files should not be able to modify the file after a given amount of time. A strong freshness guarantee is addressed so that a read request will return the latest version of the data file which helps in handling consistency issues involved in concurrent write operations.

4.1 Introduction

To enable secure write access to outsourced data, we consider a data owner who has outsourced its data at a malicious but cautious CSP (as discussed in 1.2). Now, consider that a write authorized user in collusion with the CSP can modify his own written outsourced data files until next update is written, without being detected by the data owner. Modifying a file here means altering the content of the file without creating a new version. This is a matter of concern even if the user is an authorized writer. For example, in a cloud-based daily e-newspaper publishing system, the news is created by staff writers (write authorized users) and uploaded to a third party storage service provider. Once the news articles are published, the e-newspaper CEO will not allow any modification to the published content. However, it may happen that the published news article has some unauthenticated content which may lead to some legal action or embarrass the newspaper. Meanwhile, let the staff writer collude with the service provider and update the published content at the server to avoid any legal consequences. Now nobody including the CEO can frame any charges against the staff writer as outsourced news is modified.

Similarly, in the case of user's access right revocation, i.e., the revoked user can collude with the CSP and modify their latest written version (s) of a resource whose access right is now revoked.

The existing schemes for write access control ([42, 43, 15, 12]) aim to restrict a user from modifying the outsourced data files for which it does not have write authorization. However, we found that there is still a scope to modify the self-written data files in the existing schemes. For example, consider a resource r whose last version v_i was written by a user u at time t. Now, until a new version v_{i+1} is written, the user u can modify it at the server by colluding with the CSP. It is possible even when the access permission for resource r is revoked from u. Also, if u has written the latest contiguous sequence of versions $v_j, v_{j+1}, ..., v_i$, it can modify any number of versions in that sequence. In what follows we summarize their two shortcomings.

1. A write authorized user in collusion with the CSP can modify his own written latest sequence of data records until a new version is written by another user.
2. A user whose access right is revoked, in collusion with the CSP, can still be able to modify his latest written version (s) of revoked resource (s).

In above, the *latest record* of a resource refers to the last written version of the resource. Similarly, the latest sequence of records refers to the recent contiguous chain of versions (including the last version) of the resource.

In our model each data file update operation works as follows: a data file is first read by an authorized user who will then update the file locally and send the updated version back to the server for storage. We define two types of transactions: *Get* and *Put*. Both will be initiated by the authorized users. A transaction is a sequence of operations each treated as a unit such as read and write. In Get transaction, the user will first send a read request to the CSP which verifies the request and responds back with the requested resource, if the request is valid. Else respond with "invalid request" message. In Put transaction, the user will first request the recent resource version and store it locally. Next it creates a new updated version and sends it to the CSP. Upon receiving the new data version, the CSP will authenticate the user and verify the integrity of the received version. If verified correctly, the new version is updated at the server and an acknowledgment is returned to the writer. A Get transaction is said to be committed when a reader receives the requested data file. A Put transaction is said to be committed only when a writer receives an acknowledgment.

As another requirement if the staleness is maliciously injected by any misbehavior party (including the CSP), it must be caught by the data owner. For example, a small delay in getting the current status of seat allocation in a railway reservation system may restrict you from getting confirmed seat reservation. Therefore, it is desired to minimize the read staleness (or improving freshness guarantee) in a data access system.

In the following section, we give some preliminaries used in the rest of the chapter. In Section 4.3, we review the existing schemes with respect to the security of write access against users who has written the outsourced data files. We highlight their shortcomings and give appropriate countermeasures. The modified system is implemented on Azure platform. Section 4.4 address the stronger freshness property as compared to the existing ones. Section 4.5 will discuss the related work on write access control over outsourced data. Section 4.6 summarizes this chapter. The notations used in this chapter are shown in Table 4.1.

Table 4.1: Notations and abbreviations used

Notation	Description
$h()$	Cryptographic hash function
ver_i	Version of resource i
$ver(K)$	Version of secret key K
(Pr_u, Pb_u)	Private and public key pair of entity u
ID_o	Identifier of resource o
CH_i	Chain hash corresponding to resource's version v_i
$rec_{o,i}$	Data record for j^{th} version of resource o

4.2 Preliminaries

4.2.1 A resource record

For proper maintenance and security reasons, the encrypted data file is stored along with various other attributes. We call such individual group as a record. A general record structure is shown below. In the given record structure for a resource o, ID_o represents the resource's identifier, ver_i is the version number, $E_K(o_i)$ represents the encrypted current resource version o_i and $ver(K)$ is the version number of encryption key K. MAC ([61]) is message authentication code computed over the described fields and is used for checking integrity of the record.

Table 4.2: Outsourced resource record structure proposed by Popa et al.

ID_o	ver_i	$E_K(o_i)$	$ver(K)$	Writer ID	MAC	...
resource id	version of resource	encrypted resource	version of key K		integrity verification code	

4.2.2 Chain hash

A malicious CSP can insert a file as old version, delete an old version, and change a resource version sequence. As a solution, we required a guarantee that an updated version i is always written over version $i-1$. Chain hash creates a strong binding between every pair of contiguous record versions so that no unauthorized user (including the CSP) can change the sequence of versions, insert or delete a version from in-between. For example, consider a resource with versions $v_1, ..., v_i$ with the corresponding chain hashes $CH_1, ..., CH_i$ where $CH_j = MAC_K(j_{th}$ *resource information* + $CH_{j-1})$ and K is symmetric key known to the writer who has written the updated record. K is also known to the data owner (who generates it) who can verify the chain hash during auditing process (see Section 4.2.3). Since K is only known to the writer and the data owner, no other user including the CSP can modify the outsourced resource and create the corresponding modified chain hash using K.

4.2.3 Auditing

Write integrity property ensures that only authorized writers can modify or update the outsourced content. Chain hash can prevent a malicious user (including the untrusted service provider) from altering the sequence of existing resource versions. However, the malicious CSP in collusion with an unauthorized user who has written the resource can still overwrite its latest updated version (s). One of the possible solution to address this issue is by using "audit" mechanism executed by the data owner that verifies any unauthorized outsourced data modification [15]. It is assumed that no user except the data owner is allowed to delete any outsourced data record. If data deletion is allowed by such users then to prove any unauthorized data modification may become difficult to verify, without the knowledge of old (deleted) records. Popa et al. [15] suggest time-bound auditing of outsourced data. System time is divided into epochs. Auditing is done at the end of each epoch. A general auditing mechanism is described in Algorithm 16.

In the algorithm, the data owner selects a random set of resource identifiers with version numbers to be audited and send them as a read request to the CSP. The CSP will return the corresponding records back to the data owner. The data owner will then verify the chain hash by re-computing it against each received record. If every chain hash in the received record set is correct then the algorithm returns "Success" else it returns "Failure". In case the audit process returns "Failure", the data owner will know that the integrity of the corresponding record is

Algorithm 16 *Audit*()

```
1: DO → CSP: Generate and send a random set of resource ids with version
   numbers to be audited.
2: for (each entry in the set) do
3:     CSP → DO: Sends the corresponding resource record
4: end for
   /* DO will do the following */
5: for (each received resource record) do
6:     Verify the chain hash by re-computing it
7:     if (Chain hash is correct) then
8:         Return “Success”
9:     else
10:        Return “Failure” /* If verification fails, appropriate action will be taken
           */
11:    end if
12: end for
```

beached and will take the appropriate action.

The above mechanism can detect the integrity misbehavior by the CSP but cannot prove in the court of law that the CSP misbehaved. To frame charges against the CSP, the data owner needs a concrete proof mechanism so that it can frame charges against the CSP in case the CSP is involved in the misbehavior. On the other hand, the CSP also required a shield against any wrong blame against him by the data owner.

A solution to the above discussed misbehavior is proposed by Popa et al. [15] using attestation messages or proof messages. In their scheme, a signed proof message is returned by the CSP for each read/write operation to the user (involved in read/write operation) to assure the correctness of the operation. These proof messages are then checked by the user and forwarded to the data owner. The collected proof messages at the data owner (DO) are then used in auditing process to verify the correctness of outsourced records. Now, if the integrity of the outsourced record is breached the data owner can frame charge against the CSP using the respective proof messages.

A similarly signed proof message is sent by a user while performing a read or write operation. The CSP authenticate the user, verify the proof message and store it along with the record so that at the time of dispute these proof messages can shield the CSP from any wrong charge against him.

Although the proof messages can be used to frame charges against a misbehaving party, it may create a storage bottleneck the data owner. To handle this Popa et al. uses Merkle hash tree for verifying the integrity of stored data. After auditing, the data owner, and the CSP create a Merkle hash value of the entire storage, exchange proof messages ensuring that they agree on the same hash value. Now, all proof messages from that epoch can be discarded. Although the Merkle hash tree significantly reduces the storage at the data owner, one major drawback with such tree computed over dynamic records (updated with time as different versions) is that the tree hash values need to be updated after each record update. Also, the Merkle root hash value needs to be signed (in proof messages) each time by the data owner and the CSP ensuring that both agree with the tree computation.

It must be noted that if CSP is honest then the auditing mechanism is not required. In Popa et al. [15] scheme, priorities are given to the data files so that the data with higher priority is audited more frequently than data with lower priority.

4.3 Audit-based protocols

In this section, we discuss how audit-based protocols can be used for detecting misbehavior by unauthorized writers. A secret symmetric key K_s is assumed to be known only to the data owner. The data owner will verify and attest each of the latest updated records of various resources written in the current time slot with its secret key K_s. For attesting a record, the data owner computes MAC over corresponding chain hash with K_s. The attestation is such that if the integrity of any old committed record of the attested record is breached, it must be caught by the data owner in the detailed audit process given in Algorithm 17. The k in step three of the algorithm describes the number of versions written in the last time slot for a given resource. We assume it is not very large in practice.

Algorithm 17 *Audit*()

```
1: DO $\rightarrow$ CSP: Prepare and send a set of resource ids with version numbers to
   be audited.
2: for (each entry in the set with version number $i$) do
3:    CSP $\rightarrow$ DO: Sends a contiguous sequence of corresponding resource records
      with version number $i$, $i-1$,...,$i-k$ such that $i-k$th versioned record is
      attested by the data owner
4: end for
   /* DO will do the following */
5: for (each received contoguous sequence of resource records) do
6:    Verify each chain hash by re-computing it
7:    if (all chain hashes are correct till the last attested chain hash in the se-
      quence) then
8:       Return "Success"
9:    else
10:      Return "Failure"
11:   end if
12: end for
```

4.3.1 Protocol for enforcing time-limited access

In what follows, *Protocol 1* describes the handling of the first requirement given in Section 4.1.

The protocol for handling the given requirement is described in Algorithm 18. It runs between CSP and the data owner (DO). The goal of the protocol is that even the authorized user cannot modify their own written data records after a pre-defined amount of time, i.e., after the end of the current time slot. To achieve this, the CSP will call Algorithm 18 at the end of each time slot.

In Algorithm 18, let t_i be the current time slot. At the end of time slot t_i, the CSP will create a set S of triplets corresponding to each latest data record version written in the current time slot (Step 2). A triplet corresponding to resource o contains resource identifier ID_o, its latest version number ver_j and corresponding chain hash (CH_j). Then the CSP will compute the hash of set S and t_i (to avoid replay attack). This hash is then signed by the CSP using its private key Pr_{CSP}. The signed hash is used by the data owner as a correctness proof for set S and current time slot. The signed hash and the set S are then sent to the data owner (Step 3). Upon receiving the message, the data owner will first compute and ver-

Algorithm 18 *ControlledWriteAccess*()

```
1: $t_i \leftarrow$ current time slot
   /* Create set S containing triplets <resource id, version number, chain hash (CH)> */
2: CSP creates set $S = \{< ID_o, ver_j, CH_j >\}$ for each latest record committed in $t_i$
3: CSP $\rightarrow$ DO: Send $S$ + $\{h(S, t_i)\}_{Pr_{CSP}}$
4: DO verifies $\{h(S, t_i)\}_{Pr_{CSP}}$. If verification fails, ask the CSP to send it again
5: for (each chain hash $CH_j$ in $S$) do
6:     $C \leftarrow RetrieveChain(ID_o, ver_j)$
7:     if ($CorrectChain(ID_o, C)$) then
8:         DO computes $MAC_{K_s}(CH_j)$, replace it with $CH_j$ in $S$
9:     else
10:        "Verification fails", DO takes appropriate action
11:    end if
12: end for
13: DO $\rightarrow$ CSP: Send signed updated $S$. The CSP will then overwrite each respective chain hash with the received MAC
14: CSP $\rightarrow$ DO: Send "Signed proof message" as an acknowledgment ensuring updates are written
15: DO can now delete old proof messages
```

ify the signature (Step 4). If the signature is invalid, stop the process and request the CSP to send it again. In case it repeats more than few times, an appropriate action can be taken against the CSP about not fulfilling the service agreement. If signature is correct then for each CH_j in S, the DO will do the following: retrieve corresponding latest sequence of records by calling $RetrieveChain(ID_o, ver_j)$ (Algorithm 19) and if it is correct then attest it by computing a MAC over CH_j using key K_s. The newly attested MAC is now replaced with corresponding CH_j in S (Steps $6-8$). In the function call, ver_j is the version number of resource o with id ID_o corresponding to the CH_j. Updated S is now sent to the CSP (Step 13). Upon receiving the updated S, the CSP will update the corresponding records at the server by replacing their chain hashes with received updated MACs (or chain hashes). CSP will now send a signed proof message as an acknowledgment to the data owner ensuring that the updates are successfully written in the cloud store (Step 14). The old proof messages received from the authorized users can be now deleted by the data owner (Step 15). The use of chain hash attestation using key K_s is used to reduce the storage at the data owner.

Algorithm 19 $RetrieveChain(ID_o, ver_i)$

Input: A resource id ID_o and one of its version number ver_i.
Output: It retrieves a sequence of recent records for ID_o up to version ver_i.

```
1: DO→CSP: send Get request with $ID_o, ver_i$
   /* create sequence $C$ containing latest records $rec_{o,j}$ of resource $o$ and version $j$ */
2: CSP creates and initialize set $S = \{rec_{o,i}\}$
3: $k \leftarrow i$
4: while ($rec_{o,k-1}$ and not $MAC_{K_s}(CH_{k-1})$) do
5:     $C = C \cup \{rec_{o,k-1}\}$
6:     $k \leftarrow k - 1$
7: end while
8: if ($rec_{o,k-1}$ and $MAC_{K_s}(CH_{k-1})$) then
9:     $C = C \cup \{rec_{o,k-1}\}$
10: end if
11: CSP→DO: $C$
```

Algorithm 20 $CorrectChain(ID_o, C)$

Input: A resource id ID_o and a sequence of its recent records C.
Output: It verifies each element in C by recomputing it and returns a true or false value accordingly.

```
1: for (each record $rec_{o,l}$ in $C$) do
2:     DO verifies the $CH_l$ by recomputing it and comparing it with the stored
       proof messages in the current time slot
3: end for
4: Return “True” if all chain hashes are correct, else return “False”
```

The $RetrieveChain()$ method allows the data owner to retrieve the chain (i.e., C) of given resource' id (ID_o) and version number (ver_j) from the CSP. The chain $C = \{rec_{o,k}, ..., rec_{o,i}\}$ is such that the CH_k associated to record $rec_{o,k}$ with version ver_k has MAC'ed with key K_s. Also, there is no CH_l exists with $k + 1 \leq l \leq i$ and MAC is computed with K_s.

Using $CorrectChain()$ method (Algorithm 20), the data owner verifies the given chain hash of a given resource. If correctly verified then the algorithm returns “Success' else it returns “Failure”.

In case the CSP does not send the latest version of a resource to the data owner, it will be caught by the data owner. There are two possible cases: only CSP misbehaves or the CSP will collude with the writer. In the first case since the writer is trusted, the timestamp will ensure the time of resource update. If the resource was written in the previous time slot, the CSP will be caught by looking at the timestamp. If the resource is written in current time slot but is not the latest version then also it will be caught. It is because at the end of each time slot the latest version must be attested and an intermediate version is only attested if corresponding access right is revoked. It means if the latest version of a resource is not attested at the end of the current time slot, it cannot be attested later and will be caught easily by the data owner. If the writer colludes with the CSP and will modify the timestamp then also it will be caught by the data owner. It is because the timestamp is captured in the chain hash of next version written for that resource. If there is no new version written then it will be treated as a new version.

4.3.2 Protocol to defend against revoked access

In what follows, *protocol 2* describes handling of *requirement 2* given in Section 4.1. In the protocol, immediately after a user's access right is revoked, the data owner will attest each of his latest updated records written in the current time slot. It ensures that the integrity of revoked resource will be intact even if the CSP colludes with the writer of the resource.

To handle the given requirement, we propose a protocol described in Algorithm 21. The goal of this protocol is that a user whose access right is revoked for a resource (has written previously some latest records for the resource) cannot be able to modify their own written records, even if colluded with the CSP. If a user u is revoked from accessing resource r (with identifier ID_r), the data owner will call procedure $RevokeAccess(u, ID_r)$ (i.e., Algorithm 21).

In $RevokeAccess()$ method (Algorithm 21), the data owner will first send the read request for resource r to the CSP (Step 2). If the latest version of the resource r is written in the current time slot t_i, the CSP returns the corresponding record (Steps $3-4$) else return "no such record found" (Step 12). If the received record was written by the revoked user, the data owner will compute MAC over it using secret key K_s, and send it back to the CSP (Steps $5-7$). The CSP will update the received chain hash into corresponding resource record and return a signed proof

Algorithm 21 *RevokeAccess*(u, ID_r)

Input: A user u and resource ID_r to be revoked from u.
Output: The authorization of resource ID_r is revoked from user u.

```
1: $t_i \leftarrow$ current time slot
2: DO $\rightarrow$ CSP: Send $ID_r$
3: **if** (latest $ver_j$ of resource $r$ is written in $t_i$) **then**
4:     CSP $\rightarrow$ DO: Send latest record $rec_{r,j}$ for $ID_r$
5:     **if** ($rec_{r,j}$ is written by $u$ and $CorrectChain(ID_r, RetrieveChain(ID_r, ver_j))$) **then**
6:         DO computes $X \leftarrow MAC_{K_s}(CH_j)$
7:         DO $\rightarrow$ CSP: Send $X$
8:         CSP will replace $CH_j$ with $X$
9:         CSP $\rightarrow$ DO: Send "Signed proof message" as an acknowledgment
10:    **end if**
11: **else**
12:    CSP $\rightarrow$ DO: " no new record found"
13: **end if**
```

message as an acknowledgment back to the data owner (Steps $8-9$). In case the CSP maliciously allows the revoked writer to overwrite the existing record, the data owner can frame charge against the CSP through the proof message. When a new version is written by another user, the proof message can be destroyed by the data owner.

The two shortcomings given in Section 4.1 are formalized together as follows. In the definition below, the unauthorized user refers to untrusted cloud service provider, authorized users who have written some historical data and the revoked users.

Definition 13 (Collusion-secure)**.** *A system is called collusion-secure if even in the presence of two or more unauthorized users collude together, the secret keys known to all the users does not contain any key that can be used in generating secrets that will allow any unauthorized write access in the system.*

Claim 2. Protocol 1 *and* Protocol 2 *are collusion secure.*

Proof. Upon receiving the updated sequence of triplets in *Protocol 1*, the CSP will update these chain hashes in the corresponding records. Now since chain hash of every latest resource version written in the current time slot is created using data owner's secret key K_s, no user including the writer of the resource itself can modify the resource. With the reasoning as in *Protocol 1*, in *Protocol 2* since MAC is computed over latest version' chain hash using data owner's key, the revoked user cannot be able to modify its own written records. The above two protocols are secure even if the unauthorized users will collude (Definition 13) because the secret key (K_s) used in the computation of MAC is only known to the data owner. If the chain hash in last versioned record of a resource in a time slot is not MAC'ed with K_s, the CSP is caught by data owner for violating the mutual service agreement. □

4.3.3 Experimental evaluation

We implemented the proposed write access control protocols on Microsoft's Windows Azure [62] cloud platform using Java. In the implementation, we are not focusing on Azure storage performance since the proposed system can be implemented on most of the cloud storage systems. The implementation, in particular, focuses on cryptographic models used in the system to achieve different security requirements.

The implementation consists mainly of three entities: a client machine in user premises, two virtual machines. One virtual machine works as a data owner and other works as a cloud server. The client machine consists of an Intel core 2 quad processor 2.66GHz with 3GB RAM and 16MB Cache. We choose AES-128 as the cipher for file encryption and employ MD5 as the hash function. The cloud server stores 1000 records whose size varies from 100KB to 200KB.

Figure 4.1 shows the latency chart with respect to different numbers of write requests given to the cloud service provider. Major processing time is due to the cryptographic operations performed at the server and rest of the time is used in to and fro communication between the client machine and server machine.

Figure 4.2 shows the performance of data owner machine for computing MACs over received chain hashes in Protocol 1. The execution cost of one received message with 10^7 pairs is 78.2 seconds. It includes computation of one MAC for message integrity, verification of authorization certificate and 10^7 MAC computations

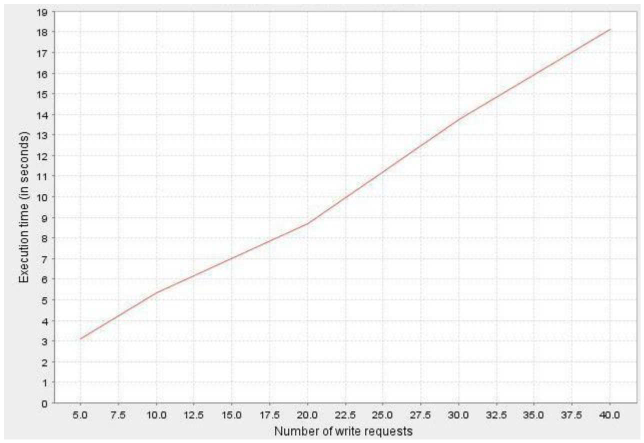


Figure 4.1: Write operation latency

over received chain hashes from the CSP.

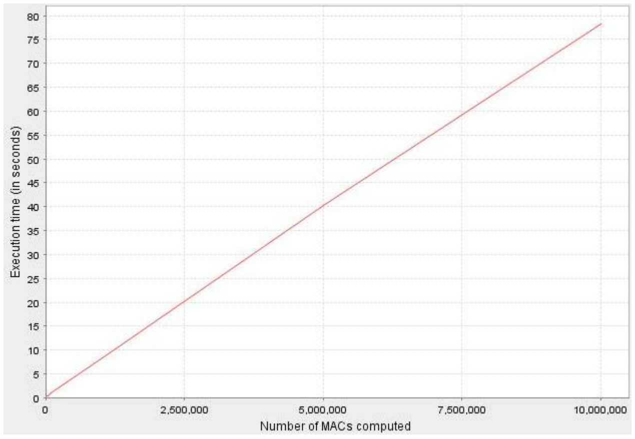


Figure 4.2: Data owner performance for computing MACs over chain hashes

Figure 4.3 analyses Protocol 2. The protocol requires two communications between the data owner and the CSP. In the first communication, the data owner sends a list of users IDs to the CSP whose write access has been revoked and receives a list of the corresponding chain hashes from the CSP. The data owner then computes the MAC over each received chain hash and sends them back to the CSP, who will then update each corresponding record and return an acknowledgment back to the data owner. In the implementation, we consider three sets of revoked users, i.e., 10, 100 and 1000. Since a revoked user can have multiple resources to be revoked, we consider 10 and 100 number of MACs are computed by the data owner. The maximum time (approx 70%) in the operation is due to communication between the two parties. Therefore we can see from the figure that time is slowly increasing with the increase in a number of revoked users. It may be noted here that in general the number of revoked users in a time slot and their respective latest updated versions (in the last time slot) are limited in number.

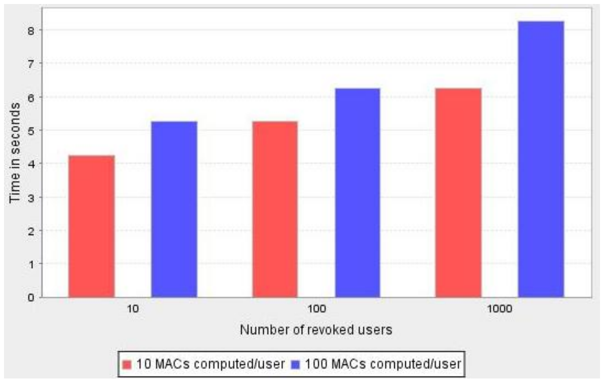


Figure 4.3: Implementation cost for Protocol 2

4.4 Freshness guarantee

If the staleness is maliciously injected by any misbehaving party (including the CSP), it must be caught by the data owner. For example, in a stock market application, a small delay in updating the bid price can cause a huge loss to a bidder. Similarly, a small delay in getting the current status of seat allocation in a railway reservation system may restrict you from getting a confirmed seat reservation. Therefore, it is desirable to minimize the read staleness (or improve freshness guarantee) in a data access system.

In order to guarantee the freshness of the received data to a user, in an ideal situation, the CSP needs to give a freshness proof (along with the data item) that will ensure the user that received data is fresh as the last update. However, there is a delay involved in sending and receiving the proof message.

4.4.1 Existing notions

Golab et al. [63] define two interpretations for staleness: version-based staleness and time-based staleness. In the first type, a read returns any of the k last updated versions (called k-atomicity [64]). In latter, a read returns the value that is at least

t time units stale (called delta-atomicity). In real systems, t depends on write latency and propagation delays. Bailis et al. [45] combine the two above metrics and define another probabilistically bounded Staleness, i.e., $< k, t >$-staleness. It ensures that a read request will return one of the last k updated versions provided we wait t seconds after the last version was written. Suppose the last 3 versions are v_1, v_2 and v_3. Now, $< 3, 4 >$-staleness ensures the returned record will be one from the last 3 (i.e., v_1, v_2, v_3) versions provided the read request was made within 4 seconds after the last record was written (i.e., v_3).

At the time of read request to resource (o) in Popa et al. [15] scheme, a user sends the resource ID (i.e., ID_o) along with a (fresh) random nonce N to the CSP. The CSP authenticates the user and return the requested resource information o_i along with a proof message.

$$Proof\ message = \{h(ID_o, ver_i, ver(K_o), h(o_i), CH_i, N)\}_{Pr_{CSP}}$$

Where $h()$ is secure hash function, ver_i is the resource version number, $ver(K_o)$ is the version number of resource encryption key K_o and o_i is the current ith resource. The presence of N in the proof message only assures that the message is computed afresh. It gives version-based staleness. In [44, 12], the timestamp (τ_i) stored securely with each record (i.e., $CH_i = MAC_{K_u}(o_i || CH_{i-1} || \tau_i)$) written by user u with secret key K_u. It captures the time when the record was written.

In the previous schemes ([44, 15, 12]), the CSP can still give stale resource in response to a read request without the misbehavior being detected by the data owner. This is because the proof messages do not guarantee that there is no record written between the time when the returned record was written and the time when the resource is sent to the user. Therefore, we are expecting that there must be a commitment message that will guarantee "the returned resource version will be the latest committed version at least till the time when the data record is sent by the CSP to a reader". However, while working with distributed cloud servers we need to give some practical interpretation to the staleness definition. A genuine issue is that there are delays in updating replicas; therefore one possible excuse from the CSP for such delays is that the requested server is disconnected (for a time being) from other servers. Now, the challenge is that can we give a better or improved staleness (or freshness) guarantee?

4.4.2 Improved notion: $< v, d, t >$-staleness

We propose a stronger interpretation to the staleness property which we call $< k, t >$-staleness [45]. It ensures that a read request which begins t time units after the write commit operation returns one of the last k updated versions.

Definition 14 ($< v, d, t >$-staleness)**.** *A system guarantees $< v, d, t >$-staleness if it assures that the version v of resource d is the latest updated version at least until time t, where t is the time when the resource was dispatched by the CSP.*

The $< v, d, t >$-staleness definition says that its respective freshness proof will ensure that the data version (i.e., v) returned is fresh as the last update at least till the proof creation time (i.e., t). In distributed scenario, no server can give such a proof since the write operation on a resource can be performed at any server. However, to protect against stale read, staleness sensitive applications require such concrete proofs so that the CSP's misbehavior is caught by the data owner during the auditing process. To achieve the stronger freshness requirement, we assume there is a dedicated write server for each resource, i.e., in the system's view, each resource is updated at only one dedicated server by the CSP and replicas are then updated. This assumption is similar to the one used by Zellag et al. [65] and by Akal et al. [66], where all resources are stored at a central database server. Every resource is first written at the central server then replicas are updated. In contrast, in our model, there is more than one such dedicated server, and each server is responsible for updating a distinct set of resources. It may happen that the CSP may not do the write updates on dedicated servers. These servers need to provide freshness guarantees in the form of proofs to the readers whenever asked. Therefore, if the CSP does not follow the required agreement, it will be exposed during auditing process with the help of proof messages (collected at the data owner from various users). It is to be noted here that the use of dedicated server may degrade the Get and Put transactions costs due to the increase in communication cost.

The dedicated server can only generate proofs for its associated resources. Upon a read request, the proof message is computed as follows.

$proof\ message = \{h(t_{curr},\ resource\_info,\ t_w)\}_{Pr_{CSP}}$

Where, t_{curr} is the current timestamp indicating the time when the proof was created, t_w is the timestamp indicating the time when the resource was written, and $resource\_info$ contains the t_w along with other resource record information. The proof message is sent to the read authorized user along with t_{curr} and resource information. It assures the user that there is no record written in-between the proof creation time and the time when the returned resource was written. If later any message is found to be written in this time interval, the proof message can be used as a proof of staleness.

A system that ensures $< v, d, t >$-staleness is called $< v, d, t >$-staleness secure. An informal proof for the staleness security property is given below.

Claim 3. *The proposed mechanism is $< v, d, t >$-staleness secure.*

Proof. The proof message by the CSP binds the committed time t_w of the resource version along with the time t_{curr} when the proof is created. Such proofs are collected at the data owner and are used while auditing. Since the proof message was signed by the CSP with its private key and created by a single dedicated server, it cannot later deny about the commitment given in the proof. In case the server tries to fool a reader by giving a stale data, the corresponding proof messages collected from authorized users must differ.

Suppose the version number of received resource record is v_i and its commit timestamp is t. While auditing, the data owner will check the timestamp (t') of version v_{i+1} (if exists). If $t' < t_{curr}$ (present in the proof message), the record was stale else it was fresh. In case v_{i+1} does not exist, the record is assumed to be fresh. Since the timestamps in data records are used in computing MAC with writer's secret key, CSP cannot forge it. Hence, no unauthorized user (including the CSP) can give a stale record to an authorized user without being detected by the data owner. $\square$

4.5 Related work

Early cryptographic file system with untrusted servers proposed by Cattaneo et al. [67] and Miller et al. [68] provide data integrity by storing a hash value and digital signatures for each data file. In SiRiUS [44] every user has one signature key and one asymmetric key for file encryption. Every file has two parts: the data part and meta-data part. Meta-data (contains access control information) file is continuously signed by the user with updated timestamp, for freshness. SUNDR [42] implements the property named "fork consistency" which guarantees that users can detect any integrity or consistency violations as long as they see each other's file modifications on the server. They use signed message called update certificate generated by a user to handle concurrent updates. This will lead to waiting for users until another user will finish their access. Also, the consistency is achieved by adding an extra user to user communication. There are schemes which provide high-level integrity for cloud-based file system, i.e., Proofs of Retrievability (PoR) [22, 69]. Stefanov et al. [69] proposed dynamic PoR that continuously monitors the operation of cloud storage service and obtain strong guarantees about the correctness and availability of entire stored file system. It is enabled through an auditing protocol that continuously monitors the correctness and availability of the entire file system. At a low level, MAC's are used for ensuring data integrity. To ensure freshness, it is necessary to authenticate not just data blocks, but also their versions. Freshness is ensured using MAC for each file block binding it with the corresponding unique version number and is updated every time the block is updated. In this work, we are however considering low-level integrity and strong consistency semantics for a malicious storage system.

4.5.1 Write-serializability

SUNDR [42] implements a weaker property named "fork consistency". To overcome the fork attack, a strong binding among the history of updates for a resource is needed. It is implemented using a user's signed message that binds together all the updates into a single hash (i.e., the i-handle using Merkle hash trees [70]) along with the latest version of every other i-handle. While a file update operation, the user acquires the global lock and access the latest version structure for each system user from the server. This set is called version structure list (or VSL). The user then updates its version structure by modifying its i-handles and/or the version numbers according to the (received) current state of the file system. A user verifies the received VSL and compares it with stored VSL. If each version structure in received VSL is no lesser than the corresponding version structure in stored VSL then update the stored one (assuming it is consistent). Now sign the updated version structure (the signed message) and sends it to the server. The server adds this updated structure to the VSL and deletes the old entries for updated i-handles. Now the user will release the file system global lock.

For concurrent updates, the user pre-declares a fetch or update operation to the server before receiving the VSL. It is done using signed messages called update certificates that contain user's next version number (next to the one which is in the stored VSL), a hash of user's VSL entry and a list of modifications to be performed. The user then sends the update certificate to the server which replies with the VSL and a list of all pending operations not yet reflected in the VSL (by other users). It is to be noted here that while signing an updated certificate, a user does not predict the version vector of its next version structure since it may depend on concurrent operations by other users.

Shraer et al. [43] addressed a stronger write-serializability property by taking help of a verifier that will verify it using client-to-client communication. The verifier stores linked or chain hash for each resource, i.e., the latest hash includes all previous versions information (in contrast to [42]). Therefore the untrusted CSP will not able to modify old resources since every latter resource is linked with previous versions. Also, each client updates others by sending consistency notification messages that contain the maximal version number. If one of the checks fails then the client broadcasts a failure message among other clients.

Write-serializability property can be addressed by storing a secure message such as a "chain hash" with each record [15, 12], as described earlier.

Although the existing schemes defend against unauthorized modification of outsourced records written by other users, a write authorized (or revoked) user in collusion with the CSP can modify his own written latest sequence of data records until a new version is written by another user.

4.5.2 Freshness

In Popa et al. scheme [15], at the time of resource read request, the authorized user sends the requested resource ID along with a random nonce N to the CSP, as shown in Figure 4.4. The CSP authenticates the user and returns the requested resource information along with hashed and signed "CSP get attestation" message. The "CSP get attestation" is computed by the CSP over returned resource information and the N. The nonce value assures that the attestation is computed afresh.

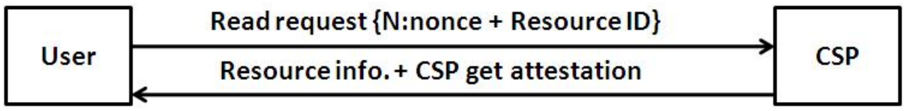

User
Read request {N:nonce + Resource ID}
Resource info. + CSP get attestation
CSP

Figure 4.4: Attestation protocol

Vimercati et al. [12] use timestamp (as a commitment message) stored securely with each record version that assures the time when it was written. However, in both of the schemes, the CSP can still give the stale resource in response to a read request without the misbehavior being detected by the data owner. Our proposed scheme guarantees that the received resource version by a user is the latest committed version at least till the time when it was sent by the CSP.

4.5.3 Revocation

Popa et al. [15] use lazy revocation ([5]) to handle access right revocation, i.e., a revoked file is re-encrypted only when the file is modified for the first time after being revoked. Therefore, the untrusted CSP can collude with the writer and modify the last written file without being detected by the data owner.

In Vimercati et al. [12] scheme, a write access revocation is handled by updating the secure write token at CSP. However, it will not restrict the collusion of untrusted CSP and the revoked user who want to modify the existing resource.

Table 4.3 summarizes the mechanisms used by existing schemes with respect to considered security properties, i.e., integrity, write-serializability, freshness and Secure revocation. In the table, 'X' indicates that the security property is not addressed. As shown in the table, integrity is addressed using Message Authentication Code (MAC [61]) in the most promising schemes [15, 12] and in the proposed scheme. Chain hash is preferred over the global lock and out-of-band communication based mechanisms for handling write-serializability by these schemes. The scheme in [15] achieve version-based staleness, i.e., a read returns one of the kth latest version. k is fixed in a system and dependents on the time required by a data record to reach a reader. In contrast, the proposed scheme uses $< v, d, t >$-staleness guarantee. Attestation mechanism is used for handling a user's access right revocation as compare to lazy revocation or token renewal.

Table 4.3: Comparison of existing schemes against the considered security properties

Security property $\rightarrow$	Integrity	Write Serializability	Freshness	Access right revocation
$\downarrow$ Scheme				
Li et al. [42]	Hash tree	Global lock	X	X
Shraer et al. [43]	Verifier	Out-of-band communication	X	X
Popa et al. [15]	MAC	Chain hash	Version-based staleness	Lazy revocation
Vimercati et al. [12]	MAC	Chain hash	X	Renew token
Proposed	MAC	Chain hash	$< v, d, t >$-staleness	Attestation

4.6 Summary

In this chapter, we demonstrated the challenges with regard to write access control for outsourced data as it requires more dependency on the untrusted service provider. In the considered data outsourcing scenario, we argued that even the authorized user should not be allowed to modify an outsourced content for an unlimited amount of time. Also, we have shown that in the existing schemes, a revoked user in collusion with the CSP can modify the old written data files for which his access is now revoked. The new system with proposed mechanism is implemented on Microsoft Azure platform and it shows that the suggested mechanism is viable in practice.

We also provide stronger freshness guarantee as compared to the existing schemes for outsourced data. It assures the reader that the received data file is fresh at least until the time when it was dispatched from the CSP. Freshness guarantee plays an important role especially when concurrent modifications to the existing data file are allowed. Read staleness is inversely proportional to the efficiency in handling concurrent updates.

For efficiency reasons, we assume that the data owner will divide the outsourced resources into four groups $(1 - 4)$ according to their security level (similar to the priorities assigned in [15]). Most secure resources are in group 1 and resources which require the least secrecy are in group 4. Group 1 resources are audited in each time slot, group 2 resources are audited very frequently but not necessarily in each time slot, group 3 are least frequently audited and the group 4 resources are never audited. To isolate the scheduled versions for auditing from the CSP, resources from groups 2 and 3 are randomly chosen. Although the percentage of resources chosen are application dependent, a number of resources chosen from the group 2 are always greater than the number of resources chosen from the group 3.

CHAPTER 5

A novel PHRMS scheme supporting unobservability and forward secrecy

Privacy and security of personal health information are the two major concerns for users of e-Health system. For privacy reasons, it is desired that third parties cannot link the outsourced e-Health documents to their owners. A scheme is proposed that employs mix networks for achieving unobservable outsourced data access. An important security requirement named forward secrecy is addressed and protocols for publishing a user's health record documents are proposed and analyzed.

5.1 Introduction

Confidentiality of PHR is an important security requirement of a PHRMS [71]. A PHR owner can identify the appropriate doctor to treat her and grant him access to an appropriate part of her record. A doctor will usually access only her specialty related information from a PHR [72]. Therefore, the PHR data can be divided according to the doctor's specialty classes for proper access control. To enforce access control, each class of information can be encrypted with distinct keys and an appropriate key can be given to the consulting doctor. However, access to future PHR information by the doctor must be restricted, since the patient may change her doctor at any time. If forward secrecy is not provided, an unauthorized doctor can see the patient's future consultation information using some old authorization, such as to whom she is consulting and what prescriptions she is getting. A solution to handle this is to encrypt each new class document with a fresh secret key. This approach requires a large amount of secret storage to store these keys with the patient.

Unlinkability between PHR owners and their PHR documents is an important privacy requirement. A patient will be highly concerned about the linkability to some of her private PHR information [73], for example, mental health diagnosis information such as depression or alcoholism, HIV, psychiatric behavior, teenagers, battered women, that may lead to her social discrimination [74]. Health Insurance Portability and Accountability Act (HIPAA [75]) outlined the legal protections for PHR privacy. However, the HIPAA rules apply only to covered entities. The entities like cloud service provider, individual doctor receiving the consultation fees in cash, researchers and surveyors are not covered entities. There are two types of existing solutions which provide user-data unlinkability. In one approach [76, 77], the data is released in groups and within a group, a person's identifier cannot be distinguished from at least a fixed number of individuals. Privacy of these methods is limited to the fixed number of individuals in each group that require trusted servers and hence are not suitable for untrusted cloud servers. The second approach uses a procedure called pseudonymisation [8], where the identifying information in a data file is replaced by a secure (encrypted) identifier called pseudonym. A user knowing respective pseudonym can only link the data file. To allow a doctor to decrypt the PHR documents of patients, the de-pseudonymisation and pseudonymisation operations must be separated. However, this separation can be done only by using asymmetric encryption schemes which require more computation cost [47]. Also, a separate third party is generally used for the de-pseudonymisation procedure.

Although, the unlinkability using pseudonymization works well when communicating parties are trusted, it will not work when one of the communicating parties is untrusted such as the CSP. Especially in case of untrusted CSP who controls the system traffic can link the communicating parties involved in a communication. Therefore, a stronger privacy notion is needed such as unobservability which will also defend against the adversary who can control the network traffic. For achieving unobservability, we first time make use of MixNet [48] in PHRMS (as best of our knowledge) so that no unauthorized user including the cloud service provider can find the link between a user and her PHR information.

It is necessary to enable data access by the secondary users such as medical researchers and surveyors ([78]). For example, surveyors studying the malaria cases in a city must get efficient access to all the related reports created by the laboratories (labs) present in that city and study them. If the lab reports are stored in encrypted form, then providing access to researchers and surveyors requires either giving them the corresponding decryption keys or, somebody on behalf of PHR owner will decrypt the documents and send to them. Since researchers or surveyors are not trusted by a PHR owner, they may leak owner's privacy. We assume that the Lab reports are stored unencrypted so that they are efficiently but securely accessed by the users including secondary users.

We propose a symmetric key based key management solution for patient-centric PHRMS. We first time introduce the forward security property in PHRMS so that a doctor with any old authorization key cannot access any newly added PHR document. The scheme supports key and user revocation. A doctor or lab cannot deny that a document is written by them if it was the case. The unobservability property in the PHR document publishing protocol is formally analyzed using ProVerif [50]. In the proposed scheme, secret storage requirement for a PHR owner will be of storing one private key and one master secret key. Also, one certificate is needed to be stored. To reduce computation cost, hash computations and symmetric key encryption/ decryption operation are generally used.

The following section gives the system overview. Section 5.3 describes and analyze the secure key management for a PHR. In Section 5.4, we give protocols for publishing a medical prescription and lab report. It formally analyzes the lab report publishing protocol with respect to privacy leakage. Section 5.5 will give few use cases for a better understanding of given PHR management system and discuss the hospital scenario. Section 5.6 gives the summary of this chapter. For better readability, notations used in this chapter are shown in Table 5.1.

Table 5.1: Notations and abbreviations used

Notation	Description
$h()$	Cryptographic hash function
$PHRid_U$	A unique id of a PHR for user U
SM_U	A symmetric master key of user U
K_i	Key of class i
$K_{i,j}$	jth key for class i in a PHR
$Cert_U$	Certificate of user U
$E()$	Symmetric encryption function
$\|\|$	bit-wise concatenation operator
(P_U, S_U)	A public and private key pair for user U

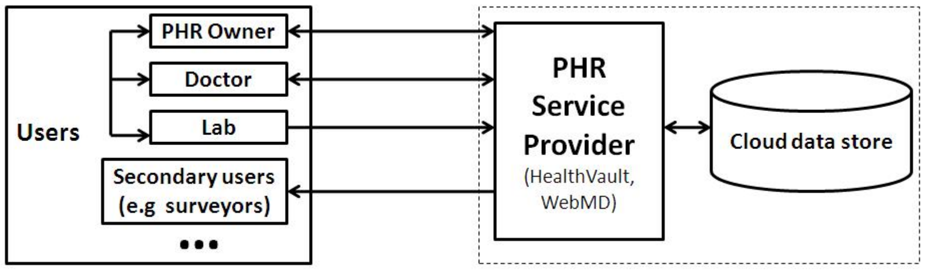


Figure 5.1: The PHRMS system

5.2 System overview

An outline model of the proposed PHRMS is shown in Figure 5.1. It consists of two primary entities: the users and PHR Service Provider ($PHRSP$).

Users a user is a PHR owner, doctor, lab, insurance company, researcher or surveyor. The responsibilities of key users are described below.

1. PHR owner is responsible for creating and changing her PHR with the help of PHRSP. We assume that the PHR owner is also responsible for generating and managing secret keys used to encrypt her PHR documents.

2. Doctor represents an entity responsible for generating medical prescriptions and progressive notes for a PHR owner, whenever requested.

3. Lab represents an entity responsible for generating patient's medical laboratory report as and when requested by a PHR owner. A lab report is such as chemical pathology report, microbiology report, immunology report etc.

PHRSP It is the central core of the PHRMS. Its responsibilities are: registering system users, maintaining system data including each (registered) patient's PHR information and, processing the read and write access requests from the authorized users. PHRSP stores the system data at the cloud data store, maintained by the cloud service provider or the PHRSP itself. PHRSP is semi-trusted, i.e., it follows the instructions given by the users correctly but can collude with the unauthorized parties and release the stored data.

Each user needs to first register itself with the PHRSP. Upon receiving a request, PHRSP will generate a unique user id u, update it in its registered user list and send it back to the user. In case the request is on behalf of a PHR owner, PHRSP also generates a unique PHR id $PHRid_u$ for the user u and send it to the user.

After the registration phase, the Certificate Authority (CA) issues a user certificate ($Cert_u$) to u that contains user type and sub-type information. User type information tells whether the user is a PHR owner, a lab, a doctor etc. User sub-type information tells about the specialty of the user. For example, sub-type of a lab will be hematology, chemical pathology, microbiology, immunology etc., and of a doctor will be Dentist, ENT specialist, Orthopedic Surgeon, Gynecologist etc. along with the doctor's medical degree (for example BMed/BM/MD etc.) We assume that the class types are known in advance and are defined by the PHRSP. A user certificate is used for authentication purposes by the other entities in the system.

A unique PHR document ID is generated by the PHRSP when requested by a lab or a doctor. A document ID is a bit string of type "$x||y||z$" where x and y identify the user's (who generates it) type and sub-type, respectively, and z is a random string generated by the service provider which uniquely identifies the document. A group of unique IDs (for a lab or a doctor) can be generated and assigned by the PHRSP in advance if requested.

5.2.1 Requirements and assumptions

The data access control policy is solely enforced by the PHR owner itself. As a non-security requirement, we require that the PHRMS data will be anonymously and efficiently accessed by the secondary users such as medical researchers and surveyors. We assume that a large number of users register with the PHRMS. The communication link between a user and the PHRSP is assumed to be secure. A large number of users are assumed to be associated with a doctor or a lab to ensure unobservability. Since a lab report contains highly technical information, it is required to be read by the doctors and the owner only. Allowing frequent access to the lab reports by its owner may disclose linkability between the two. Labs will know their own generated reports for a patient. The doctors and labs do not intentionally leak any of their known or generated patient's PHR information to anyone. A PHR owner may use a low configuration device and hence computationally intensive operations are avoided at their end. A doctor with access to a PHR document has access to all its old related documents (patient's history) to enable more accurate diagnosis of the patient. We require that the lab reports and the medical prescriptions do not contain any patient identity information like patient's name and her address. A unique id is assigned to each such document with the help of PHRSP, to avoid any ambiguity in accessing them.

Let CLab is a central lab with respect to a group of labs in an area, used for achieving unobservability between the lab reports and the patients. It is assumed that a CLab does not reveal the linkability between the communicating parties.

5.2.2 Unobservability

The unobservability property ensures that no unauthorized user including the untrusted PHRSP can observe the linkage between a PHR owner's identity with her PHR documents. Unlike medical prescriptions, we store lab reports anonymously and unencrypted at the server so that the users can efficiently access them. To anonymously store a lab report, we require that the respective lab uploads it directly to the server. Since the lab reports do not reveal their owner's identity (as possibly in the case of progress notes), we store them unencrypted. An authorized doctor downloads and read a report directly from the server with the help of PHRSP. A pictorial view of the cloud data store is shown in Figure 5.2.

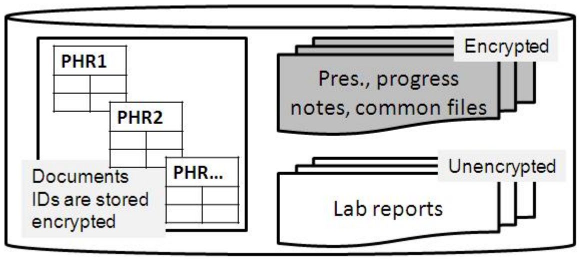

PHR1
PHR2
PHR...
Documents IDs are stored encrypted
Encrypted
Pres., progress notes, common files
Unencrypted
Lab reports

Figure 5.2: A view of cloud data store

A Mix Network (or MixNet [48]) is used for enabling unobservable communication between the users over the internet and hence the unobservability. It consists of a sequence of mix nodes which provide anonymity for a batch of inputs, by randomly changing the order of the arrived messages.

In the proposed scheme, all the PHR documents are stored independently. Since each medical prescription is stored in encrypted form, one cannot trivially link the prescription information with its owner. Also, an authorized doctor can only decrypt it other than the owner, which is assumed to be trusted and does not disclose the linkability. In the case of lab reports, to defend against an active attacker who may attempt to link input and output messages to and from a lab, we use the concept of MixNet. CLab (as a mix node) is assumed to be present corresponding to a group of labs in an area. Each patient communicates with a lab (or other entities) through the area CLab. For l patient requests to a CLab and for a group of labs, guessing the probability of user-lab link will be $1/l$. Hence, any unauthorized user including PHRSP will not able to link a lab report with its owner with probability more than $1/l$.

5.3 PHR encryption and access control

A PHR consists of all health related documents of a user. To update a PHR efficiently, we store the document IDs separately from the actual documents. Therefore, a PHR can be viewed as a group of associated documents IDs. For access control, the PHR documents IDs are divided into independent classes according to the sub-type of the doctor it is related to. For example, document IDs related to the ENT specialist are grouped in one class. The motivation behind this classification of IDs is that a doctor needs to access only its specialty related documents. 0th class is a special class, accessible to all authorized doctors. It contains the owner's information like immunizations, allergies, family history, emergency contact information, blood group, age group, etc.

For data secrecy, PHR documents IDs are encrypted. A distinct symmetric key is assigned to each class and is used for encrypting that class' IDs. We consider a PHR (excluding the 0th class) with two types of documents: medical prescriptions (or progressive notes) generated by doctors and the lab reports generated by different labs. Since a doctor's prescription may contain a continuous progress of a patient, an adversary can link it to the patient if it is left unencrypted. An adversary here is an entity including PHRSP who is always curious to know the patient's PHR information. Therefore, we suggest that every prescription must be encrypted with a key used to encrypt its corresponding ID. To avoid communication and computation cost at the PHR owner, prescriptions are directly uploaded to the server by the doctor, without the involvement of the owner.

5.3.1 Key management

The PHR documents are divided into independent classes as discussed above. An example PHR representation is shown in Table 5.2. A row in the table represents a class. Each class has an ordered sequence of corresponding encrypted document IDs. A PHR can have rows for a subset of possible classes, as needed. For example in the table, it has only two specialty classes 1 and 3, other than the 0th class.

Table 5.2: An example PHR

Class\ seq. #	1	2	3	...	m
0(Common)	$D_{0,1}$	-	-	...	-
1(ENT)	$D_{1,1}$	$D_{1,2}$	-	...	-
3(Dentist)	$D_{3,1}$	$D_{3,2}$	$D_{3,3}$	...	-

Table 5.3: Encryption keys in the example PHR

$Class\backslash seq.\#$	1	2	...	m
0(Common)	$K_{0,1}=h^m(K_0)$	$K_{0,2}=h^{m-1}(K_0)$	...	$K_{0,m}=h^1(K_0)$
1(ENT)	$K_{1,1}=h^m(K_1)$	$K_{1,2}=h^{m-1}(K_1)$	...	$K_{1,m}=h^1(K_1)$
3(Dentist)	$K_{3,1}=h^m(K_3)$	$K_{3,2}=h^{m-1}(K_3)$	...	$K_{3,m}=h^1(K_3)$

Let 1 and 3 represents ENT and $Dentist$ classes, respectively (as specified by the PHRSP). A column in the table represents the document sequence number j with $1 \leq j \leq m$, where m is the maximum number of possible documents present in a class.

Let an encrypted document ID in a class i with sequence number j be represented as $D_{i,j}$. Since each class' data is distinct; the PHR owner can independently handle each of them.

For all row and column intersection in Table 5.2, a different encryption key is assigned as shown in Table 5.3. Let each class i is associated a class key K_i computed as $h(SM_u, i||r)$, where SM_u is the symmetric master key generated by the PHR owner and r is used for handling class key revocation. The encryption keys may be revoked due to reasons like key forgery and the session with the doctor is needed to be revoked in-between, by the patient if she wants to change her doctor. Initially, $r = 1$ and is incremented by one each time the ith class key is revoked. The PHR owner will generate a hash chain for each class using the respective class key. A hash chain for class i with m elements is computed as $K_{i,m} = h^1(K_i), K_{i,m-1} = h^2(K_i), ..., K_{i,1} = h^m(K_i)$.

Updating a PHR

A PHR document with a unique ID is generated and uploaded by a doctor or a lab, whenever requested by the PHR owner. The ID is then sent to the owner over a secure channel. Upon receiving the ID (or IDs) for a specific class, the owner will compute the corresponding key in the hash chain, encrypt the ID with the computed key and send it (i.e., $D_{i,j}$) to the PHRSP who will then update it in the owner's PHR.

The length of the hash chain is bounded by m, which is the maximum number of possible documents in a class. Suppose the number of consultations per person is 13. In a study, all Australians on average go to the doctor 11 times per year. According to the data available at OECD iLibrary [79], it is maximum 13 in Japan and Korea, and over 11 in the Czech Republic, Hungary, and the Slovak Republic, it is less than 3 in Chile, Mexico, and Sweden. Therefore, considering 100 years of age of a person and 13 consultations a year, m can be typically taken as 1300. To avoid these number of hash computations each time a key in the hash chain is computed, the owner can store intermediate hash value $h^{m-l}(.)$ taking an appropriate l for each existing class.

5.3.2 Comparison

Scheme	Special h/w required	Access right revocation	Forward secrecy	Privacy method
Huang et al. [84], Chen et al. [83]	Yes	No	No	No
Odelu et al. [87], Liu et al. [88]	No	No	No	No
Li et al. [72]	No	Yes	No	No
Thilakanathan et al. [89]	No	Yes	No	Secret sharing
Neubauer et al. [90]	No	No	No	Pseudonymization
Benaloh et al. [91]	Yes	Yes	No	Searchable encryption
Moor et al. [93]	No	No	No	Pseudonymization
Proposed scheme	No	Yes	Yes	Mix node

Table 5.4: Coarse level comparison of PHRMS schemes

A comparison of existing symmetric key based PHRMSs is shown in Table 5.4. The comparison is done on the basis of following properties: whether the scheme requires any special hardware, whether access right revocation is addressed, whether forward secrecy property is addressed, and which privacy method is used. From the comparison table, we can see that a number of schemes [83, 84, 91] are hardware-based. Schemes in [87, 88, 72] use hierarchical access control. These schemes require a significant amount of system public storage for storing the key derivation hierarchy and key derivation information. Also, the key derivation cost for accessing PHR document vary with the height of the hierarchy. Li et al. [72] scheme requires a variable number of secret keys with each system user. Also, Liu et al. [88] scheme is not scalable with add or delete file operation or change in the relationship between user classes. All the existing schemes enabling privacy requirement use a TTP in their implementation. Our proposed scheme first time introduces the forward secrecy requirement using symmetric keys. For privacy, Thilakanathan et al. [89] scheme uses secret sharing mechanism and requires a TTP for data sharing service. Neubauer et al. [90] scheme is not patient centric. The solution in Benaloh et al. [91] scheme needs to create and manage multiple keys by users and service providers. Moor et al. [93] scheme uses a separate data provider rather than another TTP to perform the pre-pseudonymization process. As compare to the pseudonymization-based anonymity, our scheme uses mix node and provides stronger unobservable communication guarantee.

5.4 Protocols for publishing PHR documents

In this section, we describe the protocols for publishing a medical prescription and a lab report. The data owner initiates the protocols whenever need a document to be generated. We assume that communication between the communicating parties is secured using SSL.

5.4.1 Publishing a medical prescription

A prescription is published by a doctor as and when requested by a patient (PHR owner). The procedure for publishing a medical prescription (or progress note) is shown in Figure 5.3 and described as follows.

1. PHR owner U will send the prescription request as $<Details,\ PHRid_U,\ (i,j),\ K>$ to the doctor where $Details$ represent the disease information against which prescription is requested, $PHRid_U$ is her PHR's id and (i,j) is the index of latest document for which the doctor is to be given access. The key $K = K_{i,j+1}$ is used for encrypting new prescription. It can also be used for

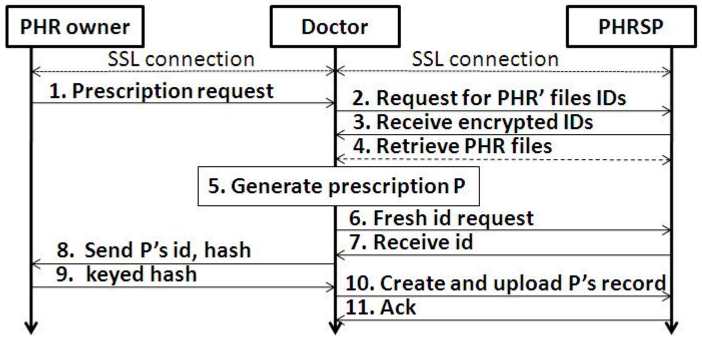


Figure 5.3: Publishing a medical prescription

computing the encryption keys of the previous documents in that class to access the patient' history.

2. Upon receiving the request, doctor will read the request $Details$ and if needed send the PHR document request to the PHRSP as $< PHRid_U, (i,j), l >$ where, l is the number of latest encrypted IDs required.

3. Upon receiving the request, the PHRSP will retrieve the most recent l encrypted IDs $\{D_{i,j}, ..., D_{i,j-(l-1)}\}$ from the server corresponding to $PHRid_U$ and (i,j), and send them back to the doctor.

4. After receiving the encrypted document IDs, the doctor will compute respective keys in the hash chain using K and decrypt the IDs. Now the IDs are used to retrieve respective documents from the server, decrypt and read them. If required, doctor can retrieve more (older) documents from the server (repeating Steps $2-4$).

5. Now, based on the current $Details$ and retrieved information of the PHR owner, doctor will generate the prescription P or append a note to existing P.

6. The doctor will now send a fresh document id request to the PHRSP (if not already received) with a random request index $r2$ for message synchronization.

7. Upon receiving the id request, PHRSP will generate an unique id IDf and send it back to the doctor with r.

8. Receiving IDf, the doctor will compute $M = h(ID||E_K(P))$ and send $< IDf, M >$ to the PHR owner.

9. Receiving $< IDf, M >$, PHR owner will compute $K' = h(K_i||j)$ and $\text{HMAC}_{K'}(M)$ where, HMAC() is the keyed-hash message authentication code [61] and is used for write integrity protection. Then send $\text{HMAC}_{K'}(M)$ to the doctor. IDf is then encrypted and updated in her PHR, with the help of PHRSP.

10. Receiving $\text{HMAC}_{K'}(M)$, the doctor will compute a signed message $\{M\}_{S_D}$ with its private key S_D and send the storage request for P to PHRSP with the following message $< IDf, E_K(P), D, \{M\}_{S_D}, HMAC_{K'}(M) >$ and wait for the acknowledgment message. Signed message using S_D is used as a proof of association between D, IDf and P so that the doctor D cannot deny later that P is created by him. If acknowledgment is not received within a specific period of time, the doctor will send the request again.

11. Upon receiving the message, PHRSP will upload it to the server and return back a message $< Ack, IDf >$ to the doctor where, Ack is an acknowledgment.

5.4.2 Publishing a Lab report

The lab publishing protocol has two goals: (1) the reports are efficiently accessible by the secondary users; and (2) the reports are unlinkable with the patients. For the first goal, we allow the lab reports to be stored unencrypted at the server. For

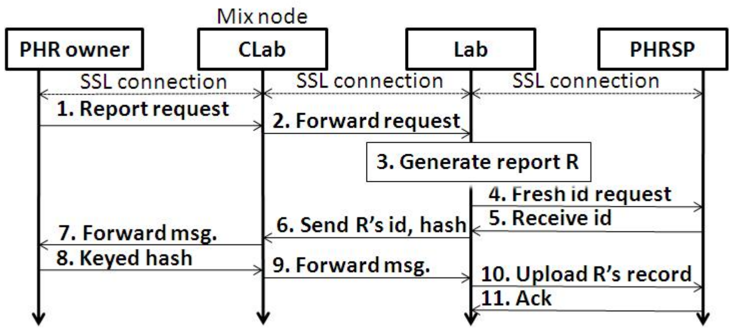


Figure 5.4: Publishing a lab report

the second goal, an anonymous channel is created between the patients and the labs, using a CLab (discussed above) and then each lab report is published by the lab itself who generates it. The procedure for publishing a lab report is shown in Figure 5.4 and described as follows.

1. The PHR owner U will compute $X = \{U, PHRid_U, req\_info, k\}_{P_L}$ where, P_L is the public key of the requested lab L, req_info represents the necessary information to generate the lab report and k is a random secret. Now, the owner will send $< L, X >$ to the CLab.

2. Upon receiving the message $< L, X >$, CLab (working as a Mix node) mix it with other received request messages and later forward X to lab L.

3. Receiving the report request, L decrypts X using its private key S_L and generates the required report R corresponding to the received request information req_info.

4. Lab L will now send a fresh document id request (if not already received) to the PHRSP with a random request index r for message synchronization.

5. Upon receiving the request, PHRSP will generate an unique id IDf and send it back to L along with r.

6. Receiving (or having) IDf, lab L will compute $E_k(IDf)$ and $M = h(IDf||R)$, and send them along with U to the CLab.

7. Receiving $U, E_k(IDf), M$, the CLab will forward $E_k(IDf), M$ to U.

8. Receiving $E_k(IDf)$ and M, the owner will compute $K' = h(K_i||j)$ and $HMAC_{K'}(M)$ similarly as in prescription publishing protocol and send message $< L, HMAC_{K'}(M) >$ to CLab. Later, PHR owner will decrypt IDf from $E_k(IDf)$ using k sent in Step 1 and securely updated it in her PHR with the help of PHRSP.

9. Receiving $< L, HMAC_{K'}(M) >$, the CLab will forward $\text{HMAC}_{K'}(M)$ to L.

10. Receiving $\text{HMAC}_{K'}(M)$, lab L will compute signed message $\{M\}_{S_L}$ with its private key S_L, create record for R as $< IDf, R, L, \{M\}_{S_L}, HMAC_{K'}(M) >$

and send it to the PHRSP. $\text{HMAC}_{K'}(M)$ is used similarly as in prescription publishing protocol. Now, L will wait for an acknowledgment from PHRSP. If the acknowledgment is not received within a specified amount of time, it will send the request again.

11. Upon receiving the request, PHRSP will upload the received report at the server and return back an acknowledgment message $< Ack, IDf >$.

Lab reports are stored at cloud server in cleartext to facilitate efficient access by authorized doctors, researchers, and surveyors. These must be anonymously stored for data secrecy reasons so that no unauthorized user can link a report with its owner. Since an outsourced report does not contain any owner's identity information; one cannot trivially link the report with its owner. However, one can get linkability when the report is communicated over the Internet or through the collection of IDs in PHR. The report is generated and outsourced by a lab. The report ID is securely sent to the patient who will then update it securely in her PHR.

We used ProVerif [50] to verify the unobservability property of the lab report publishing protocol. ProVerif is a well-known cryptographic protocol verifier that automatically verifies various security properties including unobservability.

To model the unobservability property in lab report publishing protocol, we consider two reports $R1$ and $R2$ generated by a lab for users $U1$ and $U2$ using the protocol shown in Figure 5.5. The unobservability property is specified in terms of observational equivalence between two variants of the protocol. We say, two variants are observational equivalent if an attacker cannot distinguish between the two variants by interacting with either of them. For example, in the above lab report generating protocol X, the unobservability is specified by the following

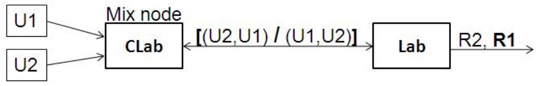


Figure 5.5: CLab handling two user requests

observational equivalence:

$$X\{R1/U1\}\{R2/U2\} \sim X\{R1/U2\}\{R2/U1\}$$

Where process $X\{R1/U1\}\{R2/U2\}$ represents that report $R1$ is for user $U1$ and report $R2$ is for user $U2$, respectively. Similarly in process $X\{R2/U1\}\{R1/U2\}$, report $R2$ is for user $U1$ and report $R1$ is for user $U2$, respectively. To further simplify the above equivalence, consider that the first report published is R, for two input requests from the patients $U1$ and $U2$. The observational equivalence can be now relaxed to $X\{R/U1\} \sim X\{R/U2\}$ where, $R \in \{R1, R2\}$.

The detailed analysis is given in Appendix C.

5.5 Security analysis

In the proposed scheme, medical prescriptions are stored in encrypted form. The encryption keys are only known to the PHR owner and the authorized doctors. PHR documents (encrypted) IDs are grouped into classes where each class belongs to a doctor's specialty. A PHR owner will securely send the necessary class key(s) to the doctor. The doctor can now access only their corresponding class information from the user's PHR. Since the doctor does not have keys of other classes, they are not able to access other class documents. On the other hand, the lab reports are stored anonymously at the server and are accessed by doctors only along with their owners. Communications between the system entities are secured with secure SSL. Therefore, any unauthorized user including PHRSP does not obtain any user's PHR information.

A doctor with a key in a class can only access documents belonging to that class. Also, since the class keys are generated using a hash chain, one can compute keys of one side of the chain. In the example shown in Table 5.3 one can compute old (left-hand side) keys. However, future (right-hand side) keys are hard to compute due to the one-wayness property of the hash function. Therefore, a doctor with access to a key in a class cannot access the future documents encrypted with right-hand side keys. In the proposed scheme, a next key in the chain is required for encryption only when consulting doctor is changed.

Revocation

The encryption keys used to encrypt the PHR documents ids and the medical prescriptions may be revoked due to the above discussed reasons. To revoke a key $K_{i,j}$, PHR owner will do the following: increment the value r corresponding to class i by one and then recomputes the class key K_i (as $h(SM_u, i||r)$). The hash chain is now computed using updated K_i as in PHR registration phase and all IDs in class i are re-encrypted with the corresponding new keys from the hash chain. The prescriptions are also re-encrypted with the updated keys used to encrypt corresponding IDs. Each document ID and prescriptions in ith class are now re-encrypted with a new key. Therefore, no unauthorized user with revoked key (of class i) can access the class documents.

In what follows, we discuss two use cases that help in understanding the working of the proposed system and then discuss the related hospital scenario.

Case 1: Non-trivial access to PHR documents

In general, a doctor can access documents related to her class. As a non-trivial case, a doctor can sometimes require access to a document belonging to another class. For example, a physician may require access to an ongoing medication (or prescription) related to an ENT class. In such cases, the doctor can obtain required access to one or more specific documents belonging to other classes. The proposed system has two types of PHR documents: the medical prescription and the lab reports. In the case of medical prescriptions, the PHR owner downloads the encrypted prescription, decrypts it and sends it to the doctor through a secure channel. It allows the doctor to access only that particular document, since the encryption key is not known to the doctor. In the case of lab reports, the PHR owner will download the corresponding encrypted report ID, decrypt it and send it to the doctor through the existing anonymous channel (using CLab). Receiving the report ID, the doctor may then retrieve the report directly from the cloud server and access it.

Case 2: Treatment to minors

A PHR is required to be created for a minor, immediately after a baby is born. However, the minor cannot handle her PHR for her initial years. She will go with their parents to the doctor, for consultation. Therefore, a secure mechanism is needed to access a minor's PHR in her initial years and later the PHR is securely handed over to the candidate. The age of handover of a PHR to its owner can differ in different states or countries as per their local laws. It is assumed that the parents securely manage the minor's PHR until they hand back to her. After getting custody of the PHR, the owner may need to restrict her parent's access to her PHR. Therefore, the owner generates a new master key and re-encrypts all PHR IDs and corresponding prescriptions with the new keys, generated using the master key as described in Section 5.3. Now, without getting access to the new master key, the owner's parents cannot access her PHR.

Discussion on hospital scenario

Consider an example scenario where a patient approaches a specialty department in a hospital that may have more than one doctor in the panel. At a time only one doctor from the panel may be present for consultation. To handle such situation, the above protocol requires one extra communication prior to the first step so that the patient knows which doctor is in the consulting room. The patient sends a request to the hospital that contains the concerned department information he wants to consult with. Receiving the request, the hospital returns the public key of the doctor currently in the consulting room along with her certificate. Receiving the doctor's certificate, the patient will verify it and if satisfied then start the above-given protocol.

A hospital may have junior doctors and nurses within each specialty department. The junior doctor(s) can receive the patient request, assign it a case ID (internal to the hospital), maintains it and finally uploads it to the server on behalf of the consulting doctor. However, the initial request message can be decrypted by the consulting doctor only by her private key (Step 1). Since they are doctors, the consulting doctor must trust them and pass the decryption key to them. Now, the junior doctor can download the patient's history (Steps 2-4) and give them to the consulting doctor. That will now generate the prescription (Step 5), compute the signature for the record and pass it to the junior doctor. The junior doctor can then generate the fresh ID if not previously exists (Steps 6-7), create the record, upload it (Steps 8-9) and send prescription ID back to the patient (Step 10).

Other important entities in the hospital are the nurses. They may require writing notes on the prescription, for example, current patient temperature, weight, medication with time etc. For each prescription request, the junior doctor will encrypt the prescription with a locally generated symmetric key known also to the department nurses (along with the doctors) and put it in a common database associated with the case id. Since key K is not known to the nurses, they cannot access any patient's PHR information other than the current prescription. Nurses can only read, decrypt, write their readings/notes, re-encrypt the current prescription and store it back to the local store. The consulting doctor can read these readings/notes and write progress notes further to the prescription. After the consultation is over, the prescription record is built and stored at the cloud server by the doctors.

5.6 Summary

We proposed an efficient symmetric key based patient-centric PHRMS for the cloud using hash chains for key management. It efficiently handles the forward data secrecy and access control in a PHR. Lab reports are stored anonymously and unencrypted so that they can be efficiently accessed by the users including the secondary users. Protocols for publishing a medical prescription and a lab report are explained. We implement the lab report publishing protocol in ProVerif calculus and successfully prove that it satisfies the unobservability property using ProVerif tool. We consider the revocation of the encryption key and a user. Non-repudiation property is met so that a writer (lab or doctor) cannot deny that the PHR document is written by them, in the case of disputes. Write integrity of the outsourced PHR data is protected.

CHAPTER 6

Conclusions

Fine-grained access control to outsourced data using a symmetric key-based cryptosystem requires each data file (that can be accessed individually) is encrypted with a distinct key. A user authorize for accessing a set of files needs to store each file's secret key. Key management hierarchies are used to efficiently manage a large set of secret keys with each user and enforce data access control. Two types of key management hierarchy are used for managing keys in data outsourcing scenario: user-based and resourced-based. We critically analyzed the two types of key management hierarchy and show that the storage requirement for resource hierarchies will be same as user-based hierarchies. They perform better in the case of dynamic operations such as extending read authorization and revoking a user without affecting other required functionalities. However, average key derivation-time in resource-based hierarchies is more than user-based hierarchies. We have implemented the two hierarchies and shown the results experimentally for the sake of our arguments. We conclude that the resource-based hierarchies can be a good candidate for key management in data outsourcing scenario.

The goal of a subscription-based key management hierarchy is to enforce time-limited (or subscription-based) access control. The key assignment for time-limited access is done using a subscription-based HKAS (SBHKAS). Existing SBHKASs exhibit a trade-off between private storage requirement by a user, system public storage requirement, and key derivation cost. Reducing public storage is not generally emphasized in designing traditional HKAS but it is relevant in data outsourcing where consumer is paying for storage-as-a-service. We have proposed a simple and efficient hash-based SBHKAS using dependent keys. The proposed

SBHKAS reduces the secret storage cost at the central authority responsible for managing the keys and system public storage without increasing other costs such as secret storage per user and key derivation cost. However, the average re-keying operation cost due to access right revocation is more in HKAS with dependent keys. In case of dependent keys, ascendant nodes keys are also sometime needs to be re-keyed. In our scheme, a parent node's key needs to be re-keyed only if the key of child node is computed using the parent key.

Access right revocation is a desired operation in many access control systems. Traditionally, it is handled using the re-keying mechanism that assigns and encrypts each affected outsourced node with a new key. It prevents the revoked user with old secrets from accessing the resource which is now encrypted with the new key. Wang et al. [9] proposed an access right revocation mechanism for outsourced data considering honest but curious CSP. The important feature of their mechanism is that it does not require re-keying procedure (used in traditional systems). However, in their mechanism, a user's access right revocation is dependent on all other users' access. Therefore, the system does not scale. In our proposed system, access right revocation can be efficiently handled using the improved certificate-based data access mechanism where each revocation is independently handled.

Recently, Vimercati et al. [40] proposed a *SBHKAS* for outsourced data in a cloud scenario. We show that their scheme has a security flaw. In their scheme, a user after withdrawing his subscription can still have access to the resources associated with his old and revoked subscription interval.

Write access control for outsourced data is more challenging as compared to read access control since it requires more on the service provider. A small malicious change to the outsourced content can put a hugely adverse effect on the data owner's business. Therefore, in the literature, a malicious but cautious CSP type is adopted for write access control. The data owner requires an auditing like mechanism to detect any misbehavior and will take appropriate action accordingly to avoid it in future. In existing schemes, it is possible that a user can modify their own written outsourced records any number of times in collusion with a service provider, without being detected by the data owner. We first time consider this property as an important security requirement and propose an audit-based mechanism to handle it. We also provide a stronger freshness guarantee for distributed cloud scenario to assure a reader that the received data file is fresh at least until the time when it was dispatched by the service provider. We argue that for freshness property, storing timestamp and version number with the outsourced data record is not sufficient. It must require some proof mechanism that can be used later at the time of disputes. Finally, although the audit-based mechanisms will defend against many write access control issues, we realize there is still an open question that can we prevent unauthorized writes without using audit-based mechanisms?

Personal health record (PHR) is a well accepted patient-centric model for cloud-based e-health. It is one of the important privacy enabled data outsourcing application. We proposed a symmetric key based PHR management system (PHRMS) for the cloud using hash chains for key management. Two of the important requirements we addressed are forward secrecy and unobservability. As best of our knowledge, we first time addressed the forward secrecy requirement in PHRMS. It will be beneficial for patient's outsourced data privacy. Privacy of PHR is a primary concern for the user it belongs. It becomes more challenging when it is outsourced to a untrusted cloud. Unobservability is a privacy property defends against traffic analysis and is important requirement when communication is through untrusted entity such as cloud service provider. We achieve unobservability using mix node and show that the forward secrecy can be achieved using one-way hash chains. Although the presence of mix node adds communication delay, it will be significantly less than the document generation time.

References

[1] Fangming Zhao, Takashi Nishide, and Kouichi Sakurai. Realizing fine-grained and flexible access control to outsourced data with attribute-based cryptosystems. In *ISPEC*, pages 83–97, 2011.

[2] Junbeom Hur and Dong Kun Noh. Attribute-based access control with efficient revocation in data outsourcing systems. *IEEE Trans. Parallel Distrib. Syst.*, 22(7):1214–1221, 2011.

[3] Sushmita Ruj, Milos Stojmenovic, and Amiya Nayak. Decentralized access control with anonymous authentication of data stored in clouds. *IEEE Trans. Parallel Distrib. Syst.*, 25(2):384–394, 2014.

[4] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM Conference on Computer and Communications Security*, pages 89–98, 2006.

[5] Kevin. E. Fu. Group sharing and random access in cryptographic file systems. *Master's thesis, MIT*, 1999.

[6] Daniel J. Solove. *Understanding Privacy*. Harvard University Press, Cambridge, USA, 2008.

[7] Andreas Pfitzmann and Marit Hansen. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management. http://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf, August 2010. v0.34.

[8] Johannes Heurix, Michael Karlinger, Michael Schrefl, and Thomas Neubauer. A hybrid approach integrating encryption and pseudonymization for pro-

tecting electronic health records. In *Proceedings of the Eighth IASTED International Conference on Biomedical Engineering*, 2 2011.

[9] Weichao Wang, Zhiwei Li, Rodney Owens, and Bharat K. Bhargava. Secure and efficient access to outsourced data. In *CCSW*, pages 55–66, 2009.

[10] Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Over-encryption: Management of access control evolution on outsourced data. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pages 123–134, 2007.

[11] Atanu Basu, Indranil Sengupta, and Jamuna Kanta Sing. Secured cloud storage scheme using ecc based key management in user hierarchy. In *International Conference on Information Systems Security (ICISS)*, pages 175–189, 2011.

[12] Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Giovanni Livraga, Stefano Paraboschi, and Pierangela Samarati. Enforcing dynamic write privileges in data outsourcing. *Computers & Security*, 39:47–63, 2013.

[13] Mariana Raykova, Hang Zhao, and Steven M. Bellovin. Privacy enhanced access control for outsourced data sharing. In *Financial Cryptography*, pages 223–238, 2012.

[14] Myrto Arapinis, Sergiu Bursuc, and Mark Ryan. Privacy-suppor -ting cloud computing by in-browser key translation. *Journal of Computer Security*, 21(6):847–880, 2013.

[15] Raluca Ada Popa, Jacob R. Lorch, David Molnar, Helen J., and Li Zhuang. Enabling security in cloud storage slas with cloudproof. In *2011 USENIX Annual Technical Conference, Portland, OR, USA, June 15-17, 2011*, 2011.

[16] Beom Heyn Kim and David Lie. Caelus: Verifying the consistency of cloud services with battery-powered devices. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 880–896, 2015.

[17] Matteo Golfarelli, Jens Lechtenbörger, Stefano Rizzi, and Gottfried Vossen. Schema versioning in data warehouses. In *Conceptual Modeling for Advanced Application Domains, ER 2004 Workshops CoMoGIS, COMWIM, ECDM, CoMoA, DGOV, and ECOMO, Shanghai, China, November 8-12, 2004, Proceedings*, pages 415–428, 2004.

[18] Marc Shapiro, Nuno M. Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In *Stabilization, Safety, and Security of Distributed Systems - 13th International Symposium, SSS 2011, Grenoble, France, October 10-12, 2011. Proceedings*, pages 386–400, 2011.

[19] Ayad F. Barsoum and M. Anwar Hasan. Enabling dynamic data and indirect mutual trust for cloud computing storage systems. *IEEE Trans. Parallel Distrib. Syst.*, 24(12):2375–2385, 2013.

[20] Peter Szolovits, Jon Doyle, William J. Long, Isaac Kohane, and Stephen G. Pauker. Guardian angel: Patient-centered health information systems. Technical report, Cambridge, MA, USA, 1994.

[21] Markle Foundation. Connecting for health. the personal health working group final report.

[22] Ari Juels and Burton S. Kaliski Jr. Pors: proofs of retrievability for large files. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 584–597, 2007.

[23] Aiiad Albeshri, Colin Boyd, and Juan Manuel González Nieto. Geoproof: Proofs of geographic location for cloud computing environment. In *32nd International Conference on Distributed Computing Systems Workshops (ICDCS 2012 Workshops), Macau, China, June 18-21, 2012*, pages 506–514, 2012.

[24] Yang Tang, Patrick P. C. Lee, John C. S. Lui, and Radia J. Perlman. Secure overlay cloud storage with access control and assured deletion. *IEEE Trans. Dependable Sec. Comput.*, 9(6):903–916, 2012.

[25] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 14-17, 2000*, pages 44–55, 2000.

[26] Jinguo Li, Yaping Lin, Mi Wen, Chunhua Gu, and Bo Yin. Secure and verifiable multi-owner ranked-keyword search in cloud computing. In *Wireless Algorithms, Systems, and Applications - 10th International Conference, WASA 2015, Qufu, China, August 10-12, 2015, Proceedings*, pages 325–334, 2015.

[27] Sabrina De Capitani di Vimercati, Sara Foresti, and Pierangela Samarati. Recent advances in access control. In *Handbook of Database Security - Appli. and Trends*, pages 1–26. 2008.

[28] Naveen Kumar, Anish Mathuria, and Manik Lal Das. Comparing the efficiency of key management hierarchies for access control in cloud. In *Security in Computing and Communications - Third International Symposium, SSCC 2015, Kochi, India, August 10-13, 2015. Proceedings*, pages 36–44, 2015.

[29] Selim G. Akl and Peter D. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Trans. Comput. Syst.*, 1(3):239–248, 1983.

[30] Mikhail J. Atallah, Marina Blanton, Nelly Fazio, and Keith B. Frikken. Dynamic and efficient key management for access hierarchies. *ACM Trans. Inf. Syst. Secur.*, 12(3), 2009.

[31] Marina Blanton and Keith B. Frikken. Efficient multi-dimensional key management in broadcast services. In *ESORICS*, pages 424–440, 2010.

[32] Wen-Guey Tzeng. A secure system for data access based on anonymous authentication and time-dependent hierarchical keys. In *ASIACCS*, pages 223–230, 2006.

[33] Shyh-Yih Wang and Chi-Sung Laih. Merging: An efficient solution for a time-bound hierarchical key assignment scheme. *IEEE Trans. Dependable Sec. Comput.*, 3(1):91–100, 2006.

[34] Giuseppe Ateniese, Alfredo De Santis, Anna Lisa Ferrara, and Barbara Masucci. Provably-secure time-bound hierarchical key assignment schemes. *J. Cryptology*, 25(2):243–270, 2012.

[35] Mikhail J. Atallah, Marina Blanton, and Keith B. Frikken. Incorporating temporal capabilities in existing key mgmt. schemes. In *ESORICS*, pages 515–530, 2007.

[36] Jason Crampton. Trade-offs in cryptographic implementations of temporal access control. In *NordSec*, pages 72–87, 2009.

[37] Naveen Kumar, Anish Mathuria, and Manik Lal Das. Simple and efficient time-bound hierarchical key assignment scheme - (short paper). In *Information Systems Security - 9th International Conference, ICISS 2013, Kolkata, India, December 16-20, 2013. Proceedings*, pages 191–198, 2013.

[38] Mikhail J. Atallah, Keith B. Frikken, and Marina Blanton. Dynamic and efficient key management for access hierarchies. In *Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005, Alexandria, VA, USA, November 7-11, 2005*, pages 190–202, 2005.

[39] Paolo D'Arco, Alfredo De Santis, Anna Lisa Ferrara, and Barbara Masucci. Variations on a theme by akl and taylor: Security and tradeoffs. *Theor. Comput. Sci.*, 411(1):213–227, 2010.

[40] Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, and Giovanni Livraga. Enforcing subscription-based authorization policies in cloud scenarios. In *DBSec*, pages 314–329, 2012.

[41] Naveen Kumar, Anish Mathuria, Manik Lal Das, and Kanta Matsuura. Improving security and efficiency of time-bound access to outsourced data. In *Proceedings of the 6th ACM India Computing Convention, COMPUTE 2013, Vellore, Tamil Nadu, India, August 22 - 24, 2013*, pages 9:1–9:8, 2013.

[42] Jinyuan Li, Maxwell N. Krohn, David Mazières, and Dennis Shasha. Secure untrusted data repository (SUNDR). In *6th Symposium on Operating System*

Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004, pages 121–136, 2004.

[43] Alexander Shraer, Christian Cachin, Asaf Cidon, Idit Keidar, Yan Michalevsky, and Dani Shaket. Venus: verification for untrusted cloud storage. In *Proceedings of the 2nd ACM Cloud Computing Security Workshop, CCSW 2010, Chicago, IL, USA, October 8, 2010*, pages 19–30, 2010.

[44] Eu-Jin Goh, Hovav Shacham, Nagendra Modadugu, and Dan Boneh. Sirius: Securing remote untrusted storage. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2003, San Diego, California, USA*, 2003.

[45] Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica. Probabilistically bounded staleness for practical partial quorums. *PVLDB*, 5(8):776–787, 2012.

[46] Naveen Kumar and Anish Mathuria. Improved write access control and stronger freshness guarantee to outsourced data. In *Accepted in 18th International Conference on Distributed Computing and Networking, IDRBT, Hyderabad, India*, 2017.

[47] Harald Aamot, Christian Dominik Kohl, Daniela Richter, and Petra Knaup-Gregori. Pseudonymization of patient identifiers for translational research. *BMC Med. Inf. & Decision Making*, 13:75, 2013.

[48] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

[49] Naveen Kumar, Anish Mathuria, and Manik Lal Das. Achieving forward secrecy and unlinkability in cloud-based personal health record system. In *2015 IEEE TrustCom/BigDataSE/ISPA, Helsinki, Finland, August 20-22, 2015, Volume 1*, pages 1249–1254, 2015.

[50] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14 2001), 11-13 June 2001, Cape Breton, Nova Scotia, Canada*, pages 82–96, 2001.

[51] Mikhail J. Atallah, Marina Blanton, and Keith B. Frikken. Key management for non-tree access hierarchies. In *SACMAT*, pages 11–18, 2006.

[52] Carlo Blundo, Stelvio Cimato, Sabrina De Capitani di Vimercati, Alfredo De Santis, Sara Foresti, Stefano Paraboschi, and Pierangela Samarati. Managing key hierarchies for access control enforcement: Heuristic approaches. *Computers & Security*, 29(5):533–547, 2010.

[53] F.K. Hwang, D.S. Richards, and P. Winter. *The Steiner Tree Problem*. North Holland, 1992.

[54] Alfred V. Aho, Michael R. Garey, and Frank K. Hwang. Rectilinear steiner trees: Efficient special-case algorithms. *Networks*, 7(1):37–58, 1977.

[55] Thomas Rothvoß. Directed steiner tree and the lasserre hierarchy. *CoRR*, abs/1111.5473, 2011.

[56] Ondrej Suchý. On directed steiner trees with multiple roots. In *Graph-Theoretic Concepts in Computer Science - 42nd International Workshop, WG 2016, Istanbul, Turkey, June 22-24, 2016, Revised Selected Papers*, pages 257–268, 2016.

[57] Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Encryption policies for regulating access to outsourced data. *ACM Trans. Database Syst.*, 35(2), 2010.

[58] Jason Crampton. Time-storage trade-offs for cryptographically-enforced access control. In *ESORICS*, pages 245–261, 2011.

[59] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *FSE*, pages 371–388, 2004.

[60] Jason Crampton. Practical and efficient cryptographic enforcement of interval-based access control policies. *ACM Trans. on Inf. Syst. Secur.*, 14(1):14, 2011.

[61] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, pages 1–15, 1996.

[62] Microsoft Corporation. Windows azure.

[63] Wojciech M. Golab, Xiaozhou Li, and Mehul A. Shah. Analyzing consistency properties for fun and profit. In *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011, San Jose, CA, USA, June 6-8, 2011*, pages 197–206, 2011.

[64] Amitanand S. Aiyer, Lorenzo Alvisi, and Rida A. Bazzi. On the availability of non-strict quorum systems. In *Distributed Computing, 19th International Conference, DISC 2005, Cracow, Poland, September 26-29, 2005, Proceedings*, pages 48–62, 2005.

[65] Kamal Zellag and Bettina Kemme. How *consistent* is your cloud application? In *ACM Symposium on Cloud Computing, SOCC '12, San Jose, CA, USA, October 14-17, 2012*, page 6, 2012.

[66] Fuat Akal, Can Türker, Hans-Jörg Schek, Yuri Breitbart, Torsten Grabs, and Lourens Veen. Fine-grained replication and scheduling with freshness and correctness guarantees. In *Proc. of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, pages 565–576, 2005.

[67] Giuseppe Cattaneo, Luigi Catuogno, Aniello Del Sorbo, and Pino Persiano. The design and implementation of a transparent cryptographic file system for UNIX. In *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference, June 25-30, 2001, Boston, Massachusetts, USA*, pages 199–212, 2001.

[68] Ethan L. Miller, Darrell D. E. Long, William E. Freeman, and Benjamin C. Reed. Strong security for distributed file systems. In *Proceedings of the 20th IEEE International Performance, Computing and Communications Conference (IPCCC '01)*, pages 34–âĂŞ40, Apr 2001.

[69] Emil Stefanov, Marten van Dijk, Ari Juels, and Alina Oprea. Iris: a scalable cloud file system with efficient integrity checks. In *28th Annual Computer Security Applications Conference, ACSAC 2012, Orlando, FL, USA, 3-7 December 2012*, pages 229–238, 2012.

[70] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, pages 369–378, 1987.

[71] J J Lindenthal and C S Thomas. Psychiatrists, the public, and confidentiality. *J Nerv Ment Dis.*, 170(6):319–23, 1982.

[72] Ming Li, Shucheng Yu, Kui Ren, and Wenjing Lou. Securing personal health records in cloud computing: Patient-centric and fine-grained data access control in multi-owner settings. In *SecureComm*, pages 89–106, 2010.

[73] Carol A. Ford, Susan G. Millstein, Bonnie L. Halpern-Felsher, and Charles E. Irwin Jr. Influence of physician confidentiality assurances on adolescents' willingness to disclose information and seek future health care. a randomized controlled trial. *JAMA*, 278(12):1029–34, 1997.

[74] Paul S Applebaum. Privacy in psychiatric treatment: Threats and response. *American Journal of Psychiatry*, 159, 2002.

[75] US Public Law. Health insurance portability and accountability act of 1996. *104th Congress*, pages 104–191, 1996.

[76] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.

[77] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. *L*-diversity: Privacy beyond *k*-anonymity. *TKDD*, 1(1), 2007.

[78] Charles Safran, Meryl Bloomrosen, W. Edward Hammond, Steven E. Labkoff, Suzanne Markel-Fox, Paul C. Tang, and Don E. Detmer. White paper: Toward a national framework for the secondary use of health data: An american medical informatics association white paper. *JAMIA*, 14(1):1–9, 2007.

[79] OECD. Consultations with doctors, in health at a glance 2011: OECD indicators, OECD publishing. 2011.

[80] Jiankun Hu, Hsiao-Hwa Chen, and Ting-Wei Hou. A hybrid public key infrastructure solution (hpki) for hipaa privacy/security regulations. *Computer Standards & Interfaces*, 32(5-6):274–280, 2010.

[81] W D Yu and M A Chekhanovskiy. An electronic health record content protection system using smartcard and pmr. pages 11–18, 2007.

[82] Wei-Bin Lee and Chien-Ding Lee. A cryptographic key management solution for hipaa privacy/security regulations. *IEEE Transactions on Information Technology in Biomedicine*, 12(1):34–41, 2008.

[83] Yu-Yi Chen, Jun-Chao Lu, and Jinn ke Jan. A secure ehr system based on hybrid clouds. *J. Medical Systems*, 36(5):3375–3384, 2012.

[84] Hui-Feng Huang and Kuo-Ching Liu. Efficient key management for preserving hipaa regulations. *Journal of Systems and Software*, 84(1):113–119, 2011.

[85] Wei-Bin Lee, Chien-Ding Lee, and Kevin I.-J. Ho. A hipaa-compliant key management scheme with revocation of authorization. *Computer Methods and Programs in Biomedicine*, 113(3):809–814, 2014.

[86] Shivaramakrishnan Narayan, Martin Gagné, and Reihaneh Safavi-Naini. Privacy preserving ehr system using attribute-based infrastructure. In *CCSW*, pages 47–52, 2010.

[87] Vanga Odelu, Ashok Kumar Das, and Adrijit Goswami. An effective and secure key-management scheme for hierarchical access control in e-medicine system. *J. Medical Systems*, 37(2), 2013.

[88] Chia-Hui Liu Chen, Tzer-Shyong, Tzer-Long Chen, Chin-Sheng Chen, Jian-Guo Bau, and Tzu-Ching Lin. Secure dynamic access control scheme of phr in cloud computing. *J. Medical Systems*, 36(6):4005–4020, 2012.

[89] Danan Thilakanathan, Shiping Chen, Surya Nepal, Rafael Calvo, and Leila Alem. A platform for secure monitoring and sharing of generic health data in the cloud. *Future Generation Computer Systems*, 35:102–113, 2014.

[90] Thomas Neubauer and Johannes Heurix. A methodology for the pseudonymization of medical data. *I. J. Medical Informatics*, 80(3):190–204, 2011.

[91] Josh Benaloh, Melissa Chase, Eric Horvitz, and Kristin Lauter. Patient controlled encryption: ensuring privacy of electronic medical records. In *CCSW*, pages 103–114, 2009.

[92] Klaus Pommerening, Markus Schröder, Denis Petrov, Marc Schlösser-Faßbender, Sebastian C. Semler, and Johannes Drepper. Pseudonymization service and data custodians in medical research networks and biobanks. In *GI Jahrestagung (1)*, pages 715–721, 2006.

[93] De Meyer F, De Moor G, and Reed Fourquet. Privacy protection through pseudonymisation in ehealth. *Studies in Health Technology and Informatics*, (141):111–118, 2008.

Chapter A
Some algorithms

The Algorithm 22 takes a one dimension chain hierarchy (named ODH) as input and produce an output hierarchy enabling 3-hop shortcut scheme (3-HS). In a 3-HS, distance between any two nodes in the hierarchy will be at most 3 edges (or hops).

Algorithm 22 3HS_Gen(ODH)

```
Input: A one-dimension hierarchy (ODH).
Output: Hierarchy with 3-HS.
1: Create a set of special nodes $S$ consisting of every $\sqrt{n}$th node in the graph. That is, initialize $S$ with $v_1$ and then add nodes $v_{j\sqrt{n}+1}$ for all $j$ such that $j\sqrt{n} \leq n$. If $v_n \notin S$, set $S = S \bigcup \{v_n\}$.
2: Insert new edges between the nodes in $S$ to form the transitive closure of the set.
3: For each node $v_i \notin S$, find $v_j \in S$ such that $j < i$ and $i < j + \sqrt{n}$. Insert an edge $(v_j, v_i)$ if it is not already present.
4: For each node $v_i \notin S$, find $v_j \in S$ such that $i < j$ and $j - \sqrt{n} < i$. Insert an edge $(v_i, v_j)$ if it is not already present.
5: return
```

Algorithm 23 creates a tree data structure in Step 1 corresponding to the input set T of time intervals and user hierarchy UH. Then it assigns secret keys, public labels and public edge values to the tree data structure (represents subscription hierarchy) corresponding to each node in the given user hierarchy.

Algorithm 23 Gen($1^k, T, UH$)

```
Input: Security parameter $1^k$, set $T$ of time intervals and a user hierarchy $UH(V_U, E_U)$.
Output: It create a tree data structure and for each node in the user hierarchy it assigns different set of secret keys, public labels and public edge values to the tree data structure.
1: Create a root node $root$ for the data structure and run $DataStuctBuild(root, T)$.
   Let $G = (V, E)$ denote the tree data structure returned.
2: For each $v \in V$, randomly choose a secret key $k_w \in \{0,1\}^k$ and an unique
   public label $l_w \in \{0,1\}^k$ associated with each node $w$ in $D(v)$, $R(v)$, and $L(v)$.
3: For each $t \in T$, randomly choose a secret key $k_t \in \{0,1\}^k$ and an unique
   public label $l_t \in \{0,1\}^k$.
4: For each $v \in V_U$, randomly choose a secret key $k_v \in \{0,1\}^k$
5: For each node $u \in V_U$, perform the following:
   (a) For each $v \in V$ , randomly choose a secret key $k_{u,w} \in \{0,1\}^k$ associated
   with each node $w$ in $D(v)$, $R(v)$, and $L(v)$.
   (b) For each $v \in V$, construct public information about each edge in $D(v)$,
   $R(v)$, and $L(v)$ using the key derivation method.
   (c) For each $t \in T$, randomly choose a secret key $k_{u,t} \in \{0,1\}^k$.
6: For each $t \in T$, compute public information to permit key derivation between
   nodes: for each edge $(u_1, u_2) \in E$ compute public information by setting $S_{u_1} =$
   $k_{u_1,t}$ and $S_{u_2} = k_{u_2,t}$ and using the key derivation method and public labels $l_{u_1}$
   and $l_{u_2}$.
7: For each $t \in T$, let $V_t \subset V$ denote the set of nodes in $G$ access to which implies
   access to $t$. Then for each $V_t$, for each $v \in V_t$ :
   (a) Find in $D(v)$ the node corresponding to the time interval $t$; call it $w$.
   (b) Create an edge from $w$ to $t$ by computing public information using en-
   abling key $k_{w,t}$'s secret key $k_t$, public label $l_t$, and the key derivation method.
   Mark such an edge with the level of $v$ and type $D$.
   (c) Repeat (a) and (b) for $R(v)$ and $L(v)$, using types $R$ and $L$, respectively.
8: Let $K$ consist of the secret keys $k_t$ for each $t \in T$ and $Sec$ consist of the remain-
   ing secret keys $k_w$. Also let $Pub$ consist of $G$, all public labels (of the form $l_w$
   and $l_t$), and public information about all edges generated above.
9: **return**
```

CHAPTER B

Formal security proof for new SBHKAS

Here, we provide the proof for our proposed *SBHKAS* scheme given in Section 3.3 using the modern notions of security which would be the first provable security style proof for any dependent *SBHKAS* in the literature. Let A_{SH} be the set of all nodes in the subscription hierarchy. Let $Pred(t_a, t_b)$ denotes the set of all nodes in the subscription hierarchy that can derive at least one leaf node t with $t_a \leq t \leq t_b$. We can say that $Pred(t_a, t_b)$ is the set of all predecessor nodes in the subscription hierarchy, with respect to any of the leaf node with time slot from t_a to t_b. $Lower(t_a, t_b)$ denotes the set of all the nodes with subscription interval (t_c, t_d) such that $t_a \geq t_c$ and $t_b \geq t_d$. Let, $S_{(t_a,t_b)} = Pred(t_a, t_b) - Lower(t_a, t_b)$ represents set of nodes in $Pred(t_a, t_b)$ but not in $Lower(t_a, t_b)$. We define set *corr* as set of secret keys known to the adversary in advance. Let, $keys(S)$ is the set of keys associated with the nodes in set S.

To prove the security of above scheme w.r.t. key recovery, we want to model all the information available to the adversary in advance. To achieve this, the game shown in Definition [key recovery] is played between the adversary and the challenger. Security of the scheme is bounded to arbitrary but fixed value in terms of security parameter.

Theorem B.0.1. *Proposed scheme is secure w.r.t. key recovery against static adversary provided h_{pre} assumption hold.*

Proof. Let $G = (V, E)$ be some subscription hierarchy with z time slots where V is the set of nodes and E is the set of edges in the hierarchy. Let, a subscription interval (t_a, t_b) in the hierarchy and let, A be a static polynomial time adversary attacking the node u_t with time interval (t, t) with $t_a \leq t \leq t_b$. Now, based on Definition [key recovery], A is having secret information corresponding to all unauthorized access nodes with subscription interval (t_c, t_d) such that there does not exist a time slot t with $t_a \leq t \leq t_b$ and $t_c \leq t \leq t_d$.

We consider three cases as follows where $|(t_a, t_b)|$ defines number of time slots in subscription interval (t_a, t_b). First two cases are special cases and the third case is the general case. We show that all possible cases can be identified as one of the three cases defined below. Later, we prove that KR adversary A has a negligible advantage in finding the key of any node in the hierarchy as described one of the cases below and hence the overall maximum advantage of A is negligible.

Case 1: $|(t_a, t_b)| = z$ i.e. $t_a = t_1$ and $t_b = t_z$.

As the Definition [collusion secure], there is no node v with subscription interval (t_c, t_d) is available such that there exists a time slot t with $t_c \leq t \leq t_d$ and $t_1 \nleq t \nleq t_z$. Hence, this is trivial case where adversary can access only Pub which contains public edge values with other public information.

We know that getting the key of any node in the subscription hierarchy will lead to knowing at least one of the encryption key in the hierarchy. Also, each node in the hierarchy can have only two type of edges; dependent edge without public value or edge with public value. Let us consider a preferable case from an adversary point of view, where a subscription node in the hierarchy has all incoming and outgoing edges with public edge values. We can argue that, if we are not able to get any non-negligible advantage in getting the key of a preferable node with the help of its all related public edge values in Pub then it implies that same argument will follow for all the other nodes in the hierarchy. Hence, the advantage of getting any key in the hierarchy is negligible.

As a most preferable case, consider a subscription node v in the hierarchy with q incoming edges and 2 outgoing edges where every edge has one associated public edge value. Let q edges between node v and its immediate predecessor nodes $u_1, u_2, ..., u_q$ has public edge values $r_{u_1,v}, r_{u_2,v}, ..., r_{u_q,v}$ respectively where $r_{u_i,v} = k_v \oplus h(K_{u_i,l_v})$ with $1 \leq i \leq q$. Similarly, let the public edge values between v and its two immediate successor nodes w_1 and w_1 are r_{v,w_1} and r_{v,w_2} respectively with $r_{v,w_i} = k_{w_i} \oplus h(K_v, l_{w_i})$, $1 \leq i \leq 2$. We can see that, all $r_{i,j}$ public values corresponding to node v, has one distinct hash component. Therefore, no two public edge values will give any simplified solution to compute target key K_v. Hence, the adversary in game defined in Definition [Key Recovery] will not gain any non-negligible advantage in knowing any encryption key in the hierarchy. Hence, KR advantage of the adversary A against full subscription interval (t_1, t_z) of z time slots in the hierarchy is defined as,

$$Adv_A^{KR}(z) < \epsilon_{KR_z} \quad (1)$$

where ϵ_{KR_z} is negligible function of security parameter τ.

Case $2(a)$: $|(t_a, t_b)| = z - 1$ with $t_a = t_2$ and $t_b = t_z$.

We can have two possible target subscription intervals with $|(t_a, t_b)| = z - 1$ as shown in Figure B.1. First, consider the target subscription interval (t_2, t_z) (as shown in Figure B.1(a)), according to Definition [collusion secure] the adversary A is given access to only subscription key $K_{(t_1,t_1)}$ (i.e. $A_{SH} - Pred(t_2, t_z)$). Figure B.1 shows partition of subscription nodes in the hierarchy into two sets: Set I and Set II. Set I contain nodes in set $A_{SH} - Pred(t_2, t_z)$ and Set II contain nodes in set $Pred(t_2, t_z)$.

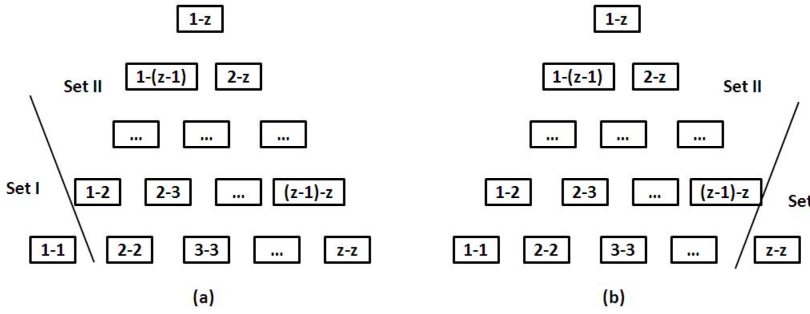


Figure B.1: Type of hierarchies in Case 2

We assume that there exists a polynomial time adversary A which can break the scheme with non-negligible probability. The probability of A outputting correct key is same as the probability of which h_{pre} problem can be broken as shown in lemma 1.

Lemma B.0.1. *The advantage of A in attacking target subscription interval (t_2, t_z) is negligible.*

Proof. Suppose that there exists an adversary A that is able to compute a key $K_{(t_1,t_p)}$ with $p \geq 2$ (i.e. a node in set $S_{(t_2,t_z)}$) with non-negligible advantage. We construct a polynomial time adversary $A_{h_{pre}}$ that, on input $(h(), L, K_{(t_1,t_1)})$, uses the adversary A to compute with non-negligible advantage the value $K_{(t_1,t_p)}$ where $h(L, K_{(t_1,t_p)}) = K_{(t_1,t_1)}$, as follows:

$A_{h_{pre}}(h(), L, K_{(t_1,t_1)})$

1. $A_{h_{pre}}$ knows key $K_{(t_1,t_1)}$ and it does not knows any key $K_{(t_1,t_q)}$ with $q \geq 2$. Let, $K_{(t_1,t_1)} = h_{l_{(t_1,t_1)}}(K_{(t_1,t_a)})$ with $q \geq 2$ where $L = l_{(t_1,t_1)}$.
2. Now there exists either a public edge value $r_{(t_1,t_a),(t_1,t_1)}$. $A_{h_{pre}}$ can compute $h_{l_{(t_1,t_1)}}(K_{(t_1,t_a)}) = K_{(t_1,t_1)} \oplus r_{(t_1,t_a),(t_1,t_1)}$.
3. Or if $K_{(t_1,t_1)}$ is dependent key then there exist a time slot t_a with $a \geq 2$ such that $K_{(t_1,t_1)} = h_{l_{(t_1,t_1)}}(K_{(t_1,t_a)})$.
4. Let $K_{(t_1,t_a)}$ is the output of A on input $(1^m, G, Pub, corr)$ where $corr = \{K_{(t_1,t_1)}\}$.
5. Output $K_{(t_1,t_a)}$.

Since, only way of getting $K_{(t_1,t_a)}$ from $K_{(t_1,t_1)}$ is by computing using $h_{l_{(t_1,t_1)}}(K_{(t_1,t_a)})$. So, we are able to construct an algorithm $A_{h_{pre}}$ which can break the h_{pre} assumption with the same success probability as that of A which was assumed earlier in the proof to have non-negligible probability. But, it is known that h_{pre} assumption is hard and so success probability of A with respect to subscription interval (t_2, t_z) is also negligible. Hence,

$$Adv_A^{KR}(t_2, t_z) < \epsilon_{KR_{z-1}} \quad (2)$$

where $\epsilon_{KR_{z-1}}$ is negligible function of security parameter τ.

$\square$

Case $2(b)$: $|(t_a, t_b)| = z - 1$ with $t_a = t_1$ and $t_b = t_{z-1}$.

In this case we consider the other target subscription interval with $|(t_a, t_b)| = z - 1$, i.e. (t_1, t_{z-1}) shown in Figure B.1(b). With the same reasoning as in case of subscription interval (t_2, t_z) above, we can show that the success probability of A in computing a key of any node in set $S_{(t_1, t_{z-1})}$ is negligible and hence,

$$Adv_A^{KR}(t_1, t_{z-1}) < \epsilon_{KR_{z-1}} \quad (3)$$

where $\epsilon_{KR_{z-1}}$ is negligible function of security parameter τ.

Combining equation (2) and (3),

$$Adv_A^{KR}(z-1) \leq 2\epsilon_{KR_{z-1}} \quad (4)$$

Case 3: General case with $|(t_a, t_b)| < (z - 1)$.

In the general case we have either $t_a > t_1$ and/or $t_b < t_z$. Figure B.2 shows the general case graphically where hierarchy of nodes can be divided into three sets: set I, set II and set III. Set II in the middle represents set $Pred(t_a, t_b)$. Nodes in set I and set III represents the set $corr = keys(A_{SH} - Pred(t_a, t_b))$. If set I exists, then we can consider this case as close to case $2(a)$ where now adversary have possession of more than one keys i.e. $Pred(t_1, t_{a-1})/Pred(t_a, t_b)$ for $a > 1$.

We assume that there exists a polynomial time adversary A which can break the scheme with non-negligible probability. The probability of A outputting correct key is same as the probability of which h_{pre} problem can be broken as shown in lemma 2.

Lemma B.0.2. *The advantage of A in attacking target subscription interval (t_a, t_b) is negligible.*

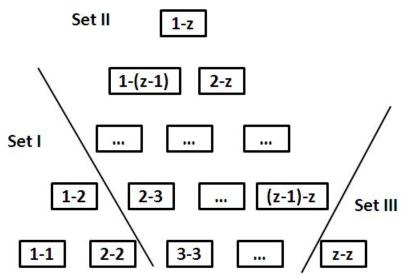


Figure B.2: Type of hierarchies in Case 3

Proof. Suppose that there exists an adversary A that is able to compute a key $K_{(t_c,t_d)}$ i.e. key of a node in set $S_{(t_a,t_b)}$ with non-negligible advantage. We construct a polynomial time adversary $A_{h_{pre}}$ that, on input $(h(), L, K_{(t_e,t_f)})$ where key $K_{(t_e,t_f)}$ is known to A (i.e. node $(t_e, t_f) \in set\ I$), uses the adversary A to compute with non-negligible advantage the value $K_{(t_c,t_d)}$ where $h(L, K_{(t_c,t_d)}) = K_{(t_e,t_f)}$, as follows:

$A_{h_{pre}}(h(), L, S_{(t_a,t_b)})$

1. Let, $A_{h_{pre}}$ knows key $K_{(t_e,t_f)}$ and it does not knows any key $K_{(t_c,t_d)}$ with $(t_c, t_d) \in S_{(t_a,t_b)}$. Let, $K_{(t_e,t_f)} = h_{l_{(t_e,t_f)}}(K_{(t_c,t_d)})$ where $l_{(t_e,t_f)} = L$.
2. Now there exists either a public edge value $r_{(t_c,t_d),(t_e,t_f)}$. $A_{h_{pre}}$ can compute $h_{l_{(t_e,t_f)}}(K_{(t_c,t_d)}) = K_{(t_e,t_f)} \oplus r_{(t_c,t_d),(t_e,t_f)}$.
3. Or if $K_{(t_e,t_f)}$ is dependent key then there exist a node (t_c, t_d) with $(t_c, t_d) \in S_{(t_a,t_b)}$ such that $K_{(t_e,t_f)} = h_{l_{(t_e,t_f)}}(K_{(t_c,t_d)})$.
4. Let $K_{(t_c,t_d)}$ is the output of A on input $(1^m, G, Pub, corr)$ where $corr = keys(set\ I)$ $\bigcup keys(set\ III)$.
5. Output $K_{(t_c,t_d)}$.

Similarly, if set III also exists along with set I, then we can consider this case as close to case $2(b)$ where now adversary have possession of additional keys i.e. $Pred(t_{b_1}, t_z) \setminus Pred(t_a, t_b)$ for $b < z - 1$. Now, the adversary knows the set of keys $corr = keys(set\ I) \bigcup keys(set\ III)$. Since, incoming edges to the nodes in set III have similar types of relationship as the nodes in set I, we follow the same security argument as discussed in case of set I.

Since, only way of getting $K_{(t_c,t_d)}$ knowing $K_{(t_e,t_f)}$is by computing $h_{l_{(t_e,t_f)}}(K_{(t_c,t_d)})$. So, we are able to construct an algorithm $A_{h_{pre}}$ which can break the h_{pre} assumption with the same success probability as that of A which was assumed earlier in the proof to have non-negligible probability. But, it is known that h_{pre} assumption is hard and so success probability of A with respect to subscription interval (t_a, t_b) is also negligible. Hence,

$$Adv_A^{KR}(t_a, t_b) < \epsilon_{KR_{|(t_a,t_b)|}} \quad (5)$$

where $\epsilon_{KR_{|(t_a,t_b)|}}$ is negligible function of security parameter τ.

□

Let, there are n number of such time intervals (t_a, t_b), then we can combine all corresponding inequalities into one as shown below,

$$Adv_A^{KR}(t_a, t_b) \leq \epsilon_{KR_n} \quad (6)$$

where ϵ_{KR_n} is addition of all n negligible functions ($\epsilon_{KR_{|(t_a,t_b)|}}$) and is again a negligible function of security parameter τ.

Inequalities given in (1), (4) and (6) includes all possible subscription intervals in the hierarchy. Hence, adding right-hand side values of all these three inequalities gives maximum KR advantage of the adversary A against given subscription hierarchy of z time slots

$$\begin{aligned} Adv_A^{KR} &\leq \epsilon_{KR_z} + 2\epsilon_{KR_{z-1}} + \epsilon_{KR_n} \\ &\leq \epsilon_{KR} \qquad (7) \end{aligned}$$

where ϵ_{KR} is negligible function of security parameter τ. Hence the proposed scheme is secure against key recovery (KR).

□

CHAPTER C

Formal security analysis of Lab report publishing protocol

We model the four communicating parties *Patient*, *CLab*, *Lab* and *PHRSP* as processes written in ProVerif calculus (see below) with corresponding public ids *P*, *CL*, *L* and *SP*, respectively. *CLab* process work as a Mix node. It randomly chooses a report request out of two input users requests and forwards it to the *Lab*. The main *process* below defines the parallel execution of above processes including two patients processes, one for each *U*1 and *U*2. Let *r*1 and *r*2 are two random numbers used for message synchronization and, *k*1 and *k*2 are two random keys, used by patients *U*1 and *U*2 respectively. Let, *kU*1*C* is the session key between *U*1 and *CL*. Similarly, *kU*2*C* is between *U*2 and *CL*, *kCL* is between *CL* and *L*, and *kLSP* is between *L* and *SP*. *prkL* and *pbkL* are the private and public keys of *L*.

```
(** main process **)
process
new r1:RandNum; new r2:RandNum;
new kCL:Key; new kLSP:Key; new k1:Key;
new k2:Key;
new prkL:prkey; let pbkL=pk(prkL)
in out(c3,pbkL);
out(c3,kLSP);
!((let U=U1 in let r=r1 in let k=k1 in
let kU1C = Sessk(U1) in Patient(kU1C)) |
(let U=U2 in let r=r2 in let k=k2 in
let kU2C = Sessk(U2) in Patient(kU2C)) |
CLab(Sessk(U1),Sessk(U2),kCL) |
Lab(kCL,kLSP,prkL) |
PHRSP(kLSP) )
```

The main process verifies the equivalence between at least two runs where in the first run, *Lab* handles request from one user and in the second run, from another user. Equivalence between these two processes implies that an adversary (including PHRSP) who can listen from the communicating channel cannot distinguish whether the (first) published report (R) request comes from user $U1$ or $U2$. We perform the above process in ProVerif and result shows that the unobservability property holds between the two processes. An important part of the code used for the verification is given below.

```
(** free channels **)
free c0,c1,c2,c3:channel.
const U1,U2,L,CL,SP:UserID.

(** functions Used **)
fun Sessk(UserID): Key [private].
fun hash(bitstring): bitstring.
fun senc(bitstring,Key):bitstring.
reduc forall m:bitstring, k1:Key;
sdec(senc(m,k1),k1)= m.
fun aenck(bitstring, pbkey): bitstring.
reduc forall m:bitstring, k1:prkey;
adeck(aenck(m,pk(k1)),k1)= m.
fun sign(bitstring, prkey): bitstring.

(** Patient Process **)
let Patient(sk:Key) =
1. new r:RandNum; new k:Key; new U:UserID;
2. let m1=aenck((U,L,r_req,r,k),pb(L)) in
```

```
3. let m2=senc((U,m1),sk) in
4. out(c0,(U,m2));                                    (* Send Msg 1 *)
5. in(c0,m3:bitstring);                               (* Receive Msg 7 *)
6. let (w1:bitstring,w2:bitstring) = sdec(m3,sk) in
7. out(c0,(U,senc(w2,sk))).                           (* Send Msg 8 *)

(** CLab Process **)
let CLab(kU1C:Key,kU2C:Key,kCL:Key) =
1. in(c0,m1:bitstring);                              (* Receive Msg 1-1 *)
2. in(c0,m2:bitstring);                              (* Receive Msg 1-2 *)
3. let (U3:UserID,m3:bitstring)=m1 in
4. let m4=sdec(m3,Sessk(U3)) in
5. let (U4:UserID,m5:bitstring)=m2 in
6. let m6=sdec(m5,Sessk(U4)) in
7. let m7=choice[choice[m4,m6], choice[m6,m4]] in
8. out(c1,senc(m7,kCL));                              (* Send Msg 2 *)
9. in(c1,m8:bitstring);                               (* Receive Msg 6 *)
10.let (U5:UserID,w1:bitstring,w2:bitstring) = sdec(m8,kCL) in
11.out(c0,senc((w1,w2),Sessk(U5)));                   (* Send Msg 7 *)
12.in(c0,m9:bitstring);                               (* Receive Msg 8 *)
13.let (U6:UserID,m10:bitstring) = m9 in
14.let m11=sdec(m10,Sessk(U6)) in
15.out(c1,senc(m11,kCL)).                             (* Send Msg 9 *)

(** Lab Process **)
let Lab(kCL:Key,kLSP:Key,prL:prkey) =
1. new Rx:Report; new r1:RandNum;
2. in(c1,m1:bitstring);                               (* Receive Msg 2 *)
3. let (y:bitstring)=sdec(m1,kCL) in
4. let (u:UserID,=L,=r_req,r:RandNum,k:Key)= adeck(y,prL) in
5. out(c2,senc((L,r1,ID_req),kLSP));                  (* Send Msg 4 *)
```

```
6. in(c2,m2:bitstring);                               (* Receive Msg 5 *)
7. let (=r1,uid:UniqueID)=sdec(m2,kLSP) in
8. out(c1,senc((u,senc(UID_to_bitstring(uid),k),hash((uid,Rx))),kCL));
                                                      (* Send Msg 6 *)
9. in(c1,m3:bitstring);                               (* Receive Msg 9 *)
10.let m4=sdec(m3,kCL) in
11.out(c2,senc((uid,Rx,L,sign(hash((uid,Rx)),prL),m4),kLSP)).
                                                      (* Send Msg 10 *)


(** PHRSP Process **)
let PHRSP(kLSP:Key) =
1. in(c2,m1:bitstring);new uid:UniqueID;              (* Receive Msg 4 *)
2. let (=L,r1:RandNum,=ID_req) = sdec(m1,kLSP) in
3. out(c2,senc((r1,uid),kLSP));                       (* Send Msg 5 *)
4. in(c2,m2:bitstring).                               (* Receive Msg 10 *)
```

List of Publications

- Naveen Kumar and Anish Mathuria. ”Improved Write Access Control and Stronger Freshness Guarantee to Outsourced Data ”. Accepted in ”18th International Conference on Distributed Computing and Networking (ICDCN’17) ”, IDRBT, Hyderabad, India, January 2017.

- Naveen Kumar, Anish Mathuria and Maniklal Das. ”Achieving Forward Secrecy and Unlinkability in Cloud-based Personal Health Record System”. 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (IEEE TrustCom/BigDataSE/ISPA (1) 2015), pp. 1249-1254, Helsinki, Finland, 20-22 August, 2015.

- Naveen Kumar, Anish Mathuria and Maniklal Das. "Comparison of Key Management Hierarchies for Access Control in Cloud". Third International Symposium on Security in Computing and Communications (SSCC’15), pp. 36-44, Kerala, 10-13 August, 2015.

- Naveen Kumar, Anish Mathuria, Maniklal Das and Kanta Matsuura. "An Efficient Time-Bound Hierarchical Key Assignment Scheme". Ninth International Conference on Information Systems Security (ICISS’13), pp. 191-198, ISI Kolkata, 16-20 December, 2013.

- Naveen Kumar, Anish Mathuria, Manik Lal Das and Kanta Matsuura. "Improving Security and Efficiency of Time-Bound Access to Outsourced Data". Sixth ACM India Computing Convention: Next generation information, computing and security (Compute’13), pp. 9:1-9:8, VIT University, Vellore, 2013.