Adaptive Cryptographic Access Control for Dynamic Data Sharing Environments

by

Anne Voluntas Dei Massah Kayem

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Doctor of Philosophy

Queen's University
Kingston, Ontario, Canada
October 2008

Abstract

Distributed systems, characterized by their ability to ensure the execution of multiple transactions across a myriad of applications, constitute a prime platform for building Web applications. However, Web application interactions raise issues pertaining to security and performance that make manual security management both time-consuming and challenging. This thesis is a testimony to the security and performance enhancements afforded by using the autonomic computing paradigm to design an adaptive cryptographic access control framework for dynamic data sharing environments.

One of the methods of enforcing cryptographic access control in these environments is to classify users into one of several groups interconnected in the form of a partially ordered set. Each group is assigned a single cryptographic key that is used for encryption/decryption. Access to data is granted only if a user holds the "correct" key, or can derive the required key from the one in their possession. This approach to access control is a good example of one that provides good security but has the drawback of reacting to changes in group membership by replacing keys, and re-encrypting the associated data, throughout the entire hierarchy. Data re-encryption is time-consuming, so, rekeying creates delays that impede performance.

In order to support our argument in favor of adaptive security, we begin by presenting two cryptographic key management (CKM) schemes in which key updates affect only the class concerned or those in its sub-poset. These extensions enhance performance, but handling scenarios that require adaptability remain a challenge. Our framework addresses this issue by allowing the CKM scheme to monitor the rate at which key updates occur and to adjust resource (keys and encrypted replicas) allocations to handle future changes by anticipation rather than on demand. Therefore, in comparison to quasi-static approaches, the adaptive CKM scheme minimizes the long-term cost of key updates. Finally, since self-protecting CKM requires a lesser degree of physical intervention by a human security administrator, we consider the case of "collusion attacks" and propose two algorithms to detect as well as prevent such attacks. A complexity and security analysis show the theoretical improvements our schemes offer. Each algorithm presented is supported by a proof of concept implementation, and experimental results to show the performance improvements.

Co-Authors

1. Anne V.D.M. Kayem, Patrick Martin, Selim G. Akl, and Wendy Powley, "A Framework for Self-Protecting Cryptographic Key Mnaagement", Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2008), Isola di San Servolo (Venice), Italy, October 20-24, 2008, (To Appear), ***is based on Chapter 5***

2. Anne V.D.M. Kayem, Selim G. Akl, and Patrick Martin, "On Replacing Cryptographic Keys in Hierarchical Key Management Systems", Journal of Computer Security, Vol. 16(3), 2008, pp. 289-309, ***is based on Chapter 4***

3. Anne V.D.M. Kayem, Patrick Martin, Selim G. Akl, and Wendy Powley, "A Self-Protective Key Management Framework", 4th International Workshop on Engineering Autonomic Software Systems, held in conjunction with the 17th International Conference on Computer Science and Software Engineering (CASCON 2007), Markham, Ontario, Canada, October 2007 (Position Paper) ***is based on Chapter 5***

4. Anne V.D.M. Kayem, Patrick Martin, and Selim G. Akl, "Heuristics for Improving Cryptographic Key Assignment in a Hierarchy", In Proceedings of the 3rd IEEE Symposium on Security in Networks and Distributed Systems, May 21-23, 2007, Niagara Falls, ON, Canada. pp. 531-536, ***is based on Chapter 3***

5. Anne V.D.M. Kayem, Selim G. Akl, and Patrick Martin, "An Independent Set Approach to Solving the Collaborative Attack Problem", In Proceedings of the 17th IASTED International Conference, Parallel and Distributed Computing and Systems, November 14-16, 2005, Phoenix, AZ, USA. pp. 594-599, ***is based on Chapter 6***

Dedication

To my parents and my three sisters: Veronique, Delphine, and Madeline.

Acknowledgments

There are no words with which to express my deepest gratitude to my supervisors, Dr. Selim G. Akl and Dr. Patrick Martin. They have been, to me, like two fathers for the past four years, prompting me always to do better than my best and encouraging me in moments of difficulty. By allowing me to explore different horizons they enabled me to discover the joy and challenge of designing a research project. Many thanks also to the members of my examination committee: Dr. Sylvia Osborn, Dr. Stafford E. Tavares, Dr. Mohammad Zulkernine, and Dr. Hagit Shakay for their helpful comments. I am also grateful to the Canadian Commonwealth Scholarship Program for financially supporting this research.

I would like to thank my father and mother who always managed to say something to uplift my mood and self-confidence. They helped me stay focused and most especially kept my dreams alive. My sisters have also been a source of constant support for which I will always be grateful. I sometimes think I do not deserve to have you all, because I have done nothing to merit it. Every PhD student needs a family like mine, to help them through the moments of stress and moodiness.

Queen's University has left a deep and very positive impression on me. There are so many I would like to thank. You made my life here memorable. In particular, I'd like to thank Wendy Powley for all the support she has given me throughout these four years. She has been to me a great source of inspiration and encouragement. I also am deeply grateful to her for helping me proof-read some chapters and for the suggestions that helped me improve the quality of this thesis.

Finally, I would like to acknowledge the support that all my friends have given me. Our discussions reminded me there's more to life than research and so much to be gained from philosophy, politics, culture, and history.

Statement of Originality

I, Anne Voluntas Dei Massah Kayem, certify that the work presented in this thesis is original unless otherwise noted. Any published (or unpublished) ideas and/or techniques from the work of others are fully acknowledged in accordance with the standard referencing practices.

Contents

List of Tables

List of Figures

List of Acronyms

ACDR:	Adaptive Collusion Detection and Resolution
ACL:	Access Control List
BLP:	Bell-Lapadula security model
BN:	Brewer and Nash security model
CA:	Central Authority or Key Server
CAC:	Cryptographic Access Control
CAT:	Collusion-Free Akl and Taylor Scheme
CL:	Capability List
CLW:	Clark-Wilson security model
CWSP:	Chinese Wall Security Policy
DAC:	Discretionary Access Control
DBMS:	Database Management System
DCFK:	Detecting Collusion-Free Keys Problem
Triple DES:	Triple Data Encryption Standard
DKM:	Dependent Key Management
FSA:	File Sharing Application
IKM:	Independent Key Management
MAC:	Mandatory Access Control
MCK:	Mackinnon et al. Scheme
MIS:	Maximum Independent Set
NP:	Class of decision problems that can be solved on a Non-deterministic Turing Machine in Polynomial time
RBAC:	Role-Based Access Control
RSP:	Rekey Scheduling Probem
SA:	Security Administrator
SPKA:	Self-Protecting Key Assignment
SPKM:	Sub-Poset Key Management
SPCKM:	Self-Protecting Cryptographic Key Management
SPR:	Sub-Poset Rekeying
TSA:	Timestamp Authority

TSKA:	Timestamped Key Assignment
TSKM:	Timestamped Key Management
TSP:	Travelling Salesman Problem
TSR:	Timestamped Rekeying
XML:	eXtensible Markup Language
XACL:	XML Access Control Language
XACML:	eXtensible Access Control Markup Language

List of Symbols

U_0	Central Authority or Key Server
S	A partially ordered set (poset) of security classes $\{U_0, ..., U_{n-1}\}$
U_i	User group at the i^{th} position in the poset
n	Maximum number of security classes in a poset
t_i	The exponent assigned to the i^{th} security class
K_i	The cryptographic secret key assigned to the i^{th} security class
d_i	Unencrypted data associated with the i^{th} security class
D_{K_i}	Data d_i encrypted with key K_i
M	Product of two large distinct primes p and q
KS	A key set $\{K_0, ..., K_{n-1}\}$
I	An independent or collusion-free set of set of vertices (keys) that is a subset of V
Γ	Time at which an event occurs
G	A subset of V comprising keys assigned to level l in the poset
h	Maximum number of levels in the access control hierarchy defined by the poset
$e_{x,y}$	Exponent situated at the y^{th} position at the x^{th} level
Y	Denotes the maximum number of security classes at any given level in the access control hierarchy
R_H	Secret registry of Verification signatures
T_i	Timestamp associated to the i^{th} security class
H_{K_i,T_i}	Verification Signature associated with the i^{th} security class
N_i	Number of replicas associated with the i^{th} security class
α_i	Degree of availability associated with the i^{th} security class
V	A set of vertices (keys), computed using an exponent set of integers assigned to the security classes or groups in the hierarchy
W_c	The $c^{\texttt{th}}$ preset monitoring interval
λ_i^c	Arrival rate of rekey requests from U_i during the W_c
$Max(\lambda_i^c)$	Preset maximum arrival rate of rekey requests during W_c from U_i
m_i	Sum of rekey requests from U_i during W_c

p_i^c	Probability prediction that current resources levels are inadequate for W_{c+1}
μ_i	Predicted number of rekey request from U_i during W_{c+1}
$e^{-\mu_i}$	Base of the natural logarithm raised to $-\mu_i$
T_R	Total rekey time during W_c at U_i
T_E	Total encryption time during W_c at class U_i
T_C	Total checkpointing time during W_c at class U_i
$E\left(T_R\right)$	Expected T_R
$E\left(T_E\right)$	Expected T_E
$E\left(T_C\right)$	Expected T_C

Chapter 1

Introduction

"A man who makes trouble for others

is making trouble for himself."

–***Chinua Achebe***

By its distributed nature, the Internet offers a considerable scope of opportunities for resource sharing, making it a prime vehicle for building collaborative Web applications. The popularity of applications like YouTube, Flickr, Facebook, and MySpace, highlight a few of a growing number of "social networking" environments whose primary goal is to facilitate information sharing [28]. Yet, collaborative Web applications inevitably raise questions pertaining to data security and availability [13]. Users want firm reassurances that their data is secure and that access to it will occur in a timely manner. Consequently, the expectation is that access control schemes will evolve from their current quasi-static status into a realm of adaptability whereby security will be on par with the performance expectations imposed by service level agreements.

The autonomic computing paradigm has emerged as an interesting approach to resolving problems that involve dynamic adaptation [20, 56]. It has gained popularity as an approach to designing adaptive systems because of its principle of extending existing architectures as opposed to re-designing them [20, 36, 56, 56, 48]. Essentially, the autonomic computing can be defined as an approach to designing computing systems that are self-managing in the sense that they have the ability to control the functioning of applications with very little input from a user, and are self-configuring to ensure that the system is managed efficiently [76, 91]. Therefore, systems modeled using the autonomic computing paradigm have the capacity to run themselves, and at the same time keep the system's complexity invisible to the user [99].

Autonomic computing gives security management a new perspective because it has the potential to facilitate security management in complex scenarios. In comparison to manually managed security systems, autonomic security management has the advantage of being less time-consuming, less prone to error, and less challenging, for the security administrator (SA). Autonomic management will, in essence, allow the SA to set a number of base parameters that will enable a system to manage itself and only revert to the SA for unusual cases. Self-protecting (adaptive) access control can, therefore, be described as an approach to security that is based on the autonomic computing paradigm whereby a system is designed to be self-managing (to reduce human error) and self-configuring (to provide efficient management). Hence, self-protecting access control schemes have the ability to match security with performance in dynamic scenarios. However, although this proposition has gained popularity in many computing domains, its application in the security domain has been met with skepticism and reluctance.

Skepticism is generated by the fact that security mistakes (failures) can have far-reaching consequences. We recall security incidents like the one that occurred in January 2007 when hackers broke into Winners'[1] computer system and stole customer credit card information [79]. Yet, failure in other areas does not generate the same outrage, probably because there is some assurance, or alternative solution. For instance, power failures/surges do not raise the same concerns and yet can be equally, perhaps even more, risky. Reluctance results from the concern security failures raise. Costly law suits and loss of revenue make business owners seek conservative approaches on which they have some assurance of dependability and security weaknesses are covered by clauses that limit their liability.

Hence, there are a number of issues that need to be addressed to motivate users to accept the idea of autonomic control in security schemes [56]. First, is there a way of determining when too much control has been handed over to the system? Second, how can we guarantee that the source of a breach will be easier to trace than in standard security schemes? Although the answers to these questions are not straightforward, the growing complexity of computing system management undeniably tips the balance in favor of autonomic control. Breaches are currently difficult to trace and prevent, and the problem will only become worse with time [20, 99, 67].

1.1 Motivation

Our motivations for pursuing the argument in favor of adaptive security enforced through self-protecting access control schemes are founded on three observations. The first is that even though the ease of use of password authentication schemes

[1]Department store in the US and Canada specialized in clothing, shoes and accessories.

has made them the *de facto* standard of enforcing access control on the web, their susceptibility to increasingly sophisticated cyber-security attacks make cryptographic alternatives or support for authentication schemes attractive. Cryptographic access control (CAC) schemes, unlike authentication schemes that rely on system-specific security policies, have the advantage that their security does not rely on the physical security of the system on which the data resides [2]. Moreover, since CAC schemes use data encryption to enforce access control, unauthorized access is more difficult because the data remains encrypted irrespective of its location, and only a valid key can be used to decrypt it.

Secondly, access control models are prone to failures (security violation or the inability to meet its objectives) that stem from the fact that security designers tend to assume that if security schemes are correctly specified, failure is unlikely.

Thirdly, CAC schemes have failed to gain usage popularity in Web applications because they rely on key management algorithms that place a heavy processing cost on the system. Cryptographic key management (CKM) algorithms handle data security by updating the key that is used to encrypt a data object whenever a situation arises that might compromise the rules of access control. While generating new cryptographic keys can be done within a relatively short period[2], data encryption is time consuming. When key updates occur frequently[3] and involve re-encrypting large amounts of data[4], the key server is unable to complete a re-encryption before the next request arrives. This can result in a cascading effect, where the key server continually stops and restarts the key generation and encryption process, that results in longer

[2]Depending of the key generation function used, it takes about 0.0015 seconds to generate 1 Triple DES key and about 10 seconds to encrypt a file that is ≈ 32 MB in size

[3]Here frequently implies that the intervals between key update requests are about 4-7 seconds long

[4]Data sizes of ≥ 32 MB

response times. Longer response times imply a wider vulnerability window[5] and reduced data availability to satisfy user queries. Queries cannot be handled during the re-encryption process so, longer encryption times affect the query response time.

The CKM algorithms that have been proposed to address the three concerns cited above, can be broadly classified into two categories: ***independent*** and ***dependent*** key management algorithms. Independent key management algorithms operate by assigning each security class in an access control hierarchy a separate key and access to lower classes is only granted to a user belonging to a higher level class if he/she holds the key that explicitly authorizes him/her access. Dependent key management algorithms, on the other hand, assign each class a separate key and users belonging to classes at higher levels are granted access to lower level classes if the key in their possession can be used to mathematically derive the required lower level key. In both cases the reverse is not possible since users belonging to lower level classes are not assigned keys that allow them access to information at higher levels.

Although independent key management algorithms are more flexible and easier to implement in practice, the cost of key distribution is high in comparison to that in dependent key management algorithms because all the keys a user group requires are physically distributed to it. Moreover, the independent key management approach opens up more possibilities for security violations due to mis-managed or intercepted keys [40]. Dependent key management algorithms alleviate these problems by minimizing the number of keys distributed to any group (class) in the hierarchy thus making for a less cumbersome security management policy. Thus, supporting a CAC scheme with a dependent key management algorithm falls in line with the first of the

[5]Period between the emission of a key update request and its satisfaction (key generation, data re-encryption, and distribution of the updated key) by the key server. A wide vulnerability window allows a malicious user, who is supposed to have left the system, more time to cause damage.

three observations we made at the beginning of this section, that CAC schemes make unauthorized access more difficult by using data encryption to protect data irrespective of its location. However, dependent key management algorithms do not perform well in dynamic scenarios because the inter-dependence between the assigned keys in the hierarchy implies that updating any of the keys results in a change of keys and data re-encryptions throughout the entire hierarchy to preserve data security. When these changes occur within short intervals of each other and involve large amounts of data, dependent key management schemes face delays that can result in a failure to meet pre-specified security and performance objectives.

1.2 Thesis Statement

Cryptographic key management schemes based on a dependent key management approach can be extended to provide a basis for adaptive security in web-based data sharing applications. Standard approaches focus on security correctness and efficiency in key management, but overlook cases that involve dynamic key updates [71, 86]. Consequently, when faced with situations that require frequent key updates, the key replacement process opens up windows of vulnerability and creates delays that negatively impact performance.

A number of problems need to be solved to achieve adaptability in key management schemes. First, in previous schemes, replacing any of the keys in the hierarchy triggers a change throughout the entire hierarchy. This method of key replacement is computationally expensive, because it implies additionally, that all the data associated with the updated keys be re-encrypted. Therefore, an algorithm that minimizes the cost of key replacement in a dependent key approach is needed.

Second, in cases where key replacement occurs frequently, the time-lapse between a request for key replacement and the distribution of the updated key can create delays that not only impede performance by widening the window of vulnerability but also increase the response time experienced by non-requesting users awaiting the updated key. The problem of minimizing the window of vulnerability and the response time can be resolved by using a predictive algorithm to anticipate requests for key updates and adjust the associated resources accordingly.

Finally, since the incorporation of adaptability into key management schemes reduces the necessity of interventions by the security administrator, an algorithm for monitoring key generation and usage is required to prevent security violations by authentic users.

1.3 Contributions

The contributions of this thesis are as follows:

1. We present a comparative analysis of standard access control approaches in relation to their vulnerabilities and inefficiencies, showing that, they, like all systems, are prone to failure and need to be supported by fault tolerance solutions, in order to be self-protecting.

2. We propose two new algorithms to minimize the cost of key assignment and replacement. The first algorithm, that we refer to hereafter as the *"Sub-Poset Key Assignment (SPKA)"* scheme, uses a distance based heuristic to guide the assignment of the exponents used to compute user group keys such that a collusion attack is avoided. Collusion occurs when two or more users at the same

level in a hierarchy cooperate using their respective keys and a contrived function to compute a key that belongs to a user class higher up in the hierarchy, to which they are not entitled. The second algorithm, that we refer to hereafter as the *"Sub-Poset Rekey (SPR)"* scheme, operates by using an integer factorization metric to select exponent replacements in a way that bounds the growth of the keys linearly. Additionally, the SPR algorithm allows the key server to replace keys only in the sub-poset associated with the security class affected by the key replacement. These two contributions offer the advantage of a flexible access control hierarchy in the sense that keys can be replaced, deleted, or inserted into the poset without having to re-define a new hierarchy as is the case in previous schemes.

3. We propose a new algorithm to minimize the cost of key replacement (*rekeying*) by associating a timestamp with each valid instance of a key and computing a verification signature from both the timestamp and the key. In order to gain access to data, a user's timestamp and key must yield a currently valid verification signature. Rekeying is handled by updating the timestamp associated with a key, computing a new verification signature that is stored in a secret registry, and secretly transmitting the updated timestamp to the current members of a group. Thus, instead of rekeying and re-encrypting the data in the portion of the poset associated with the key that needs to be replaced, as is the case in the *"Sub-Poset Rekey (SPR)"* scheme [52], the *"Timestamped Rekey (TSR)"* scheme allows the central authority (e.g. key server or security provider) to update only the group's timestamp and verification signature.

4. The second and third contributions minimize the cost of rekeying in general,

but are faced with the problem that the size of the window of vulnerability and the response time per request increases with an increasing arrival rate of rekey requests. The fourth contribution addresses these problems with a key management framework that uses the autonomic computing paradigm [56] to incorporate adaptivity and self-protection into a key management algorithm. The approach allows the key management algorithm to make predictions about future arrival rates of rekey requests and to adjust the number of resources (keys and replicated data) in anticipation. The framework allows a CKM scheme to adapt to changing scenarios by minimizing the response time and the size of the vulnerability window created by frequent rekeying. The functionalities of the framework are structured into six components: the *sensor*, *monitor*, *analyzer*, *planner*, *executor* and *effector*, that are linked together to form a feedback control loop. The feedback control loop continually monitors the arrival rate of rekey requests at the key server and, at regular intervals, computes an acceptable resource (keys and encrypted replicas) allocation plan to minimize the overall cost of rekeying. Each component of the framework contributes to enhancing a standard CKM scheme's performance without changing its underlying characteristics. The advantage of this solution is that the response time for handling an increased arrival rate of rekey requests is minimized, in comparison to the previous two approaches we proposed, because all the key server needs to do is to transmit the new key when a request arrives as opposed to waiting to generate a new key and re-encrypt the associated data when a rekey event occurs.

5. Finally, we present a solution to the problem of "collusion attacks" in key management hierarchies. Our algorithm adaptively assigns keys at each level in the hierarchy such that the possibility of the keys being combined to generate illegal keys is minimized. The algorithm works by mapping the key set onto a graph whose vertices represent the keys generated, and the edges, the possibility that their end points can be combined to generate a "collusion attack". The "collusion free" key set is obtained by using a heuristic to compute an independent set of the vertices. Since the problem of computing a maximum independent set of keys (i.e., a key set that is both collusion free and covers every node in the hierarchy) is NP-complete, we deduce that the problem of detecting all the possible key combinations that can result in collusions is also NP-complete.

1.4 Organization of Thesis

The rest of the thesis is organized as follows. In the next chapter we review the literature on standard access control schemes. We show that, they are prone to failure (security vulnerabilities) and need to be supported by fault[6] tolerant solutions, to be self-protecting. The pros and cons of each scheme are highlighted, evolving gradually through the state of the art to support our argument in favor of adaptive security. Our evaluation of the access control approaches in the literature indicates that the main drawbacks they face are a result of the fact that access control approaches are designed with the assumption that, if correctly specified, failure (security violation or the inability to meet its objectives) is unlikely.

[6]The terms 'fault' and 'vulnerability' are used interchangeably throughout this thesis, though in principle, a vulnerability is a special case of a fault.

In Chapter 3 we present two algorithms that improve on the cost of key assignment and replacement in comparison to previous CKM schemes. We use examples to clarify the explanation of the algorithms. Towards the end of the chapter, a complexity analysis, security analysis, and experimental results indicating the performance improvements, effectiveness, and scalability offered by the proposed algorithms are presented and discussed. However, in the worst case the change occurs at the highest point in the hierarchy, resulting in a change of keys and data re-encryptions throughout the entire hierarchy as is the case in previous schemes.

Chapter 4 addresses this concern by building on the algorithms presented in Chapter 3, with the proposition of a timestamped key management scheme to handle the worst key replacement case. This approach resolves this problem by associating each key with a timestamp. The timestamp and key are used to compute a verification signature that is used to authenticate users before data access is granted. So whenever group membership changes, instead of rekeying and re-encrypting the associated data, only the timestamp is updated and a new verification signature computed. The new scheme is analyzed using a time complexity, security, and experimental analysis and is shown to provide performance improvements as well as providing effective security.

The results presented in Chapters 3 and 4 highlight the fact that, while extensions to standard schemes result in performance improvements for relatively static scenarios where user behavioral patterns can be predicted a priori, their reliance on manual management makes it difficult for these CKM schemes to handle dynamic scenarios effectively. This is the case in particular when rekey events occur frequently and at random points in the key management hierarchy. In Chapters 5 and 6, we present two adaptive security schemes that address the need for dynamic key replacements and also automatic detections of attack possibilities. Specifically, we address the need for adaptive security with a framework and prototype implementation for self-protecting cryptographic key management (SPCKM) that is presented in Chapter 5. We explore the benefits gained by using the autonomic computing paradigm to design the SPCKM framework. A prototype implementation and experiments showing performance improvements demonstrate the effectiveness of the proposed framework.

Following this line of thought, since automated key management implies a lesser degree of control by the SA, Chapter 6 considers a case of how illegal keys might be generated using authentic keys and presents a "collusion" detection algorithm and a "collusion" resolution algorithm for detecting and replacing any such keys. Finally, Chapter 7 summarizes the main contributions of the thesis, offers a critical assessment of the thesis, and identifies some suggestions for further research.

Chapter 2

Distributed Access Control

"Nothing is more difficult, and therefore
more precious, than to be able to decide."
– ***Napoleon Bonaparte***

Distributed systems like the Internet are inherently vulnerable to security threats because of their open architecture and ability to facilitate interactions amongst heterogeneous systems. Although a lot of work has gone into designing schemes to protect data from unauthorized access, the ever evolving applications that arise on the Web create scenarios that increasingly require that good security be matched with performance. Yet, the idea of matching security with performance has received very little attention in the computer security research community because the primary goal of security schemes is to provide good protection rather than efficiency. A good example of such a case is that of cryptographic key management schemes. These schemes provide access control with an added layer of security that makes violation more difficult, but have not gained widespread popularity because of the cost of implementation and the lack of scalability.

Our running example is that of a collaborative web application because it seems to be a suitable application to use to emphasize the need for adaptive security. The proposed collaborative web application involves a hypothetical scenario of a file sharing platform that users can join and leave spontaneously. A security administrator centrally oversees updates and sanctions bad behavior by policing (i.e., a user is completely expelled or suspended temporarily). The group that a user chooses to join determines the privileges of access (read, write, modify, and/or delete) that they are allowed. Real-world collaborative web applications include Chat Systems, Shared White boards, and “social networking” environments like Facebook [32], YouTube [103], Flickr [34], and MySpace [73].

The aim of this chapter is to review the literature on distributed access control paradigms in relation to the issue of matching good security with performance. We show that access control is intertwined with the notion of dependability and therefore access control schemes need to be supported by fault tolerance in order to be self-protecting. As mentioned in the introductory chapter, our hypothesis is that incorporating fault tolerance into a security scheme endows it with the ability to adapt to changing scenarios resulting in better performance and security. For clarity, we begin with some definitions of the common terminology and then proceed to discuss models of access control in distributed systems in relation to the problems we evoked in Chapter 1, Section 1.2.

2.1 Terminology

The notion of “access” control typically suggests that there is an active entity (i.e. a user or application process), with a desire to read or modify a data object (file, database, etc). For simplicity, we will hereafter refer to an entity as a user and a data object as a file. Access control typically involves two steps: *authentication* and *authorization*. In order to authenticate an active user, the distributed system needs some way of attesting that a user is in reality who he/she claims to be. Standard methods of *authentication* include passwords and digital signatures. On the other hand, *authorization* to access a file relies on a set of rules. Although the rules can be specified in a natural language format, this choice is typically avoided in order not to generate ambiguities, inconsistencies, and omissions. A more formal and scientific approach therefore, is to specify the access control rules or *security policies* with a series of mathematical axioms that indicate which users get to access a file.

A *privilege* allows a user to perform a number of well-defined operations on a file. For example, in the collaborative Web application that we described earlier, the security administrator can choose to schedule automatic virus checks with an anti virus application. In this way, the application gets assigned the privilege of scanning all the hard disks and memory on the computers on the network (system) with the aim of eliminating viral threats.

Privileges usually imply some sort of stratification of users each with a clearly defined *role*. A role is a set of *operations* that a user is allowed to perform. A user can have more than one role and more than one user can have the same role [10].

Partial orderings are used to compare the levels of privileges associated with a set of security policies. A partial ordering $\preceq$ on a set of security classes S, such that $S = \{U_0, .., U_{n-1}\}$ where U_i denotes the i^{th} security class, is a relation on $S \times S$ which is:

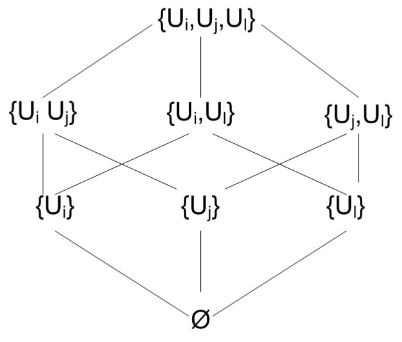


Figure 2.1: Hasse Diagram

- *Reflexive:* For all $U_i \in S$, $U_i \preceq U_i$ holds;
- *Transitive:* For all $U_i, U_j, U_k \in S$, if $U_i \preceq U_j$ and $U_j \preceq U_k$ then $U_i \preceq U_k$
- *Antisymmetric:* For all $U_i, U_j \in S$, if $U_i \preceq U_j$ and $U_j \preceq U_i$ then $U_i = U_j$

Hasse diagrams are a graphical representation of partially ordered sets (posets). A Hasse diagram is a directed graph where the security classes are the elements of the set and the edges in the diagram define the relations that support the partial ordering. So, for $U_i, U_j \in S$ an edge is placed from U_i to U_j if and only if:

- $U_i \preceq U_j$
- There exists no $U_l \in S$ such that $U_i \preceq U_l \preceq U_j$

This definition implies that $U_i \preceq U_j$ holds if and only if there is an edge between U_i and U_j. The Hasse diagram for the partially ordered set $(\{U_i, U_j, U_l\}, \preceq)$ is given in Figure 2.1.

An access control scheme is considered to exhibit *security correctness* when it can provably be shown to provide both *confidentiality* and *integrity* to the system it needs to protect. Confidentiality is the quality that allows a security system ensure that information is accessible only to those authorized to have access. While integrity is provided by specifying rules that prevent malicious or accidental altering of information.

A security system is *dependable* when it ensures availability, reliability, and safety. Availability implies that users with the right privileges always get access to the data, while reliability implies that the data has not been tampered with and is therefore correct. Safety ensures that the system employs a set of rules that prevent a user from making changes that will damage the data.

2.2 General Access Control Models

Security models for access control in distributed systems can be generally classified as either ***discretionary*** or ***mandatory*** [22, 75, 83]. In the discretionary access control approach, access authorizations are determined at the discretion of the owner. Mandatory access control assumes an opposite view in the sense that access control is regulated by controlling the flow of information among communicating users by assigning labels to files to restrict accessibility to authorized users. In the following, we review both the discretionary and mandatory models of access control, highlighting their pros and cons in relation to enforcing adaptive security.

2.2.1 Discretionary Access Control

Discretionary access control (DAC) is based on the privileges a user has with respect to a file. A lot of Web applications use the DAC principle because it is simple and straight-forward to implement in a distributed environment. Using a DAC mechanism allows users control over the access rights to their files without their

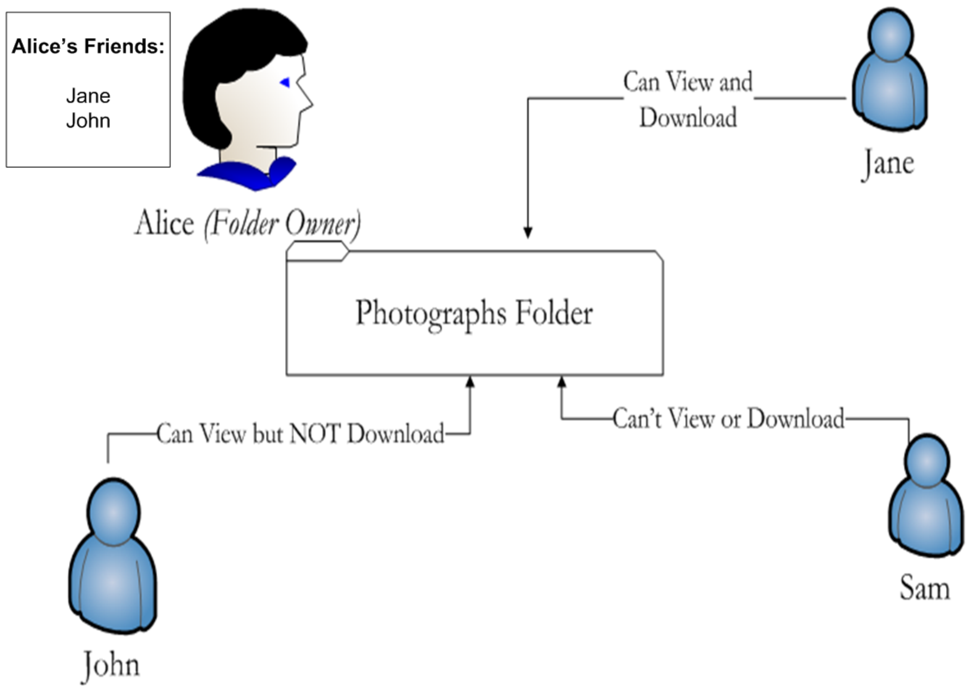


Figure 2.2: Discretionary Access Control

needing to comply with a set of pre-specified rules. When these rights are managed correctly, only those users specified by the file owner may have some combination of read, write, execute, etc. permissions (privileges) on the file [78, 22]. An example of how a DAC model might be used to enforce access control in our collaborative file sharing application is given in Figure 2.2. In this case, a user, say Alice, can choose to create a folder containing photographs she would like to share and allow access only to members who belong in her repertoire of "friends". As Figure 2.2 shows, Jane has the right to view and download the photographs, John only has permission to view the photographs while Sam has no privileges at all with respect to the *Photographs Folder*.

The access control matrix is perhaps the most widely used model for enforcing simple security policies according to the DAC method [37]. Access rights can be

	Photographs Folder	***Skype.exe***	***Movies Folder***
Sam	-	$\{Execute\}$	-
Jane	$\{View, Download\}$	$\{Execute\}$	$\{Execute, Download, Upload\}$
John	$\{View\}$	$\{Execute\}$	$\{Execute, Download\}$

Figure 2.3: An Access Control Matrix

defined individually for each combination of users and files/directory in the form of an access control matrix. An access control matrix that enforces the rules of access depicted in Figure 2.2 is given in Figure 2.3. In this case, the example is extended to allow all three users different rights of access on three different files/directories. Here, the three files/directories are the *Photographs Folder*, *Skype.exe*, and *Movies Folder*. As shown in Figure 2.3, Jane can view and download files from the *Photographs Folder* whereas John can only view the photographs and Sam has no privileges at all with respect to the *Photographs Folder*. On the other hand, all three users can execute *Skype.exe*, as well as download movie files from the *Movies Folder*, but only Jane has the right to upload movies to *Movies Folder*.

The access control matrix is typically not implemented directly if the number of users and files is large or if the groups of users requiring access to the data, and the content of the file, change frequently because of the risk of creating ambiguities or overlaps in the security policy specifications [37]. The access control matrix is sparse, so storing it by rows gives *capability lists*, and by columns gives *access control lists*. A *capability list* is analogous to a ticket for a concert, that is, a user with a "ticket" (access privilege) is allowed entry into the concert hall (file). The *capability list* (CL) specifies privileges of access to various files held by a user. The *access control list* (ACL) on the other hand, is analogous to a reservation book at a restaurant (file) where a customer (user) is allowed seating in the reservation-only restaurant if his/her name appears in the reservation book. The access control list specifies the permissible rights that various users have on a file [22]. Although both implementation approaches appear to be equivalent, they differ in the following ways [22, 64]:

1. **Authentication:** In implementing an ACL, the identification of the user (e.g. driver's license of the restaurant's owner) needs to be authenticated, whereas in CLs, it is the capability (ticket for the concert) that must be authenticated. So, in the case of CLs, the authenticity of the holder of the ticket (user) is secondary and not always required.

2. **Review of Access Rights:** Reviewing access control rights allows a security administrator to determine which users are allowed to access what. In ACLs reviews are trivial since the list contains all the information and is centrally located at the server. ACLs minimize the storage requirements for access control information by using groups to avoid enumerating all users that have common access rights. In CLs on the other hand, it is difficult to review access rights unless an activity log is kept for all the users that are given the capability.

3. **Access Rights Propagation:** In ACLs, the propagation of access rights is explicitly initiated by a request to the data server, which then modifies or adds an entry to its ACL. In CLs, access rights can be propagated from user to user without the intervention of the data server, which can result in a completely uncontrollable system. One of the ways of avoiding this is to oblige the propagation to go through the data server.

4. **Access Rights Revocation:** In ACLs revoking access rights is simplified by

the fact that the privileges granted to any user on the system can be removed simply by deleting them from the list that is typically located at the access control server. On the other hand, in CLs, the process is not as straightforward because the privilege granting user cannot revoke a granted privilege until the period for which it is granted expires. Only then can the granting user either deny giving the requesting user a similar capability or give them one that grants less or more access rights.

Discussion:

Although the access control matrix (ACM) is a good way of providing a standard framework for enforcing access control requirements in distributed systems, it has disadvantages that limit how an access control scheme can meet some security requirements. An example of such a problem is the confinement problem that Lampson [61] cited, which is to determine whether there is a mechanism by which a user authorized to access a file may leak information contained in that file to users that are not authorized to access that file. Harrison et al. [37] formally showed that the confinement problem is undecidable due to the characteristic of discretionary transfer of access rights between users in the ACM model. An added consideration is that although the DAC model is effective for specifying security requirements and is also easier to implement in practice, its inability to control information flow implies it is not well-suited to the context of Web-based collaborative applications where central control in some form is desirable [46]. Moreover, since users applying a DAC security model do not have a global picture of the data on the system, it is difficult to take the semantics of the data into consideration in assigning access rights, so information might unknowingly be revealed to unauthorized users. Table 2.1 summarizes the pros and cons of the ACL and CL approaches to implementing ACMs in the DAC model.

Table 2.1: Comparison: Access Control Lists and Capability Lists

Approach	***ACL***	***CL***
Authentication	User identification	User capability
Review Access Rights	Easy to Review	Not easy unless user capability log is kept
Access Right Propagation	Server Initiated	User to user
Access Right Revocation	Server Initiated	Server waits till rights expire then denies subsequent requests
Location	Server Side	User Side
Storage Requirements	Minimized	Increases

2.2.2 Mandatory Access Control

Apart from the confinement and information semantics problems inherent in the access control matrix model, a key problem that the DAC model faces is vulnerability to Trojan Horse attacks [89]. In order to viloate confidentiality, Trojan Horse attacks exploit two possibilities of access rights management:

- Changes in access rights to a file are handled by the file owner and are not centrally controlled so a malicious user can masquerade as the file owner and grant read-access to a file against the owner's desire

- Users authorized to access a file are typically allowed to create copies of the file, so a malicious user can create a copy of the file or part of it and grant read-access to users to whom the owner has not authorized access

The mandatory access control (MAC) model counters these threats by controlling access centrally. An ordinary user (i.e., not the central authority) cannot change the access rights a user has with respect to a file, and once a user logs on to the system the rights he/she has are always assigned to all the files he/she creates. This procedure allows the system use the concept of information flow control to provide additional security [37]. Information flow control allows the access control system to monitor the ways and types of information that are propagated from one user to another. A security system that implements information flow control typically classifies users into security classes and all the valid channels along which information can flow between the classes are regulated by a central authority or security administrator.

MAC models are typically designed using the concept of information flow control [31, 12]. Information flow control is different from regulating accesses to files by users, as in the ACM model, because it prevents the propagation of information from one user to another. In the MAC model, each user is categorized into a security class and the files are tagged with security labels that are used to restrict access to authorized users [47, 83]. All the valid channels along which information can flow between the classes are regulated [31]. The collaborative file sharing example shown in Figure 2.3 can be extended to handle a security scenario in which a security administrator prevents transitive disclosures, by the users accessing Alice's *Photographs Folder*, by using data labels to monitor information flow. Each data object is tagged with the security clearance labels of each of the users in the system. As shown in Figure 2.4(a.), by extending the discretionary access example we gave in Figure 2.3, a transitive disclosure could occur if a user, in this case Sam, gains access to Alice's photographs folder because he belongs in Jane's (who incidentally is on Alice's list

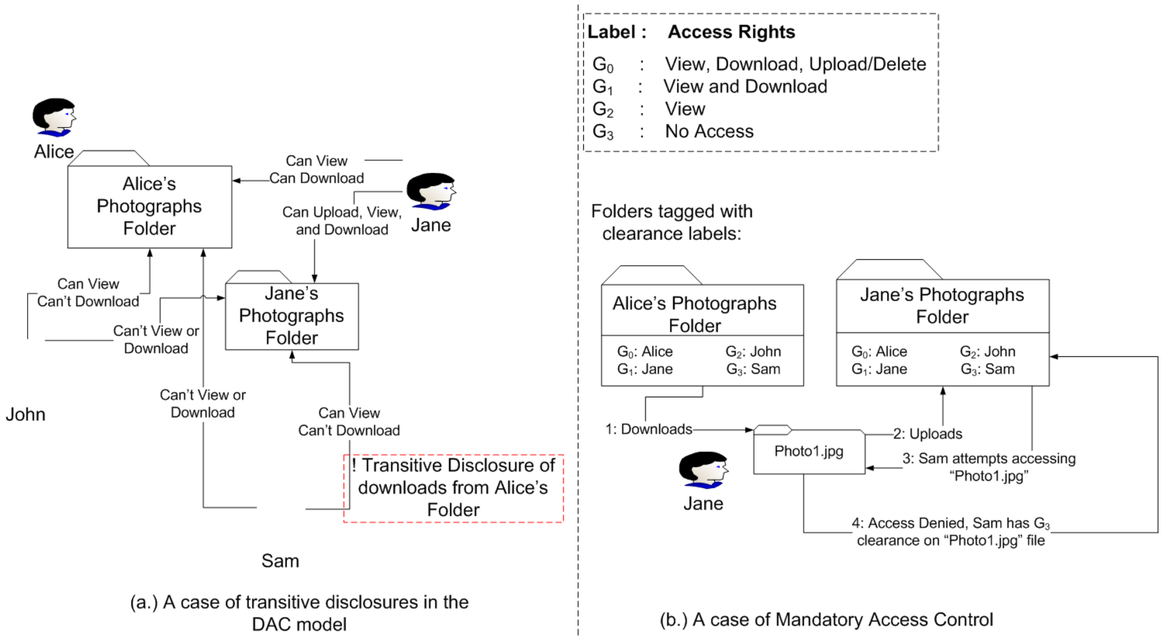

Alice
Alice's Photographs Folder
Can View
Can Download
Can Upload, View, and Download
Jane
Jane's Photographs Folder
Can View
Can't Download
Can't View or Download
John
Can't View or Download
Can View
Can't Download
! Transitive Disclosure of downloads from Alice's Folder
Sam
(a.) A case of transitive disclosures in the DAC model
Label : Access Rights
G0 : View, Download, Upload/Delete
G1 : View and Download
G2 : View
G3 : No Access
Folders tagged with clearance labels:
Alice's Photographs Folder
G0: Alice G2: John
G1: Jane G3: Sam
Jane's Photographs Folder
G0: Alice G2: John
G1: Jane G3: Sam
1: Downloads
Photo1.jpg
2: Uploads
3: Sam attempts accessing "Photo1.jpg"
Jane
4: Access Denied, Sam has G3 clearance on "Photo1.jpg" file
(b.) A case of Mandatory Access Control

Figure 2.4: Mandatory Access Control

of friends) list of "friends". The MAC model prevents such disclosures by defining a hierarchy such as the one depicted in Figure 2.4(b.) to monitor information flow centrally. The users are assigned labels according to their security clearance and information flow is regulated by authenticating a user and then granting access to the file based on their privileges. Since each file is labeled with a security clearance tag, Sam can no longer access files that Jane downloads from Alice's Photographs Folder because Sam does not have a security clearance that allows him access. When the access control policy of a system is based on the MAC model, the security of the system ceases to rely on voluntary user compliance but rather is centrally controlled, making it easier to monitor usage patterns and prevent violations.

2.2.3 Role-Based Access Control

Role-based access control (RBAC) is a combination of mandatory and discretionary access control. In the role-based access control model, a *role* is typically a job function or authorization level that gives a user certain privileges with respect to a file and these privileges can be formulated in high level (e.g. in simple English) or at a low level (e.g. formally specified and hard coded into an application). RBAC models are more flexible than their discretionary and mandatory counterparts in the sense that a user can be assigned several roles and a role can be associated with several users. Unlike the access control lists (ACLs) used in traditional DAC approaches to access control, RBAC assigns permissions to specific operations with a specific meaning within an organization, rather than to low level files. For example, an ACL could be used to grant or deny a user modification access to a particular file, but it does not specify the ways in which the file could be modified. By contrast, with the RBAC approach, access privileges are handled by assigning permissions in a way that is meaningful, because every operation has a specific pre-defined meaning within the application.

Moreover, the RBAC approach to access control is more flexible than the one in the DAC and MAC models in the sense that roles can have overlapping responsibilities and privileges, so users belonging to different roles may need to perform common operations.

Thus, the role in which a user gained membership is not mutually exclusive of another role for which the user already possesses membership. The operations and roles can be subject to organizational policies or constraints and, when operations overlap, hierarchies of roles are established. Instead of instituting costly auditing to monitor access, organizations can put constraints on access through RBAC. For example, it may seem sufficient to allow all the users on the system (Jane, John and Sam) to have 'view' and 'download' access to the *Photographs Folder*, if their accesses are monitored carefully. By using role-based access control, constraints can be placed on user accesses so that they do not tamper with contents of the *Photographs Folder*.

RBAC assumes that all permissions needed to perform a job function can be neatly encapsulated. In fact, role engineering has turned out to be a difficult task [37]. The challenge of RBAC is the contention between strong security and easier administration. On the one hand, for stronger security, it is better for each role to be more granular, thus having multiple roles per user. On the other hand, for easier administration, it is better to have fewer roles to manage. Organizations need to comply with privacy and other regulatory mandates and to improve enforcement of security policies while lowering overall risk and administrative costs. Meanwhile, web-based and other types of new applications are proliferating, and the Web services application model promises to add to the complexity by weaving separate components together over the Internet to deliver application services.

An added drawback that RBAC faces is that roles can be assigned in ways that create conflicts that can open up loopholes in the access control policy. For example in the scenario in Figure 2.2, we can assume that Alice is the security administrator for the *Movies Folder*, and that she chooses to assign roles to users in a way that allows him/her to either download or upload movies but not both. Now suppose that at a future date Alice decides to assign a third role that grants a user, say Sam, the right to veto an existing user's (e.g. Jane's) uploads. In order to veto Jane's uploads, Sam needs to be able to download as well as temporarily delete questionable uploads, verify the movies and, if satisfied, reload the movies to the site. So, essentially Sam has the right to both download and upload movies to *Movies Folder*, a role assignation that conflicts with the first two Alice specified. Since policy combinations create conflicts that can open up vulnerabilities in a security system designed using the RBAC model, extensions to enable adaptability need to be evaluated with care.

2.2.4 Multilevel Access Control

The multilevel security (MLS) model is essentially a special case of how the MAC model is implemented for different contexts or scenarios. In the MLS model, a security goal is set and information flow is regulated in a way that enforces the objectives determined by the security goal [82]. Practical implementations of security schemes based on the MLS concept include the Bell-Lapadula (BLP), Biba Integrity Model, Chinese Wall, and Clark-Wilson models [82, 8, 23, 17, 43, 65]. In the following, we briefly discuss each of these four MLS models but for a detailed exposition of the field one should see the works of McLean [70], Sandhu [85], Nie et al. [74], and Gollmann [37].

- ***The BLP and BIBA models:***

  In the BLP model [8], high level users are prevented from transmitting sensitive information to users at lower levels, by imposing conditions that allow users at higher levels only read data at lower levels but not write to it. On the other hand users at lower levels can modify information at higher levels but cannot read it. Although this method of information flow control prevents sensitive information from being exposed, allowing users at lower levels to write information to files at higher levels that they cannot read can create situations of violations of data integrity that are difficult to trace and correct [65]. The Biba integrity model [11] addresses this problem of data integrity by checking the correctness of all write operations on a file. However, this approach opens up the possibility of security violations that result from inferring high level information from low level information.

- ***The Chinese Wall model:*** In 1989, Brewer and Nash proposed a commercial security model called the Chinese wall security policy [17]. The basic idea is to build a family of impenetrable walls, called Chinese walls, amongst the datasets of competing companies. So, for instance, the Chinese wall security policy could be used to specify access rules in consultancy businesses where analysts need to ensure that no conflicts of interest arise when they are dealing with different clients. Conflicts can arise when clients are direct competitors in the same market or because of ownerships of companies; therefore, analysts need to adhere to a security policy that prohibits information flows that cause a conflict of interest. The access rights in this model are designed along the lines of the BLP model but with the difference that access rights are re-assigned and re-evaluated at every state transition whereas they remain static in the BLP model. Unfortunately, their mathematical model was faulty and the improvements proposed have failed to completely capture the intuitive characteristics of the Chinese wall security policy [105, 63, 64].

- ***The Clark-Wilson (CLW) Model:*** Like the BIBA model, the CLW model addresses the security requirements of commercial applications in which the importance of data integrity takes precedence over data confidentiality [23]. The CLW model uses programs as an intermediate control level between users and data (files). Users are authorized to execute certain programs that can in turn access pre-specified files. Security policies that are modeled using the CLW model are based on five rules:

1. All data items must be in a valid state at the time when a verification procedure is run on it.
2. All data transformation procedures need to be set a priori and certified to be valid.
3. All access rules must satisfy the separation of duty requirements.
4. All transformation procedures must be stored in an append-only log.
5. Any file that has no access control constraints be transformed into one with one or more access control constraints before a transformation procedure is applied to it.

The CLW model is therefore more of a security policy specification framework that extends the concepts in the BIBA model to the general case.

The discussion in this section also illustrates that access control models are typically designed with a set goal and that the scenarios that they are designed for are assumed to be static. Although no single access control scheme can be designed to handle every possible security scenario, web-based security scenarios are increasingly difficult to predict and control manually, which adds a further complication to the problem of designing good security frameworks in the Web environment. In these cases, therefore, the need for good security is intertwined with performance because the delays created in trying to address new situations manually can be exploited maliciously and also affect performance negatively from the user's perspective. In the next section we cover the cryptographic approaches that have been proposed to address the access control problem. Hierarchical cryptographic access control schemes offer the advantage of being simpler to model mathematically and so lessen the SA's burden of security policy specification.

2.3 Cryptographic Access Control

Hierarchical cryptographic access control (CAC) schemes emerged in an attempt to design MLS models that are more general and capable of providing security in different contexts without requiring extensive changes to the fundamental architecture [2, 44, 66, 28]. For instance, in situations that require data outsourcing CAC schemes are useful because the data can be double encrypted to prevent a service provider from viewing the information but yet be able to run queries or other operations on the data and return a result to a user who can decrypt the data using the keys in their possession [28]. CAC schemes are typically modeled in the form of a partially ordered set (poset) of security classes that each represent a group of users requesting access to a portion of the data on the system. Cryptographic keys for the various user groups requiring access to part of the shared data in the system are defined by classifying users into a number of disjoint security groups U_i, represented by a poset $(S, \preceq)$, where $S = \{U_0, U_1, ..., U_{n-1}\}$ [2, 66]. By definition, in the poset, $U_i \preceq U_j$ implies that users in group U_j can have access to information destined for users in U_i but not the reverse. The following paragraphs present a comparative discussion of cryptographic key management (CKM) schemes that are based on the concept of posets of security classes, highlighting the pros and cons of each approach in relation to designing self-protecting access control frameworks.

2.3.1 Key Management Models

Models of key management (KM) are based on the concept of posets and can generally be divided into two main categories: *independent* and *dependent key management* schemes. Independent key management (IKM) schemes originate from the multicast community where the concern is securing intra-group communications efficiently. In these protocols, the focus is on how to manage keys within a group in a way that minimizes the cost of key distribution when the membership of the group changes [40, 104].

IKM schemes approach hierarchical KM by assigning each security class all the keys they need to access information both at their level and below. Accesses are granted only if the user requesting access holds the correct key [40]. So for instance in Figure 2.5(a.), to access the data located at the security classes below it a user U_0 needs to hold all the keys K_1, K_2, K_3, K_4, and K_5 in addition to their own key K_0.

While this method of KM is easier to implement in practical systems because of its flexibility, the cost of key distribution as well as the possibility of security violations due to mis-managed or intercepted keys, is higher than that in dependent key management schemes [40]. In fact, in the worst case scenario where all the keys in the hierarchy are updated, $2n + 1$ keys are redistributed (n represents the maximum number of security classes in the hierarchy), making key re-distribution more costly in comparison to the dependent key management approach where only n keys are re-distributed [40]. As shown in Figure 2.5(a.), to access data $D_{K_0}, D_{K_1}, D_{K_2}, D_{K_3}, D_{K_4}$, and D_{K_5} at classes U_0, U_1, U_2, U_3, U_4, and U_5, a user situated at class U_0 must hold the keys K_1, K_2, K_3, K_4, and K_5 in addition to their own key K_0. So, if a key say K_4, is updated then the new key needs to be re-distributed to all the users in the classes

U_0, U_1, U_2 and U_4 that use it.

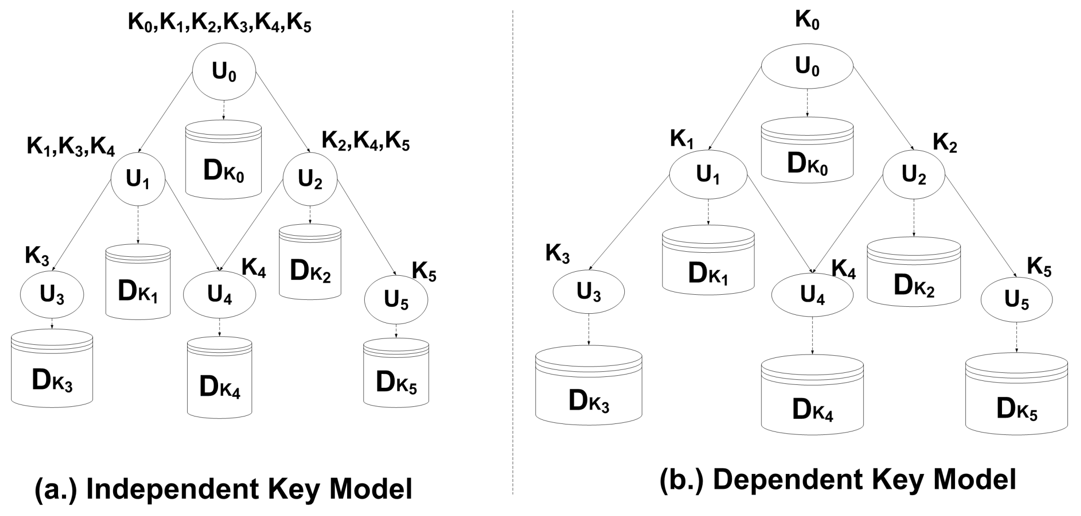


Figure 2.5: Independent versus Dependent Key Management Models

A good way to alleviate these problems is to design the KM scheme in a way that minimizes the number of keys distributed to any security class in the hierarchy. This model, typically referred to as the dependent key management (DKM) scheme, defines a precedence relationship between the keys assigned to the security classes in the hierarchy whereby keys belonging to security classes situated at higher levels in the hierarchy can be used to mathematically derive lower level keys. Access is not possible if the derivation function fails to yield a valid key. So for instance, in Figure 2.5(b.), the data d_1, d_2, d_3, d_4, and d_5 is encrypted with the keys K_1, K_2, K_3, K_4 and K_5 to obtain $D_{K_1}, D_{K_2}, D_{K_3}, D_{K_4}$, and D_{K_5}. Therefore, possession of the key K_1 allows access to D_{K_1}, D_{K_3}, and D_{K_4} since the key is associated with the security class U_1 that is situated at a higher level than the classes U_3 and U_4, and by the partial ordering $U_3, U_4 \preceq U_1$. The reverse is not possible because keys belonging to the lower classes cannot be used to derive the required keys and access information at higher levels.

However, there are a number of drawbacks that impede the performance of dependent key management schemes in dynamic scenarios. For instance, as shown in Figure 2.6, an update request typically requires that the key associated with the security class concerned be replaced.

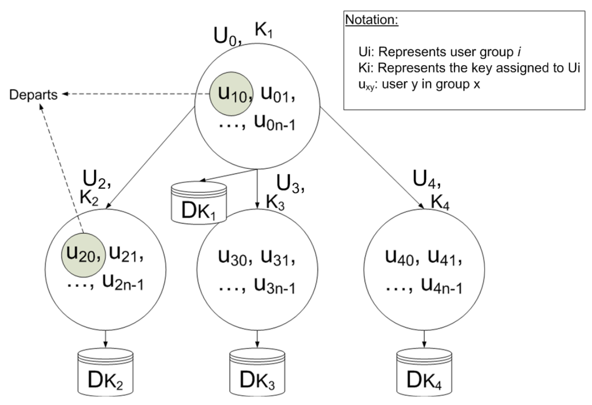


Figure 2.6: Adjusting to Key Updates in a Dependent Key Management Scheme

So when a user u_{10} departs from U_1 both K_1 and the correlated keys K_2, K_3, and K_4 need to be changed to prevent the departed user u_{10} from continuing to access $D_{K_1}, D_{K_2}, D_{K_3}, D_{K_4}$. Likewise, when u_{20} departs from U_2 the keys K_1, K_2, K_3, K_4 need to be changed as well so that K_1 can derive the new K_2 and more importantly to guarantee that the new K_2 does not overlap with K_1, K_3, or K_4 and unknowingly grant U_3 or U_4 access to D_{K_2} or vice versa. This approach to key assignment is not scalable for environments with frequent[1] group membership changes where meeting the goals of service level agreements is an additional constraint. Table 2.2 summarizes the pros and cons of both key management approaches [40, 104].

[1]Here, "frequent" implies that the interval between two rekey events is shorter than the time it takes the key management scheme to define a new one-way function, check its uniqueness, and generate a new key for the node (class) concerned.

Table 2.2: Key Management Models: Comparison

	DKM Model	***IKM Model***
Security	Fewer keys distributed	More Keys distributed
Cost	More re-encryption	Less re-encryption
Inter-group communication	Not flexible	More Flexible
Effect of Rekeying	Changing one key implies updating the whole hierarchy	Change only affected keys, and distribute users requiring the keys
Key Distribution Cost (Number of keys transmitted)	n keys	$2n + 1$ keys

2.3.2 One-Way Function Schemes

Akl and Taylor [2] proposed the first cryptographic key management scheme based on the DKM model. According to their scheme, a central authority (e.g., a security provider or key server) U_0 chooses a secret key K_0 as well as two distinct large primes p and q. The product $M = p \times q$, is computed and made public whereas p and q are kept secret. The access control hierarchy is composed of a maximum of n security classes and each class (group) U_i is assigned a public exponent (an integer) t_i that is used, together with a one-way function[2], to compute a key K_i for the group. In the schemes that Akl and Taylor proposed, this one-way function is expressed as: $K_i = K_0^{t_i} \bmod M$. For simplicity, throughout this section, in analyzing the complexity of a KM scheme, we will assume that a hierarchy is comprised of a maximum of n security classes. We assume also that every key K_i is less than M ($K_i < M$) and that each key K_i requires $O(\log M)$ bits to be represented.

Akl and Taylor suggested two algorithms for assigning the exponents used to compute the group keys. The first algorithm (referred to hereafter as the Ad-hoc AT

[2]A function which is easy to compute but whose inverse is computationally infeasible.

scheme) uses an ad-hoc assignment of exponents that is efficient in time and space complexity, but is vulnerable to collusion attack. The second algorithm (referred to hereafter as the CAT scheme) assigns each group U_i a distinct prime p_i and the exponents are computed from the product of all the p_j's associated with the classes U_j in the poset such that $U_j \not\preceq U_i$. The size of the n^{th} largest prime, in an n group hierarchy is $O(n \log n)$ [58]. Considering that the formula $K_i = K_0^{t_i} \bmod M$ is used to compute the keys, if $K_i < M$ then this implies that K_i requires $O(\log M)$ bits to be represented. In the CAT scheme the largest exponent, t_i, is a product of n primes, so t_i is $O((n \log n)^n)$. Each key in the hierarchy is computed by raising a key to the power of t_i, therefore $\log t_i$ multiplications are required to compute a single key, that is, $O(n \log n)$ multiplications of $O(\log M)$ bit numbers. Finally, to generate keys for the whole hierarchy, since n keys are required in total, we need $O(n^2 \log n)$ multiplications of $O(\log M)$ bit numbers. In the Ad-hoc AT scheme, the size of t_i is $O(\log n)$ since the integer values assigned to the exponents grow linearly with the size of the hierarchy. Therefore generating keys for the whole hierarchy requires $O(n \log n)$ multiplications of $O(\log M)$ bit numbers. In both schemes, key replacements trigger updates throughout the entire hierarchy, so the costs of key generation and replacement are equivalent. We note, however, that when the CAT scheme is used key updates are more expensive than in the Ad-hoc scheme. Therefore, when key updates are triggered frequently throughout the entire hierarchy, rekeying using the CAT approach is computationally expensive.

Mackinnon et al. [66] address this problem with an improved algorithm designed to determine an optimal assignment of keys by attributing the smallest primes to the longest chains in the poset that defines the hierarchy. They use a heuristic algorithm to reduce the number of primes used to obtain the decompositions because obtaining an optimal decomposition of a poset in polynomial time remains an open problem [66]. This improves the time efficiency for key generation in the CAT scheme, because the largest exponent t_i, is a product of approximately half the total number of primes and so, in comparison to the CAT scheme, reduces the total number of multiplications required to compute a key by a half. For example, in Figure 2.7(a.), the initial

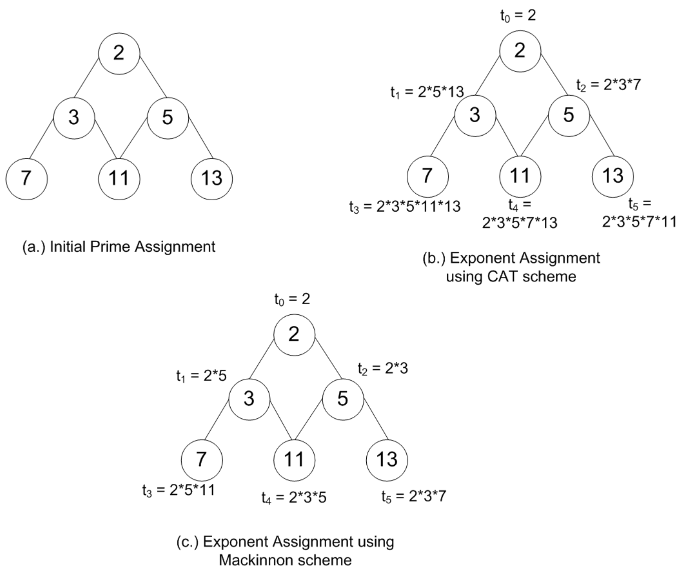


Figure 2.7: Exponent Assignments: A comparison between the CAT and Mackinnon schemes

assignment of primes is $2, 3, 5, 7, 11, 13$ and therefore using the exponent generation algorithm in the CAT gives the exponent assignment shown in Figure 2.7(b.). By contrast, using the exponent generation algorithm in the Mackinnon scheme yields the assignment given in Figure 2.7(c.). Notice that in the last case the largest exponent is a product of $2, 5, 11$ while the largest exponent in the CAT scheme is a product of $2, 3, 5, 7, 13$. The Mackinnon scheme reduces the cost of exponent generation and consequently key generation, in the best case but in the worst case, when the algorithm fails to obtain an optimal decomposition, the complexity bounds for key generation and replacement remain unchanged from what they are in the CAT scheme.

Sandhu [84] proposed addressing the issue of rekeying and security class additions/deletions with a key assignment/replacement scheme that is based on a one-way function. According to his scheme, the security classes are organized in the form of a rooted tree with the most privileged security class, say U_0, being at the root and the least privileged at the leaves. In order to generate keys for the security classes in the hierarchy, the central authority starts by selecting an arbitrary secret key K_0 for the class U_0. The keys for each of the successor classes are created by assigning the classes an identification parameter, $Id(U_i)$, that is then encrypted with the key belonging to its direct ancestor class. For instance, if a class, say U_j, has s children (i.e. security classes at level i directly below it) then the keys for each of the children classes will be computed as follows:

$$\begin{aligned} K_{i_1} &= E_{K_j}(Id(U_{i_1})), \\ \ldots &= \ldots, \\ K_{i_s} &= E_{K_j}(Id(U_{i_s})) \end{aligned}$$

This scheme makes it easier to completely delete or insert new security classes into the hierarchy, because insertions/deletions only require the creation/deletion of the identification parameter associated with the class and encrypting the identification parameter with the ancestor key. However, key replacement at any security class in the hierarchy still requires updating the whole hierarchy to prevent departed users at higher levels from continuing to derive lower level keys and to allow high level keys to continue to be able to derive low level keys. Key replacements also imply updating the affected one-way functions and re-encrypting the data with the new keys. Ensuring the uniqueness of the one-way functions can be challenging in complex hierarchies when group membership changes frequently.

In order to evaluate the complexity of the Sandhu scheme, we assume that the keys are equivalent in size to those obtained in the CAT scheme, i.e. $K_i < M$, where M is the product of two large primes, and K_i requires $O(\log M)$ bits to be represented. We assume also, that in the worst case, the number of encryptions needed to obtain the largest key, in an n group hierarchy is equivalent to the largest prime, so $O(n \log n)$ encryptions of keys of size $O(\log M)$ bits are required to compute a key. Since n keys are required for the whole hierarchy, we need $O(n^2 \log n)$ encryptions of $O(\log M)$ bit numbers to compute all n keys. Rekeying results in an update of keys throughout the entire hierarchy, so the complexity bounds for key generation and rekeying are equivalent.

In order to overcome the drawbacks of the Sandhu scheme, Yang and Li [100] proposed a scheme that also uses a family of one-way functions but limits, to a pre-specified number, the number of lower level security classes that can be directly attached to any class in the hierarchy. The keys for the security classes in the hierarchy are assigned on the basis of the maximum accepted number of classes that can be attached to a security class. A high level class can only derive the key associated with the n^{th} lower level security class attached if there is a path that links it to the lower class and if the high level class key can be used to derive the required low level class key [100]. Limiting the number of lower level security classes attached to a higher level security class reduces the number of keys and encryptions needed but the complexity bounds for generating and replacing the keys remain the same as in the Sandhu scheme. Moreover, Hassen et al. [40] have shown that the scheme does not obey the confidentiality requirements expected of a key management graph (i.e. high level nodes can only derive keys belonging to lower level nodes if they hold a key that is valid and authorizes them access). According to Hassen et al. [40], in the Yang and Li scheme, deleted keys can still be used by users to continue to derive lower level keys since the lower levels keys are not updated when a change occurs at a higher level security class.

Other schemes based on the concept of one-way functions and the DKM model include that proposed by Harn and Lin [38], Shen and Chen [88], and Das et al. [27]. Harn et al. proposed using a bottom-up key generation scheme instead of the top-down approach that the Akl and Taylor schemes use. In the Harn et al. scheme, the smallest primes are assigned to the lower level security classes and the larger primes to the higher level classes. The Harn et al. scheme allows a security administrator to insert new classes into a key management hierarchy without having to update the whole hierarchy. Key insertions/deletions are handled by updating only the keys belonging to higher level classes that have access privileges with respect to the new class. Therefore, key insertions or deletions do not result in key updates throughout the entire hierarchy. Moreover, in comparison to the CAT scheme, the cost of key derivation in the Harn et al. scheme is lower if we consider that key derivation is bounded by $O(\log(t_i/t_j))$ operations (i.e., $O(n \log n)$ time) when $U_i \preceq U_j$. However, the complexity bounds for key generation and replacement remain unchanged from what they were in the CAT scheme because the sizes of the largest prime and key remain the same and rekeying still requires updating all the keys in the hierarchy.

In order to address these issues Shen and Chen [88], and Das et al. [27], proposed using asymmetric cryptography, the Newton interpolation method and a predefined one-way function to generate keys. As in the Harn et al. scheme, both schemes allow security class additions/deletions without requiring updates throughout the entire hierarchy. Key generation is handled by assigning each class (group) a secret key K_i, such that K_i is relatively prime to a large prime number p_i, and a positive integer b_i, such that $1 \leq b_i \leq p_i$. Both K_i and b_i are used to compute the interpolation polynomial $H_i(x)$ and the public parameter Q_i associated with a class U_i. Both $H_i(x)$ and Q_i are made public while b_i and K_i are kept secret. Key derivation is possible if a user's secret key allows them to derive first the b_l associated with the lower class U_l, and then the required secret key K_l. Key updates are handled by replacing only the interpolation polynomial ,$H_i(x)$, and the public parameter Q_i associated with U_i, as well as the old secret key K_i that can be updated to a new one, say K_i'. So all the other keys do not need to be changed.

We know, from our analysis of the CAT scheme, that in an n class hierarchy, the size of the n^{th} largest prime p_i is $O(n \log n)$. Therefore, if $b_i < p_i$, the size of b_i is $O(n \log n)$ [58]. A key K_i requires $O(\log M)$ bits to be represented. According to Knuth [58], computing a k degree interpolation polynomial requires $O(k^2)$ divisions and subtractions and since computing Q_i requires that K_i be raised to the power $1/b_i$, this implies that we need $O(\log n)$ multiplications of $O(\log M)$ bit numbers, in addition to the interpolation time $O(k^2)$, to compute one key. Therefore the n keys in the hierarchy are obtained by $O(n \log n)$ multiplications of $O(\log M)$ bit numbers in addition to $O(nk^2)$ interpolation time. Key derivation is a two step process, first $O(k^2)$ interpolations are required to obtain b_l and next $O(\log n)$ multiplications of

$O(\log M)$ bit numbers are required to obtain K_l. It is also worth noting that these schemes have been shown to be vulnerable to "collusion attack" [107, 41].

We note that in all of the KM schemes that we have discussed in this section, rekeying requires updating the whole hierarchy, so the costs of key generation and replacement are equivalent. We also note that these costs are high, because key replacement requires that the associated data be re-encrypted.

In order to alleviate the costs of key replacement, Atallah et al. [5] proposed a method of updating keys locally, i.e. in the sub-hierarchy associated with the affected security class. In the Atallah et al. scheme, a user belonging to a higher level class uses a hash function to derive the key belonging to a lower level class and the authors show that the scheme is secure against collusion attack. This scheme has the advantage over previous approaches that security classes can be inserted into, and deleted from, the hierarchy without having to change the whole hierarchy. In fact, since each of the paths between the higher and lower level classes is associated with a cryptographic hash function replacing, inserting, or deleting a key is achieved by creating (in case of replacement or insertion), or deleting a key and recomputing a hash function for the affected paths using the new key. Cycles are eliminated from the key management graph, with a shortcut algorithm [4] that operates by dynamically creating direct access paths from high level classes to low level classes. While this improves on the cost of key derivation it adds more paths to the key management graph which goes to further complicate security management and open up potential loopholes for security violations due to mismanaged edge function assignments.

There are a number of additional points worth noting regarding the Atallah et al. scheme. First, in order to derive the key belonging to a lower level class, a user belonging to a higher level class must derive all the keys along the path leading to the target lower level class. By contrast, in the CAT scheme, key derivation is a one step process. Although, the Atallah et al. scheme concludes with some pointers on how to reduce the number of hops performed to reach a lower level class, this comes at the price of additional public key information. Second, when the replacements need to be done throughout most of the hierarchy, rekeying is more costly than in previous schemes because it implies replacing the keys, re-computing all the hash functions on all the affected edges, and re-encrypting the data associated with the updated keys. So, in essence the scheme does well in the best and average cases but performs worse than previous schemes in the worst case scenario. Third, because there is a lot of public information that is computed from the secret key, there is a greater chance that an adversary can correctly guess at the value of the secret key. Finally, although the authors claim that their scheme is secure against collusion, a closer look indicates that this is not the case. In the Atallah et al. scheme replacement requires computing a new secret edge value $y_{i,j}$, and since the old descendant class (group) key is not replaced, once a high level user performs a key derivation operation to obtain a key, he/she continues to have access to the lower level classes until the classes are rekeyed.

In order to evaluate the complexity requirements of the Atallah et al. key management schemes, we assume that there are n groups in the hierarchy. We assume also that largest key $K_i < M$, the size of each secret key is bounded by $O(\log M)$ bits. The largest access key, $y_{i,l}$, is obtained from the function $y_{i,l} = K_l - H(K_i, l_l) \bmod 2^\rho$ where $H(K_i, l_l)$ is a cryptographic hash function, K_l is the key associated with the lower level class connected by the edge to the higher level class, and ρ is a prime number greater than any key value. Since $(H(K_i, l_l) \bmod 2^\rho) < 2^\rho$, this implies that

Table 2.3: One-Way Function Schemes: Time Complexity Analysis Comparison (Here n is the number of security classes in a hierarchy, M is the product of two large primes and is used to generate a key K_i such that $K_i < M$, and s is the number of descendant classes directly accessible from a class U_i.)

Scheme	***Generation***	***Rekeying***
Ad-Hoc AT (Random)	$O(n \log n) \times O(\log M)$	$O(n \log n) \times O(\log M)$
CAT (Primes)	$O(n^2 \log n) \times O(\log M)$	$O(n^2 \log n) \times O(\log M)$
Mackinnon et al.	$O(n^2 \log n) \times O(\log M)$	$O(n^2 \log n) \times O(\log M)$
Sandhu	$O(n^2 \log n) \times O(\log M)$	$O(n^2 \log n) \times O(\log M)$
Yang and Li	$O(n^2 \log n) \times O(\log M)$	$O(n^2 \log n) \times O(\log M)$
Shen and Chen	$O(n \log n) \times O(nk^2)$ $\times O(\log M)$	$O(\log n) \times O(k^2)$ $\times O(\log M)$
Atallah et al.	One $O(\log M)$ bit key and s subtractions of $O(\rho)$ bit numbers from $O(\log M))$ bit numbers	n $O(\log M)$ bit keys and $n-1$ subtractions of $O(\rho)$ bit numbers from $O(\log M))$ bit numbers

$H(K_i, l_l)$ mod 2^ρ requires $O(\log(2^\rho)) = O(\rho)$ bits to be represented. Therefore, in the case where a single class has s edges connecting it to the lower classes directly below it, we need one randomly generated $O(\log M)$ bit key and s subtractions of $O(\rho)$ bit numbers from $O(\log M)$ bit numbers to obtain all the keys required. Since there are n classes in total in the hierarchy, each with s edges connecting it to its direct descendant classes, we need, n randomly generated class keys and a total of $n-1$ subtractions of $O(\rho)$ bit numbers from $O(\log M)$ bit numbers to obtain all the keys required. Table 2.3 summarizes the worst-case complexity costs of all the one-way function key generation algorithms that we have discussed in this section. Additional tables (Tables A.1, A.2, and A.3) summarizing key management schemes are given in Appendix A.

All of these solutions handled the access control problem as though keys assigned to users were intended to be for an indeterminate period and that the only time when it was necessary to regenerate keys was when a new user joined or left the system. In practical scenarios, such as the examples evoked in relation to a collaborative web application, it is likely that users may belong to a class for a short period of time and then get a higher role or be eliminated from the group, which may give them the right to a higher security clearance or disallow them access completely.

2.3.3 Time-Bound Schemes

Tzeng [93] proposed using time bounded keys to avoid replacing most of the keys each time a user is integrated into (e.g., subscriptions to newsletters where new users are not allowed to view previous data), or excluded from, the system. His solution supposes that each class U_j has many class keys K_j^t , where K_j is the key of class U_j during time period t. A user from class U_j for time t_1 through t_2 is given an information item $I(j, t_1, t_2)$, such that, with $I(j, t_1, t_2)$, the key K_i^t of U_i at time t can be derived from K_j^t if and only if $U_i \preceq U_j$ and $t_1 \leq t \leq t_2$. The scheme is efficient in key storage and computation time because the same key can be used with different time bounds for multiple sessions.

The scheme allows a user to keep only their information item $I(i, t_1, t_2)$, which is independent of the total number of classes for the time period from t_1 to t_2, in order to be able to derive all the keys they are entitled to. However the computation of a cryptographic key, say K_j^t, requires expensive public-key and Lucas computations [107, 102] and therefore has limited the implementation of this scheme in the distributed environment. Moreover, Yi and Ye have shown that Tzeng's scheme is vulnerable to collusion attack [102].

Chien [21] addresses the collusion problem in Tzeng's scheme with a solution based on a tamper resistant device but as Yi has shown, this scheme is also vulnerable to collusion attack [101]. More recently, Wang et al. [95] have proposed a time-bound scheme based on merging that creates a correlation between group keys by compressing them into single keys that are characterized with time-bounds. Although this results in better efficiency for key derivation in the average case, the time complexity for key generation is in $O(n \log n)$ (here, n is the maximum number of security classes in a poset that is representative of the access control hierarchy) and that for computing the number of time intervals z associated with a key, in $O(z^2)$. More recent time-bounded key management schemes include [21, 102, 101, 95, 6, 29]. However, it is worth emphasizing that time bounded schemes are not practically efficient for dynamic scenarios where user behavior is difficult to foresee since it is hard to accurately predict time bounds to associate with keys. Table 2.4 summarizes our analysis of the comparative time complexities of the overhead created by the one-way function, and time-bounded schemes.

Table 2.4: A Comparison of the Time Complexities of the Overhead created by the different Key Management Approaches

Approach	***Rekeying***	***Collusion***
One-Way Functions	$O(n^2 \log n) \times O(\log M)$	No for most cases
Time-Bounded	$O(z^2) \times O(n \log n)$	Yes for most cases

2.3.4 Other CKM Schemes

In order to minimize the amount of information distributed during key replacements variants of independent key management schemes that appear in the literature, [88, 59] propose ways of making key updates (distributions) easier and more secure by encrypting the keys that are to be distributed with a public key. The encrypted keys are then placed in some public location and a secret key is transmitted to each group. Access to a particular set of keys is only allowed if a user is in possession of the correct secret key. This makes it easier to exclude users that are compromised and reduces the number of keys distributed but the advantage comes at the cost of added public key information that increases the chances of an adversary correctly guessing at the secret keys being used [26].

Other approaches in the area of secure group communications have proposed batching key update requests in order to minimize the long term cost of rekeying [62]. Batching operates by accumulating requests for key updates during a preset interval at the end of which the keys are then replaced. Although, this improves on the cost of rekeying, it widens the vulnerability window of the key management scheme.

Still along this line of batching key update requests, Crampton has suggested using lazy re-encryption to minimize the cost of data re-encryptions [25]. Lazy re-encryption operates by using correlations in data updates to decide whether or not to update a key and re-encrypt the old data when group membership changes. In this way, since data re-encryption accounts for the larger part of the cost of key replacement, re-encryption is only performed if the data changes significantly after a user departs or if the data is highly sensitive and requires immediate re-encryption to prevent the user from accessing it. The problem of having to re-encrypt the data after a user's departure still remains. Moreover, if the file is a sensitive file that does not change frequently, lazy re-encryption can allow a malicious user time to copy off information from the file into another file and leave the system without ever being detected.

More recently, Ateniese et al.[7] have proposed an improvement on the variant of IKM schemes that Blaze et al. [14] proposed in 1998 whereby proxy-reencryption is used to assign users access to particular files associated with another user or group. Basically, each group or user in the hierarchy is assigned two pairs of keys (a master and a secondary key). The secondary key is used to encrypt files and load them into a block store where they are made accessible to users outside of the group. In order to access encrypted data from the block store a user must retrieve the data and present both the data and their public key to an access control server. The access control server re-encrypts the data in a format that is decryptable with the user's secret key, only if the presented secondary public key authorizes them access. The problem of having to reencrypt, update and distribute new keys when group membership changes remains.

Therefore irrespective of how a key management scheme is designed, rekeying is handled by replacing the affected key and reencrypting the associated data. Rekeying is time-consuming and increases the vulnerability window in a CKM scheme, making it susceptible to two issues: delayed response time in handling key updates and an increased possibility of security violations during the vulnerability window. Additional tables summarizing our comparisons of CKM schemes are given in Appendix A.

2.4 Other Access Control Paradigms

In the previous sections, we presented and discussed standard distributed access control models highlighting the inefficiencies and vulnerabilities that they face in handling dynamic security scenarios on the Web. Although CAC schemes offer a number of advantages over the DAC and MAC models in terms of security, their reliance on costly KM algorithms has hindered their widespread implementation in real world applications. The objective of this section, therefore, is to explore other access control paradigms that have emerged since the creation of the Internet in order to show the extent to which they address the issue of providing access control in dynamic environments like the Web.

2.4.1 Overview

The principal paradigm in distributed systems before the emergence of the World Wide Web had been the client-server architecture [92, 37]. The client-server architecture in its simplest form allows the server to protect itself by authenticating a client requesting access. Kerberos is an example of an authentication service designed for such an environment [92]. This client-server architecture has however changed in many aspects. For instance, when a client looks at a web page, the client's browser will run programs embedded in the page. So, instead of handling simple accesses either to an operating system or a database, programs are being sent from the server to be executed at the client side. Clients receive programs from servers and can store the session states in "cookies". The World Wide Web has also created a new paradigm for software distribution. Software can be downloaded from the Internet and many organizations have learned the hard way to restrict the kinds of programs that they allow their employees to download.

However, while the Internet has not created fundamentally new security problems, it has changed the context in which security needs to be enforced. Consequently, the design of access control paradigms is currently going through a transitory phase in which standard paradigms are being re-thought and evolved to cope with the scenarios that arise on the Internet. The following sections explore some of the changes that are occurring in access control paradigms, highlighting the pros and cons of each in relation to the problem of designing adaptive security schemes that ensure self-protecting access control.

2.4.2 Cookies

The http protocol (hypertext transfer protocol), is a stateless protocol that was originally designed to transfer HTML documents, and is the workhorse of the World Wide Web [37]. Http requests are treated as independent events even when they are initiated by the same client. Web browsers overcome the problem of having to repeat all management tasks associated with a transaction by storing the information entered with the first request and automatically including that information in all subsequent replies to the server. For instance (see Figure 2.2), when Alice allows Jane to download movies from *Movies Folder*, Jane's web browser needs to keep a record of the state of the download operation so that it is able to return a consistent state between Jane's client (browser) and the server *Movies Folder* in case the download operation is interrupted. The browser stores the state of the operation on the client side as a cookie and the server can retrieve the cookie to learn about the client's current state in the operation.

At a basic level, cookies in themselves are not a security problem in the sense that they are executable pieces of code that the server stores at the client-side and so do not pose a problem of confidentiality. A server will only store a cookie on a client that has passed an authentication test. There are however, a couple of application-level attacks that exploit the behavior of cookies. For instance (see Figure 2.2), if

Alice sets up a bonus points loyalty scheme for users of the online telephony system, *Skype.exe*, a client could increase the score to get higher discounts, with a *cookie poisoning attack*. In a *cookie poisoning attack*, the attacker could be a third party that makes an educated guess about a client's cookie and then uses the spoofed cookie to impersonate the client.

Clients can protect themselves by setting up their browsers to control the placement of cookies, obliging the server to request permission before storing a cookie or block cookies, but this can become a nuisance [96]. There is also the option of deleting the cookies at the end of a session or allowing the server to protect itself by encrypting cookies. Spoofing attacks can then be prevented by using proper authentication. However, we note again that in this case, all the attack prevention strategies are statically implemented and as in the approaches we have already discussed, attack types are assumed to be known beforehand.

2.4.3 XML Access Control and Limitations

According to R. Chandramouli [19] eXtensible Markup Language (XML) and XML schema specification languages gained acceptance as standards for representing, interchanging, and presenting both meta data and complex content models in a platform-independent fashion because the XML schema provides a very extensible means for specifying document structures through a comprehensive type definition language. Hence, advocates for XML access control hold that XML is a good candidate for a linguistic framework that is needed to express an access control model that embodies multiple policy requirements.

Considerable effort has gone into developing XML-based frameworks for the specification of access control information. The Organization for the Advancement of Structured Information Standards (OASIS) Extensible Access Control Markup Language (XACML) and IBMs XML Access Control Language (XACL) are access control policy specification frameworks that are mainly geared towards securing XML documents, but they can be applied to other system resources as well. Currently these languages do not provide direct support for representing standard access control models such as DAC or MAC, but a recent extension to XACML incorporates RBAC support [77].

The Extensible Access Control Markup Language (XACML) is a general-purpose language for specifying access control policies [42]. In XML terms, it defines a core schema with a namespace that can be used to express access control and authorization policies for XML objects. Since it is based on XML, it is, as its name suggests, easily extensible. XACML supports a broad range of security policies [19, 42], and uses a standardized syntax for formatting requests so that any one of the following responses to an access request will be valid:

- *Permit:* action allowed
- *Deny:* action disallowed
- *Indeterminate:* error or incorrect/missing value prevents a decision
- *Not Applicable:* request cannot be processed.

As shown in Figure 2.8, XACML's standardized architecture for this decision-making uses two primary components: the Policy Enforcement Point (PEP) and the Policy Decision Point (PDP). The PEP constructs the request based on the user's attributes, the resource requested, the action specified, and other situation-dependent information through Policy Information Point (PIP). The PDP receives the constructed request, compares it with the applicable policy and system state through the Policy Access Point (PAP), and then returns one of the four replies specified above to the PEP. The PEP then allows or denies access to the resource. The PEP and PDP components may be embedded within a single application or may be distributed across a network.

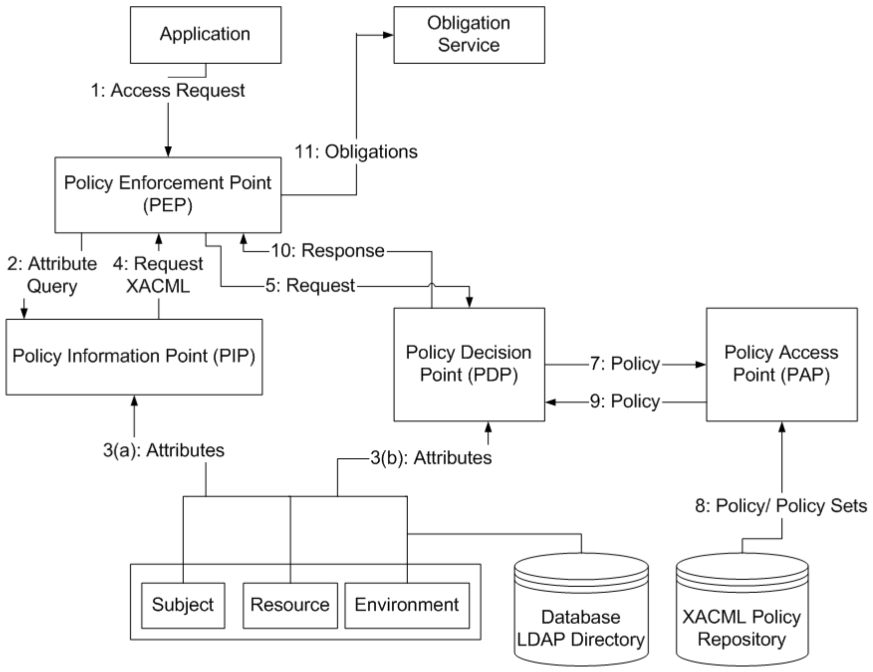


Figure 2.8: XACML Access Control Model [94]

In order to make the PEP and PDP work, XACML provides a policy set, which is a container that holds either a policy or other policy sets, plus links to other policies. Each individual policy is stated using a set of rules. Conflicts are resolved through policy-combining algorithms. XACML also includes methods of combining these policies and policy sets, allowing some to override others. This is necessary because the policies may overlap or conflict. For example, a simple policy-combining algorithm is “Deny Overwrites”, which causes the final decision to be “Deny” if any policy results in an “Overwrite”. Conversely, other rules could be established to allow an action if any of a set of policies results in “Allow”.

Determining what policy or policy set to apply is accomplished using the “Target” component. A target is a set of rules or conditions applied to each subject, object, and operation. When a rule’s conditions are met for a user (subject), object, operation combination, its associated policy or policy set is applied using the process described above.

The associated access control data for a given enterprise domain can then be encoded in an XML document, and the conformance of data to the enterprise access control model can be obtained by validating the XML document against the XML schema that represents the enterprise access control model using XML parsers. These XML parsers are based on standard application programming interfaces such as the Document Object Model (DOM), and the parser libraries are implemented in various procedural languages to enable an application program to create, maintain, and retrieve XML-encoded data.

Although, XML-based and other access control languages provide capabilities for composing policies from scratch, allowing users to specify access control policies, together with the authorizations through the programming of the language, they lack a formal specification language for access control constraints (like historical-based and domain constraints) that prevent assigning overlapping privileges. As an example, consider the case of constraints that require the manipulation and recording of access states (such as granted privileges). This is in order to avoid creating situations that result in users who were previously denied access to certain files being unknowingly granted access in a future state. Like most access control languages, XACML does not provide tools for the expression of historical constraints for historical-based access control policies, thus leaving the completeness of the constraint logics to the policy writer. This case is similar to the one that was evoked in Section 2.2.3 where Alice unknowingly grants Sam a combination of "view" and "download" rights with respect to the *Movies Folder*, by allowing Sam to veto Jane's uploads to the site.

Domain constraints are based on the semantic information pertaining to an enterprise context; a grammar-based language cannot deal with content-based constraints. So, an XML schema is insufficient for a complete specification of the RBAC model for an enterprise since the latter contains content-based domain constraints. An example is not allowing more than one user to be assigned to the role of security administrator (role cardinality constraint) and not allowing the roles viewer and unloader to be assigned to the same user (separation-of-duty constraint).

Here, again we note as before that the specification languages assume a static environment where changes in access control policies are generally effected manually by a security administrator. So in essence, although XML-based access control languages provide features that enable them to specify a broad range of policies, a formal specification is still needed in order to define constraint rules adaptively.

2.4.4 Anti-Viruses, Intrusion Detection, and Firewalls

Other methods of protecting computer systems from unauthorized access include anti-viruses, firewalls and intrusion detection/prevention mechanisms. Anti-virus programs work by detecting and preventing malicious programs from causing damage to the system [87, 16]. The anti-viral program keeps a registry of known malicious code signatures, and removes or cleans any files on the system that are infected with the malicious signature. Since anti-virus programs depend on a registry of known malicious code signatures, the registry needs to be updated constantly to prevent newer variants of viruses from infecting the computer system. The ease with which viruses can spread over the Internet further goes to emphasize the need for adaptive anti-viral programs.

Viruses and malicious code installation on a computer system create amongst other problems that of ***denial of service***. Denial of service attacks that come from one or two sources can often be handled quite effectively. Matters become much more difficult when the denial of service attack is distributed. In distributed denial of service attacks, a collection of malicious processes jointly attempt to bring down a networked service. Typically, the attackers install malicious code on computers that are not well protected and then use the combined power of these computers to carry out the attack. There are basically two types of denial of service attacks: *bandwidth* and *resource depletion.*

Bandwidth depletion attacks are accomplished by sending messages to a single machine with the effect being that normal messages have difficulty getting to the receiver. Resource depletion attacks on the other hand deceive the receiver into using up resources on useless messages. An example of a resource depletion attack is TCP SYN-flooding [92]. Here, the attacker initiates a large number of connections to a server but never responds to the acknowledgments from the server. The result is that the server keeps resending acknowledgment messages that consume bandwidth and slow down network communications.

Since these attacks occur by secretly installing malicious software (malware) on badly protected network devices, intrusion detection algorithms aim to detect and remove such snippets of code from a system before the malware does damage. Intrusion detection systems work to detect malicious behavior in order to alert a security administrator so that some action can be taken to stop the intrusion before the system is damaged irreparably [15, 35].

Firewalls play an added protection role in distributed systems. Essentially, a firewall disconnects any part of a distributed system from the outside world. All outgoing and incoming packets are routed through a special computer and inspected before they are passed. Unauthorized traffic is discarded and not allowed to continue. An important point is that the firewall itself needs to be protected against any form of security threat and should never fail.

There are essentially two categories of firewalls: the packet-filtering gateway and the application-level gateway. The packet-filtering gateway filters incoming and outgoing packets. In contrast to the packet-filtering gateway, the application-level gateway inspects the content of an incoming or outgoing message. An example of an application-level gateway is a mail gateway that discards incoming or outgoing mail exceeding a certain size. More sophisticated mail gateways exist that are capable of filtering spam e-mail.

The common pattern inherent in all the approaches discussed above is the inability to forecast violation scenarios or adapt to new scenarios dynamically. Norton's Symantec Anti-virus software is taking steps towards building pre-emptive anti-virus software that incorporates adaptivity by using machine learning and data mining techniques, which is an indication that professional organizations also recognize the

Table 2.5: Comparison: DAC, MAC, CAC, RBAC, Cookies, and XACML

	DAC	***MAC***	***CAC***	***RBAC***	***XACML***
Control Point	User	Server	Server	Server	Server/User
Authentication (Control Point)	User	Server	Server	Server	Server/User
Review of Access Rights	User	Server	Server	Server	Server/User
Access Right Propagation	User	Server	Server	Server	Server/User
Access Right Revocation	User	Server	Server	Server	Server/User
Implementation	ACL or CL	Lattice Model	Partially ordered Hierarchy	Lattice	Server-based Security Policy
Information Flow Control	None	Yes	Yes	Yes	None unless security policy specified
User-Reliant Security Policy?	Yes	No	No	No	No unless authorized in security policy

need for an evolution towards adaptive security mechanisms [39, 48]. Adaptive intrusion detection algorithms are also still at a budding stage but the idea of moving towards schemes that can adjust to new scenarios is inherent in all these approaches. Table 2.5 summarizes our discussion and comparative analysis of the the DAC, MAC, CAC, RBAC, Cookies, and XACML approaches to access control.

2.5 Autonomic Access Control

The paradigm of autonomic computing emerged in a bid to design applications with the ability to adaptively handle scenarios of varying complexity [56]. Yet, these methods have not gained as much popularity in the domain of access control due to skepticism and reluctance on the part of the users towards autonomic approaches [20]. The main reason behind the skepticism and reluctance is that security breaches create scandals that are expensive to handle and so business owners prefer to opt for security schemes that react in pre-specified and predictable ways, as opposed to those that adapt and evolve dynamically. However, web applications are increasingly faced with scenarios that are difficult to predict a priori, which makes manual security management challenging and prone to error [56, 20]. As mentioned before in Section 2.4.4, breaches created by errors in security policy specifications are currently difficult to trace and prevent, and this will become even harder as systems become more complex [20].

2.5.1 The Autonomic Security Model

Security via the autonomic computing paradigm was first proposed by Chess et al. in 2003 [20]. In order to address the challenge of handling complex situations for which security needs to be ensured, they suggest using the paradigm of autonomic computing that IBM proposed in 2001 [57, 56]. The paradigm of autonomic computing supposes that a system can be designed to self-regulate by using automatic reactions to defend, optimize and heal. The functions of an autonomic system are modeled using a feedback control loop that has two major components: the *autonomic manager* and the *managed resource*. The autonomic manager adjusts the behavior of the managed resource on the basis of recorded observations.

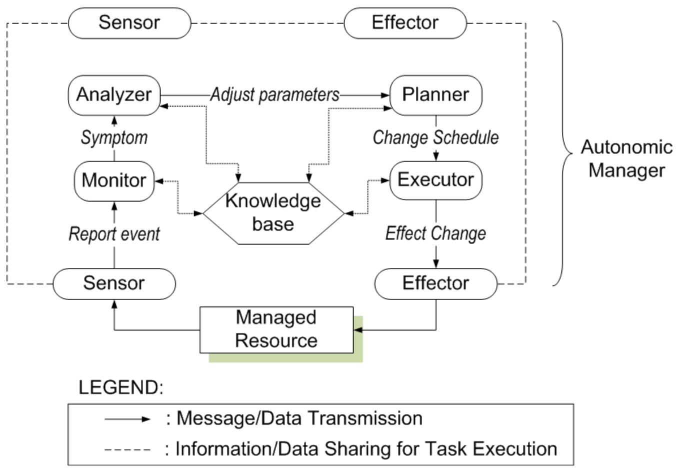


Figure 2.9: The Autonomic Computing Feedback Control Loop

The autonomic model shown in Figure 2.9, is comprised of six basic functions: the *sensor*, *monitor*, *analyzer*, *planner*, *executor*, and *effector*. The sensor captures information relating to the behavior of the managed component and transmits this information to the monitor. The monitor determines whether or not an event is abnormal by comparing observed values to threshold values in the knowledge base. The analyzer, on reception of a message from the monitor, performs a detailed analysis to decide what parameters need to be adjusted and by how much, and transmits this information to the planner where a decision is made on the action to take. The executor inserts the task into a scheduling queue and calls the effector to enforce the changes on the managed resource in the order indicated by the planner.

Autonomic computing aims to provide survivability and fault-tolerance for security schemes [20, 48]. Johnston et al. [48] propose a preliminary approach that uses reflex autonomic computing in the development of a multi-agent security system. This is an interesting approach to self-protecting security, but the authors indicate that real-world implementation of their prototype system would require additional security controls and does not support the ability of a security class to operate independently. As Moreno et al. [72] pointed out, the connection to the rest of the system is lost. We note also that this work on autonomic access control focuses mainly on security policy definitions and restrictions on the messages sent and received by entities (users and/or agents) in the system as opposed to key management for cryptographic access control. The problem of designing adaptive CAC schemes to support a specified security policy definition in general still needs to be addressed.

2.5.2 Perspectives and Discussions

The discussion in this chapter has been centered on the state of the art in distributed systems access control. The pros and cons of each approach were highlighted in relation to the problems that arise in ensuring access control in collaborative web-applications where user group membership is dynamic. Proactive security approaches like access control, are popular because it is easier to prevent damages that result from security loopholes than to wait for a violation to occur and then try to repair the resulting damage caused.

Table 2.6: Comparison: Conventional versus Autonomic Access Control Approach

	Conventional	***Autonomic***
Access Control	Statically operated	Dynamic or Adaptive
Resilience to change	Not Efficient	Aims for enhanced efficiency through adaptivity
Information Flow Control between Security classes	Yes if supported	Yes if supported
Rekey Effect (Key Management)	React on demand	Anticipate demand and adjust accordingly

In analyzing each approach, we noted that access control methods typically face either one or all of three weaknesses: vulnerability to security violations, inefficiency in management resulting in delays as well as reduced availability, and a lack of inbuilt mechanisms that allow them handle new scenarios adaptively (i.e., without a security administrator having to intervene manually). Moreover, the fundamental assumption that all the security solutions make is that if specified correctly, failure (security violation) is unlikely.

However, security attacks show that access control schemes need not only to be supported by some form of fault tolerance (way of minimizing the chances of a vulnerability being exploited by malicious users) but also need to be designed in ways that enable them to adjust their behavior in order to cope with changing scenarios. In the following chapters we focus specifically on cryptographic access control schemes, that are designed using the DKM approach. We show that adaptability can enhance the performance and security of cryptographic access control schemes, without necessarily changing their underlying security specifications. In Table 2.6, we summarize the main attributes of conventional and autonomic access control approaches.

Chapter 3

Efficient Key Management: Heuristics

"Management must manage!"

– ***Harold S. Geneen***

Efficient key assignment and replacement are probably the most controversial aspects of key management in cryptographic access control schemes. Certain schools of thought hold the belief that cryptographic key management should be dynamic to minimize the risk of security violations due to stolen keys. Therefore keys should be updated throughout the entire hierarchy whenever any of the keys need to be updated. On the other hand, the other school of thought holds that key updates typically involve data encryptions that are expensive and therefore it makes sense to reduce the number of keys updated at any time; particularly if the updates occur fairly frequently.

For instance, a common security concern in collaborative web-applications is that of secure access to shared data. Security administrators are faced with two challenges, that of guaranteeing users, *data security* and *availability*. However, while cryptographic key management schemes are designed to provide good security, issues pertaining to availability tend to be relegated to a secondary role. In dependent key management (DKM) schemes in particular, better security comes at the price of expensive key replacements since replacing any of the keys in the hierarchy results in rekeying the entire hierarchy and re-encrypting the associated data.

In this chapter we look at the problem of minimizing the cost of rekeying when a key in the hierarchy needs to be updated. We consider the cases of deleting and adding keys into the hierarchy as being special cases of rekeying, in the sense deletion is similar to the case of updating a key with a new key that is not available to any user (either by direct access or derivation). Insertions are also considered to be a special case of rekeying, where the creation of the new key requires adjusting the hierarchy to incorporate the addition. Our sub-poset key management (SPKM) scheme draws its inspiration from the DKM scheme that Akl and Taylor [2] proposed in 1983. The SPKM scheme comprises two algorithms for exponent assignment and replacement whose goals are to minimize the cost of initial key assignment and subsequent rekeying. The first algorithm uses a distance based heuristic to guide the assignment of the exponents used to compute user group keys such that collusion attacks are avoided. Collusion attacks occur when two or more users at the same level in a hierarchy cooperate using their respective keys and a contrived function to compute a key that belongs to a user class higher up in the hierarchy, to which they are not entitled. The second algorithm employs an exponent replacement algorithm that allows the system to track the validity of the exponents used to form the keys, in order to avoid re-assigning keys that have already been used or that are currently in use. Since our proposed SPKM scheme extends the Collusion-Free Akl and Taylor (CAT) scheme, we begin by briefly reviewing how it operates.

3.1 An Overview of the CAT Scheme

In the Ad-hoc AT and CAT schemes, a security administrator (SA) classifies users into exactly one of a number of disjoint security classes U_i and generates keys for each of the classes using the formula:

$$K_i = K_0^{t_i} \,(\mathrm{mod} M) \qquad (3.1)$$

where $M = p \times q$ is the product of two large primes, t_i is the public exponent that defines a user group's relationship vis-à-vis other groups in the hierarchy and K_0 is a secret key that is selected by the SA situated at the highest level in the hierarchy, usually at U_0. Each of the computed keys K_i is secretly distributed to its respective group U_i (here we assume that that transmission channels are secure, so the keys can be transfered without their being intercepted by a third party).

When the property $U_i \preceq U_j$ holds, and t_j divides t_i evenly (i.e., $t_i \div t_j$ gives an integer value), U_j can derive K_i from K_j with the following computation:

$$K_i = K_0^{t_i} \,(\mathrm{mod}\ M) = K_0^{t_j(t_i \div t_j)} \,(\mathrm{mod}\ M) = K_j^{(t_i \div t_j)} \,(\mathrm{mod}\ M) \qquad (3.2)$$

The security of this algorithm relies on the fundamental assumption that no efficient algorithm exists for extracting roots modulo M, if M is the product of two unknown large primes [81].

Although the ad-hoc assignment of exponents in the Ad-hoc AT scheme is efficient in time and space complexity, Akl and Taylor showed that it is vulnerable to collusion

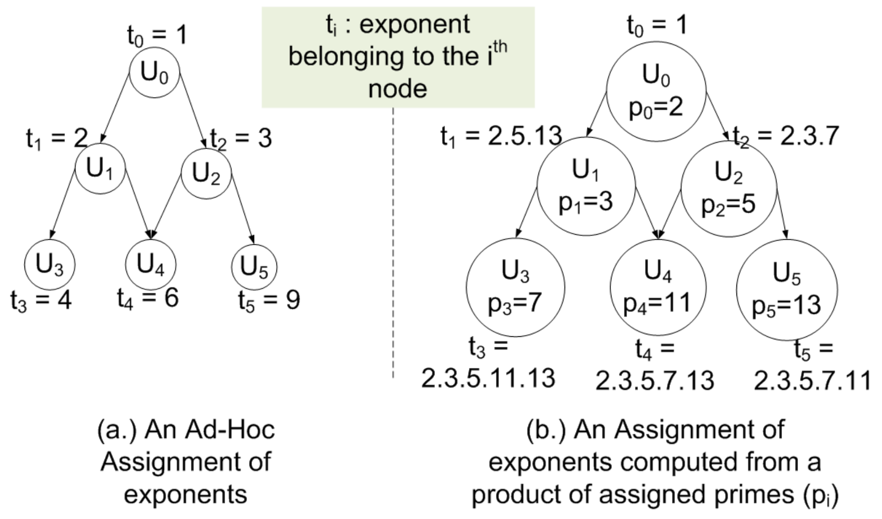


Figure 3.1: Exponent Assignment under the Akl and Taylor Scheme

attacks. For instance in Figure 3.1(a.), two users from U_3 and U_5 can deduce K_0 from the combined product of their keys K_3 and K_5 as follows:

$$(K_3)^{-2}K_5 \bmod M = (K_0^4)^{-2}K_0^9 \bmod M = K_0^{-8}K_0^9 \bmod M = K_0 \bmod M$$

In order to remove the vulnerability to collusions, Akl and Taylor proposed a second algorithm for exponent assignment where the exponents are computed using the formula:

$$t_i = \prod_{U_j \not\leq U_i} p_j \qquad (3.3)$$

where p_j is a product of all the primes that are not associated with the security class under consideration or those that are the sub-poset of the class under consideration. For example, as shown in Figure 3.1(b.), $t_1 = 2 \times 5 \times 13$ which eliminates p_1, p_3, and p_4. Each exponent is a composition of a series of primes and is resistant to collusion attack [2]. However, as we explained in Chapter 2, Section 2.3.1, since this scheme is based on the DKM model, updating, inserting or deleting any of the keys in the hierarchy requires that the whole hierarchy be rekeyed to avoid assigning overlapping keys that could result in violations of the access control policy. In the following sections we present two algorithms that alleviate this problem by assigning the exponents in a way that ensures only the keys in the sub-poset associated affected key are replaced.

3.2 Exponent Assignment Algorithm

Since the performance of any key management (KM) scheme based on the Akl and Taylor model depends on the size of the exponent t_i, our exponent assignment algorithm aims to alleviate the cost of KM by linearly bounding the growth of the size of the exponents assigned to the classes. In order to do this, the exponent assignment algorithm must enforce two conditions, the first is that the exponents be assigned using a distance-based heuristic to bound the growth of the exponents linearly and the second is that an integer factorization metric be used to ensure that the exponents assigned cannot be used to provoke "collusion" attacks [50]. Both conditions can be expressed more formally as follows:

- ***Property 1:*** Avoiding a collusion attack by verifying that the $GCD\{t_i : U_i \in G\}$ does not divide any t_j [2]. Here, G represents the keys that are assigned to the security classes at a level in the hierarchy and t_j represents any of the exponents associated with keys belonging to security classes that are higher up in the hierarchy. For instance in Figure 3.2(a.), collusion is possible because the greatest common divisor at both levels 1 and 2 are equal to 1. On the other hand, collusion is not possible if the assignment in Figure 3.2(b.) is used.

- ***Property 2:*** The growth function of t_i must be linear in order to minimize the cost of key assignment. For instance in Figure 3.2(b.), assigning t_2 a value of 4 as opposed to 20 would mean 3 modular exponentiations in terms of key generation time if equation (3.1), ($K_i = K^{t_i} \bmod M$) is used, as opposed to 8 modular exponentiations.

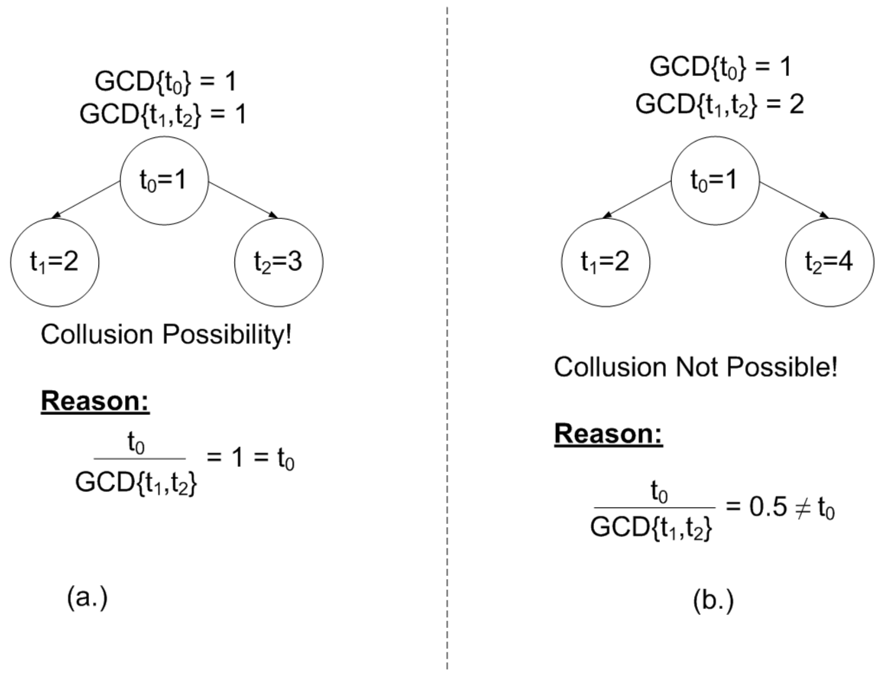


Figure 3.2: An Illustration of a collusion-liable and collusion-free exponent assignment

Therefore, any scheme that obeys the first property provides security against collusion attack [2, 66], while the second property bounds the growth of the exponents linearly to minimize the cost of key assignment and/or derivation. Since we use a poset to model the key management hierarchy, each security class is assigned an exponent t_i that is used to compute the corresponding key K_i, using equation (3.1). In order to guarantee access to the security classes that are lower levels in the hierarchy, the exponent belonging to the higher level security class must be a divisor of the exponent belonging to the lower level class. This allows keys belonging to the lower level security classes to be derived from those belonging to the higher level security classes. Thus, in addition to properties 1 and 2 cited above, secure and efficient key management in the SPKM scheme is conditional on a third property which can formally be stated as follows:

- ***Property 3:*** Accessibility is conditional on the precedence relationship between security classes and exponent divisibility. The precedence relationship is verified by the condition that $U_i \preceq U_j$ and exponent divisibility by $t_i \div t_j$. Hence access to a lower class is granted if and only if t_j belongs to a higher level security class and t_j is a divisor of the exponent t_i belonging to the lower level security class.

3.2.1 Algorithm

The security administrator specifies the size and depth of the hierarchy and hands control to the central authority (CA) *e.g. key server*, where each class in the hierarchy is assigned an exponent that will be used to compute the required secret key. In Figure 3.3, for instance, $e_{j,0}$ indicates the exponent assigned to the class situated at level j and position 0. Integer values are allocated to each class such that the greatest common divisor (GCD) of the exponents at any level, say i (see Figure 3.2), does not divide any of the exponents belonging to higher level , say j, security classes in the poset. A distance based heuristic is used to achieve this by selecting an exponent for the leftmost node at any level such that it does not divide any of the exponents belonging to classes at higher levels. Subsequent values assigned to the classes at the same level are multiples of the value of the leftmost class.

Thus the GCD of the exponents at any level, say i, is equal to the value belonging to the leftmost node and *Properties 1,2, and 3* are verified. The procedure is repeated until every node in the hierarchy has been assigned an exponent. The

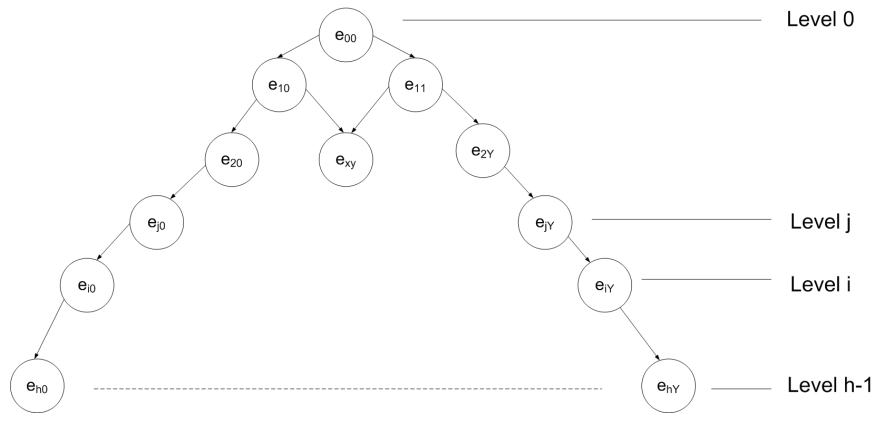


Figure 3.3: An Example of a Key Management Hierarchy

exponents allocated are stored in a secret registry to keep track of their validity and also to facilitate replacement later on. The pseudo code for the exponent assignment algorithm is given in Algorithm 1.

Once the exponents have been assigned successfully, the key server proceeds to select two distinct large primes p and q that are used to compute a product $M = p \times q$. M is made public whereas p and q are kept secret. Finally, the CA selects a secret key K_0 and using equation (3.1), computes keys for the whole hierarchy. The position of an exponent in the KM hierarchy is specified by labeling the exponent in terms of the level and position at which it is situated in that level in the hierarchy. For instance, $e_{x,y}$ indicates the exponent assigned to the security class situated at level x and position y. So we can re-write equation (3.1) as follows:

$$K_{x,y} = K_0^{e_{x,y}} (\mathrm{mod} M) \qquad (3.4)$$

and the keys generated using equation (3.1) are distributed secretly to the users

Algorithm 1 : Exponent Generation (h, nd)

```
Require: $h \geq 1$; $nd \geq 1$
Ensure: $e_{0,0} \leftarrow val$  /*val $\geq 1$: random value in $[1..10]$ */
1:  for $k = 1$ to $h - 1$ do
2:     $e_{k,0} \leftarrow e_{k-1,0} \times \left(nd^k - 1\right)$
3:     if $((e_{k,0} \leq e_{k-1,0}) \vee ((e_{k,0} \bmod e_{k-1,0}) \neq 0))$ then
4:        $e_{k,0} \leftarrow e_{k-1,0} \times 2$  /*Get new value*/
5:     end if;  $j \leftarrow 0$  /*Parent node position*/
6:     for $i = 1$ to $nd^k - 1$ do
7:        if $((i \bmod nd) = 0)$ then
8:           $j \leftarrow j + 1$ /*Shift to next parent node*/
9:        end if;  $e_{k,i} \leftarrow e_{k,0} \times (i + 1)$
10:       $temp \leftarrow i$  /*Bounds growth of $e_{k,i}$ linearly.*/
11:       while $((e_{k,i} = e_{k,i-1}) \vee ((e_{k,i} \bmod e_{k-1,j}) \neq 0))$ do
12:          $e_{k,i} \leftarrow e_{k,0} \times temp$;
13:          $temp \leftarrow temp + 1$
14:       end while
15:    end for
16: end for
```

belonging to the respective security classes in the access control hierarchy and the access control hierarchy is formed.

3.2.2 Exponent Assignment Example

The operation of the exponent assignment algorithm is better explained with the example shown in Figure 3.4, where the security administrator specifies the size and form of the access control hierarchy. In this case the security administrator decides to make each security class have two classes directly attached to it and to have three levels in the hierarchy. A secret registry is used to store both the currently public exponents and those that have already been used in order to keep track of those that have been assigned (exponent validity).

In order to allocate exponents that will be used to compute keys for the security classes in the hierarchy, the CA begins by selecting an exponent $e_{0,0}$ for the highest security class in the hierarchy. In this case $U_{0,0}$ is the only class at the highest level of the hierarchy and it gets assigned a value of 1. Exponents for the remaining classes in the hierarchy are then generated by assigning values to the nodes $e_{1,0}$ and $e_{1,1}$, at level 1 in the exponent graph. An arbitrary value of 2 is selected for node $e_{1,0}$, and since 2 is greater than its ancestor $e_{0,0} = 1$, and Property 1 is verified, $e_{1,0}$ the leftmost node at level 1, takes on 2. The next exponent $e_{1,1}$ then takes on a multiple of $e_{1,0}$, $e_{1,0} \times 2 = 4$. This is the last node at level 1, so we can proceed to level 2. In the same way, at level 2, $e_{2,0}$ is assigned a value of 6 because it is not in the registry and is greater than any of the exponents that have already been assigned to higher classes. As before, this value of 6 is then checked to ensure that Properties 1,2, and 3 are

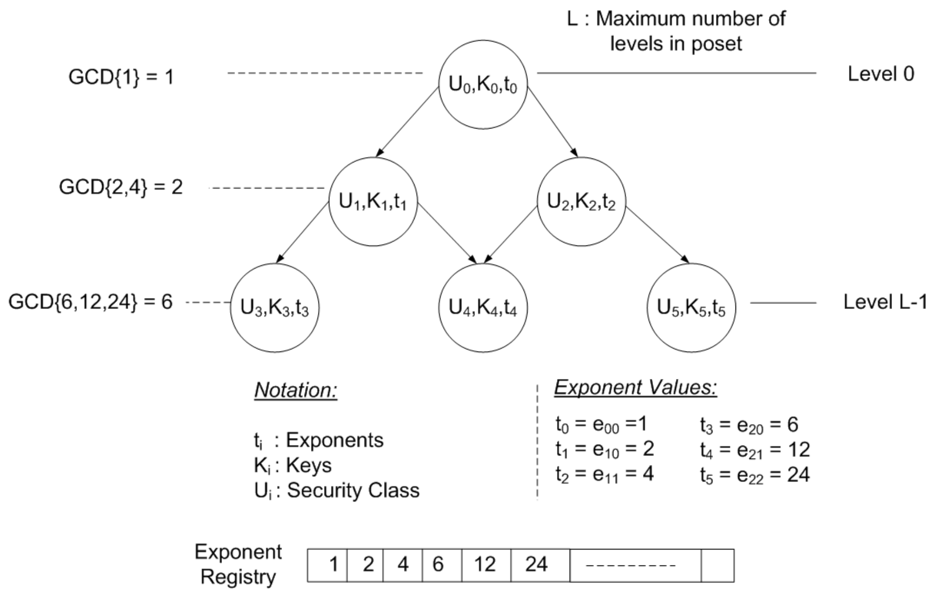


Figure 3.4: Exponent Generation and Allocation

verified. In this case, all three conditions are verified since 6 is a multiple of both 1 and 2 (the exponents at $e_{0,0}$ and $e_{1,0}$ respectively) so Properties 2 and 3 are satisfied, and since the greatest common divisor at level 2 is 6 Property 1 is satisfied as well. The next exponent $e_{2,1} = 12$. Initially, $e_{2,2} = 18$ but $e_{1,1} = 4$ is not a divisor of 18, so this value is discarded in favor of the next 24. This is the exponent associated with the last class at level 2 and also the last in the hierarchy so the CA will now proceed to generate keys for each of the security classes in the hierarchy using equation (3.1).

3.3 Enforcing Hierarchy Updates

In replacing exponents we tackle three aspects: key replacements, deletions, and insertions. Key replacements occur when for some reason the CA decides to update the key associated with a security class. This is the case for instance when there is a change in the membership of a group of users that share a common secret key. In such a situation the CA may choose to update the shared group key to prevent a departed user from continuing to access data available to users in possession of the old key. Key deletions occur when the CA decides to completely eliminate a group from the hierarchy or merge them with another one and finally, key insertions occur when the CA decides to create a new user group. In all three cases some form of key update occurs that results in an adjustment of the KM hierarchy. The following paragraphs outline the operation of the algorithms.

3.3.1 Replacement, Insertion, and Deletion: Algorithm

We first consider what happens when a user departs from any of the groups in the hierarchy. As mentioned before, when this occurs the key belonging to that group needs to be replaced to continue to guarantee data security. The weakness inherent in previous methods was that whenever the key belonging to any class needs replacing, keys for the whole hierarchy were re-generated [66, 2, 50, 43]. The exponent replacement algorithm overcomes this drawback by replacing only the exponent belonging to the affected security class and those that belong to the security classes in its sub-poset.

Algorithm 2 : Exponent Replacement (rk, ri)

```
Require: $rk, ri \geq 0$   /*Position of $e_{k,i}$ being replaced*/
Ensure: $R \neq \emptyset$   /*Registry of exponents*/
 1: for $k = rk$ to $h-1$ do
 2:    $j \leftarrow 0$; $i \leftarrow 1$
 3:    while $(i \leq ri)$ do
 4:       if $((i \bmod nd) = 0)$ then
 5:          $j \leftarrow j+1$  /*Parent node position*/
 6:       end if  $i \leftarrow i+1$
 7:    end while  $e_{k,ri} \leftarrow e_{k,0} \times (ri+1)$
 8:    $temp \leftarrow ri$   /*Bounds growth of $e_{k,ri}$ linearly.*/
 9:    while $((e_{k,ri} \in R) \vee ((e_{k,ri} \bmod e_{k-1,j}) \neq 0))$ do
10:       $e_{k,ri} \leftarrow e_{k,0} \times temp$  /*Multiplicity rule*/
11:       $temp \leftarrow temp + 2$  /*Invalid Exponent,next?*/
12:    end while
13:    if $\left(ri \neq nd^{k-1} - 1\right)$ then
14:       for $i = ri + 1$ to $nd^{k} - 1$ do
15:          if $((i \bmod nd) = 0)$ then
16:             $j \leftarrow j+1$  /*Shift to next parent node*/
17:          end if  $e_{k,i} \leftarrow e_{k,0} \times (i+1)$
18:          $temp \leftarrow i$
19:          while $((e_{k,i} \in R) \vee ((e_{k,i} \bmod e_{k-1,j}) \neq 0))$ do
20:             $e_{k,i} \leftarrow e_{k,0} \times temp$
21:             $temp \leftarrow temp + 1$
22:          end while
23:       end for
24:    end if
25: end for
```

Intuitively, the replacement algorithm operates as follows. On reception of a message indicating a user's wish to depart from the system, the CA computes a new exponent for the node concerned and checks to ensure that it is not in the registry to prevent re-using exponents with the same key K_0 (see equation (3.1)). Next, the exponent is checked to ensure it is a multiple of the exponent belonging to the leftmost node and is a factor of the exponents belonging to the nodes that are lower than it. If this is the case, the new exponent is recorded in the registry and assigned to the node. In the worst case, when no valid exponent can be found that satisfies the properties above, the CA will resort to selecting a new set of exponents for the whole hierarchy or changing the key K_0 and re-assigning keys to the whole hierarchy (this is the case in previous schemes). The pseudo code in Algorithm 2 summarizes the exponent replacement procedure.

In order to delete a key, the CA proceeds by replacing the exponent, associated with the security class concerned, with a value of -1 or 0, and the CA can ensure that Property 3 is not violated by extending the condition to prevent any divisions with 0 or -1. As before, to prevent previous users from continued access, a new key for the class is computed using either exponent -1 or 0 and the data re-encrypted with the new key.

Inserting a new group in the hierarchy requires that an exponent be created at a given level, say k, and a position l. The exponent insertion algorithm works in the same way as the exponent assignment algorithm with the sole difference being that instead of creating a hierarchy we only create a number x of security classes. In order to avoid replicating exponents, the exponent insertion algorithm generates an exponent in conformity with Properties 1,2, and 3 and then checks the exponent registry to ensure that the exponent generated does not already exist. If the exponent does not exist in the registry, a key is created using equation (3.4) and the generated key is secretly distributed to the users wishing to join the newly formed group. Otherwise, the exponent selection process is repeated until a suitable exponent is found.

3.3.2 Insertion, Deletion and Replacement: Example

Consider the case where two users decide to depart from U_5 and U_2 at times T, and $(T+2)$ respectively. Assuming U_5's message arrives before U_2's, the CA selects a new exponent for $e_{2,2}$. As shown in Figure 3.5(a), the key server initially assigns $e_{2,2}$ a value of 30. However, since 30 is not a multiple of $e_{1,1} = 4$, this value is discarded in favor of the available next multiple of $e_{2,0} = 6$, which is 36. The new value for $e_{2,2}$ is recorded in the registry and a new key generated and assigned to the users in U_5.

In order to handle the change in exponents required at U_2 at time $T+2$, as shown in Figure 3.5(b.), the key server initially assigns $e_{1,1}$ a value of 6. However, since 6 already exists in the exponent registry, the next available multiple of $e_{1,0} = 2$ is selected. In this case, $e_{1,1} = 8$ is a multiple of $e_{0,0} = 1$, therefore the new value of 8 is retained for $e_{1,1}$ and recorded in the exponent registry. This assignment implies that the data associated with U_4 and U_5 will no longer be accessible to users at U_2 because 8 is neither a multiple of 12 nor 36. The key server addresses this problem by assigning $e_{2,1}$ the next available multiple of $e_{2,0}$, in this case 42. Since 42 is not a multiple of $e_{1,1} = 8$, the value is discarded in favor of the next multiple of $e_{2,0} = 6$, 48. It turns out that 48 is a multiple of 8 so $e_{2,1}$ is assigned the value 48 that is recorded in the registry. Likewise values of $54, 60$, and 66 are eliminated in favor of 72 in selecting a value for $e_{2,2}$. The new assignment is shown in Figure 3.5(c.).

As shown in Figure 3.5(d.), to delete a key, say K_5, the key server assigns $e_{2,2}$

a value of 0 or -1 and uses the new value of $e_{2,2}$ to compute a new $K_5 = K_{2,2}$ using equation (3.4). Likewise, to insert a security class at level 1 for instance (see Figure 3.5(e.)), the key server creates a new class by assigning $e_{1,2}$ a value of 16. So, at level 1, the $GCD\{2, 4, 16\} = 2$ as before, and the value of 16 is chosen because it obeys Properties 1,2, and 3, and is not in the exponent registry.

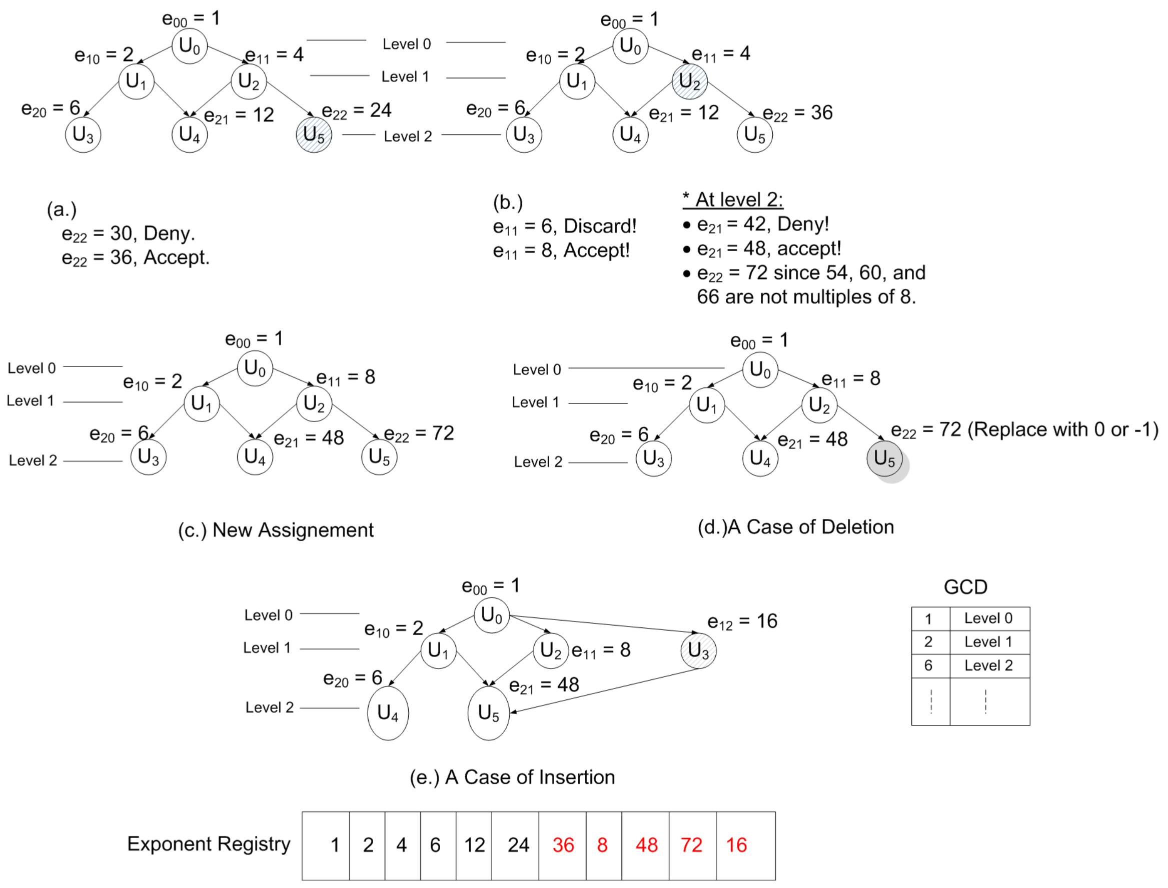


Figure 3.5: Exponent Replacement: Example

When the exponent being replaced falls on a leftmost node, say $e_{2,0}$, all the nodes at that level are replaced to avoid collusion. For example, replacing $e_{1,0}$ with 3 results in $e_{1,1}$ being replaced with 9, $e_{1,2}$ with 18 and $e_{2,1}$ with 54. Likewise, replacing the exponent $e_{0,0}$ could result in a complete change of the hierarchy. We have assumed however, that in practical scenarios, replacements at the root node occur rarely and leftmost nodes can be used for user groups where membership changes occur rarely.

3.4 Analysis

This section presents a security and complexity analysis of the algorithms proposed, in comparison to the CAT scheme and the Mackinnon et al. (MCK) scheme. This is because the SPKM and MCK schemes are both inspired by the CAT scheme. The other schemes mentioned in Chapter 2 (section 2.3), either have complexity bounds for key generation and management that are similar to the MCK scheme, are not adapted to highly dynamic scenarios (e.g. time-bounded schemes), and/or sacrifice security for performance [62, 84, 93, 95]. The security analysis gives an indication of the security guarantees that the proposed SPKM scheme provides, while the complexity analysis presents a theoretical analysis of the cost of key management (generation, replacement, and derivation).

3.4.1 Security Analysis

The security analysis of the SPKM scheme is centered on the security of the key K_i. With respect to the security of K_i, there are two possible attacks. In the first attack a user at some level l, belonging to a class U_l attempts to access information at a higher level or any other group in the hierarchy that they are not authorized to access. The second attack involves two or more users at some level l colluding to compute an authorized key K_i higher up in the hierarchy.

Akl and Taylor showed that if the greatest common divisor of the exponents at level l does not divide any of the exponents at level i, then collusion is not possible [2]. A special case involves a user at level l attempting to derive a key (K_i) at level i. From equation (3.2), that is $K_i = K_0^{t_i} \bmod M = K_0^{t_l(t_i \div t_l)} \bmod M = K_l^{t_i \div t_l} \bmod M$, since $t_i \div t_l$ does not yield an integer value, hence K_i will not be derivable from K_l.

Concerning the security of the exponent t_i, a malicious user who no longer belongs in a group might attempt to guess at the value of t_i associated with the key K_i. This attack is forestalled by keeping secret the key K_0 that is used to compute the new K_i with the newly generated t_i. Moreover, the central authority avoids reassigning exponents that have already been applied to keys by parsing the registry before assigning a new exponent to a group.

3.4.2 Complexity Analysis

We analyze the SPKM scheme in relation to the CAT and MCK schemes, because the three schemes use the same key generation function, i.e. $K_i = K_0^{t_i} \bmod M$. Assume that there are n groups in the hierarchy, and that the size of the largest distinct prime p_j assigned to a group U_j, is $O(n \log n)$ bits. In the CAT and MCK schemes the formula $t_i = \prod_{U_j \not\preceq U_i} p_j$ is used to obtain the largest t_i and t_i is $O((n \log n)^n)$, so the space required to store any t_i is $O(n \log n)$ bits. If $K_i < M$, the size of each key K_i, is bounded by $O(\log M)$ bits. Therefore, $O(n \log n)$ multiplications of $O(\log M)$ bit numbers are required to compute a key K_i. Generating keys for the whole hierarchy (n groups) requires $O(n^2 \log n)$ multiplications of $O(\log M)$ bit numbers. In the CAT and MCK scheme, rekeying involves changing the keys throughout the entire hierarchy, therefore the cost of rekeying is equivalent to the cost of key generation for the entire hierarchy.

Table 3.1: Comparative Time Complexity Analysis: SPKM, CAT, and MCK Schemes

	SPKM	CAT and MCK
Generation	$O\left(n \log n\right) \times O(\log M)$	$O\left(n^2 \log n\right) \times O(\log M)$
Replacement	$O\left(n \log n\right) \times O(\log M)$	$O\left(n^2 \log n\right) \times O(\log M)$
Derivation	$O\left(\log n\right)$	$O\left(n \log n\right)$

In the SPKM scheme, assuming also that we have a hierarchy of n groups and that the largest t_i is less than or equal to the value of the nodal degree raised to the power of n, we deduce that $O(\log n)$ bits are required to represent each t_i. Since $K_i < M$ and K_i requires $O(\log M)$ bits to be represented, computing each K_i requires $O(\log n)$ multiplications of $O(\log M)$ bit numbers. Therefore, in order to obtain all n keys in the hierarchy, we need $O(n \log n)$ multiplications of $O(\log M)$ bit numbers. In the worst case, the cost of rekeying in the SPKM scheme is equivalent to the cost of generating keys for the whole hierarchy, so the costs of key generation and rekeying are equivalent. In order to derive K_i from K_j the user in U_j needs to perform $O(\log(t_i/t_j))$ operations and so derivation takes $O(n \log n)$ time using the CAT scheme whereas the SPKM scheme requires $O(\log n)$ time. Table 3.1 summarizes the complexity analysis.

3.5 Experimental Setup and Results

This section describes our experimental setup we used to run experiments to evaluate our SPKM scheme in comparison to the CAT and MCK schemes, and experimental results showing performance evaluations. We have chosen to limit the comparison to both the CAT and MCK schemes because all three schemes rely on the same key generation function (i.e. equation (3.1)) and the main factor that causes the difference in key generation or replacement times, is the size of the exponent used to compute the keys. We begin by describing the criteria used to perform the evaluations and then proceed to describe the platform on which we conduct the experiments.

3.5.1 Implementation and Experimental Setup

In order to perform our evaluations we use four metrics: key generation time, data encryption time, key replacement time, and the size of the window of vulnerability. We explain the criteria in a little more detail below:

- *Key Generation Time:* This is the time it takes the algorithm to generate the exponents required to compute the keys and then use the generated exponents to compute the keys required for all the security classes in the hierarchy
- *Data Encryption Time:* This is the time required to encrypt a data object (in our case, a file)
- *Key Replacement Time:* This is the combined time it takes to both generate a new key and encrypt the data associated with the affected keys
- *Window of Vulnerability:* This is the combined time it takes to transmit a key replacement request to the key server and the key replacement time.

These four metrics allow us to compare the behavior of all three schemes with the aim of confirming the observations we made in our complexity analysis. An added consideration is that the four metrics we have selected are the more cost-intensive operations that the key management schemes need to handle both during the initial setup of the access control system and when keys need to be replaced. As shown in

Figure 3.6, in the implementation setup, the key server generates exponents for each of the classes in the hierarchy and uses the exponents to generate keys. The keys are then used to encrypt the data and once this is done, the keys are distributed to the users requiring access. Key replacement requests are directed to the key server and users remaining in the group wait to receive a new key before continuing accesses.

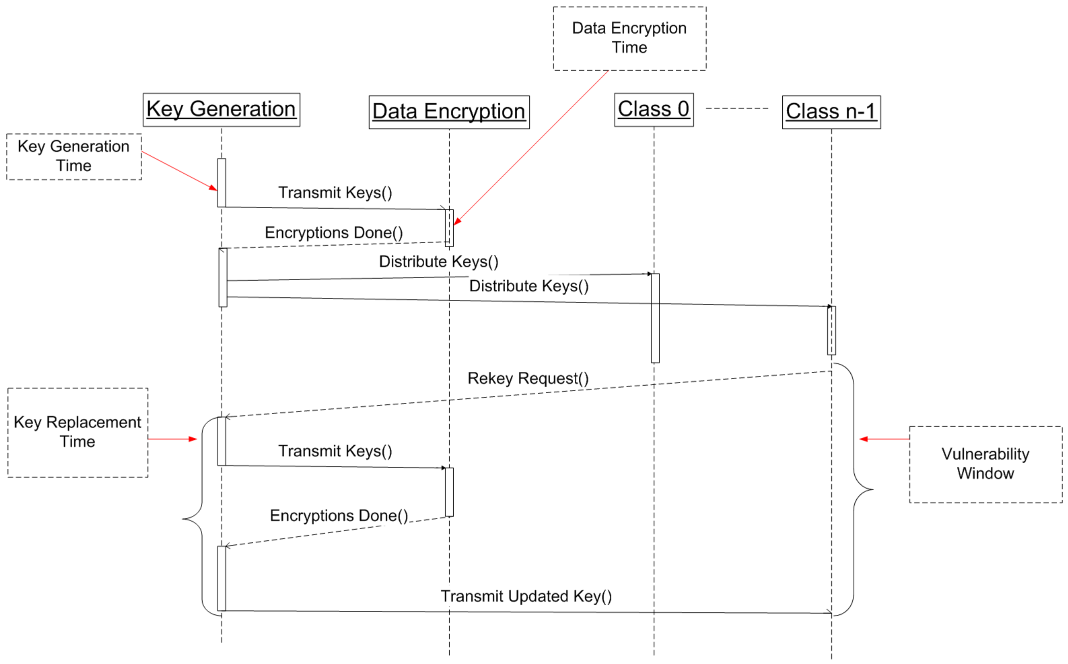


Figure 3.6: Sequence Diagram describing SPKM,CAT, and MCK Implementation

We implemented our exponent generation and replacement algorithms on a Microsoft Windows XP platform using the Java 2 Standard Development Kit and Eclipse [87, 16]. In our implementation, the key generation function generates Triple DES (Data Encryption Standard)[97] encryption/decryption keys and the data encryption module encrypts the files. Each security class is associated with a file of size $\approx$ 32MB. Although a Triple DES key is not the state of the art in cryptographic key usage, it seemed like a good choice because it is simple to use for implementation purposes and is strong enough to provide good security as well as to highlight the need for a scheme like the SPKM one we have suggested for addressing issues of performance management in dealing with data encryptions and key replacements. The experiments are conducted on an IBM Pentium IV computer with an Intel 2.66Ghz processor, 512MB of RAM. We limited the size of our hierarchy to 111 security classes in total because of the computational limitations of the machine we used for testing. (Diagrams of the hierarchies used are given in Appendix B.)

3.5.2 Cost of Key Generation

The first experiment evaluates the total cost of key generation per poset size in the SPKM scheme in comparison to the CAT and the MCK schemes. In each case key generation is handled for the whole hierarchy and for simplicity, we conducted the experiments on poset sizes of 3 to 111 using a 32MB file per security class. The experiment was repeated 10 times for each hierarchy instance and the results averaged and plotted in Figure 3.7. The error bound for each data point in the CAT, MCK and SPKM schemes respectively is ± 0.005 seconds.

As indicated in Figure 3.7, the experimental results confirm the theoretical analysis that the CAT and the MCK schemes perform worse than the SPKM scheme. This is to be expected because the exponent generation algorithms that the CAT and the MCK schemes use results in a geometric growth in the size of the exponent whereas in the SPKM scheme the exponent generation algorithm bounds the growth of the sizes of the exponents linearly.

We noted also that although the cost of key generation in the MCK scheme is generally lower than that of the CAT scheme, in the worst case, as indicated by the data point (85 classes,0.164 seconds), the cost of key generation is higher. However, the difference between the times produced by the CAT and the MCK schemes is approximately 0.0484 seconds so this reasonable. This is because the scheme fails to

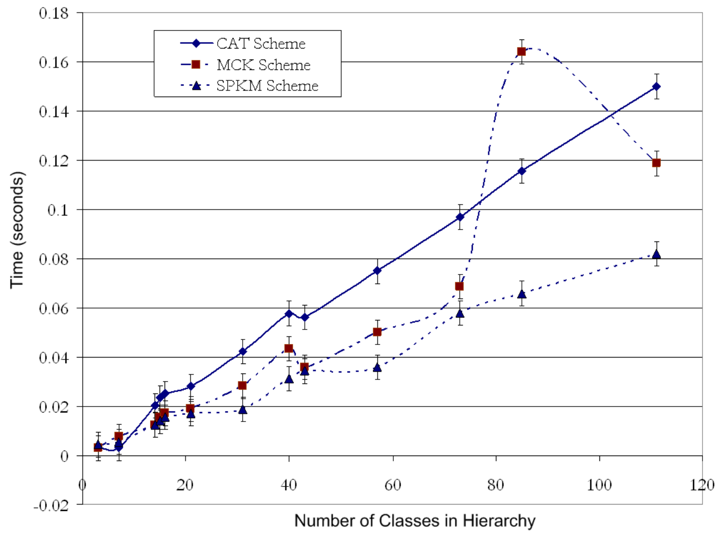


Figure 3.7: Cost of Key Generation Only

find an optimal assignment and so the combined time of searching for the assignment and then proceeding to generate the keys results in an overall higher key generation cost in the MCK scheme than in the CAT scheme.

3.5.3 Cost of Data Encryption

In the second experiment we evaluated the cost of data encryption throughout the entire hierarchy during key generation. This encryption cost does not include the cost of key generation but rather the total cost of data encryption for all the classes in the hierarchy. Each class has a 32MB file associated with it. The experiment was repeated 10 times for each hierarchy instance and the results averaged and plotted in Figure 3.8. The error bound for each data point in the CAT, MCK and SPKM schemes respectively is ±20 seconds.

As shown in Figure 3.8, the cost of encryption in the MCK scheme is better than that in the CAT scheme and likewise the SPKM scheme performs better than the first two. This indicates the the size of the exponent defines the size of the key and consequently the time it takes to perform encryptions. Again, this is not surprising

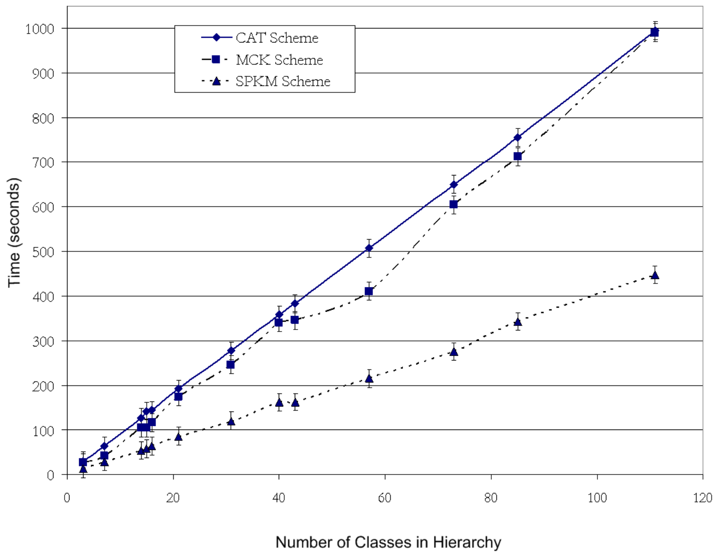


Figure 3.8: Total Cost of Data Encryption per Hierarchy Size

because encryptions are performed by dividing the file to be encrypted into blocks that are then encrypted with the provided cryptographic key. The higher the value of the cryptographic key the closer it is to its maximum bit size (in this case we are using a triple DES key so the maximum key size is 128 bits and provides 112 bits of security), and consequently the longer it takes to create a cipher.

3.5.4 Cost of Key Replacement

We evaluate the cost of key replacement, in all three schemes, in the worst case when the whole hierarchy is rekeyed. The cost of key replacement is the combined cost of key generation and encryption for the whole hierarchy. The experiment was repeated 10 times for each hierarchy instance and the results averaged and plotted in Figure 3.8. The error bound for each data point in the CAT, MCK and SPKM schemes respectively is ±20 seconds.

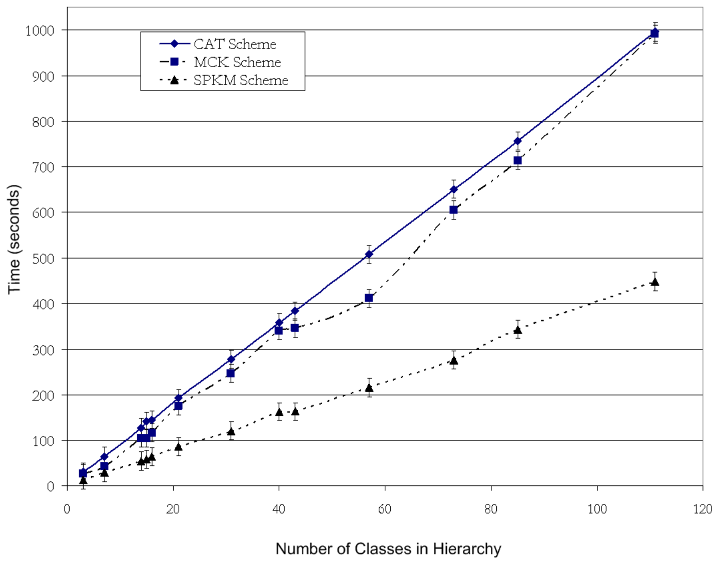


Figure 3.9: Total Cost of Key Replacement in Relation to Hierarchy Size

As indicated by the results plotted in Figure 3.9, the cost of replacement is similar to the cost of data encryption because the cost of key generation is very small in comparison to the cost of encryption (see Figure 3.8). However, the performance of the SPKM scheme is better than that of the previous two schemes. An added point worth noting is that the cost of key generation in itself is reasonable (bounded by 2 seconds) but the cost of encryption is a factor of the key type. More complex keys imply higher encryption costs.

3.5.5 Window of Vulnerability

The size of the window of vulnerability is determined by the combined time it takes to transmit a rekey request to the key server and the time it takes to create a new key, re-encrypt the data associated with the replaced key and transmit the updated key to the users remaining in the group. The experiment was repeated 10 times for

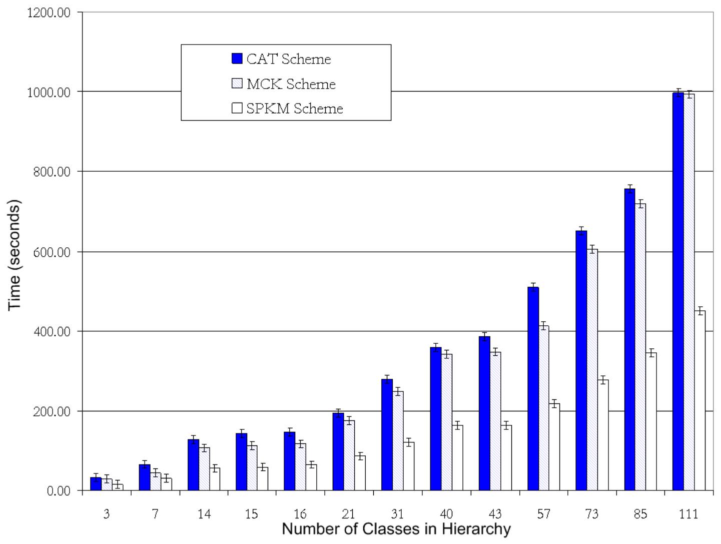


Figure 3.10: Size of Window of Vulnerability with Respect to Number of Classes Replaced

each hierarchy instance and the results averaged and plotted in Figure 3.10. The error bound for each bar plotted in the CAT, MCK and SPKM schemes respectively is ±10 seconds.

The results shown in Figure 3.10 confirm the observations made in sections 3.5.1 - 3.5.3, that although the cost of encryption and key generation costs during rekeying grows with an increasing hierarchy size, the SPKM scheme still performs better than the CAT and the MCK schemes.

3.6 Discussions

We have presented an SPKM scheme that uses two algorithms for exponent generation and replacement to minimize the cost of key management and replacement in a hierarchy. The first uses a distance based heuristic to control the growth of the exponents used to generate keys for access control in the tree hierarchy. While the second proposes a key replacement scheme that minimizes the number of keys that need to be changed whenever there is a change in user group membership by re-assigning keys only to portions of the hierarchy rather than to the whole poset (hierarchy) as is the case in previous schemes.

The security and complexity analysis and experimental results indicate that the algorithms are secure, scalable, and perform effectively in comparison to both the CAT and MCK schemes. The complexity analysis and experimental results were used to evaluate both the theoretical and practical improvements on the time required to generate and/or replace the exponents used to compute keys for the hierarchy.

Since the verifications required to obtain valid exponents can be computationally intensive, potentially creating overhead, we propose that this process be performed off line or during periods when the system is idle. Additional questions that may come to mind in relation to the proposed scheme include the possibility of key reuse. We answer this question in the negative, arguing that the registry of exponents and the exponent hierarchy allow the system to keep track of the exponents (t_i's) that have been used or that are currently in use. Furthermore, in the extreme case when no other combinations of exponents (t_i's) are possible, the central authority (security administrator or key server) has the option to select a new secret key K_0 and re-assign new keys to the whole hierarchy in order to preserve data security.

The size of the window of vulnerability and data re-encryptions affect both the security and performance of all the key management schemes, but is reduced significantly in the SPKM scheme. The objective of the next chapter is to present a method (algorithm) that overcomes this drawback by avoiding data encryptions.

Chapter 4

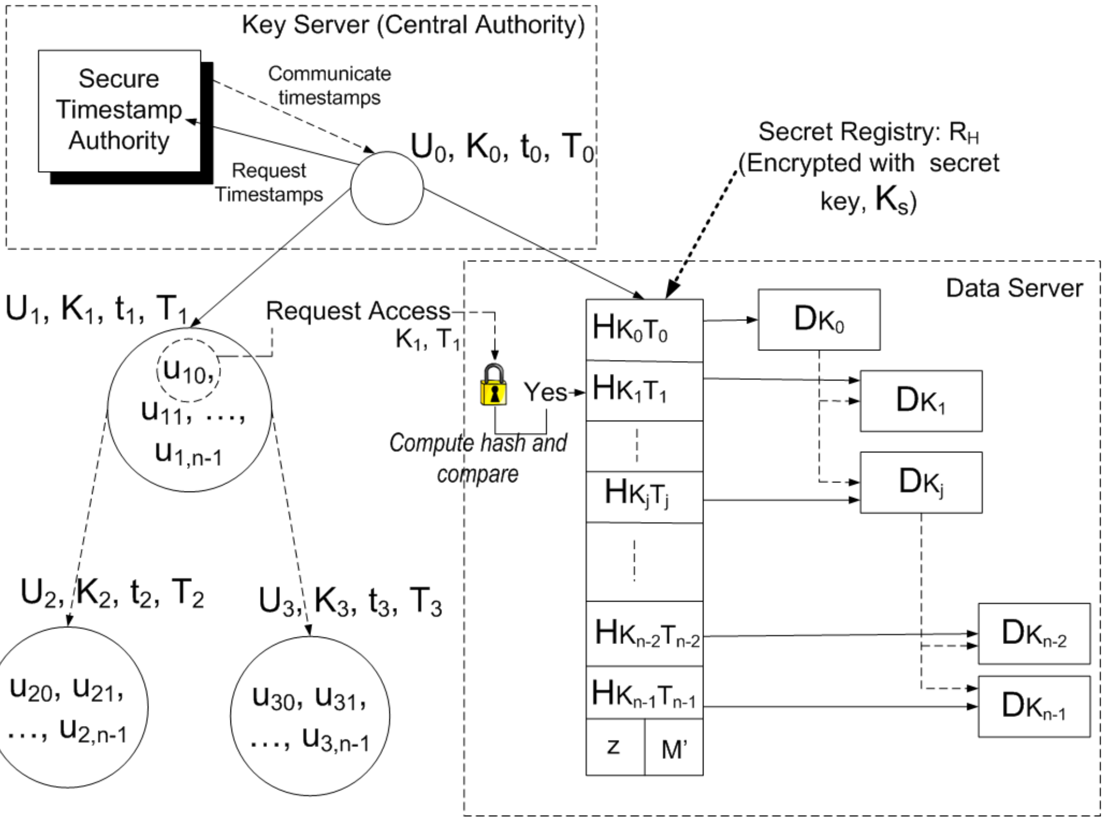
Timestamped Key Management

"Defer no time, delays have dangerous ends"

– ***William Shakespeare***

This chapter describes a second approach to efficient rekeying. As mentioned before, one of the main issues that arises in rekeying is that of minimizing the size of the window of vulnerability created during the rekey process. In the previous chapter, we noted that the requirement of rekeying and re-encrypting the associated data to preserve data security creates a wide vulnerability window [5, 52]. Since this process is time consuming and causes delay, the approach we propose in this chapter associates a timestamp to each key and uses both pieces of information to compute a verification signature that can be used for authenticating users [51]. Replacement is handled by updating both the timestamp and the verification signature as opposed to updating the whole hierarchy and re-encrypting all the associated data, as is the case in the SPKM scheme that we proposed previously.

As illustrated in Figure 4.1, the assumption is that there exists a single trusted secure central authority U_0, as well as a trusted (secure) timestamp authority (TSA), in charge of key generation and timestamp creation respectively. The timestamp authority communicates the timestamps to U_0, where they are associated with the keys generated and transmitted securely to the user groups in the hierarchy. Data is only accessible to users if his/her key and timestamp yield a verification signature (hash value) that corresponds to the one associated with the data.



Key Server (Central Authority)
Secure Timestamp Authority
Communicate timestamps
Request Timestamps
U_0, K_0, t_0, T_0
Secret Registry: R_H (Encrypted with secret key, K_s)
Data Server
U_1, K_1, t_1, T_1
Request Access K_1, T_1
u_{10}, u_{11}, …, $u_{1,n-1}$
Yes
Compute hash and compare
HK_0T_0
HK_1T_1
HK_jT_j
$HK_{n-2}T_{n-2}$
$HK_{n-1}T_{n-1}$
z
M'
DK_0
DK_1
DK_j
DK_{n-2}
DK_{n-1}
U_2, K_2, t_2, T_2
U_3, K_3, t_3, T_3
u_{20}, u_{21}, …, $u_{2,n-1}$
u_{30}, u_{31}, …, $u_{3,n-1}$

Figure 4.1: Timestamped Key Management

The central authority U_0, ensures secure access by transmitting the key and timestamp pair for each group to the data server where a hash value (verification signature) is computed for each key and timestamp pair, and stored in a secret registry R_H. It is assumed that the registry is kept secret by encrypting its contents with a secret key that is held by the data server. Each slot in the registry points to the data encrypted under the group key, associated with the signature. For instance, in Figure 4.1, the verification signature H_{K_0,T_0}, grants access to $D_{K_0}, ..., D_{K_{n-1}}$ because all the other keys $K_1, ..., K_{n-1}$ are derivable from the key K_0 associated with the verification signature H_{K_0,T_0}.

In order to access data, members of a security class (group) U_j, transmit their key K_j and timestamp T_j, securely to the data server where a pre-defined one-way hash function, H_i, is applied to the pair to compute a verification signature H_{K_j,T_j}. The computed signature H_{K_j,T_j} is then compared to the value in R_H for U_j and if the computed signature matches the currently valid signature for U_j then access is granted, otherwise it is denied. In the case where access is granted, the user in U_j can then access all data packages that obey the precedence relation, $U_i \preceq U_j$.

When the membership of a group changes, U_0 contacts the TSA for a new timestamp to replace the old value of T_j. On reception of the new timestamp, U_0 transmits this securely to the members of U_j and recomputes a new hash value H_{K_j,T_j} which is sent to the data server in order to prevent users who have departed from continuing to access data.

The advantage of this approach is that it removes the requirement of updating keys in whole portions of the hierarchy whenever the key, K_j belonging to a class U_j needs to be updated. As a consequence, the window of vulnerability created between key replacements is significantly reduced in comparison to the CAT, MCK and SPKM schemes [2, 66, 52, 62]. We begin with an overview of how the keys, timestamps, and verification signatures are generated and assigned to the various classes in the poset and then proceed to show how replacement works.

4.1 Timestamped Key Assignment

Our solution for timestamped key assignment has the following form. The central authority, U_0, begins by defining the size and form of the poset that defines the access control hierarchy. Next, U_0 selects a secret key K_0 as well as two large primes p and q, whose product M is made public while p and q are kept secret. In order to assign exponents to the various classes in the hierarchy, U_0 uses an exponent generation algorithm such as the one outlined in Chapter 3 (SPKM scheme). Once the exponents have been generated and assigned, a key K_i for each class in the hierarchy is then computed using the formula, $K_i = K_0^{t_i}(\text{mod} M)$.

The TSA generates timestamps T_i for the keys K_i, by selecting an integer value that is relatively prime to the product $\phi = (p-1)(q-1)$, of the two distinct large primes p and q used in the computation of the keys (see equation (3.1), Chapter 3, section 3.1 : $K_i = K_0^{t_i} \bmod M$ where $M = p \times q$). These values of p and q are made known to the TSA by U_0 and the reason for selecting a value that is relatively prime to ϕ is in order to guarantee the uniqueness and security of the timestamp [81]. An integer value that is relatively prime to ϕ is selected and assigned to T_i.

Once a valid timestamp has been selected, the key server proceeds to compute a hash value (verification signature) H_{K_i,T_i} for the given user group U_i using the formula:

$$H_{K_i,T_i} = (h_1(K_i, T_i) + zh_2(K_i, T_i))(\text{mod} M) \qquad (4.1)$$

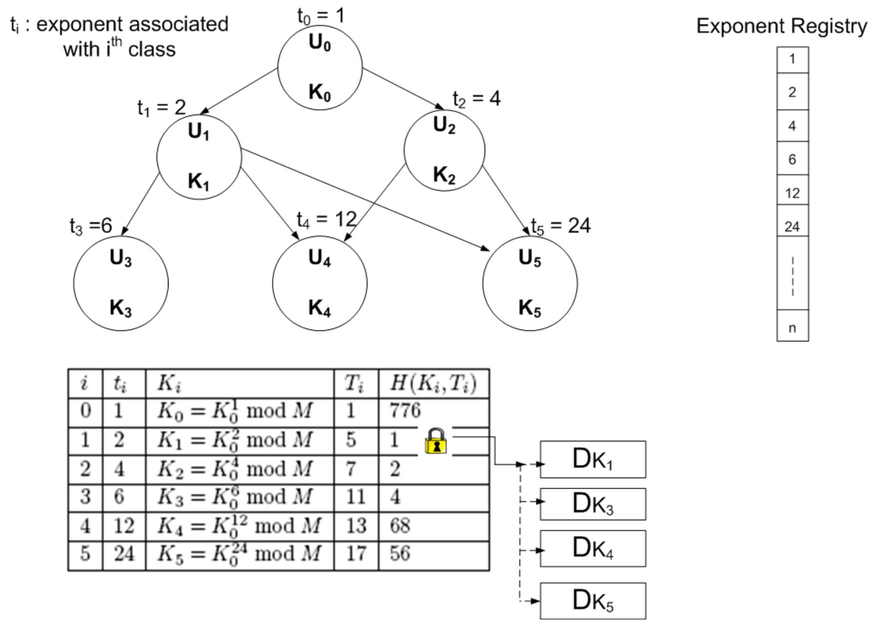
where

$$h_1(K_i, T_i) = K_i^{T_i} \bmod M \qquad (4.2)$$

and

$$h_2(K_i, T_i) = 1 + (K_i^{T_i} \bmod M') \qquad (4.3)$$

Here, p, q are the large prime numbers selected by U_0, M' is chosen to be slightly less than M (say $M - 1$) and z is an arbitrary integer value selected in order to avoid collisions in the secret registry (R_H). The values of z and M' are transmitted securely to the data server alongside the computed hash values, where they are then recorded in the secret registry R_H.



i	t_i	K_i	T_i	$H(K_i, T_i)$
0	1	$K_0 = K_0^1 \bmod M$	1	776
1	2	$K_1 = K_0^2 \bmod M$	5	1
2	4	$K_2 = K_0^4 \bmod M$	7	2
3	6	$K_3 = K_0^6 \bmod M$	11	4
4	12	$K_4 = K_0^{12} \bmod M$	13	68
5	24	$K_5 = K_0^{24} \bmod M$	17	56

Figure 4.2: Key Assignment and Corresponding Hash Values Generated

For instance, in Figure 4.2, using $\phi = (p-1)(q-1) = 288$ (i.e. $p = 17$ and $q = 19$), and an arbitrary value for $z = 12$, we obtain the hash values $776, 1, 2, 4, 68, 56$, for the various combinations of K_i and T_i. The hash values obtained are dependent on the bit length of the keys K_i. As illustrated, in Figure 4.2, in order for a user in U_2 to be granted access to data encrypted with K_5, the data server needs to authenticate their key K_2, and timestamp T_2, by applying the hash function (see equation (4.1)) to the values supplied by the user. If the computed value corresponds to the current value in the registry, access is granted. Otherwise it is denied. In the case where access is granted the user can then proceed to derive K_5 from K_2 using equation (3.2).

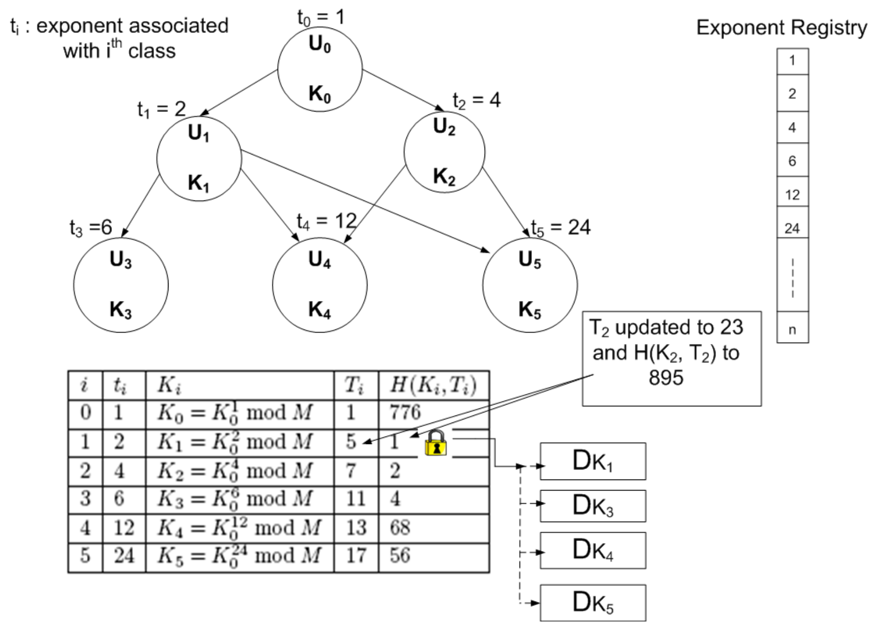
If the division of t_5 and t_2 (i.e. $t_5 \div t_2$, in this case, $t_5 = 24$ and $t_2 = 4$ so $t_5 \div t_2 = 6$) yields an integer value then the user can proceed to derive K_5 as follows:

$$\begin{aligned} K_5 &= K_0^{24}(\text{mod} M) \\ &= K_0^{4(24 \div 4)}(\text{mod} M) \\ &= K_2^{24 \div 4}(\text{mod} M) \\ &= K_2^{6}(\text{mod} M) \end{aligned}$$

Since $K_2^6(\text{mod} M) = K_5$, the user can now use this key to access data in D_5 that is encrypted with the key K_5.

4.2 Timestamped Rekey Scheme - Algorithm

When a user departs from a group, say U_j, security is preserved by updating the timestamp T_j associated with the secret key K_j. As mentioned before, in order to guarantee that the T_j selected is unique, the key server updates T_j by selecting a new value that is not in the registry and is relatively prime to $\phi = (p-1)(q-1)$. When a valid timestamp is obtained, the key server proceeds to compute the new hash value (H_{K_j,T_j}) using equations (4.1),(4.2), and (4.3) above. The new hash value (H_{K_j,T_j}) is transmitted securely to the data server and the timestamp is also securely transmitted to valid members of the group U_j. A user who has left the group will be unable to access data because his/her timestamp is invalid. For instance, in Figure 4.3, if the membership of U_2 changes, all U_0 does is update T_2 and compute a new hash H_{K_2,T_2}. In this case, the new timestamp T_2 is 23 and the new hash value H_{K_2,T_2}, is 895. The new hash value is recorded in the registry, R_H to avoid repetitions that could potentially lead to security violations.



i	t_i	K_i	T_i	$H(K_i,T_i)$
0	1	$K_0 = K_0^1 \bmod M$	1	776
1	2	$K_1 = K_0^2 \bmod M$	5	1
2	4	$K_2 = K_0^4 \bmod M$	7	2
3	6	$K_3 = K_0^6 \bmod M$	11	4
4	12	$K_4 = K_0^{12} \bmod M$	13	68
5	24	$K_5 = K_0^{24} \bmod M$	17	56

Figure 4.3: Rekeying with the *TSR* scheme: Example

This approach has the advantage of removing the requirement of updating both the key affected by the change as well as those that are derivable from it. Indeed, the cost of data re-encryption is no longer a factor since updating the timestamp and verification signature does not affect the associated key that was used in the encryption process. As a consequence, the window of vulnerability created between key replacements is decreased significantly in comparison with the SPKM scheme.

4.3 Analysis

The scheme is analyzed with respect to its security properties and its complexity. The security analysis shows cases of attacks that might occur against the proposed timestamp scheme and the counter measures taken in designing the schemes. The complexity analysis, on the other hand, compares the efficiency of Timestamped Key Management with that of the SPKM, MCK, and the CAT schemes. We chose to make the comparison with these three schemes since the SPKM, MCK, and the *TSR* schemes extend the method of key management in the CAT scheme [2].

4.3.1 Security Analysis

There are three security issues that need to be addressed, namely the security of K_i, the security of T_i and the security of the hash value H_{K_i,T_i}. With respect to the security of K_i, there is one possible attack. In this attack two or more users at some level l collude to compute an authorized key K_i higher up in the hierarchy. As mentioned before, when the greatest common divisor of the exponents at level l does not divide any of the exponents at level i, then collusion is not possible [2]. A special case involves a user at level l attempting to derive a key (K_i) at level i. From equation (3.2), that is $K_i = K_0^{t_i}(\text{mod}M) = K_0^{t_l(t_i \div t_l)}(\text{mod}M) = K_l^{t_i \div t_l}(\text{mod}M)$, since $t_i \div t_l$ does not yield an integer value, K_l will not pass the test of divisibility and hence K_i will not be derivable from K_l.

With respect to the security of T_i, a possible attack is that a lower level user attempts to derive a T_i at a higher level. Considering that T_i is a very large integer, guessing at the exact value of T_i using $gcd\{T_i, \phi\} = 1$, even when $\phi = (p-1)(q-1)$ is known, is a hard problem, since T_i is generated using a cryptographically secure random number generator and $gcd\{T_i, \phi\} = 1$ is not easy to solve for large values of T_i and ϕ [81].

Finally, concerning the security of the hash value (verification signature) H_{K_i,T_i}, a malicious user who no longer belongs in a group might attempt to guess at the value of T_i associated with the key K_i that they still have. The function for computing the signatures forestalls this attack by using a secure timestamp authority to generate timestamps that are difficult to guess [71, 78]. Furthermore, the security of the keys and the fact that the verification signature is obtained by the means of a one-way hash function, makes deriving the value of H_{K_i,T_i} hard [71, 78].

4.3.2 Complexity Analysis

The time complexity of the *TSR* scheme is examined in relation to the SPKM, MCK, and CAT schemes because all three are based on the CAT key management scheme [52]. Assuming that the sizes of the largest t_i and T_i (exponent and timestamp respectively), used to compute the keys and hash values in the TSR scheme, are equivalent and require $O(\log n)$ bits to be represented; we can conclude (from the discussion in Section 3.4.2) that $O(n \log n)$ multiplications of $O(\log M)$ bit numbers are required to generate keys and timestamps for the whole hierarchy of n groups (classes). With respect to rekeying, since the TSR scheme only needs to replace the timestamp associated with the affected class, the TSR scheme only needs to perform $O(\log n)$ multiplications of $O(\log M)$ bit numbers to compute the new hash value. Moreover, while the CAT, MCK, and SPKM schemes need to re-encrypt all the data that was encrypted with the old keys, in the TSR scheme, the data does not need to be re-encrypted. A summary of the complexity analysis for all three schemes is given in Table 4.1.

Table 4.1: Comparative Analysis: Time complexity in multiplications per key size in $O(\log M)$ bits

	CAT and MCK Scheme	SPKM Scheme	TSR Scheme
Generation	$O(n^2 \log n)$	$O(n \log n)$	$O(n \log n)$
Rekeying	$O(n^2 \log n)$	$O(n \log n)$	$O(\log n)$

4.4 Experimental Setup and Results

This section describes the experimental setup we used to run experiments to evaluate our SPKM scheme in comparison to the CAT and MCK schemes, and experimental results showing performance evaluations. As mentioned in Section 3.5, we limit the comparison to the SPKM, CAT and MCK schemes because all four schemes rely on the same key generation function (i.e. equation (3.1)) and the main factor that causes the difference in key generation or replacement times, is the size of the exponent used to compute the keys.

4.4.1 Implementation and Experimental Setup

The performance and scalability of the TSR scheme proposed in this chapter were evaluated with a set of experiments conducted on an IBM Pentium IV computer with an Intel 2.66GHz processor and 512MB of RAM and a 32MB file per security class in the hierarchy. As in Section 3.5, our evaluations are based on four metrics: key generation time, data encryption time, key replacement time (rekeying), and the size of the window of vulnerability.

The sequence diagram, given in Figure 4.4, shows how the TSR scheme operates. The key server generates a series of exponets using the SPKM scheme exponent generation algorithm and then calls the timestamp generation function to create timestamps. Once the timestamps are created the verification signature function is called to generate signatures that are communicated to the key generation function. On reception of the signatures, the key generation calls the data encryption function

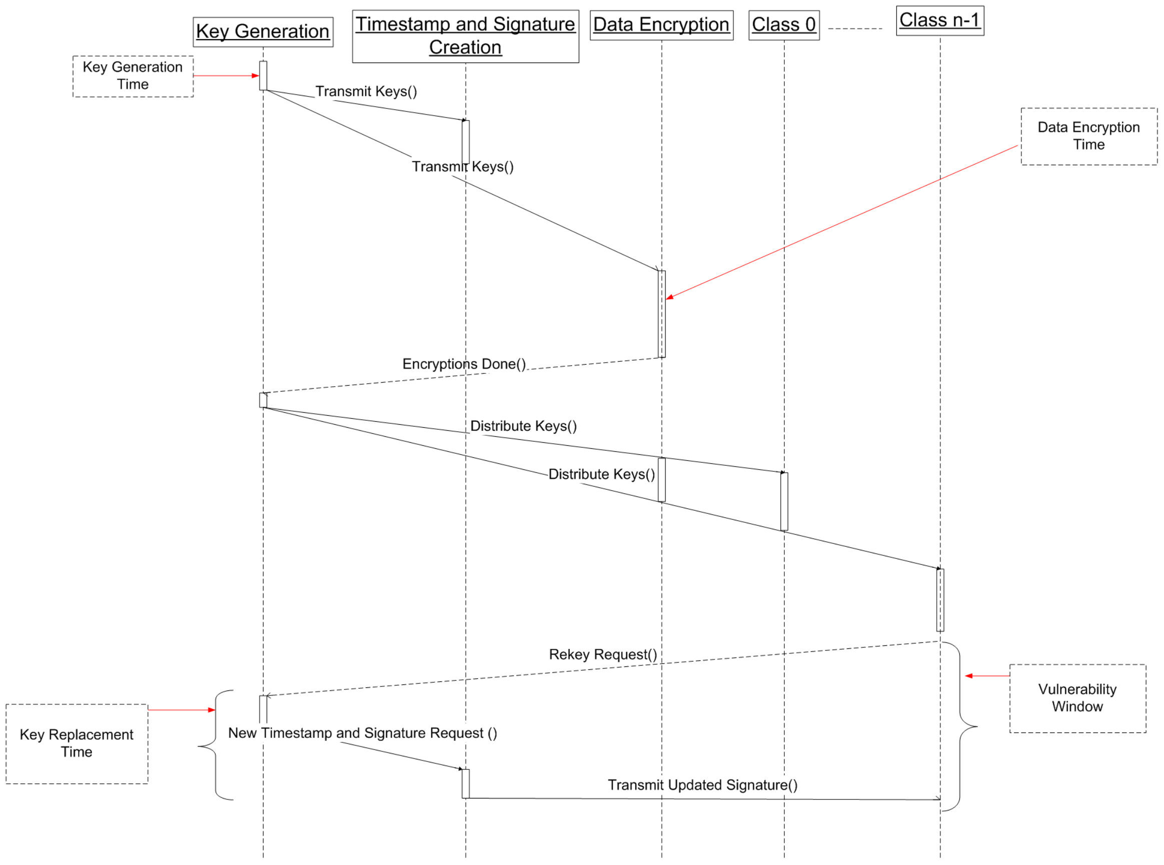


Figure 4.4: Sequence Diagram describing the TSR Implementation

to encrypt the data and then proceeds, on completion of the encryption process, to transmit both keys and signatures to the user groups. Replacement is handled by calling the timestamp generation function and signature function to create a timestamp and verification signature repectively.

We implemented our exponent generation and replacement algorithms on a Microsoft Windows XP platform using the Java 2 Standard Development Kit and Eclipse [87, 16]. In our implementation, the key generation function generates Triple DES (Data Encryption Standard)[97] encryption/decryption keys and the data encryption module encrypts the files. Each security class is associated with a file of size $\approx$ 32MB. Although a Triple DES key is not the state of the art in cryptographic key usage, it seemed like a good choice because it is simple to use for implementation purposes and is strong enough to provide good security as well as to highlight the need for a scheme like the SPKM one we have suggested for addressing issues of performance management in dealing with data encryptions and key replacements. The experimental results show that the TSR scheme reduces the vulnerability window and increases data availability in comparison to the SPKM, CAT, and MCK schemes [52]. Although the test scenarios are not exhaustive, the experiments give an intuition about the general performance of the schemes under consideration.

4.4.2 Timestamped Key Generation - Server Cost

In the first two experiments we study the effect of varying the size of the poset on the cost of key generation (time required to generate keys and encrypt a 32MB file per security class in the poset). The CAT [2], MCK [66], SPKM [52], and the TSR schemes were run 10 times for instances of 7 to 111 security classes respectively. The computation times were averaged and reported in Figure 4.5 and Figure 4.6. The error bound for each data point plotted is ± 0.03 seconds for key generations and ± 20 seconds for the combined cost of key generation and encryption, in each of the respective four schemes. (Diagrams of the hierarchies used are given in Appendix B.)

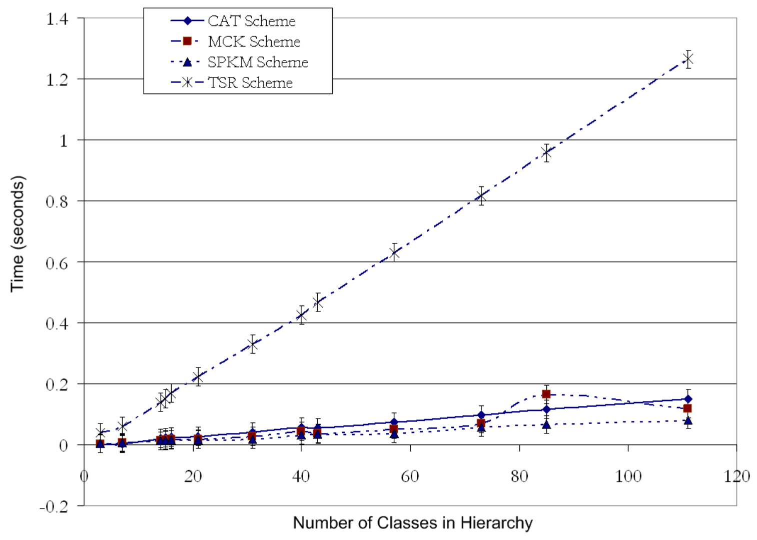


Figure 4.5: Cost of Key Generation Only

The cost of key generation in the TSR scheme is higher than that of the CAT, MCK, and SPKM schemes, which is reasonable given that the TSR scheme requires additional time on top of key generation, to generate a timestamp and hash value for each group in the hierarchy. However, the additional cost does not create a significant effect on the combined cost of key generation and data encryption. In fact, as shown in Figure 4.6 the cost of key generation and data encryption is better in the TSR scheme than in the CAT scheme and the MCK scheme and marginally worse than that of the SPKM scheme.

4.4.3 Timestamped Rekeying - Server Cost

The second experiment evaluates the cost of key replacement with respect to the size of the poset, using all four schemes. Arbitrary instances of numbers and positions of class keys to be replaced were selected. The algorithm was run 10 times for each rekey instance of 7 to 111 security classes per hierarchy and the times obtained averaged to plot the graph in Figure 4.7 with the error bounds for each test point in the CAT scheme, MCK and SPKM schemes being ±20 seconds and ±0.02 seconds in the TSR scheme.

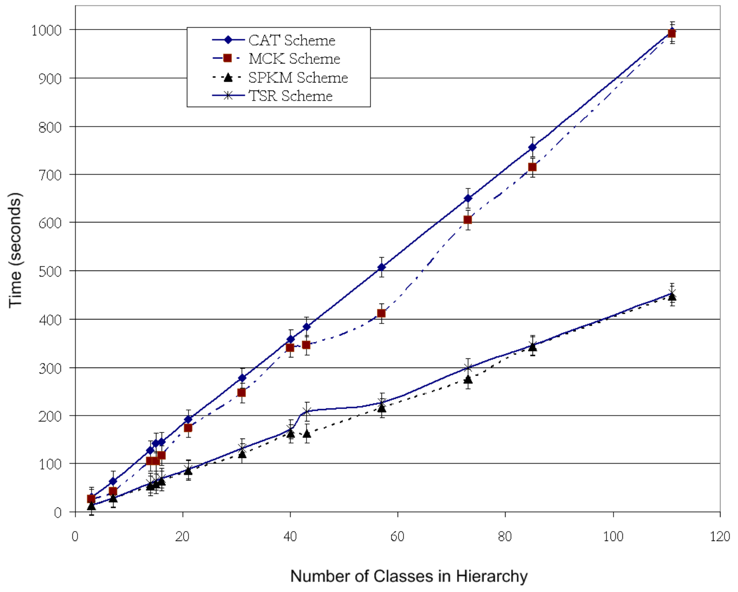


Figure 4.6: Cost of Key Generation and Data Encryption per Size of Poset

In the CAT and MCK schemes each rekey event triggers an update of the whole hierarchy. In the SPKM scheme replacement was triggered at a point in the hierarchy that would result in a complete update of the hierarchy. In the TSR scheme, timestamp updates were triggered at every class in the hierarchy. This was to ensure a fair comparison between the previous schemes and the proposed TSR scheme. The cost of rekeying is evaluated as simply being the cost of generating a new key or timestamp and where need be encryption. It does not include the cost of key transmission.

The cost of rekeying was observed to grow linearly in the *TSR* scheme whereas in the other schemes, the cost of rekeying stays fairly constant, in relation to the combined cost of key generation and data encryption (see Figure 4.6), with respect to the number of classes involved. In both cases this is to be expected because, in the SPKM scheme, in the worst case (i.e. when the key belonging to the highest class in the poset needs to be updated) the whole hierarchy is rekeyed to prevent illegal access to data (or preserve data security). Likewise, in the CAT and MCK schemes

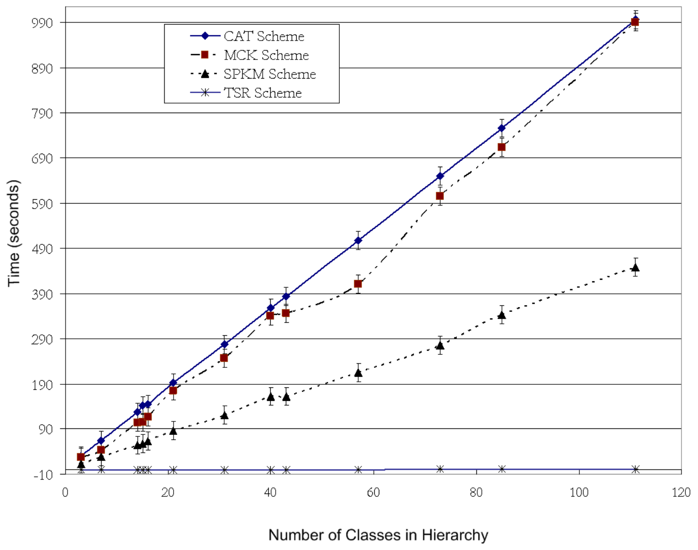
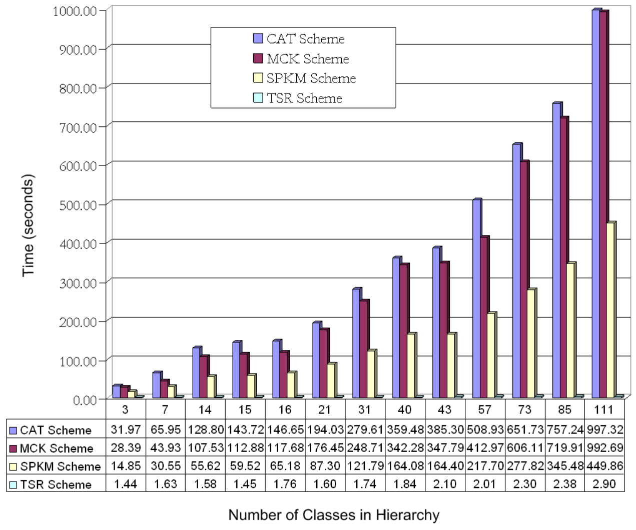


Figure 4.7: Cost of Rekeying per Number of Class keys Replaced

rekeying requires updating the whole hierarchy. By contrast, in the *TSR* scheme the cost of rekeying is much lower because rekeying only involves updating the timestamp and verification signature (hash value).

4.4.4 Window of Vulnerability

The third experiment evaluates the size of the window of vulnerability generated in the CAT, MCK, SPKM, and TSR schemes. In this case the classes we selected for rekeying resulted in changes throughout the hierarchy in all four schemes. We define the window of vulnerability as the sum of the time it takes to communicate a rekey request to the key server, generate the new key(s), re-encrypt the affected data with the new key and transmit the updated key to the users in the affected classes. We randomly selected instances and numbers of classes to be replaced in a poset that comprised 384 classes. The algorithm was run 10 times for each rekey case and the times (size of the window of vulnerability) obtained averaged to plot the graph in Figure 4.8, with the error bounds for each of the points plotted in the CAT, MCK, and SPKM schemes being ±20 seconds respectively and ±0.02 seconds in the TSR scheme.



	3	7	14	15	16	21	31	40	43	57	73	85	111
▨ CAT Scheme	31.97	65.95	128.80	143.72	146.65	194.03	279.61	359.48	385.30	508.93	651.73	757.24	997.32
■ MCK Scheme	28.39	43.93	107.53	112.88	117.68	176.45	248.71	342.28	347.79	412.97	606.11	719.91	992.69
□ SPKM Scheme	14.85	30.55	55.62	59.52	65.18	87.30	121.79	164.08	164.40	217.70	277.82	345.48	449.86
□ TSR Scheme	1.44	1.63	1.58	1.45	1.76	1.60	1.74	1.84	2.10	2.01	2.30	2.38	2.90

Figure 4.8: Size of Vulnerability Window created with respected to the Number of Keys Replaced

We observe that the size of the window of vulnerability created by the schemes grows linearly with the number of classes replaced. The reason for this behavior is that the window of vulnerability in this case includes the time required to generate and distribute the new (updated) keys as well as the data re-encryption time. By contrast, since the TSR scheme only updates the timestamp and verification signature, the size of the window of vulnerability is equivalent to the combined cost of message and key distribution as well as the cost of timestamp and verification signature computation stays fairly constant with respect to the size of the rekey instance.

4.5 Discussion

Throughout this chapter the focus has been on securing access to shared data efficiently by reducing the size of the vulnerability window and the delay incurred by users during rekeying, in comparison with previous schemes modeled on the approach in the CAT scheme. As mentioned before, the reason for focusing on shared data scenarios is that they make for a good example to emphasize the effects of the cost of using cryptographic access control in dynamic environments. Both in this chapter and in chapter 3, algorithms were presented that improve on the costs of key management in models based on a poset in which the keys are inter-dependent. Using posets of inter-dependent keys has the advantage of minimizing the number of keys held by any one group of users thus making for better data security and efficiency in key management (rather than other schemes that assign each group all the keys they require [40]). However, as noted before, these hierarchical schemes that use inter-dependent keys, handle key updates (changes) by replacing not only the key concerned, but also those in the sub-hierarchy affected by the update [62, 106]. In fact, in the worst case all the keys associated with all the classes in the entire poset need to be updated and all the associated data re-encrypted. Additional problems include the vulnerability window created between key replacements, and data unavailability during the rekey period (users cannot access data while they have not received a valid key). Evidently, these methods of rekeying are computationally expensive, which is why recent approaches have favored assigning user groups all the keys they require instead of using a poset of inter-dependent keys [2, 66, 62, 106, 52].

We have addressed the problem of rekeying in access control hierarchies, by proposing a solution that associates a timestamp T_i to each key K_i and uses both to compute a verification signature (hash value) that is used to authenticate users before they are granted access to data. So, instead of replacing all the keys in U_i's sub-poset whenever a key change is necessary, all the central authority (e.g., key server or security provider) needs to do is to replace the secret timestamp associated with the key K_i and recompute a new verification signature. Thus, to access data in D_l or D_k encrypted with K_l or K_k, a user belonging in U_i must be able to first derive the new verification signature associated to U_i using their key K_i and timestamp T_i, and secondly be able derive the respective keys K_l and K_k from the key K_i which they possess. An added advantage of the new scheme is that the data does not need to be re-signed (re-encrypted) since the key remains unchanged. Thus, the amount of out-of-sync data lost and the size of the window of vulnerability are both minimized, since only one class is affected as opposed to several, which is the case in previous schemes like the CAT, MCK, SPKM, time-bounded, batching and merging schemes [95, 62, 52].

There are a number of scenarios in which the proposed Timestamped Key Management and *TSR* schemes can be applied. As an illustration, consider a collaborative web-based community like Facebook where various users communicate. Users can be grouped by their privileges of access to data on the system. Here, information broadcast could be an invitation to a party. The problem arises when the information needs to be broadcast to certain users but not to others. Another example is in battlefield scenarios, where certain information may be broadcast to soldiers depending on their location and the situation with which they are faced. Other areas of application include sensor networks and distributed systems where communicating entities need some assurance that the communicating parties can be trusted.

Other issues that can arise with respect to this scheme include the possibility of collision in the verification signatures since they are obtained via a hash function. Our argument is that the hash function we have proposed is based on a one-way function and also uses double hashing to avoid collisions. In the experiments that involved executing the algorithm 10 times for each replacement instance in a hierarchy of 3 to 111 classes we noted that there were no collisions.

The timestamped scheme associates each key with a timestamp, and both the timestamp and key are used to compute a verification signature that is used as an authentication procedure before data access is granted. In this way whenever group membership changes, instead of rekeying and re-encrypting the data associated with the keys, only the timestamp is updated and a new verification signature computed. Although this scheme makes it possible to significantly reduce the cost of rekeying, making it interesting for dynamic scenarios, the scheme's reliance on authentication makes it vulnerable. That is, if a malicious user holding a valid key finds a way of generating correct timestamps, there is no straight-forward way of detecting or even preventing them from accessing the system.

However, computing the timestamps and verification signatures may require varying amounts of computation time depending on the verifications that need to be performed to ensure security which is why in Chapter 5, we suggest using a self-adaptive function to support key management. The method we describe in Chapter 5 is modeled around the scheme that we proposed in Chapter 3 (SPKM scheme), but it can also be applied to the timestamp scheme that we have proposed in this chapter. In such a case the adaptive model will monitor the system's behavior and instruct the key server to generate timestamps and verification signatures in anticipation of periods of high demand when rekey requests may occur within short intervals of each other.

In the next chapter we present a method of adaptive key management that extends the approach that we presented in Chapter 3, but it can also be applied to the timestamp scheme. We have chosen to use the SPKM scheme as the basis for the adaptive framework because it is more encryption intensive than the timestamped scheme and is a better way of highlighting the advantage of adaptation over previous schemes. Moreover, the timestamped scheme operates more like an authentication scheme and therefore is more susceptible to attacks that are triggered by correct guesses at the timestamps.

Chapter 5

Self-Protecting Key Management

"Adapt or perish is nature's inexorable imperative"

– ***H.G. Wells***

In the previous two chapters we presented ways in which CKM schemes can be extended for better performance. Efficient CKM schemes are good for handling security, because they allow an access control scheme to match security with performance and so provide better guarantees of satisfying service level agreements between various web applications. However, emerging applications like data outsourcing, collaborative project development, and pay-TV, make it necessary to take adaptability into account in designing security schemes.

This need for adaptability in security schemes has arisen mainly because of the complexity of managing security for the multiple and varied scenarios that occur in Web-applications, where security management is complicated by the fact that the scenarios for which security needs to be ensured are difficult to predict a priori. For instance, in the CKM context, it is difficult to manually predict where and when a rekey request will occur and so manual security management in this kind of scenario results in delayed responses and a wider period of vulnerability.

This chapter presents a framework based on the autonomic computing paradigm [20, 57, 56] that allows a CKM scheme to adapt to changing scenarios by minimizing the response time and the size of the vulnerability window created by frequent rekeying. By supporting a CKM scheme with autonomic functionalities, we allow the scheme to assume the quality of being self-protecting in the sense that the CKM scheme acquires the ability to self manage and self-configure in order to handle new key management situations adaptively. The framework is composed of six functionalities: the *Sensor*, *Monitor*, *Analyzer*, *Planner*, *Executor*, and *Effector*, that are connected together in the form of a feedback control loop (FBCL) [53, 54]. The FBCL continually monitors the arrival rate of rekey requests at the key server and, at regular intervals, computes an acceptable resource (keys and encrypted replicas/data versions) allocation plan to minimize the overall cost of rekeying. Each component of the framework contributes to enhancing a standard CKM scheme's performance without changing its underlying characteristics. A prototype implementation and experiments showing performance improvements demonstrate the effectiveness of the proposed framework.

5.1 Self-Protecting Cryptographic Key Management (SPCKM) Framework

Our Self-Protecting Cryptographic Key Management (SPCKM) framework handles the rekey problem by supporting a standard KM scheme with an autonomic model. Each class in the access control hierarchy represents a group of users authorized to access a portion of the shared data. A rekey request is triggered by transmitting a message to the key server indicating a change in group membership. The message can be explicit in the sense that a user actively indicates his/her wish to leave the group, or it can be implicit, for instance if a user remains inactive for a long time then the KM system can lock him/her out for safety reasons.

In the framework, the adaptive rekey process is initiated by monitoring, over an initial period, the rate at which the key server (central authority) receives requests. This information is then used to generate keys and encrypted versions of the data to anticipate future rekey requests. Therefore, in comparison to quasi-static key management approaches, the SPCKM framework allows a key management scheme minimize the response time and the size of the vulnerability window created in handling rekey requests.

As in the schemes we have presented in previous chapters, we assume that there exists a single trusted central authority (key server) U_0 in charge of key generation/assignment and data encryption. Cases of central server crashes are assumed to be handled by some fault-tolerance solution like server replication. The key server encrypts the data on the data server according to the rules of access specified by the security administrator (SA) and distributes the keys secretly to the user groups requiring access. Data is only accessible to users if the key in their possession allows them to directly decrypt, or to derive the keys required to decrypt that data. For instance, as shown in Figure 5.1, u_{20} can access data $D_{K_2}, D_{K_{n-2}}$, and $D_{K_{n-1}}$ since his/her key can be used to directly decrypt D_{K_2} and derive the keys K_{n-2} and K_{n-1} required to access $D_{K_{n-2}}$, and $D_{K_{n-1}}$ respectively.

Rekey requests can be explicitly formulated by a user wishing to depart from a group in which case the user transmits a message encrypted with his/her key to the key server. Alternatively, the key server can monitor a group's behavior and decide

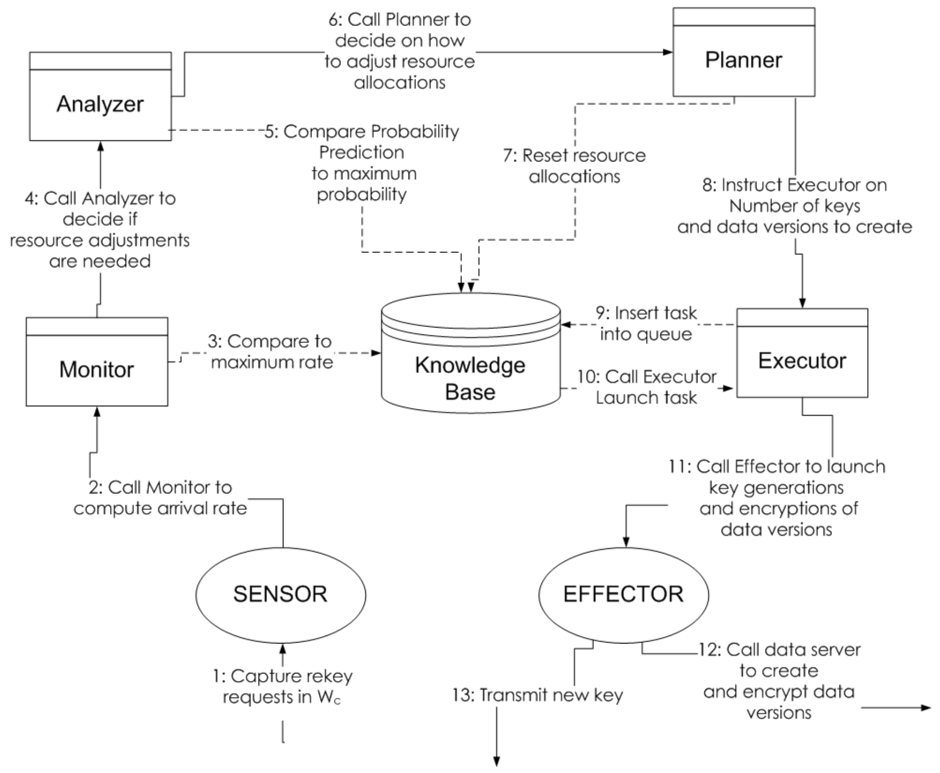

6: Call Planner to decide on how to adjust resource allocations
Planner
Analyzer
5: Compare Probability Prediction to maximum probability
7: Reset resource allocations
8: Instruct Executor on Number of keys and data versions to create
4: Call Analyzer to decide if resource adjustments are needed
9: Insert task into queue
Knowledge Base
Executor
Monitor
3: Compare to maximum rate
10: Call Executor Launch task
11: Call Effector to launch key generations and encryptions of data versions
2: Call Monitor to compute arrival rate
SENSOR
EFFECTOR
12: Call data server to create and encrypt data versions
1: Capture rekey requests in W_c
13: Transmit new key

Figure 5.1: Self-Protecting Key Management Framework

whether or not to exclude a user from a group. Knowledge of the membership of a group is located in a registry in the knowledge base. The registry contains the group and user identification of every user in the system as well as their associated secret keys.

As shown in Figure 5.1, our framework is comprised of six basic components that are interconnected in the form of a feedback control loop situated at the key server. The SA defines an initial observation period during which key replacement is handled solely by the conventional approach and data is collected to start off the adaptive

KM process. This initial period is divided into two time windows that are denoted by W_1 and W_2. During W_1 the sensor captures and transmits all rekey requests to the monitor. At the end of W_1, the monitor computes, on a per class basis, the arrival rate of rekey requests and compares this value to a preset maximum arrival rate in the knowledge base. If the current arrival rate is greater than the maximum arrival rate, the maximum arrival rate is reset to the current arrival rate and a message is transmitted to the analyzer indicating that the maximum arrival rate has been exceeded while monitoring continues in the next interval W_2. When the current arrival rate is below the maximum arrival rate, the event is discarded and monitoring continues in the next interval, W_2.

The analyzer computes a probability prediction to determine whether or not the current resource (keys and data versions) allocation will satisfy the next series of rekey requests that arrives in the next time interval, W_2, and communicates this value to the planner where an acceptable number of resources (keys and data versions) is generated in anticipation of the next series of rekey requests. The planner then calls the executor to define a schedule for creating and assigning the resources, and the executor instructs the effectors to perform each task according to the priority defined, in this case, first come first served. The maximum values (against which observed values are compared) are located in the knowledge base and are set by the SA on the basis of empirical observations. In the meantime, the monitor restarts the adaptive cycle by computing the arrival rate that occurs in W_2. The FBCL cycles continuously over time, analyzing the arrival rate for each subsequent period W_x that occurs after W_2. Copy consistency is maintained on the data versions by periodically checkpointing the state of the primary on to the data versions.

This approach has two main advantages. First, the response time and the size of the vulnerability window between key replacements is reduced since rekeying is handled in the background proactively as opposed to reactively waiting to generate the keys and encrypt the data when the rekey request arrives at the key server. Second, the job of the SA is made easier, since the SA no longer needs to take care of every key replacement scenario but rather, the SA presets specific parameters and allows the scheme to run on its own. Cases directly requiring the SA would henceforth be limited to situations that require the consent/advice of the SA to proceed.

5.1.1 Mathematical Model Supporting Framework

In the mathematical model that supports our framework, we use a Poisson process to describe the rate of arrival of rekey requests at the key server because this model is suited to modeling the occurrence of random events in time [30]. The arrival rate of rekey requests at the key server has this property of randomness since there is no way of predicting the exact number of rekey requests that will occur in a given period. Moreover, similarly to the basic foundational axioms of the Poisson model where occurences are viewed as being independent of the past and future, the observed arrival rate at any point in time is independent of both present and future arrival rates.

Since we are using a Poisson process to model the arrival rate of rekey events that originate from a group U_i in the KM hierarchy, we can assume that the requests arrive independently at a rate λ and denote:

The sensor captures rekey requests transmitted to U_0 over a preset period W_c and transmits this information (number of rekey requests and size of the monitoring

Table 5.1: Used Notations

Notation	***Meaning***
W_c:	The $c^{\mathtt{th}}$ predefined monitoring period (time window) for rekey requests arriving from U_i, such that $0 \leq c \leq \gamma - 1$ and γ is the maximum number of time windows
λ_i^c:	The arrival rate of rekey requests from group U_i during W_c
$Max(\lambda_i^c)$:	The maximum arrival rate of rekey requests for group U_i that the key server anticipates handling during W_c
m_i^c:	Total number of rekey requests that originate from group U_i during W_c
p_i^c:	The probability prediction that the key server will not be able to satisfy all the rekey requests that will arrive from U_i during the next monitoring period W_{c+1}.
α_i:	The degree of data availability that a user can expect inspite of the overhead generating activities (rekeying, encryption,...) on the system.

period W_c) to the monitor. At the end of the period W_c, the monitor computes the sum of the rekey requests received as well as the arrival rate λ_i^c. The arrival rate is computed with the formula:

$$\lambda_i^c = \frac{m_i^c}{W_c} \qquad (5.1)$$

where the arrival rate is measured in terms of number of requests per second. In the next step, the monitor compares the value of λ_i^c to a preset value $Max(\lambda_i^c)$ that is located in the knowledge base. The preset maximum values are set by the SA on the basis of empirical observations. If $\lambda_i^c > Max(\lambda_i^c)$, the maximum arrival rate is reset to the current arrival rate, i.e. $Max(\lambda_i^c)$ is assigned λ_i^c and a message is transmitted to the analyzer indicating that the previous maximum arrival rate has been exceeded (reset).

Using the formula for computing the probability mass function of the Poisson variable m_i, the analyzer computes a probability prediction to determine whether or not the current number of keys and encrypted replicas (versions) of data will be enough to satisfy a subsequent arrival rate of at least λ_i^c in W_{c+1}, without creating significant delays. There are two reasons for using this formula: the first is that it forms a part of the properties of the Poisson model that facilitate making predictions on the basis of very little information, and the second because it serves as a simple prediction tool. The formula for computing the probability p_i^c is given by [30]:

$$p_i^c = \frac{\mu_i^{m_i}}{m_i!} * e^{-\mu_i} \qquad (5.2)$$

where

$$\mu_i = Max(\lambda_i^c) * W_{c+1} \qquad (5.3)$$

is a prediction of the number of rekey requests that are expected during W_{c+1}.

The analyzer decides on whether to increase or decrease the resources (number of keys and encrypted data versions), by comparing p_i^c to a preset probability prediction value, ϵ. If $p_i^c = \epsilon$, the value is discarded. Otherwise, if $p_i^c < \epsilon$ the analyzer calls the planner with an instruction to decrease the resources, and if $p_i^c > \epsilon$, the planner is called with an instruction to increase the resources. This is to ensure that an optimal number of resources is always maintained so that the costs of updating data versions do not outweigh the benefits of adaptive KM.

On reception of the value of p_i^c and instructions regarding how the resources should be adjusted, the planner proceeds to compute a degree of availability. The degree of availability allows the planner to determine how to adjust the number of resources to keep the cost of running the system within acceptable limits. Availability, is defined as the fraction of time that the key server is in a position to satisfy any rekey request it receives. We consider that the key server can be in one of two states: the ***normal*** state or the ***idle*** state.

- ***Normal State:*** This is the state in which the key server performs two kinds of activities: key distribution or key generation. As shown in Figure 5.2 during this period, key generation and key distribution occur. Let T_R be the random variable representing the total time that the system spends rekeying during the window W_c.

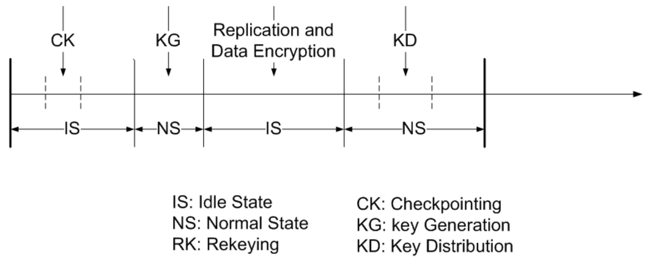


Figure 5.2: An Example of a Timeline of System States

- ***Idle State:*** As shown in Figure 5.2, this is the state in which all the required keys have been generated and the key server has relegated the tasks of data encryption and checkpointing to the data server [68, 69, 45]. Let T_E be the random variable that represents the total encryption time during the window W_c and T_C be the random variable that represents the total checkpointing time.

Our method of computing the degree of availability is inspired by the approach proposed by Jalote [45] where the availability of D_{K_i} is expressed as a probability function of the overhead generating activities in the KM system, expressed with the following formula:

$$\alpha_i = 1 - \frac{O_i}{W_{c+1}} \qquad (5.4)$$

where O_i is the overhead generated in maintaining update consistency on N_i replicas at each security class U_i. The overhead O_i is computed using the formula:

$$O_i = E(T_R) + E(T_E) + E(T_C) \qquad (5.5)$$

where $E(T_R)$ is the expected rekey time (i.e. the expected time required to generate and distribute a key to satisfy a rekey request), $E(T_E)$ is the expected encryption time, and $E(T_C)$ is the expected time required to copy updates from the primary data version onto the backup data versions (checkpointing).

Our framework is not designed to handle requests that are not completed during W_c, so we will assume that all rekey requests that arrive during the time window W_c are completed before the end of W_c, otherwise they are processed during the next time window W_{c+1}. So, W_c is set with respect to the time it takes to encrypt the data and copy updates from the primary replica (primary data version) on to the backup data versions during W_c. In order to compute the expected overhead during the window W_{c+1}, we need to compute the expected rekey time, encryption time, and checkpointing time. We compute all three values $E(T_R), E(T_E)$, and $E(T_C)$ using equations (5.6),(5.7) and (5.8).

Since p_i^c expresses the probability prediction that, if the current arrival rate occurs during the window W_{c+1}, the key server will not have the resources to satisfy all the requests that arrive in W_{c+1}, we multiply p_i^c by W_{c+1} to get a theoretical estimate of the expected rekey time. So, $E(T_R)$ can be computed using the following formula:

$$E(T_R) = p_i^c * W_{c+1} \qquad (5.6)$$

Considering that rekeying implies re-encrypting data, we can say that the encryption time is conditioned by the number of rekey requests that will occur during W_{c+1} and the number of data copies that need to be encrypted/re-encrypted. We denote

N_i as the number of data versions (replicas) that will need to be maintained at class U_i to satisfy μ_i rekey requests in W_{c+1} and compute a theoretical estimate of the total expected encryption time using the following formula:

$$E(T_E) = p_i^c * W_{c+1} * N_i \qquad (5.7)$$

This expresses the fact that all the replicas are encrypted and this time is taken into account in computing an estimate of the total re-encryption time $E(T_E)$.

In order to compute $E(T_C)$, we note that the theoretical estimate of the total time during which the system would not be in a state of rekeying during W_{c+1} (i.e. $E(T_E) + E(T_C)$) can be expressed as $[(1 - p_i^c) * W_{c+1} * N_i]$. Since we know $E(T_E)$ from equation (5.7), we can compute $E(T_C)$ using the following formula:

$$E(T_C) = [(1 - p_i^c) * W_{c+1} * N_i] - E(T_E) \qquad (5.8)$$

where N_i is the number of replicas required to ensure an optimal level of availability at class U_i during W_{c+1}.

Standard KM schemes are not supported by replication, so the overhead is given by: $O = E(T_R) + E(T_E)$ in which case $\alpha_i = 1 - (2 * p_i^c)$. On the other hand in our proposed SPCKM framework, overhead is calculated using equation (5.5). By substituting equations (5.6),(5.7) and (5.8) into equation (5.4), we can re-express availability as follows:

$$\alpha_i = |1 - (N_i + p_i^c - p_i^c N_i)| \qquad (5.9)$$

The growth of the number of replicas is controlled with a heuristic that bounds the availability degree by 1, and positive values of availability are ensured by expressing the results of equation (5.9) as absolute values.

If a rekey request arrives when there is no existing data version available to satisfy it, then the SPCKM scheme reverts to generating a new key, re-encrypting the primary with the new key and distributing the new key to the users that are left in the group. So in the worst case, the cost of rekeying reverts to the cost of KM in a standard scheme.

5.1.2 An Example

We use an example of a simple read-intensive scenario to explain how our framework operates. In this case, suppose that the observations of the key server during the initial monitoring period W_1 result in a prediction that one rekey request is going to arrive from U_2 during a future monitoring period W_x. In order to handle the rekey request, the analysis from the feedback control loop at the key server determines that one new key and encrypted data version needs to be generated in anticipation of this request. As shown in Figure 5.3 the key server creates a new key K_2, for the group

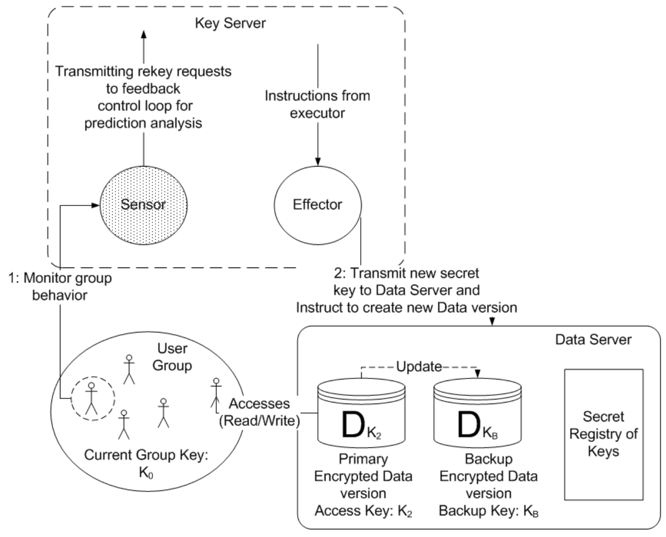

Key Server
Transmitting rekey requests to feedback control loop for prediction analysis
Instructions from executor
Sensor
Effector
1: Monitor group behavior
2: Transmit new secret key to Data Server and Instruct to create new Data version
User Group
Current Group Key: K0
Accesses (Read/Write)
Data Server
Update
DK2
DKB
Primary Encrypted Data version Access Key: K2
Backup Encrypted Data version Backup Key: KB
Secret Registry of Keys

Figure 5.3: Initial Replacement Scenario with no Update Requests

U_2, and transmits this key to the data server where it is kept in a secret registry.

On reception of the new key K_B and instructions to create a copy of D_{K_2}, the data server proceeds to create a new copy of D_{K_2} that it re-encrypts with K_B. In order to maintain copy consistency, updates on D_{K_2} are checkpointed onto D_{K_B} by periodically creating copies of D_{K_2} and re-encrypting it with with K_B to obtain an updated version of D_{K_B}.

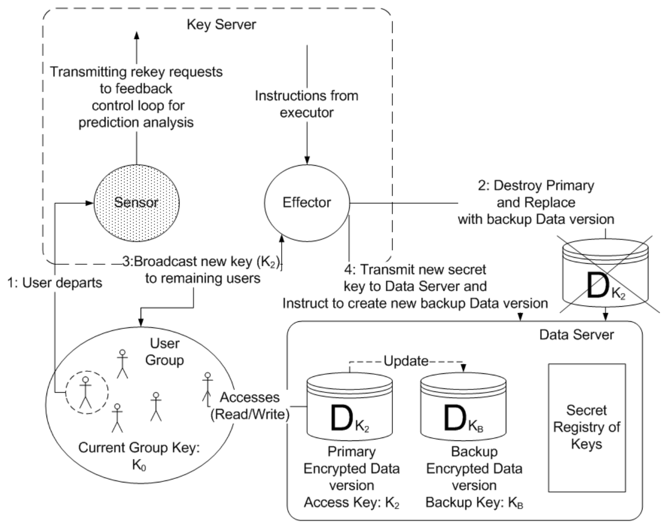

Key Server
Transmitting rekey requests to feedback control loop for prediction analysis
Instructions from executor
Sensor
Effector
2: Destroy Primary and Replace with backup Data version
1: User departs
3:Broadcast new key (K2) to remaining users
4: Transmit new secret key to Data Server and Instruct to create new backup Data version
D K2
Data Server
User Group
Update
Accesses (Read/Write)
D K2
D KB
Secret Registry of Keys
Current Group Key: K0
Primary Encrypted Data version Access Key: K2
Backup Encrypted Data version Backup Key: KB

Figure 5.4: Scenario in which u_{21} departs

As shown in Figure 5.4, when the key server receives a departure request during W_x it proceeds to destroy the primary data version and assign K_2 the value of K_B. The primary is replaced with the next data version in line (in this case $D_{K_2} \leftarrow D_{K_B}$) and the new key $K_2 \leftarrow K_B$ broadcast to the users remaining in the group. Finally it creates a new key, K_B, that it transmits to the data server and as before the data server creates a new data version D_{K_B} in anticipation of future requests.

5.2 Implementation and Experimental Setup

This section presents the prototype implementation of our proposed framework and experimental results evaluating its performance in comparison to a basic key management (BKM) scheme that is not supported by the paradigm of autonomic computing. For simplicity we have implemented the prototype as though the access control hierarchy were comprised of only one single class. The reason for doing this was to simplify the evaluation process since handling several nodes in the hierarchy requires a scheduling algorithm that determines a priority for satisfying requests in a way that minimizes the overall cost of replication and key replacement. A single node still allows us to evaluate the impact of autonomic control on KM.

5.2.1 Experimental Setup

We evaluate the performance and scalability of the SPCKM framework proposed in this paper with a set of experiments conducted on an IBM Pentium IV computer with an Intel 2.66GHz processor and 512MB of RAM. Our evaluation is conducted on a write-intensive file to simulate a scenario in which re-encryption for data security is necessary. Therefore, we do not compare our approach with the lazy re-encryption technique which as we mentioned before (see Section 2.3.4) is better suited to read-intensive scenarios. We compare both approaches using various metrics, namely the degree of availability, cost of key generation and data encryption, the cost of key replacement, cost of message communications, and the size of the window of vulnerability. Although the test scenarios are not exhaustive, the experiments give an intuition about the general performance of the schemes under consideration. Results for each case are obtained from averages of over 10 runs, with random numbers of rekey requests expressed as proportions of a user group with a maximum of 100 members and files (versions of the shared data) of size $\approx 32MB$.

5.2.2 Prototype Description

We build our prototype on the Microsoft Window XP platform using the Java 2 Standard Development Kit and Eclipse [87, 16]. We design the prototype in the form of a chat system using socket programming and a client-server model. In the access control class, the clients play the role of users and the server that of the key and data server, supplying both keys and allowing access to data. The key server in our prototype, generates Triple DES (Data Encryption Standard) encryption/decryption keys and the data server keeps an encrypted log file of each group's communications.

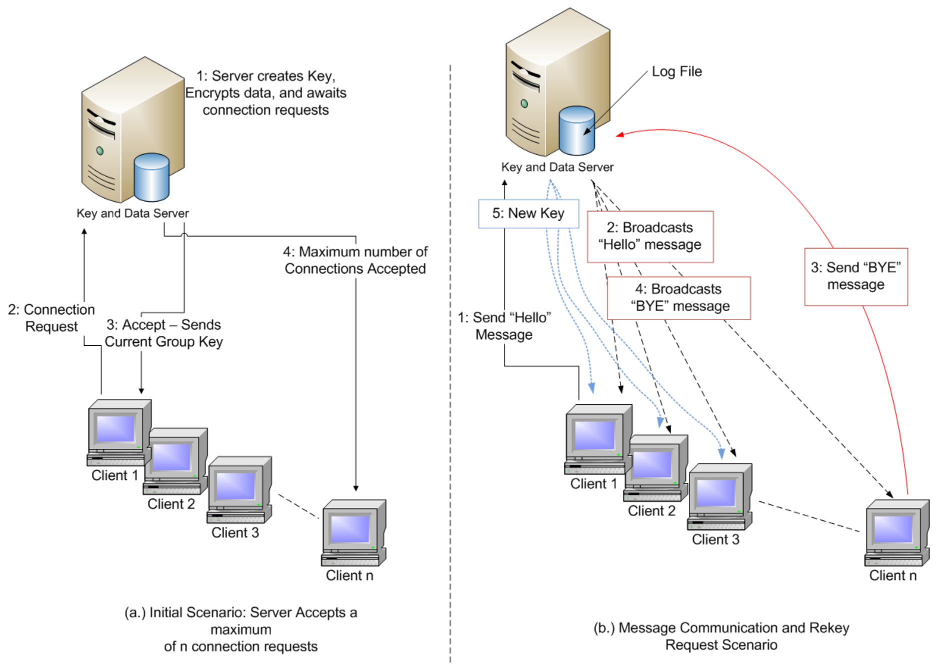

1: Server creates Key, Encrypts data, and awaits connection requests
Key and Data Server
2: Connection Request
3: Accept – Sends Current Group Key
4: Maximum number of Connections Accepted
Client 1
Client 2
Client 3
Client n
(a.) Initial Scenario: Server Accepts a maximum of n connection requests
Log File
Key and Data Server
5: New Key
2: Broadcasts "Hello" message
4: Broadcasts "BYE" message
3: Send "BYE" message
1: Send "Hello" Message
Client 1
Client 2
Client 3
Client n
(b.) Message Communication and Rekey Request Scenario

Figure 5.5: Prototype Scenario for the SPCKM Implementation

The operation of our prototype is better explained using Figure 5.5. As shown in Figure 5.5, the clients (users) communicate via the server and all communications are saved into a log file. Access to the log file is granted only if a user holds the correct group key. Once the server is initialized it waits until it receives a connection request at which point it checks to ensure that it has a free socket. If this is the case, it spawns a thread that allows the client to connect to it. The server then displays the current group key on the client's message board as well as the communications that have occurred since they joined the group. A client disconnects from the group by transmitting a 'BYE' message to the server, and the server, on reception of this message, closes the connection and broadcasts a message to the other group members indicating the departure event. The message also contains the updated group key.

The prototype starts off the self-protecting model by using the initial connection and disconnection requests to collect data empirically. This data is stored in a file that serves as the knowledge base for the server and the server uses this data to start running the feedback control loop that performs adaptive KM.

5.2.3 Performance Criteria

The performance criteria we use to evaluate the SPCKM framework are the Response Time, Size of the Window of Vulnerability, Percentage of requests satisfied with respect to rekey request arrival rate, and Server processing cost. We explain these four criteria below.

- *Response Time:* We measure response time by calculating the time it takes the server to transmit a new key in response to a rekey request. In the case of the SPCKM framework, if a new key exists, this is the time required to transmit the new key to all the users remaining in the group. In the BKM scheme, this is the time required to generate a new key, re-encrypt the data, and distribute the updated key to the users remaining in the group.

- *Window of Vulnerability:* The window of vulnerability is the period that starts from the time a rekey request leaves a user to the point when the users remaining in the group receive the updated key. So, it is calculated as the rekey request transmission time to the server plus the response time.

- *Percentage of requests satisfied with respect to rekey request arrival rate:* We calculate the percentage of requests satisfied with respect to the arrival rate as the number of keys that get transmitted in response to X number of rekey requests that occur during a time window W_x.

- *Server Processing Cost:* These are the costs the server incurs in handling adaptive KM and include the average number of replicas (backup data versions) maintained to ensure that the observed request arrival rate is handled within the current time window, the costs of updating, and the average percentage of requests satisfied.

5.2.4 Experimental Results

In the first experiment we evaluate the comparative response times per request of the BKM and SPCKM schemes with respect to the rekey request arrival rate. We define response time in this case as the time it takes the key server to assign all the group users a new key after it has received a rekey request or voluntarily decided to replace the current group key. The size of the monitoring window is set statically to a value of 60 seconds. The sum of rekey requests is computed by adding the number of requests that arrive during the monitoring period and the monitor computes the arrival rate using equation 5.1. The experiment was repeated 10 times for each case with an average interval of 2 to 5 seconds between each request, and results averaged and plotted in Figure 5.6. The error bound for each point plotted is ±10 seconds in the basic scheme (no replication, i.e., no backup data versions), ±10 seconds in the SPCKM scheme (varied or adaptive replication, i.e., varied number of backup data versions) and ±5 seconds in the SPCKM scheme (static replication : 1 replica, i.e., one backup data version).

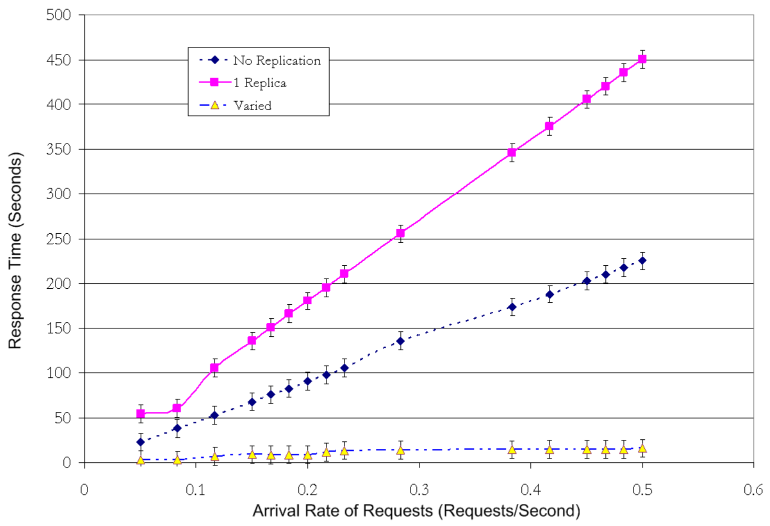


Figure 5.6: Average Request Satisfaction Time with respect to number of Replicas

We observed that the response time in the basic key management (BKM - no replication) and the SPCKM (static replication) schemes grows linearly with an increasing arrival rate of rekey requests whereas the response time in the SPCKM (adaptive replication) scheme as the number of rekey requests increases, little or no additional time is required. These results allow us draw the conclusion that the response time in the BKM (no replication) scheme is affected by an increased arrival rate of rekey requests which is a consequence of the fact that the key server keeps stopping and restarting the key generation and re-encryption process for each rekey request it receives. So for high rekey request arrival rates (i.e. an increase in the number of rekey requests), as time goes on, it takes longer on average to respond to a rekey request. The condition is made worse with static replication, because an increased arrival rate implies that the server needs to create a new backup data version as soon as a replacement occurs, and when the arrival rate is high this implies that the server will still be re-encrypting the new data version when it receives a new rekey request.

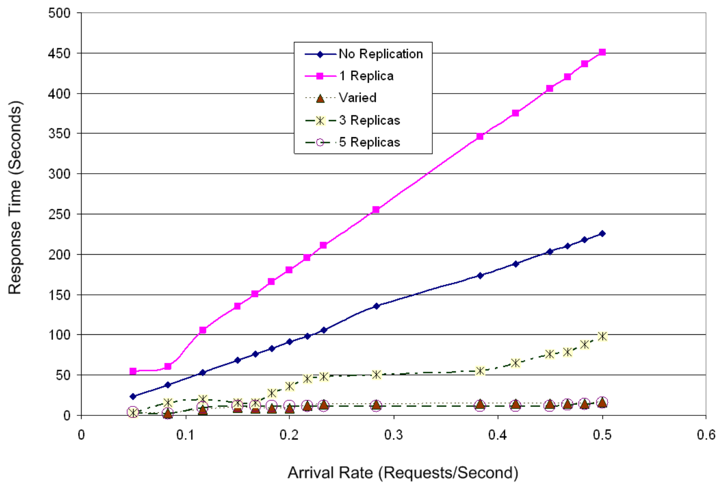


Figure 5.7: Effect of Static Replication on Average Request Response Time

Again the server keeps starting and stopping encryptions, and the condition is worse than when no replication is used because the server needs to create both two data versions each time. By contrast, in the SPCKM (adaptive replication) scheme, since the key server creates a new backup data version of the file and creates supporting keys in anticipation of a predicted volume of requests, response time is generally equivalent to the average time it takes to transmit the new key on average. Moreover, the SPCKM (varied replication) scheme further minimizes its key replacement costs by adaptively adjusting the number of replicas (data versions) in response to the arrival rate of rekey requests.

Since we noted that the adaptive scheme only requires a maximum of 10 replicas (data versions) generally, we also tested cases where the number of replicas in the SPCKM scheme is fixed statically at $1, 3$, and 5, and compared the performance of the SPCKM scheme to a BKM scheme in which there is no replication at all. From the plot shown in Figure 5.7, we observed that a larger number of static replicas keeps the response very close to the values obtained in the static scheme. However, when this value is significantly smaller than the number required to satisfy the requests, the server takes much longer to respond to key requests (see plot with 3 or 1 backup data versions) than it does when it is not supported by replication or uses adaptive replication.

Our second experiment evaluates the size of the window of vulnerability created in both the BKM and SPCKM schemes during rekeying. The window of vulnerability is defined as the time elapsed between an emission of a rekey request by a user to the key server up until the time of reception of the new key by the users remaining in the group. We ran each experiment 10 times, each time over a 60 second time window and the results were averaged and plotted in Figure 5.8. The error bound for each of the plotted bars is ± 2 seconds.

We noted that the size of the vulnerability window in the BKM and the SPCKM (static replication) schemes grows linearly with an increasing arrival rate of rekey requests whereas that in the SPCKM (adaptive replication) scheme, as the number of rekey requests increases little or no additional time is required. This supports the result we obtained in the first experiment where an accumulation of rekey requests in the BKM and SPCKM (static replication) schemes results in a longer average response time per request. We note also that the SPCKM (adaptive replication) scheme not only overcomes the drawback of delayed response times in the BKM and SPCKM (static replication) schemes but also makes for better security by reducing the size of the window of vulnerability between rekey requests.

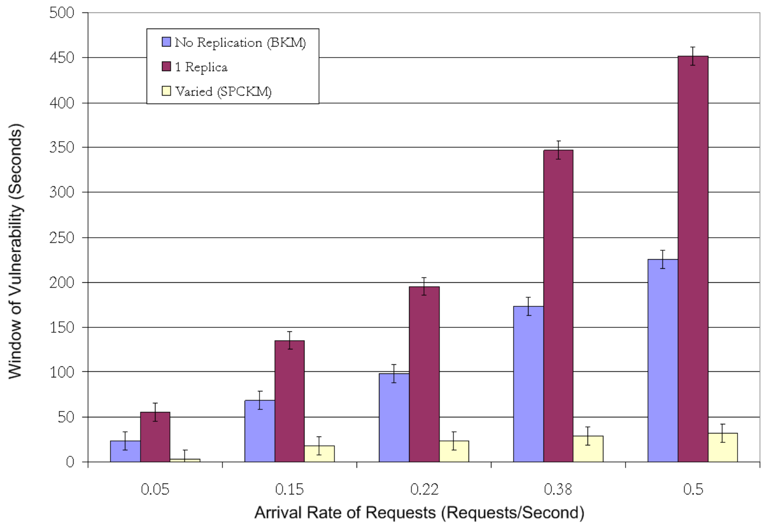


Figure 5.8: Variation in Vulnerability Window Size with Respect to Request Arrival Rate

Finally, we discuss the processing cost incurred by the key and data servers. We noted that for an average rekey request arrival rate of 0.256 requests/second the SPCKM (adaptive replication) scheme uses an average of four data versions against one in the SPCKM (static replication) and none in the BKM schemes. In the SPCKM (adaptive replication) scheme, the average time spent updating (copying messages on to the primary copy and re-encrypting in the case of rekeying) the backup data versions is 0.17 seconds in the BKM scheme and 3.3 seconds in the SPCKM scheme with static replication, as opposed 6.6 seconds in the SPCKM scheme. However, the SPCKM (adaptive replication) scheme makes up for its shortcomings on the backup data version creation and update side by satisfying an average of 52.27% of the requests during the 60 second montoring interval while the basic scheme only satisfies 20.96%, and the SPCKM with static replication only satisfies 8.69%. In fact as shown

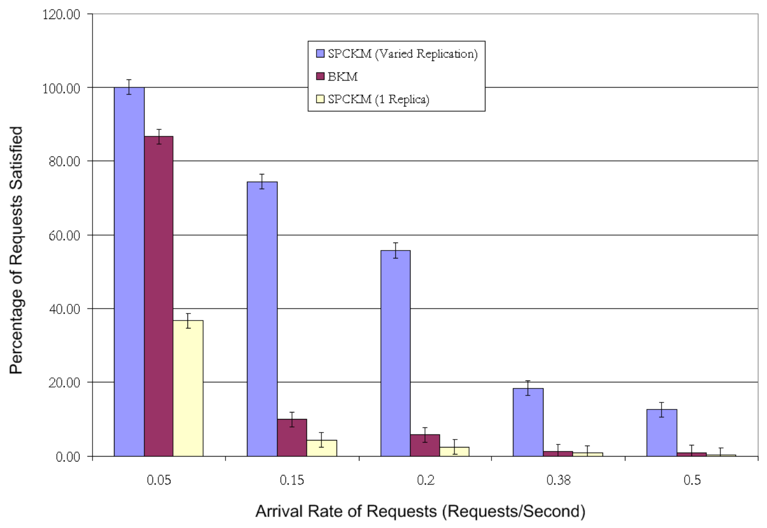


Figure 5.9: Variation in Percentage of Rekey Requests Satisfied with Respect to Request Arrival Rate

in Figure 5.9, in our experiments when the arrival rate was 0.5 requests/second during the 60 second monitoring interval, the BKM and the SPCKM (static replication) schemes only satisfied less than 1% and 0.33% of the requests they received while the SPCKM (adaptive replication) scheme initially, (i.e. before adjusting to the new rate) satisfied 12.67% of the requests. The results are summarized in Table 5.2.4.

Table 5.2: Effect of Replication (data version creations) on Key Management Scheme Performance

	None	***Static***	***Adaptive***
Number of backup data versions	0	1	4
Update Costs (seconds)	0.17	3.3	6.6
% Requests Satisfied	20.96	8.69	52.27

5.3 Discussions

This section summarizes the contributions of our adaptive key management framework and shows that the decision version of the problem of adaptive rekeying in a hierarchy is NP-complete. The reason for doing this is to demonstrate that performance improvements are only possible if done in a way that minimizes the overall cost of rekeying. Therefore rekeying in hierarchies needs to be handled by some form of priority scheduling to minimize the risk of cyclical rekeying which would create an added management cost to the key server. In fact the performance degradation in this case could even result in the adaptive scheme performing worse than any of the quasi-static key management schemes that we have discussed so far, which will defeat the advantage of adaptivity in key management.

5.3.1 Contributions of the SPCKM Framework

In the preceding sections we outlined some of the reasons behind the hesitancy to adopt the autonomic computing paradigm into security frameworks. For reasons pertaining to cost and credibility, business owners prefer to have control over their security mechanisms. Our aim therefore, was to argue that self-protecting approaches can enhance the performance of standard KM schemes without necessarily changing their underlying security, thus making the job of the SA easier.

Specifically, we considered the problem that arises in shared data scenarios where access is controlled with a CKM scheme. In these scenarios, several users hold a secret key that is used to encrypt commonly accessible data. When group membership changes, data security is maintained by updating the shared group key and transmitting it to the users remaining in the group. Therefore, KM is expensive when changes occur frequently and involve large amounts of data.

We proposed a simple but effective SPCKM framework to address this problem. The framework does not detract from the basic qualities of the standard security scheme but rather enhances its capabilities with a combination of a stochastic model and replication. The stochastic model determines an acceptable degree of replication to maintain based on an observed arrival rate and the potential impact of checkpointing on the overall performance of the system. In this way data versions and keys are generated to preemptively handle situations of high demand making for better performance than in standard schemes. The SA now only has to preset required parameters and let the system run, without having to be present to manually handle every change. Experimental results show that the SPCKM approach reduces the vulnerability window and increases data availability in comparison to the BKM.

Regarding the security of the scheme, if we assume that a key generated in the BKM scheme using the Triple DES (Data Encryption Standard) scheme is secure then it is safe to say that both the BKM and SPCKM schemes are secure in the sense that the SPCKM scheme only enhances the performance of the basic scheme by adding in data replication. Checkpointing on data versions are secured by ensuring that updates are accepted only from the primary replica. Rekeying results in a destruction of the primary replica associated with the updated key and the selection of a new primary copy by the key server.

5.3.2 Some Challenges in Adaptive Rekeying

Other challenges include finding other statistical distributions that are more effective than the Poisson model in modeling rekey arrival rates, and finding a good way to define adequate monitoring thresholds. Issues of copy consistency can also arise in situations where there is a high volume of communications between users (frequent updates on the primary) and rekey requests occur within very short intervals of each other. One of the more interesting problems that can occur in this scenario of adaptive rekeying is that of cyclical rekeying when rekey events occur within very short intervals of each other.

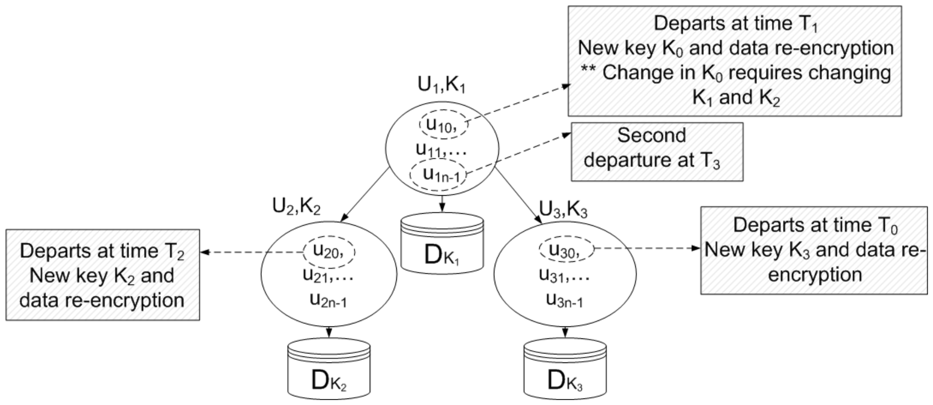


Figure 5.10: A Case of Cyclical Rekeying

A case of cyclical rekeying is given in Figure 5.10. In this case, four rekey requests R_0, R_1, R_2, R_3 are emitted from different classes U_3, U_1, U_2 at times T_0, T_1, T_2, T_3. Assuming that the rekey requests are totally ordered, we can infer that the key server will respond to the rekey requests in the order in which they arrive. The key server responds by transmitting a new key to the users remaining in a group and does not handle a new request until the current one is completed. The problem of cyclical rekeying occurs in the DKM scheme, when a rekey request requires repeating the key transmission process for a class that has just been rekeyed. In the case shown inf Figure 5.10, the request R_1 requires changing the keys K_1, K_2, and K_3, which implies rekeying K_3 that had just been updated when R_0 occurred. Likewise, for R_2 and

R_3 doing one after the other results in a loop situation because K_2 is updated at T_2 and reupdated at T_3 when R_3 occurs requiring a change of the keys K_1, K_2, and K_3. When requests like these that occur within short intervals of each other they create a situation of cyclical rekeying which is expensive not only in terms of key generation but also in data version encryptions. In fact, as we show in the following section this problem is NP-hard [86, 24] which explains why we used a priority heuristic to handle adaptive rekeying in the experiments conducted throughout the entire hierarchy.

5.3.3 The Adaptive Rekey Scheduling Problem

The adaptive rekey scheduling problem (RSP) can be expressed in the form of a complete graph in which there are n vertices. The vertices represent the user groups in the hierarchy and the associated keys and the RSP is one where a central authority wishes to assign keys to every node (vertex) in the RSP graph in way that minimizes the overall cost of key assignment. In order to minimize the total cost of key assignment the key server needs to compute a schedule that guarantees that for each set of scheduling tasks, each vertex will be assigned a key exactly once and the path that defines the rekey schedule contains no cycles (i.e. there is no repeated rekeying).

In order to show that the decision version of the RSP is NP-complete, we need to show two things. First, that the problem belongs in NP and secondly, that the problem is NP-hard. Suppose there is an integer cost $c(i, j)$ to rekey a security class say U_i before U_j and that the key server needs to select a sequence whose total cost is minimum, where total cost is defined as the number of cycles that will occur when U_i is rekeyed before U_j. The decision version of this problem can be expressed as follows:

$$\begin{aligned} RSP = < G, c, k > \quad : \quad & G = (V, E)) \textit{ is a complete graph} \\ & c \ \textit{ is a function from } V \times V \rightarrow Z \\ & k \in \ Z \\ & G \textit{ has a RS path with cost at most } k \end{aligned}$$

Theorem 5.3.1. *The decision version of the RSP is NP-complete.*

Proof.

First, we show that the RSP belongs in NP. Given an instance of the problem, we use as a certificate a sequence of n nodes in the rekey schedule. The verification algorithm checks that this sequence contains each vertex exactly once, sums up the edge cost and checks whether the sum is at most k. This process can be done in polynomial time.

In order to prove that RSP is NP-hard, we show that the traveling salesman problem (TSP) is reducible to the RSP (i.e. TSP$\leq_P$RSP) [24]. The TSP basically involves modeling a problem as a complete graph with n vertices, and requires a salesman to visit all n vertices (cities) visiting each vertex exactly once and using a tour whose total cost is minimum. The total cost of the tour is the sum of the costs along the edges of the tour.

Suppose $G = (V, E)$ is an instance of the TSP, then an instance of the RSP can be constructed as follows. First, we define a complete graph $G' = (V, E')$, where $E' = (i, j) : i, j \in V \textit{ and } i \neq j$ and the associated cost function along an edge in the graph is defined by:

$$c(i, j) = \begin{cases} 0 & \text{if } (i, j) \in E \\ 1 & \text{if } (i, j) \notin E \end{cases}$$

Since G is undirected, it has no self-loop, so $c(u, v) = 1$ for all vertices $(u, v) \in V$. The instance of the RSP is then $< G', c, 0 >$ which is easily formed in polynomial time.

In order to show that the graph G has a traveling salesman tour if and only if graph G' has a tour of cost at most 0, we can suppose that G has a traveling salesman tour h. Each edge in h belongs to E and thus has a cost of 0 in G'. Thus h is a tour in G' with cost 0. Conversely, suppose that graph G' has a schedule h' of cost at most 0. Since the costs of the edges E' are 0 and 1, the cost of tour h' is exactly 0 and each edge on the schedule path must have cost 0. Therefore h' contains only edges in E and it can be concluded that h' is a TS tour in graph G. □

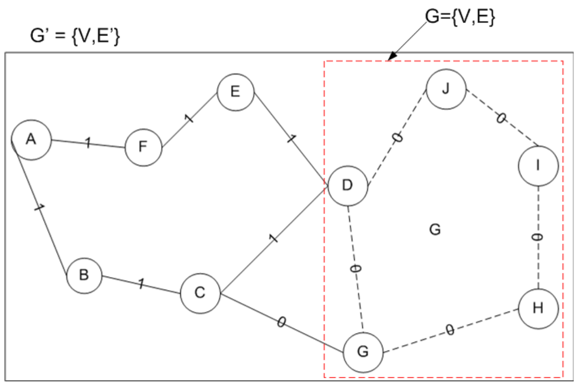


Figure 5.11: Illustration of polynomial-time reduction from TSP to RSP

The proof is better explained with the example shown in Figure 5.11. Here, the graph G' has edges of cost 0 and 1 respectively. So in this case an example of a tour h is $GHIJDG$ while one of h' is $ABCDEFA$. In this case, it is clear that the graph G is a subset of the graph G' which supports the argument that TSP$\leq_P$ RSP.

Chapter 6

Collusion Detection and Resolution

"Imagination is more important than knowledge."

*–**Albert Einstein***

A fundamental challenge in designing secure hierarchical cryptographic access control schemes is protecting against attacks perpetrated by valid users in possession of "correct" keys. Attacks that occur due to inference and covert channels as well as through the use of keys that are illegally obtained by applying contrived mathematical functions to known keys, highlight a few of the possibilities of internal security violations. Therefore, attacks that are carried out by valid users are more difficult to detect and so more dangerous to the security of any system.

We now consider one of the attacks, termed a "collusion attack", that can be provoked by valid users. As mentioned before, a collusion attack occurs when two or more users at the same level in the hierarchy collaboratively compute (using a contrived mathematical function), from their respective keys, a key (higher up in the hierarchy) to which they are not entitled. Collusion detection and resolution is an important issue in guaranteeing the security of any key management scheme because it enables the scheme to enforce precautionary measures against collusion attacks which can be carried out by users in possession of valid keys.

In previous chapters we noted that evaluating the security of a KM scheme involved determining whether or not the scheme is provably secure. The aim of our adaptive collusion detection and resolution framework is to show that the process of evaluating the security of a generated key set can be done automatically and also that in certain cases, it can be more cost effective to use a less secure scheme that is supported by some form of verification algorithm.

We begin by showing that the problem of optimally identifying all the possible collusion liable key combinations within a key set is NP-hard, and is hard to solve algorithmically in polynomial-time. Our proposed solution is based on two heuristic algorithms. The first algorithm maps the key set onto a graph according to known collusion functions while the second algorithm uses a heuristic based on computing an independent set of a graph. For simplicity and coherence, we have designed our algorithms around the Ad-hoc AT scheme since this is an example based on the same key generation function that the SPKM, CAT and MCK schemes use, and is vulnerable to “collusion attack”. In order to evaluate the performance of our collusion detection and resolution algorithms, we use them to support the Ad-hoc AT scheme and make the comparison with the CAT scheme. This is because both schemes represent the two extremes - the CAT scheme is secure against collusions but generally more costly to run than either the SPKM or MCK schemes, and the Ad-hoc scheme represents the other extreme - less costly to run than the CAT, SPKM, or MCK schemes - but vulnerable to “collusion attack”.

6.1 On Detecting Collusion Possibilities

In this section we show that the decision version of the problem of determining the largest set of "collusion free" keys within a key set (referred to hereafter as the DCFK problem), is NP-complete. The term "collusion free" is used to denote the keys within a key set that cannot be combined using any known mathematical function to illegally obtain other keys in the hierarchy. For instance, when a KM scheme is based on the model that Akl and Taylor proposed, the Properties 1,2, and 3 (see Chapter 3, Section 3.2) need to be enforced by generating and assigning keys in a way that prevents violation. The DCFK problem is hard to solve algorithmically because it requires determining the largest sub-set of collusion-free keys within a key set, such that all the security classes requiring keys are assigned different keys and also so that there are enough extra ones to handle the predicted request arrival rates during future monitoring intervals, say W_{c+1}.

6.1.1 The DCFK problem

The DCFK problem can be expressed in the form of an undirected graph comprised of z vertices, where the vertices represent the keys assigned to the security classes at level i in the hierarchy and the edges represent the probability that their end points can be combined to provoke a "collusion attack". The DCFK problem requires the key server to compute the largest "collusion-free" key set from a given key graph. Restating this optimization problem as a decision problem, we wish to determine whether a graph has a "collusion-free" set of size z, and we can define a language to describe the problem as follows:

$$DCFK = \{\langle G, z\rangle \; : \; \textit{graph } G \textit{ has a “collusion-free” key set of size } z\}.$$

Theorem 6.1.1. *The decision version of the DCFK problem is NP-complete.*

Definition 6.1.1. Given a graph, $G = (V, E)$ where V represents the set of vertices in G and E the set of edges that connect the vertices, a set of vertices $I \subset V$ is called ***independent*** if no pair of vertices in I is connected via an edge in G; an independent set is called ***maximal*** if by including any other vertex not in I, the independence property is violated [1, 50].

Proof.

In order to show that the DCFK problem is NP-complete we need to show that Theorem 6.1.1 is true by proving two things, first that the DCFK problem belongs in NP and secondly that the DCFK problem is NP-hard [24]. First we show that the DCFK problem belongs in NP. Suppose we are given a key graph, $G = (V, E)$ and an integer z for a level l in the key management hierarchy. The verification algorithm needs to check that the graph contains a subset of unconnected keys V', such that $|V'| = z$ where z is the number of security classes at level l, in the hierarchy, requiring keys. Since the value of z is known, the verification algorithm can check the set V in polynomial time to determine whether or not there is a set V' within V such that $|V'| = z$. Therefore the DCFK problem belongs in the set of NP problems.

In order to show that the DCFK problem is also NP-hard, we show that the maximum independent set (MIS) problem is reducible to the DCFK problem (i.e.$MIS \leq_P DCFK$) [24]. Essentially, the decision version of the the MIS problem is to determine whether a given undirected graph $G = (V, E)$ contains an independent set of cardinality at least z. The independent set is called maximal when the subset V' that is extracted from V is such that $|V'| = z$ and no other independent set in V completely contains V'.

Suppose that a graph G has an MIS such that $V' \subseteq V$ with $|V'| = z$ and a DCFK such that $V - V' \subseteq V$ with $|V - V'| = |V| - z$. Now, let u and v be two arbitrarily selected vertices in V connected by an edge, so that at least one of u or v belongs in V' since every vertex in V' is not connected with an edge in E. Equivalently, at least one of u or v belongs in $V - V'$ and the set $V - V'$ which has a size of $|V| - z$ forms a DCFK for the graph $G - G'$.

Conversely, suppose that $G - G'$ has a DCFK such that $V - V' \subseteq V$, where $|V - V'| = |V| - z$. Then for all $(u, v) \in V - V'$, if (u, v) is connected by and edge in E, then either of $u \in V - V'$ or $v \in V - V'$ or both belong in $V - V'$. The contrapositive of this implication is that for all $(u, v) \in V$ if $u \notin V - V'$ and $v \notin V - V'$, then $(u, v) \in V'$. In other words, $V - |V - V'|$ is a MIS and it has a size $|V| - |V - V'| = z$ □

6.1.2 Example

The following example helps to simplify and clarify the proof given above. In this case, we use the example of a graph G that contains a MIS and some connected vertices. As shown in Figure 6.1, graph G includes both graphs G and $G - G'$ where graph G' has connectivity 0 and graph $G - G'$ has connectivity 1. Here, connectivity 0 implies that there is no edge between the two adjacent vertices and connectivity 1 implies that there is an edge between the two adjacent vertices. Clearly, since graph G' includes vertices $GHIJ$ it forms a MIS for graph G. Hence, adding any vertex from either G' into $G - G'$ or vice-versa violates the rules of connectivity in either case. So, the problem of computing a MIS of G is equivalent to that of computing the DCFK of a key set.

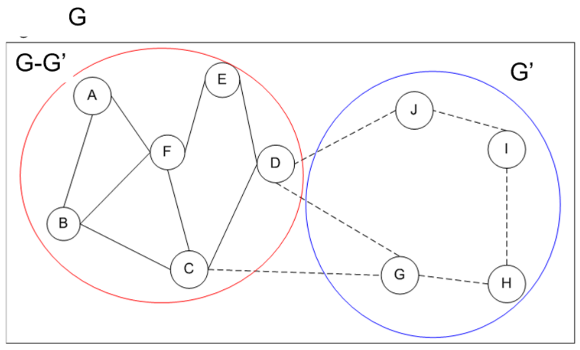


Figure 6.1: Illustration of polynomial-time reduction from MIS to DCFK

Since the DCFK problem is NP-Complete, we support our adaptive collusion detection and resolution (ACDR) framework with two polynomial-time "heuristic" algorithms to obtain a near-optimal solution to the DCFK problem. The size of the set of keys produced by the algorithms covers all the security classes and is provably collusion-free.

6.2 Adaptive Collusion Detection and Resolution (ACDR) Framework

The Adaptive Collusion Detection and Resolution (ACDR) framework is comprised of four basic components: the Key Generation Module (KGM), the Collusion Detection Module (CDM), the Collusion Resolution Module (CRM) and the Rule Set (RS), embedded within the *Effector* in the Self-Protective Cryptographic Key Management (SPCKM) framework (see Figure 5.1), that we presented in Chapter 5. As shown in Figure 6.2, our framework is comprised of four basic components embedded with the *Effector* that is situated at the key server. When the *Effector* receives instructions from the *Executor* to create keys to handle a rekey request, the

Effector begins by generating the set of keys, required to satisfy the demand, within the Key Generation Module (KGM).

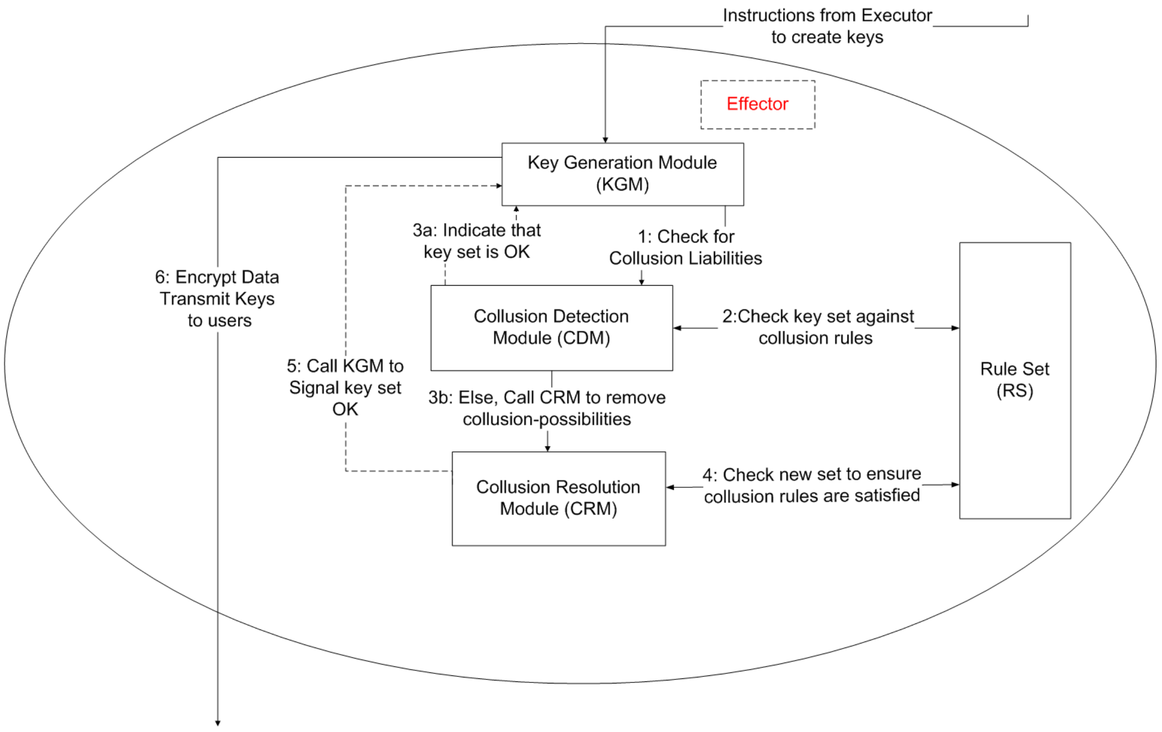

Instructions from Executor to create keys
Effector
Key Generation Module (KGM)
3a: Indicate that key set is OK
1: Check for Collusion Liabilities
6: Encrypt Data Transmit Keys to users
Collusion Detection Module (CDM)
2:Check key set against collusion rules
Rule Set (RS)
5: Call KGM to Signal key set OK
3b: Else, Call CRM to remove collusion-possibilities
Collusion Resolution Module (CRM)
4: Check new set to ensure collusion rules are satisfied

Figure 6.2: Collusion Verification and Resolution Framework

The KGM generates a set of keys to handle the predicted arrival rate during W_{c+1}, and transmits the key set to the CDM where they are checked to identify any combinations that could be exploited to provoke collusions. In order to check for collusions, the CDM uses a collusion detection algorithm that operates by mapping all generated keys onto a key graph where each vertex represents a key and the edges indicate the probability that their end points can be combined to generate an illegal key that can be used to provoke a “collusion attack”. The CDM establishes which keys are “collusion-liable” by consulting the Rule Set (RS).

The RS contains rules or values that a key set needs to maintain in order to avoid collusions. If the key set passes the verification step (stage at which collusions are detected), then the key set is transmitted to the data server for data encryption. On the other hand if the verification step fails to produce a collusion-free key set, the CDM transmits the key set to the Collusion Resolution Module (CRM) where an algorithm is applied to the key set to break the connectivity between the keys. Once the graph has been formed, the CDM sends the graph to the CRM where an algorithm that is based on the concept of maximum dispersion, implemented by computing an independent set of the key graph, to minimize the connectivity of the keys and therefore reduce the occurrence of collusion attacks. Once a collusion-free key set is obtained the key set is transmitted to the data server for data encryptions, after which the keys are distributed to the users.

The framework allows the key server to verify that the keys that are newly generated do not inadvertently open up possibilities for violating the conditions to prevent collusions. A further advantage is that the SA can pre-set these conditions and allow the scheme to run on its own and only handle abnormal cases that require the SA's consent/advice to proceed. We begin by stating the assumptions and basic definitions that support our collusion detection and resolution algorithms and then proceed to outline the operation of the algorithms.

6.2.1 Preliminaries and Assumptions

We assume that the rules of construction of the poset remain valid and that the central authority (CA) generates keys for each level in the hierarchy in a manner that enforces these rules. Additionally, we also assume that the keys generated, at each level, can be mapped on to a graph whose structure is defined by the number of possibilities of combining keys to generate a "collusion attack". The keys represent the vertices in the graph while an edge between any two vertices indicates that both keys can be combined to compute an illegal key. As an example, Figure 6.3(a.) depicts a hierarchy with a series of possibilities for collusions. One such case is that of combining the keys $K_3 = K_0^4 \bmod M$ and $K_5 = K_0^9 \bmod M$ to generate the key K_0 which belongs to the highest class in the hierarchy. In this case, K_3 and K_5 can be used to trigger a collusion attack through the following function:

$$\left(K_0^4\right)^{-2}\left(K_0^9 (\text{mod } M)\right) = K_0^1 \,(\text{mod } M) = K_0 \qquad (6.1)$$

whereas, by the same function, combinations of K_3 and K_4 or K_4 and K_5 as shown below

$$\left(K_0^4\right)^{-2}\left(K_0^6 (\text{mod } M)\right) = K_0^{-8} K_0^6 \,(\text{mod } M) = K_0^{-2} \,(\text{mod } M) \qquad (6.2)$$

$$\left(K_0^6\right)^{-2}\left(K_0^9 (\text{mod } M)\right) = K_0^{-12} K_0^9 \,(\text{mod } M) = K_0^{-3} \,(\text{mod } M) \qquad (6.3)$$

yield results that do not belong in the hierarchy. If either or both conditions are verified the edge between the classes is labeled with a value of 1, otherwise it is labeled with a value of 0. So, as shown in Figure 6.3(b.), according to this collusion function, there would be an edge between $K_3 = K_0^4 \bmod M$ and $K_5 = K_0^9 \bmod M$, whereas there is no edge between $K_3 = K_0^4 \bmod M$ and $K_4 = K_0^6 \bmod M$ or between $K_4 = K_0^6 \bmod M$ and $K_5 = K_0^9 \bmod M$. This can lead to a very sparse graph or a very dense graph in the best and worst cases, respectively. The following three definitions are mainly to extend the above explanations to the general case.

Definition 6.2.1. Key connectivity is implied by the ease with which any two keys at a level, say j, can be combined to generate a key at a higher level, say i. Thus, adjacent vertices would be more likely to be combined successfully to derive an illegal key than non-adjacent vertices.

Definition 6.2.2. The graph $G = (V, E)$ is the representation of key connectivity where V represents the keys and E the edges (probability that their end points can be combined to provoke a "collusion attack") between these keys (see Figure 6.3(b.))

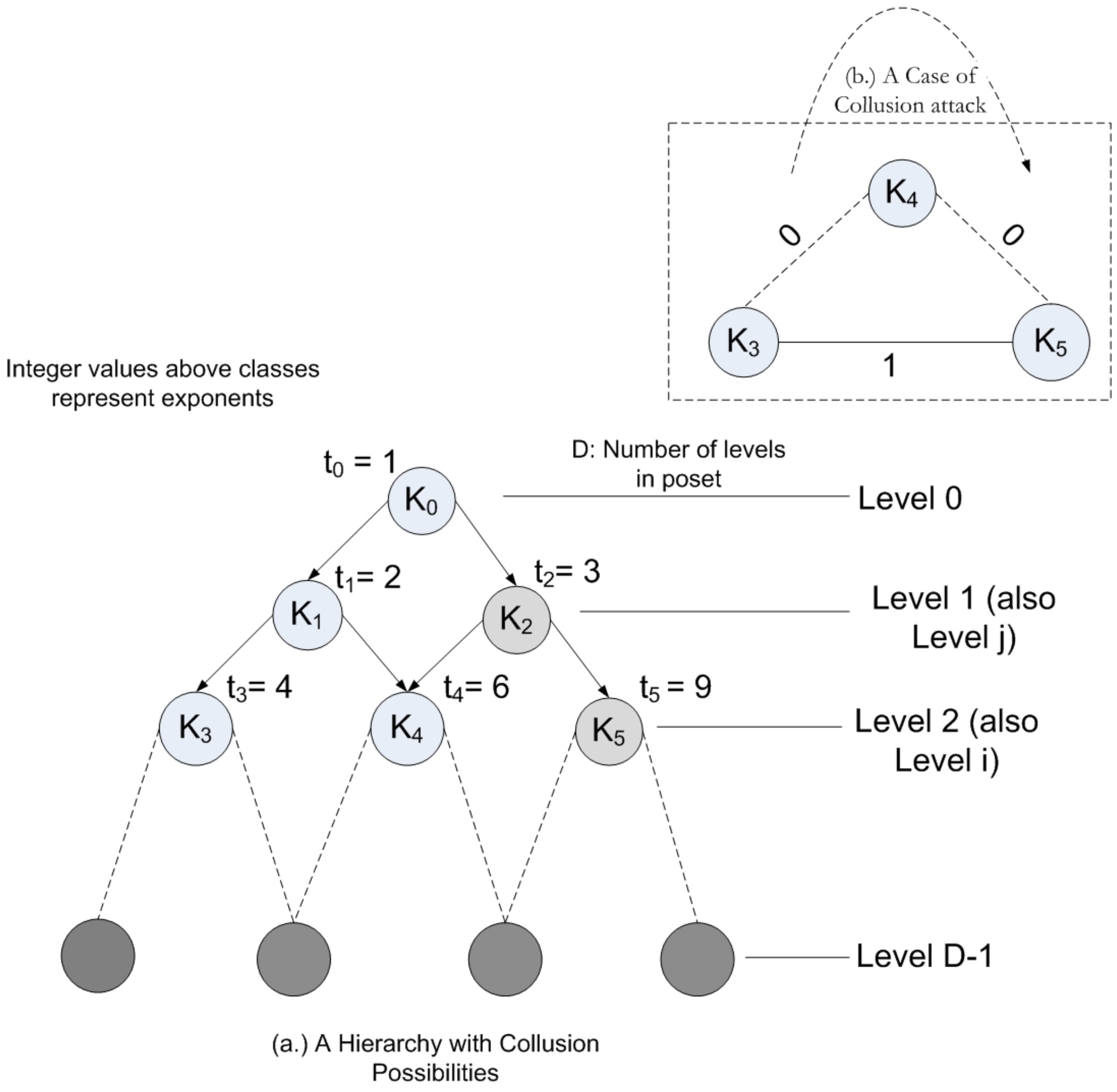


Figure 6.3: Example of a Hierarchy with Collusion Possibilities

The need to find large independent sets typically arises in dispersion problems where the objective is to identify a set of mutually separated points. For example in our case, we want to assign keys at every level in the access control hierarchy such that no two keys are close enough to be combined to derive another key (particularly if that key belongs to a higher level). The principal weakness of many CKM schemes lies in their vulnerability to attacks based on key connectivity.

Note that the edges of G are obtained based on all best current knowledge of possible attacks. The absence of an edge between two keys in no way guarantees that these two keys cannot be used in a "collusion attack".

The largest independent set would indicate the maximum possible number of keys that can be attributed such that the conditions for security are not violated, and that the set derived can accommodate the demands of the system in order that all the classes in the system get a unique key. Since the problem of determining a largest independent set is NP-hard we use a heuristic to obtain an approximate solution in polynomial time.

6.2.2 Collusion Verification

On reception of the instruction to create a key set to satisfy the predicted number of rekey requests, the KGM generates a set of keys according to the rules of access in the hierarchy that the SA specified. When the scheme is based on the principle of key management in the Ad-Hoc AT scheme, integer value exponents are randomly assigned to each of the security classes in the hierarchy. The generated exponent set is then transmitted to the Collusion Detection Module (CDM) where it is checked to identify whether any of the exponents that exist therein will result in the formation of collusion liable keys. In order to identify collusion liable keys the CDM uses a collusion detection algorithm that intuitively works as follows.

Basically, the algorithm works by checking the possibility of any of the keys being combined to provoke "collusions". The collusion detection algorithm checks for collusion possibilities based on known combination rules. These rules are located in the rule set (RS). Since the algorithms we have proposed extend the CAT scheme, the RS contains the greatest common divisor of the exponents at each of the levels in the hierarchy, as well as the registry of assigned exponents at every level in the hierarchy. The RS is formed by computing the GCD at every level of the hierarchy and inserting the computed value and the values of the exponents into the table. In order to determine whether or not a key combination is collusion liable, a greatest common divisor heuristic (see Section 3.2) is used to compute the GCDs of the exponents used to compute the keys at level l. We express the conditions that the collusion detection algorithm uses to determine collusion-possibilities as follows:

- ***Condition 1:*** Let $e_{l,y}$ be the exponent that is used to compute the key belonging to class y at level l. A collusion-possibility arises if the the greatest common divisor (GCD) of all the exponents at level l is equal to any of the GCDs of the exponents at levels $l-1, ..., 0$. More formally we can express this as follows:

  $$\gcd(e_{l,0}, ..., e_{l,y}) = \gcd(e_{0,0}, ..., e_{0,y}) \vee ... \vee \gcd(e_{l-1,0}, ..., e_{l-1,y})$$

  where $GCD[l] = \gcd(e_{l,0}, ..., e_{l,y})$

- ***Condition 2:*** The GCD of one or of the pairs of exponents assigned at level l is a divisor of one or more of the exponents at levels 0 to $l-1$, indicating that there is a collusion attack possibility at level l. we express this more formally as follows:

$$\gcd(e_{l,0}, e_{l,1}) \vee ... \vee \gcd(e_{l,y-1}, e_{l,y}) \mid (e_{0,0} \vee ... \vee e_{l-1,y})$$

Algorithm 3 : Collusion Detection $(GCD[], h, nd)$

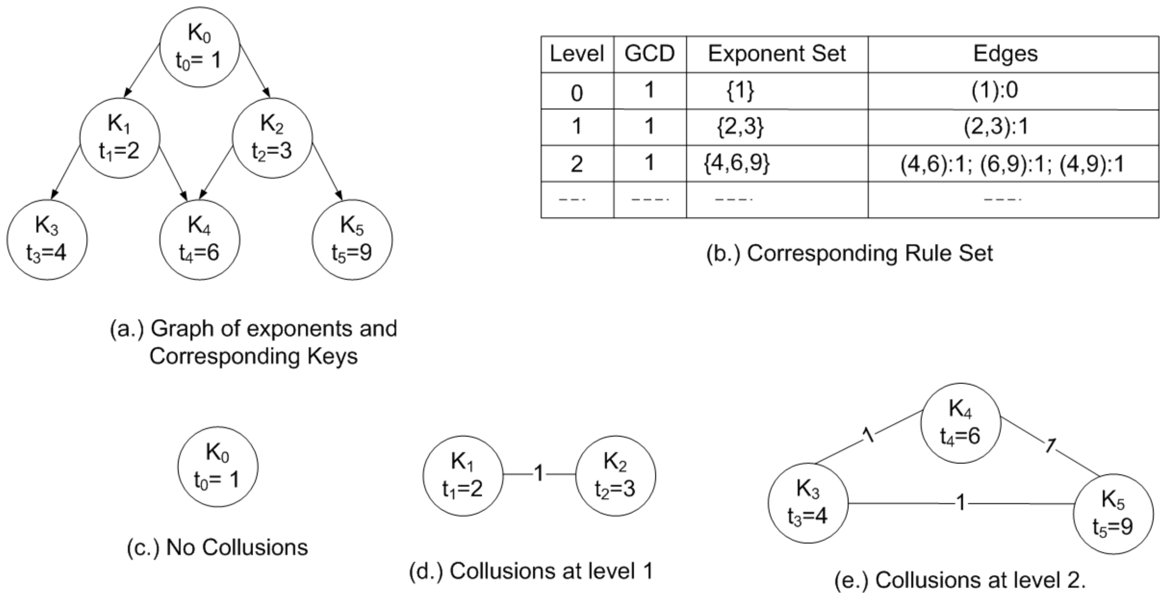
```
Require: $h \geq 1$; $nd \geq 1$   /*h:depth of hierarchy, nd: number of paths linking a
    security class to those directly below it*/
Ensure: $GCD[0] \vee ... \vee GCD[h-1] \nleq 0$   /*GCD[x]: greatest common divisor of
    exponents at level x*/
 1: $k \leftarrow 0$
 2: $GCD[0] \leftarrow 1$
 3: for $i = 1$ to $nd^k - 1$ do
 4:   $GCD[0] = \gcd(GCD[0], e_{k,i})$  /*Compute GCD at level 0*/
 5: end for
 6: for $k = 1$ to $h - 1$ do
 7:   $GCD[k] \leftarrow e_{k,0}$  /*Start collusion detection process*/
 8:   for $i = 1$ to $nd^k - 1$ do
 9:     $GCD[k] = \gcd(GCD[k], e_{k,i})$
10:   end for
11:   $index = k$
12:   for $k = index$ downto 1 do
13:     if $(GCD[k] = (GCD[k-1] \vee ... \vee GCD[0]))$ then
14:       for $i = 1$ to $nd^k - 1$ do
15:         if $(\gcd(e_{k,i-1}, e_{k,i}) = (e_{0,0} \vee ... \vee e_{k-1,i}))$ then
16:           $Edge[e_{k,i-1}, e_{k,i}] = 1$
17:         else
18:           $Edge[e_{k,i-1}, e_{k,i}] = 0$
19:         end if
20:       end for
21:     end if
22:   end for
23: end for
```

If either or both conditions are verified, the edge between the classes is labeled with a value of 1, otherwise it is labeled with a value of 0. When no collusion liable exponents are found in the exponent set, the CDM transmits the exponent set to the KGM where they are used to generate keys. The keys are computed using equation (3.1) (i.e. $K_i = K_0^{t_i} \bmod M$). Otherwise the collusion resolution module (CRM) is called to rectify the situation by re-computing a new exponent set that is collusion-free to break the potential connectivity between the keys. Alternatively, when the collusion-free exponent set is obtained, the exponent set is transmitted to the KGM for key generation. The algorithm is summarized in the pseudo-code give in Algorithm 3.

6.2.3 Example of Collusion Detection

Consider for example the case shown in Figure 6.4(a.) where a hierarchy is formed on the basis of the exponent set $V = \{1, 2, 3, 4, 6, 9\}$. From the directed graph in Figure 6.4(a.), we can infer that the rule set will be as shown in Figure 6.4(b.). In this case the greatest common divisor (GCD) at level 0 is 1, level 1, 1 and level 2, 1 (see Algorithm 3, lines 3-10). Since the GCDs at all three levels are the same, it implies that there are collusions at levels 1 and 2 (see Algorithm 3, lines 13 and 14). The collusion detection algorithm by computing the GCD of all pairwise exponents to form the graphs shown in Figures 6.4(c.), 6.4(d), and 6.4(e) (see Algorithm 3, line 15).

We note that in each of the cases collusion is possible either because the combined GCD at a given level yields a value that is a divisor of some or all of the exponents at the higher levels (here levels 0 and/or 1) or is a divisor of the combined GCD of the exponents at levels 0 and/or 1. For instance, the edge between K_3 and K_4 is labeled with a 1 because the $GCD\{4, 6\} = 2$ and since dividing $t_1 = 2$ by this value yields an integer, we can deduce that $t_3 = 4$ and $t_4 = 6$ can be combined in some form to yield a collusion-liable key. The same is true for the combinations:K_1, K_2; K_4, K_5;K_3, K_5;


K_0 $t_0= 1$
K_1 $t_1=2$
K_2 $t_2=3$
K_3 $t_3=4$
K_4 $t_4=6$
K_5 $t_5=9$
(a.) Graph of exponents and Corresponding Keys
(b.) Corresponding Rule Set
(c.) No Collusions
(d.) Collusions at level 1
(e.) Collusions at level 2.
1

Level	GCD	Exponent Set	Edges
0	1	{1}	(1):0
1	1	{2,3}	(2,3):1
2	1	{4,6,9}	(4,6):1; (6,9):1; (4,9):1
---	---	---	---

Figure 6.4: An Illustration of Collusion Detection in a Hierarchy

and/or K_3, K_4, K_5.

6.2.4 Collusion Resolution Algorithm

Menezes et al. [71] showed that any hierarchical cryptographic access control protocol in which the central authority (CA) selects a single RSA [11] modulus M as the basis of key generation, inevitably allows for the derivation of illegal keys in the system. Hence there is always a possibility that, no matter how well a scheme is defined, if the keys generated are not tested for the possibility of their being combined to generate "illegal keys", some may exist therein that can be used to cause "collusion attacks".

When the graphs of collusion liabilities have been computed the collusion resolution algorithm proceeds to replace the collusion liable keys with ones that break the connectivity in such a way as to form an independent set. This problem is similar to that of determining a maximal independent set (MIS) and since the MIS problem is an NP-hard problem, we use an approximation heuristic to obtain a solution in polynomial time.

Algorithm 4 : Collusion Resolution ($GCD[], h, nd$)

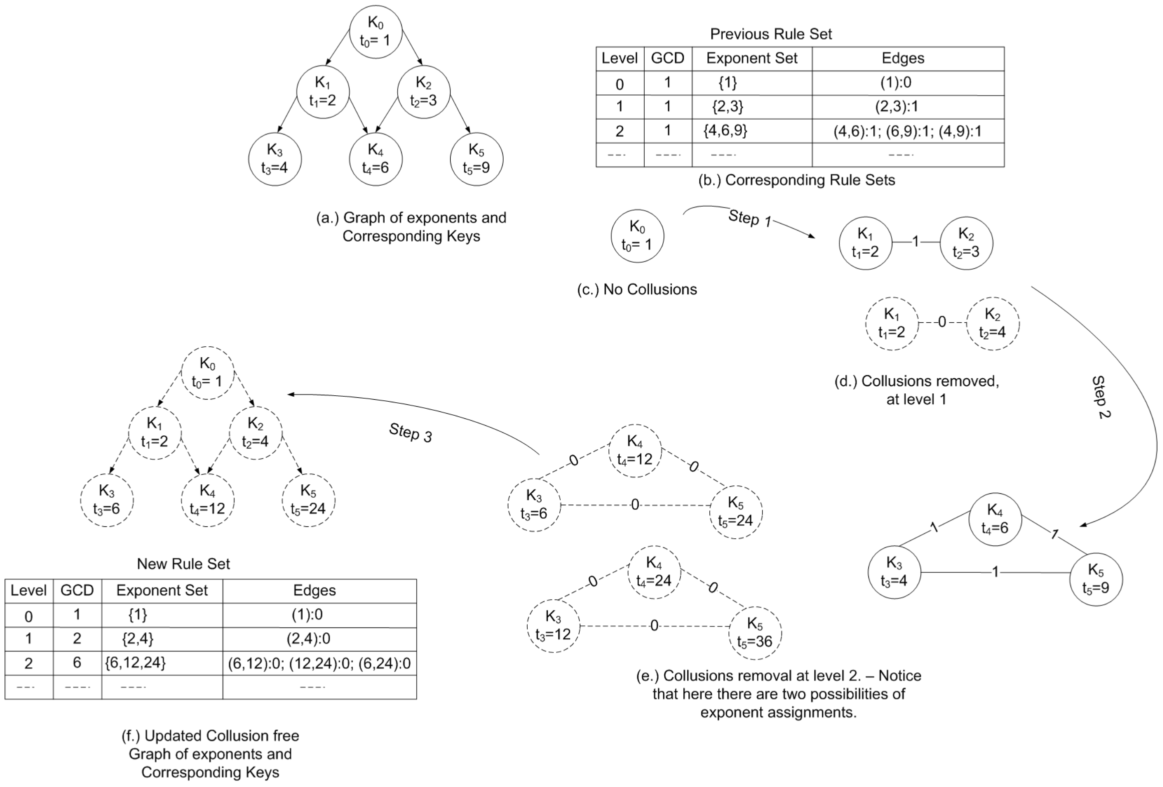
```
Require: $h \geq 1$; $nd \geq 1$   /*h:depth of hierarchy, nd: number of paths linking a
    security class to those directly below it*/
Ensure: $GCD[0] \vee ... \vee GCD[h-1] \not\leq 0$   /*$GCD[x]$: greatest common divisor of
    exponents at level x*/; $R$ : Exponent Registry;
 1: for $k = 1$ to $h - 1$ do
 2:   if $(GCD[k] = (GCD[k-1] \vee ... \vee GCD[0]))$ then
 3:     for $i = 1$ to $d^k - 1$ do
 4:       if $((i \bmod nd) = 0)$ then
 5:         $j \leftarrow j + 1$ /*Selecting new exponent. Shift to next parent node*/
 6:       end if; $e_{k,i} \leftarrow e_{k,0} \times (i + 1)$
 7:       $temp \leftarrow i$ /*Bounds growth of $e_{k,i}$ linearly.*/
 8:       while $((e_{k,i}{=}e_{k,i-1}) \vee ((e_{k,i} \bmod e_{k-1,j}) \neq 0))$ do
 9:         $e_{k,i} \leftarrow e_{k,0} \times temp$ /*new exponent selection*/
10:         $temp \leftarrow temp + 1$
11:       end while
12:       if $e_{k,i} \in R$ then
13:         GOTO line 4
14:       else
15:         $R \leftarrow e_{k,i}$
16:         GOTO line 1
17:       end if
18:     end for
19:   end if
20: end for
```

In order to remove collusion liable keys, the collusion resolution algorithm proceeds to correct the assignment (that is to remove the possibility of collusion), by selecting a random class (usually the class with the least number of edges that are labeled with a 1) and assigning a new random value to it. In a scheme like the Ad-hoc AT scheme where collusion liable keys result from the exponent assignment used to generate the keys, the heuristic used by the collusion resolution algorithm to achieve this search in polynomial time is to assign exponent values in a way that ensures that the greatest common divisor (GCD) at level l is different from those at levels 0 to $l-1$. Additionally, the assigned exponents must continue to be multiples of the exponents assigned to the security classes that are superior to it and that are authorized to access data at level l. Once a completely collusion-free set is found for level l, the procedure is repeated for all of the other levels in the hierarchy. On average, this algorithm converges after $O(\log |V|)$ iterations for each level in the hierarchy, where V is the number of security classes at level l in the hierarchy. The pseudo-code given in Algorithm 4 summarizes the operation of the collusion resolution algorithm.

6.2.5 Example of Collusion Resolution

The following example is aimed at clarifying how the collusion resolution algorithm works. Consider for example the hierarchy depicted in Figure 6.5(a.) where $V = \{1, 2, 3, 4, 6, 9\}$. In the first step, the algorithm is executed for level 0 and since there is nothing higher than t_0, t_0 retains the value of 1. At level 1, as shown in Figure 6.5(b.), the $GCD\{2, 3\} = 1$ which is equal to t_0, this indicates that there is a possibility that the keys K_1 and K_2 that are formed from $t_1 = 2$ and $t_2 = 3$ can be used to provoke a collusion attack. In order to prevent this occurrence, the algorithm then proceeds to select a random value for t_2 such that $GCD\{t_1, t_2\} \neq 1$ and both t_1 and t_2 remain multiples of t_0 (see Algorithm 4, lines 2 and 7). A pseudo-randomly chosen value of 4 is selected and since $GCD\{2, 4\} = 2$ which is not a divisor of the GCD at level 0, and both are multiples of t_0, t_2 retains the randomly assigned value of 4 (see Algorithm 4, lines 8-11).

Likewise at level 2, as shown in Figure 6.5(c), first $t_3 = 4$ which has been assigned to t_2 , and $GCD\{4, 9\} = 1 = t_0$, $GCD\{4, 6\} = 2 = t_1$ and $t_5 = 9$ is not a multiple of t_2. Hence, the algorithm needs to re-assign integer values to t_3, t_4, and t_5 such that GCDs of the pairs of t_i at level 2 are not a factor of any GCDs at levels 0 and 1 and that additionally, the divisibility condition continues to hold. The random assignments in Figure 6.5(c.) present two possibilities of assignments, $t_3 = 12, t_4 = 24, t_5 = 36$ and $t_3 = 6, t_4 = 12, t_5 = 24$ (see Algorithm 4, lines 12-14). So one of the sets is chosen and finally in Figure 6.5(d.) the new assignment of exponents is such that collusion is prevented.


K0 t0= 1
K1 t1=2
K2 t2=3
K3 t3=4
K4 t4=6
K5 t5=9
(a.) Graph of exponents and Corresponding Keys
(b.) Corresponding Rule Sets
K0 t0= 1
Step 1
K1 t1=2 —1— K2 t2=3
(c.) No Collusions
K1 t1=2 --0-- K2 t2=4
(d.) Collusions removed, at level 1
Step 2
K4 t4=6
K3 t3=4
K5 t5=9
1
Step 3
K4 t4=12
K3 t3=6
K5 t5=24
0
K4 t4=24
K3 t3=12
K5 t5=36
(e.) Collusions removal at level 2. – Notice that here there are two possibilities of exponent assignments.
K0 t0= 1
K1 t1=2
K2 t2=4
K3 t3=6
K4 t4=12
K5 t5=24
(f.) Updated Collusion free Graph of exponents and Corresponding Keys

Previous Rule Set

Level	GCD	Exponent Set	Edges
0	1	{1}	(1):0
1	1	{2,3}	(2,3):1
2	1	{4,6,9}	(4,6):1; (6,9):1; (4,9):1
---	---	---	---

New Rule Set

Level	GCD	Exponent Set	Edges
0	1	{1}	(1):0
1	2	{2,4}	(2,4):0
2	6	{6,12,24}	(6,12):0; (12,24):0; (6,24):0
---	---	---	---

Figure 6.5: Example of a Applying the Collusion Detection Algorithm to Remove Collusions

It is obvious from this illustration that several different combinations would generate correct and valid independent sets. We could use a heuristic to control the size of the GCD of the exponent pairs at all of the levels in the hierarchy. However, we do not consider having different exponent sets to be a disadvantage but rather an advantage because different sets could be generated off line and attributed when there is a demand for a new set of keys. This would be the case when a user joins or leaves the system and the original structure is maintained. In this way the method can in the best case contribute to an improvement in the efficiency of the key generation scheme [66].

6.3 Experimental Setup and Results

This section describes the experimental setup we used to run experiments to evaluate our collusion detection and resolution algorithms. We used our collusion detection and resolution algorithms to support the Ad-hoc AT scheme and compared its performance to the CAT scheme. In the Ad-hoc scheme, the exponents are randomly assigned, whereas in the CAT scheme each exponent is computed from the the product of the primes that are assigned to all the classes that do not belong to the class under consideration or those in its sub-poset.

6.3.1 Implementation and Experimental Setup

The experiments were conducted on an IBM Pentium IV computer with an Intel 2.66GHz processor and 512MB of RAM and a 32MB file per security class in the hierarchy. We implemented both schemes on a Microsoft Windows XP platform using the Java 2 Standard Development Kit and Eclipse [87, 16]. In our implementation, the key generation function generates Triple DES (Data Encryption Standard)[97] encryption/decryption keys and the data encryption module encrypts the files.

In the Ad-hoc AT scheme, each key set contained an average of 8.68% of collusion-liable keys. The hierarchies were comprised of 7 to 111 classes and implemented in the form of a tree graph with an average of 2-4 successor classes per class. (Diagrams of the hierarchies used are given in Appendix B.) Our evaluations are based on four criteria namely, cost of collusion detection, cost of collusion resolution, cost of key generation, and the combined cost of key generation and data encryption. We explain these criteria in a little more detain below:

- *Cost of Collusion Detection:* This is the time it takes the algorithm to identify, based on known collusion functions, the key combinations of keys that can be used to provoke collusion attacks.

- *Cost of Collusion Resolution:* This is the time it takes the algorithm to resolve the collusions by randomly selecting a new key and testing it.

- *Cost of Key Generation:* This is the cost of creating keys. In the Ad-hoc AT scheme, this includes the cost of collusion detection and resolution.

- *Cost of Key Generation and Data Encryption:* This is the cost of creating the keys and encrypting all of the files associated with each one of the security classes.

6.3.2 Cost of Collusion Detection

In the first experiment, we evaluate the cost of detecting collusions with respect to the size of the hierarchy. We used hierarchies in which the exponents used to form the keys were randomly generated (e.g. in the Ad-hoc AT scheme) with an average of 8.68% of the exponents being collusion liable. The experiment was run 10 times for each hierarchy of size 7 to 111, and the results averaged and plotted in Figure 6.6. The error bound for each point plotted is ±3 seconds.

We observed that the time taken to detect collusions increased with the size of the hierarchy which is reasonable since the detection algorithm needs to make a greater number of comparisons to establish collusion liabilities in a larger hierarchy than in a smaller one. However, we note that the time it takes to detect collusions is proportionate to the size of the hierarchy. For example, it takes an average of 112.6 seconds to identify approximately 14 collusion cases in a hierarchy with 156 classes, and approximately 1 second to detect 1 collusion case in a 7 class hierarchy. This

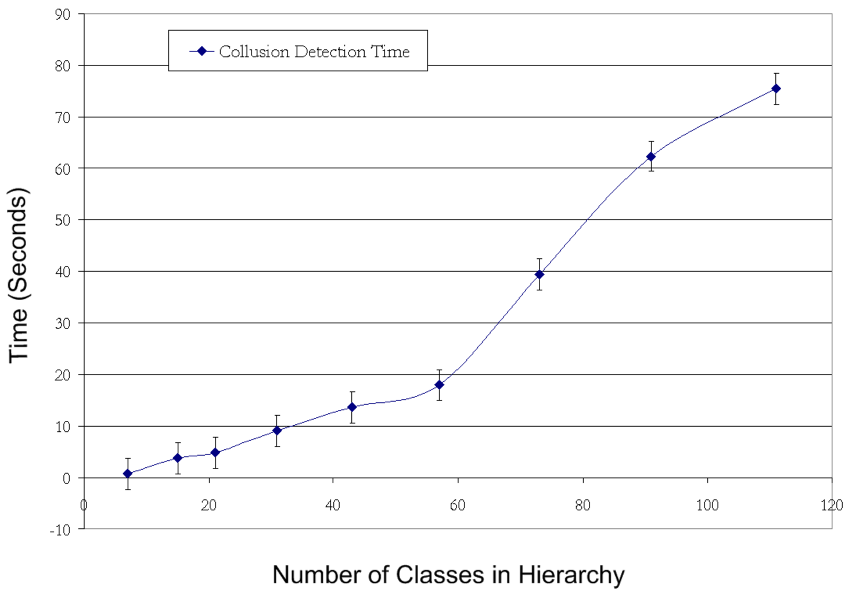


Figure 6.6: Cost of Collusion Detection

time cost can be evaluated as being reasonable when we consider that the detection algorithm needs to perform an average of $h(V!)$, where h is the number of levels in the hierarchy and V the number of security classes at each level in the hierarchy, comparisons to detect all the collusions in the hierarchy.

6.3.3 Cost of Collusion Resolution

Our second experiment evaluates the cost of correcting collusions in order to establish a collusion-free key set. In this case, we also used hierarchies of size 7 to 111 and ran the experiment 10 times for each case. The results obtained were averaged and plotted in Figure 6.7 and the error bound for each point plotted is ± 10 seconds.

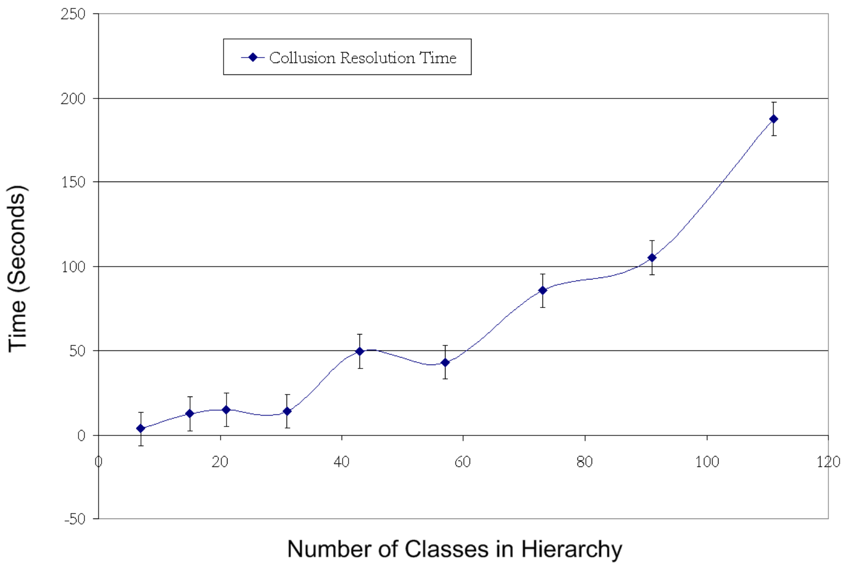


Figure 6.7: Cost of Collusion Resolution

We observed that the cost of collusion resolution stays fairly constant in the smaller hierarchies and begins to grow geometrically with an increasing hierarchy size. This is as a result of the fact that our collusion resolution algorithm uses a principle that is similar to the one we use in the SPR scheme. Therefore, collusion resolution in smaller hierarchies requires fewer checks and exponent adjustments whereas in larger hierarchies more verifications are required and consequently replacements. We note however, that time taken to correct the remains proportionate to the number of collusions that need to be corrected. For example, it takes 0.32 seconds to correct 1 collusion occurrence in a 15 class hierarchy while it take 7.04 seconds to correct an average of 14 collusion occurrences in a 111 class hierarchy.

6.3.4 Cost of Key Generation

In our third experiment we support the Ad-hoc AT scheme that is vulnerable to collusion with our collusion resolution scheme and compare the cost of key generation to that in the "Collusion-Free" AT scheme. The algorithm was run 10 times for each

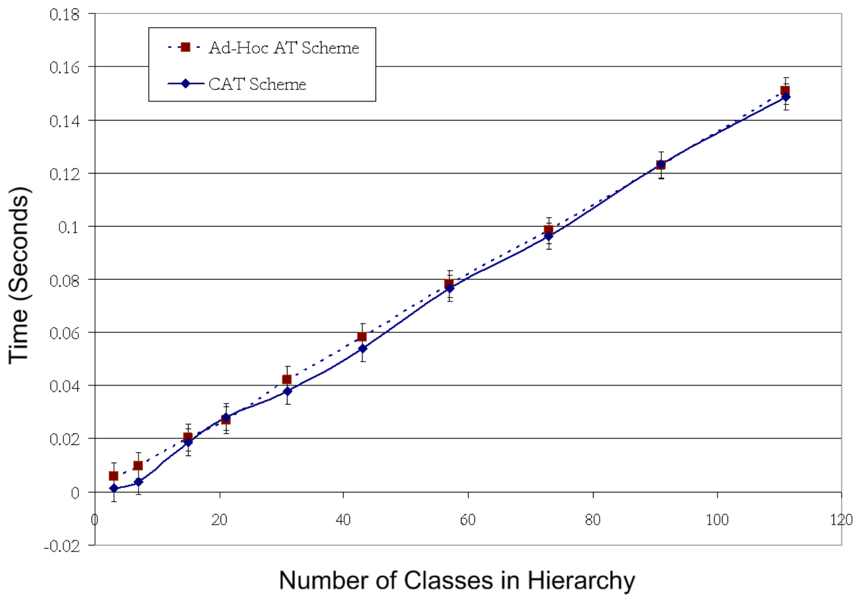


Figure 6.8: Comparative Cost of Key Generation

hierarchy is made up of 7 to 111 security classes and the results averaged to obtain the plots in Figure 6.8. The error bound for each data point in both plots is ±0.005

seconds.

In this case our initial observation is that the simple Akl and Taylor scheme performs worse than the Ad-hoc AT scheme. However, a closer look shows that the time difference between each of the points is not very significant ($\approx 0.45\%$) on average. Moreover, considering that the Ad-hoc AT scheme has the added overhead of checking for collusions after the initial exponent assignment, this performance is actually not as bad as it appears initially.

6.3.5 Cost of Key Generation and Data Encryption

Our fourth experiment evaluates the combined cost of key generation and data encryption. Again in this case the experiment is run 10 times to obtain each data point and the results for each one averaged to obtain the plot in Figure 6.9. The size of the

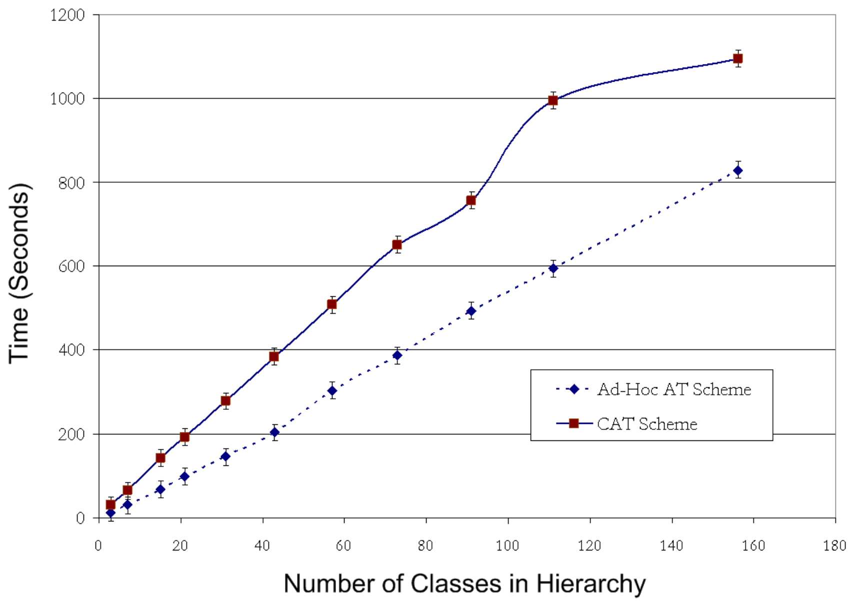


Figure 6.9: Overall Comparative Cost of Key Generation and Data Encryption

file that we used is approximately 32MB and the error bound for each of the points plotted is ± 20 seconds.

In this case, we noted as before, that the size of the key plays a big role in the cost of data encryption. What the Ad-hoc AT scheme loses in terms of performance during key generation, it gains back during data encryption. Moreover, as the plots in the graph show, the gains are even more significant for larger hierarchies.

6.4 Discussions

In this chapter, we have presented a solution to the problem of detecting and removing collusion liable keys from a key set generated using a CKM scheme. We also showed how both algorithms fit into the SPCKM framework, that we presented in Chapter 5, to support our goal of designing adaptive security schemes that are self-protecting of their environments. The approach we adopted for identifying collusion liable keys was to map the keys on to a graph where the vertices represent the keys and the edges, the probability of their being combined to provoke collusions. Adjacent keys signify a higher attack likelihood than non-adjacent keys. In order to remove the collusion liable keys we use a collusion removal algorithm that is based on the principle of computing an independent set from the vertices in a graph.

Using the independent set approach to resolve this problem shows that the problem of removing the collusion liabilities in a key set is similar to the classic graph theory problem of computing a largest independent set. As the problem is NP-hard, a heuristic was used to achieve an efficient (but perhaps suboptimal) solution in polynomial time. Nevertheless, as illustrated, the solution is feasible.

The drawback of this scheme is that it adds computational overhead on the system. The algorithm requires $O(\log |V|)$, where V is the number of vertices (exponents), time at least, for the key selection process at each level in the hierarchy. Additionally, the algorithm needs $O(n \log n)$ time to resolve the collusions.

Chapter 7

Summary and Conclusions

"Adapt yourself to the things among which your lot has been cast and love sincerely the fellow creatures with whom destiny has ordained that you shall live."

– ***Marcus Aurelius***

Throughout this thesis, we have proposed a number of approaches to extending the cryptographic access control schemes to incorporate adaptability. Specifically, we focused on cryptographic key management (CKM) in a hierarchy, where access is controlled by assigning each security class (group) a single key that allows the users to directly access information or derive the key required when the conditions of access are satisfied. We noted that while these CKM schemes afford better security than CKM schemes in which the keys are independent of each other, key management is expensive because updates result in costly data re-encryptions and changes throughout the entire hierarchy because of the inter-dependencies between the keys.

We argued that security schemes face difficulties in matching security with performance because the focus in designing security schemes has generally been on correctness as opposed to performance. In fact, designers of security schemes, unlike other areas of computing, tend to assume that if security schemes are properly designed, failure (security violations or inefficiency) is unlikely. The first two solutions we propose show that while performance enhancements can be gained by using a number of heuristics to tweak a security scheme, cases that involve more dynamic scenarios result in delays that not only impede performance, but also in the long-run, create security vulnerabilities on the system. We suggested resolving this problem with an adaptive security scheme and proposed a framework for self-protecting key management that is based on the paradigm of autonomic computing. By its property of adaptivity, our framework evolved a quasi-static access control scheme into one that has the ability to be self-protecting. We defined self-protecting access control as the ability of a software system to ensure continued security by adjusting its access control parameters in response to perceived changes in its environment. Finally, since self-protecting security implies a lesser degree of control by a human security administrator, we considered the problem of attacks that can be perpetrated by users in possession of valid keys. Specifically, we focused on the problem of "collusion attacks" and proposed an adaptive algorithm for resolving collusions. The algorithm monitors key assignments and checks to ensure that collusion attacks are prevented. In the following we first summarize the contributions of this thesis and then proceed to evoke ways in which this work can be extended.

7.1 Summary and Critique

In Chapter 1 we presented the hypothesis that underlies our arguments in favor of adaptive security. Essentially, the hypothesis is that security schemes can be extended to incorporate adaptivity without necessarily changing their underlying specifications. We explained why we opted to base our solution on the dependent key management model for cryptographic access control, by the fact that it offers better security and more efficient key management than its independent key management counterpart. In dependent key management schemes that are modeled using a partially ordered set (poset) of disjoint security classes, access is controlled by classifying every user into exactly one of a number of disjoint security classes that are partially ordered. In the partial order hierarchy, cryptographic keys that are associated with lower level classes are mathematically derivable from higher level keys, but not the reverse. In other words, access to data is granted if a user holds the "correct" key, or can derive the key from the one in their possession. (This is the case when a user belongs to a class that is higher up in the hierarchy). The drawback inherent in this approach to cryptographic access control in a hierarchy is that changes in group membership imply not only a group key update but additionally, trigger changes throughout the entire hierarchy due to the inter-dependencies between these keys. We then highlighted three scenarios centered on the idea of using cryptographic key management to support access control in collaborative web applications where access to shared data is an issue and where the cost of key replacement poses an impediment to efficient access control.

Chapter 2 reviews the literature both on cryptographic access control in particular and other access control models in general. In each one we noted that while security is a primary concern, performance is always secondary which is probably reasonable because the goal of a security scheme is first and foremost to protect. Adaptivity, however, is essential to good security in scenarios that change frequently but researchers have only recently begun to address this issue when designing security schemes. Our discussion begins with mandatory and discretionary access control models that are probably the oldest known access control models in the area of distributed systems security. Next, we discuss the different models of cryptographic key management for access control highlighting their pros and cons. Other newer access control paradigms include role-based access control, code-based access control, cookies, XML access control, anti-viruses, intrusion detection and firewalls. Finally, we conclude with a discussion on autonomic access control which is still at a budding stage.

Chapter 3 presents a heuristic algorithm (sub-poset scheme) for minimizing the cost of key assignment and replacement. Basically, the algorithm uses an integer factorization metric and a distance-based heuristic to assign exponents to the security classes in the hierarchy in a way that minimizes the cost of key generation and encryption and ensures that the size of the exponents does not grow geometrically. Since this algorithm extends the one that Akl and Taylor proposed to control access in a hierarchy [2], we compared the performance of our heuristic algorithm to both the CAT scheme and the MCK scheme that all have the same root (i.e. designed according to the principle of key management that Akl and Taylor proposed). The complexity analysis and experimental results showed that the heuristic algorithm performs better than the previous two schemes. From this we concluded that the size of the key is a determining factor in the time it takes to encrypt data. So, it is desirable to have smaller keys not just for storage purposes (cases of small devices that have limited storage capacity), but also in terms of reducing encryption time.

Since the heuristics scheme still faces the drawback of requiring that the whole hierarchy be rekeyed when the replacement event involves the highest security class in the hierarchy (this is the case in the previous schemes), we proposed using timestamps to limit the necessary changes. In essence, this scheme that we call the timestamp scheme, operates by associating a timestamp to every key in the hierarchy and computing a verification signature for each timestamp and key pair. Access to data is verified by checking that the timestamp and key pair in a user's possession yields a correct verification signature. Rekeying is handled by replacing the timestamp associated with a security class and computing a new verification signature for the security class. So, in this case the window of vulnerability created during key replacement is significantly reduced in comparison to the SPR, CAT, and MCK schemes.

However, although these first two schemes make for better performance than the CAT and MCK schemes, they do not cope well with high demand situations. In Chapter 4, we proposed a framework to handle these scenarios effectively by anticipating user requests and adjusting resource allocation accordingly. In order to model this adaptive key management approach, we used the feedback control loop drawn from the paradigm of autonomic computing. Finally, we discussed the benefits of adaptive rekeying showing that it provides better performance than a basic KM scheme and significantly reduces the size of the window of vulnerability.

Since adaptive key management implies a lesser degree of physical intervention by a human security administrator, we consider some of the cases of violations that might be perpetrated by authentic users. Specifically, we focused on the case of "collusion attack". We designed an adaptive collusion resolution algorithm that uses the concept of deriving an independent set from a key graph to eliminate keys that are liable to be used to provoke such attacks. We analyzed the performance of the collusion resolution algorithm using the Akl and Taylor scheme that is vulnerable to collusion attack and compare its performance to that of the Akl and Taylor scheme that is secure against collusions. The results indicate that while the collusion resolution algorithm takes longer on average to generate a set of collusion-free keys, the combined cost of data encryption and key generation is lower than in the CAT scheme. This indicates that the cost of data encryption (recall, that we are using Triple DES keys whereas AES keys allow for faster encryptions) is proportional to the key size when the key is obtained by exponentiation. Therefore schemes based on an exponentiation function benefit in terms of overall performance if they avoid raising keys to high exponents.

We conclude Chapters 5 and 6 with a critique of the adaptive security solutions we presented. Essentially, we noted that the decision version of the problem of adaptive key replacement in hierarchies is NP-complete and showed that it is reducible to the traveling salesman problem. Likewise, the decision version of the problem of detecting and removing all the possible key combinations, within a key set, that can be exploited to provoke collusion attacks is NP-complete. We showed that the collusion detection and resolution problem is reducible to the independent set problem. This observation is emphasized by the fact that our collusion resolution and detection algorithm, does not specifically consider the case of multiple party collusions provoked by key combinations from different levels in the hierarchy. Lastly, we discussed the benefits of adaptive security with a probabilistic analysis of both schemes.

7.2 Potential Extensions

In this section we discuss the potential of extensions of this work. Some of the problems we look at stem from the shortcomings of our work while others are problems in the area of key management that are not directly related to this work but are worth mentioning for completeness.

7.2.1 Internal Violations

In spite of the protection access control protocols provide against illegal access to sensitive information, the issue of indirect access remains a serious problem. The problem of inference channels in multilevel secure databases was first considered in the 1980s [18, 33]. Basically, inference occurs when users are able to piece together information through legal access channels, in order to deduce information at another security level.

Research has aimed to address the problem through formal methods which focus on minimal classification updates, partial disclosure and classification of data repositories to prevent disclosure via knowledge discovery [78, 90, 98]. The general approach is to try to control the query types accepted, or limit the data provided in response to a query. However, this solution affects system availability whereas the proportion of users who seek to use indirect queries to gain unauthorized access to data, generally comprise a very small section of this group. This security threat will be further aggravated in the autonomic database environment where inferences based on autonomic element behavior, usage monitoring and the application of "trick questions" may occur.

Covert channels represent another manifestation of indirect violations of security policies in the context of hierarchical access control [83]. In multilevel secure databases, a covert channel refers to a transfer of information, from one level in the

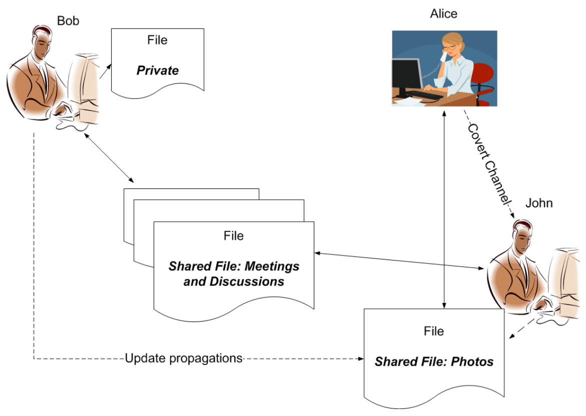


Figure 7.1: Indirect information access via inference channels

hierarchy to another, which violates the partial order between the security classes. This occurs when, for example, a higher level user employs their legal key to access information at their level and then deposits this information in a memory or storage location that can be accessed by the user with the lower security clearance. Solutions presented in the literature are many and varied. Keefe et *al.* [55] present a formal framework for secure concurrency control in multilevel databases. In [60] Lamport presents a solution to the problem of secure read/write. Although these solutions have been shown to be secure, they do not allow for serializable schedules for transactions and they also suffer from the problem of starvation where lower priority transactions (that get preempted by higher priority transactions) may be delayed indefinitely [3]. The solution proposed in [9] presents algorithms which generate serializable schedules but both suffer from starvation.

Figure 7.1 depicts a scenario based on a collaborative Web application (e.g., Facebook), where indirect access to information is achieved via inference channels and covert channels. In this case a user, say Alice, has been included in the "friends" profile of several users. Imagine for instance that she seeks to obtain information on another user, say Bob, whose "friends" profile she does not belong to. She, however is on John's "friends" profile and John in turn is on Bob's "friends" profile, so when Bob propagates updates to his profile to John, Alice can, from reading information on John's public space, infer information related to Bob. Furthermore with a little cooperation from John, a covert channel can be opened between herself and John that allows her to directly receive all the updates that Bob propagates. These weaknesses are not easy to handle with standard cryptographic key management schemes and also usually occur because of weakness in functional, multivalued and join dependencies in the databases that support these systems [18, 49, 78]. The detection and removal of these channels are vital steps in the provision of secure shared data environments

7.2.2 Adaptive Rekeying

Further implementation and experimentation is needed throughout the entire hierarchy with the aim of evaluating the performance of the adaptive scheme against a quasi-static scheme. Other challenges that need to be addressed include finding other statistical distributions that are more effective in modeling rekey arrival rates than the Poisson model and finding a good way to define adequate monitoring thresholds. A good prediction algorithm for handling future arrival rates of requests is also needed. An example of how this might be done would be to compute a moving average as opposed to using the maximum arrival rate. Issues of copy consistency can also arise in situations in which there is a high volume of communications between users (frequent updates on the primary copy) and rekey requests occur within very short intervals of each other.

Additionally, determining an optimal complexity bound for handling out-of-sync data continues to remain an issue. The out-of-sync problem occurs when a user in possession of a new key attempts to decrypt and update a file encrypted with a key that is no longer valid [62].

7.2.3 Key Selection

In the algorithms we presented to detect and remove "collusion-liable" keys from a key set, we noted that the algorithms depend on known collusion detection functions. Therefore, if an attacker were to come up with a function that is not known a priori, the system has no way of detecting and/or preventing the attack. Finding ways of determining collusion functions that are not known a priori remains a challenge. One way of approaching the problem would be to try to find correlations that can be used to provoke attacks and define some sort of language to model combinations that could result in collusions. Additionally, we need to find ways of optimizing the collusion detection and resolution process for better performance in larger hierarchies than the ones we considered in our experiments.

References

[1] M. Adams. A parallel maximal independent set algorithm. *In Proceedings. 5th Copper Mountain Conf. on Iterative Methods*, 1998.

[2] S.G. Akl and P.D. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Transactions on Computer Systems*, 1(3):239–248, August 1983.

[3] P. Ammann and S. Jajodia. A timestamp ordering algorithm for secure, single-version, multilevel databases. *Database Security, V: Status and Prospects,C.E. landweher, ed., Amsterdam, Holland*, 1992.

[4] M.J. Atallah, M. Blanton, and K.B. Frikken. Key management for non-tree hierarchies. *In Proceedings, ACM Symposium on Access Control Models and Technologies, Lake Tahoe, California, USA*, pages 11–18, 2006.

[5] M.J. Atallah, K.B. Frikken, and M. Blanton. Dynamic and efficient key management for access hierarchies. *In Proceedings, ACM Conference on Computer and Communications Security*, pages 190–202, 2005.

[6] G. Ateniese, A. De Santis, A.L. Ferrara, and B. Masucci. Provably-secure time-bound hierarchical key assignment schemes. *In Proceedings of 13th ACM*

Conference on Computer and Communications Security (CCS'06), pages 288–297, 2006.

[7] G Ateniese, K Fu, M Green, and S. Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Transactions on Information and System Security (TISSEC)*, 9(1):1–30, February 2006.

[8] D. Bell and L. Lapadula. Secure computer systems: Mathematical foundations and model. *MITRE Report, MTR2547*, page 2, 1973.

[9] E. Bertino, S. Jajodia, L. Mancini, and I Ray. Advanced transaction processing in multilevel secure file stores. *In Proceedings. IEEE Transactions on Knowledge and Data Engineering*, 10(1):120–135, 1998.

[10] E. Bertino and R. Sandhu. Database security - concepts, approaches, and challenges. *IEEE Transactions on Dependable and Secure Computing*, 2(1):2–19, 2005.

[11] K. Biba. Integrity considerations for secure computer systems. *Technical Report ESD-TR-76-372, ESD/AFSC, Hanscom AFB, Bedford, MA, April*, 1977.

[12] J-C. Birget, X. Zou, G. Noubir, and Ramamurthy B. Hierarchy-based access control in distributed environments. *In Proceedings. IEEE International Conference on Communications*, 1:229–233, 2001.

[13] K. Birman, R. van Renesse, and V. Werner. Adding high availability and autonomic behaviour to web services. *Proc. of* 26th *International Conf. on Software Engineering (ICSE'04)*, pages 17–26, 2004.

[14] M. Blaze, G. Bleumer, and M. Strauss. Divertible protocols and atomic and atomic proxy cryptography. *In Proceedings of Eurocrypt '98*, 1403:127–144, 1998.

[15] A. Boukerche, R.B. Machado, K.R.L. Juca, J.B.M. Sobral, and M.S.M.A. Notare. An agent based and biologically inspired real-time intrusion detection and security model for computer network operations. *Computer Communications*, (30):2649–2660, March 2007.

[16] J.T. Bradley, S.T. Gilmore, and J. Hillston. Analysing distributed internet worm attacks using continuous state-space approximation of process algebra models. *Journal of Computer and System Sciences*, March (*Article in Press*) 2007.

[17] D.D.C. Brewer and M.J. Nash. The chinese wall security policy. *IEEE Symposium on Security and Privacy, Oakland*, pages 206–214, May 1988.

[18] A. Brodsky, C. Farkas, and S. Jajodia. Secure databases: Constraints, inference channels and monitoring disclosures. *In Proceedings. IEEE Transactions on Knowledge and Data Engineering*, 12(6):900–919, 2000.

[19] R. Chandramouli. A policy validation framework for enterprise authorization specification. *In Proceedings of the 19th Annual Computer Security Applications Conference, 2003*, pages 319–328, 2003.

[20] D.M. Chess, C.C. Palmer, and S.R. White. Security in an autonomic computing environment. *IBM Systems Journal*, 41(1):107–118, 2003.

[21] H-Y. Chien. Efficient time-bound hierarchical key assignment scheme. *IEEE Transactions on Knowledge and Data Engineering*, 16(10):1301–1304, October 2004.

[22] R. Chow and T.J. Johnson. *Distributed Operating Systems and Algorithms.* Addison Wesley Longman Inc., 1998.

[23] D.R. Clark and D.R. Wilson. A comparison of commercial and miltrary computer security policies. *In Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 184–194, 1987.

[24] T.R. Cormen, C.E. Leiserson, R.I. Rivest, and C. Stein. *Introduction to Algorithms.* McGraw-Hill Book Company, 2001.

[25] J. Crampton. Cryptographically-enforced hierarchical access control with multiple keys. *In Proceedings of 12th Nordic Workshop on Secure IT Systems (NordSec 2007)*, pages 49–60, 2007.

[26] J Crampton, K Martin, and P. Wild. On key assignment for hierarchical access control. *In Proceedings, 19th IEEE Workshop on Computer Security Foundations, S. Servolo Island, Italy*, pages 98–111.

[27] M.L. Das, A. Saxena, V.P. Gulati, and D.B. Phutak. Hierarchical key management scheme using polynomial interpolation. *SIGOPS Oper. Syst. Rev.*, 39(1):40–47, 2005.

[28] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Over-encryption: Management of access control evolution on outsourced data. *In Proceedings, VLDB 2007*, pages 123–134, Spetember 23-28 2007.

[29] A.L. De Santis, A. Ferrara and B. Masucci. New constructions for provably-secure time-bound hierarchical key assignment schemes. *In Proceedings of 12th ACM Symposium on Access Control Models and Technologies (SACMAT'07)*, pages 133–138.

[30] M H DeGroot and M J Schervish. *Probability and Statistics.* Addison Wesley, Third Ed., New York, 2002.

[31] D.E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, 1976.

[32] Facebook. Facebook. *http://www.facebook.com/*, 2007.

[33] C. Farkas and S. Jajodia. The inference problem: A survey. *ACM SIGKDD Explorations Newsletter*, 4:6–11, 2002.

[34] Flickr. Flickr. *http://www.flickr.com/*, 2008.

[35] D. Frincke, A. Wespi, and D. Zamboni. From intrusion detection to self-protection. *Computer Networks*, (51):1233–1238, 2007.

[36] A.G. Ganek and T.A. Corbi. The dawning of the autonomic computing era. *IBM Systems Journal*, 42(1):5–18, 2003.

[37] D. Gollmann. *Computer Security.* John Wiley and Sons, Ltd, 2005.

[38] L. Harn and H. Lin. A cryptographic keys generation scheme for multilevel data security. *Computer Security*, 9:539–546, 1990.

[39] K. Harrison, B. Munro, and T. Spiller. Security through uncertainty. *Elsevier - Network Security*, pages 4–7, February 2007.

[40] R.H. Hassen, A. Bouabaallah, H. Bettahar, and Y. Challal. Key management for content access control in a hierarchy. *Computer Networks*, (51):3197–3219, 2007.

[41] C.L. Hsu and T.S. Wu. Cryptanalyses and improvements of two cryptographic key assignment schemes for dynamic access control in a user hierarchy. *Computers and Security*, 22(5):453–456, 2003.

[42] V.C. Hu, E. Martin, J. Hwang, and T. Xie. Conformance checking of access control policies specified in xacml. *In Proceedings of the 31st Annual International Computer Software and Applications Conference, 2007. COMPSAC 2007 - Vol. 2.*, pages 275–280, 2003.

[43] Q. Huang and C. Shen. A new mls mandatory policy combining secrecy and integrity implemented in highly classified secure level os. *Proc. of* 7^{th} *International Conf. on Signal Processing (ICSP '04)*, 3:2409–2412, 2004.

[44] M.-S. Hwang, C.-C. Chang, and W.-P. Yang. Modified chang-hwang-wu access control scheme. *IEE Electronics Letters*, 29(24):2095–2096, 1993.

[45] P. Jalote. *Fault Tolerance in Distributed Systems.* Pearson Education: Prentice Hall, NJ, 1998.

[46] M-A. Jeong, J-J. Kim, and Y. Won. A flexible database security system using multiple access control policies. *In Proceedings. 4th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'2003)*, pages 236–240, 2003.

[47] Y. Jiang, L. Chuang, Y. Hao, and Z. Tan. Security analysis of mandatory access control model. *In Proceedings. IEEE International Conference on Systems, Man. and Cybernetics*, 6:5013–5018, 2004.

[48] S.E. Johnston, R. Sterritt, E. Hanna, and P. O'Hagan. Reflex autonomicity in an agent-based security system: The autonomic access control system. *4th IEEE International Workshop on Engineering Autonomic and Autonomous Systems (EASe'07)*, pages 68–78, 2007.

[49] N. Jukic, S. Nestorov, and S. Vrbsky. Closing the key loophole in mls databases. *SIGMOD Record*, 32(2):15–20, 2003.

[50] A.V.D.M. Kayem, S.G. Akl, and P. Martin. An independent set approach to solving the collaborative attack problem. *Proc. of* 17th *IASTED International Conf. on Parallel and Distributed Computing and Systems (PDCS 2005), Phoenix, Arizona*, pages 594–599, November 2005.

[51] A.V.D.M Kayem, S.G. Akl, and P. Martin. On replacing cryptographic keys in hierarchical key management systems. *Journal of Computer Security*, 16(3):289–309, 2008.

[52] A.V.D.M. Kayem, P. Martin, and S.G. Akl. Heuristics for improving cryptographic key assignment in a hierarchy. *In Proceedings,* 3rd *IEEE Int'l Symposium on Sceecurity in Networks and Distributed Systems (Niagara Falls, Canada)*, pages 531–536, May 21-23, 2007.

[53] A.V.D.M Kayem, P. Martin, S.G. Akl, and W. Powley. A self-protective key

management framework. *In Proceedings, 4^{th} International Workshop on Engineering Autonomic Software Systems 2007 (EASS 2007), held in conjunction with CASCON 2007, Toronto, ON, Canada), (Position Paper)*, Oct 23-24, 2007.

[54] A.V.D.M Kayem, P. Martin, S.G. Akl, and W. Powley. A framework for self-protecting cryptographic key management. *In Proceedings, 2^{nd} IEEE International Conference on Self-Adaptive and Self-Organizing Systems, Venice, Italy, (To Appear)*, Oct 20-24, 2008.

[55] T.F. Keefe, W.T. Tsai, and J. Srivastava. Multilevel secure database concurrency control. *In Proceedings. 6th IEEE International Conference on Data Engineering*, pages Los Angeles, CA, USA, 1990.

[56] J.O. Kephart. Research challenges of autonomic computing. *In Proceedings. 27th International Conference on Software engineering, St. Louis, MO, USA*, pages 15–22, 2005.

[57] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.

[58] D.E. Knuth. *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms, Second Edition.* Addison Wesley, Reading, MA, 1981.

[59] F. Kuo, V. Shen, T. Chen, and F. Lai. Cryptographic key assignment scheme for dynamic access control in a user hierarchy. *IEE Proceedings - Computers and Digital Techniques*, 146(5):235–240, 1999.

[60] L. Lamport. Concurrent reading and writing. *ACM Communications*, 20(11), 1997.

[61] B.W. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, 1973.

[62] X. Li, Y. Yang, M. Gouda, and S. Lam. Batch rekeying for secure group communications. *WWW10*, 99(7):525–534, January 1999.

[63] T.Y. Lin. Chinese wall security model and conflict analysis. *24th IEEE Computer Society International Computer Software and Applications Conference (COMPSAC 2000), Taipei, Taiwan*, pages 122–127, Oct. 25 - 27 2000.

[64] T.Y. Lin. Managing information flows on discretionary access control models. *2006 IEEE International Conference on Systems, Man, and Cybernetics*, pages 4759–4762, Oct. 8 - 11 2006.

[65] Y. Liu and X. Chen. A new information security model based on blp model and biba model. *In Proceedings. 7th International Conference on Signal Processing (ICSP '04)*, 3:2643–2646, 2004.

[66] S.J. Mackinnon, P.D. Taylor, H. Meijer, and S.G. Akl. An optimal algorithm for assigning cryptographic keys to control access in a hierarchy. *IEEE Transactions on Computers*, c-34(9):797–802, September 1985.

[67] T. Marshall and Dai Y-S. Reliability improvement and models in autonomic computing. *In Proceedings, 11th International Conference on Parallel and Distributed Systems*, 2:468–472, July 20-22 2005.

[68] M. Mat Deris, J.H. Abawaly, and A. Mamat. An efficient replicated data access approach for large-scale distributed systems. *Future Generation Computer Systems*, (In Press.) 2007.

[69] M. Mat Deris, J.H. Abawaly, and A. Mamat. Rwar: A resilient window-consistent asynchronous replication protocol. *In Proceedings,* 2^{nd} *Int'l Conf. on Availability, Reliability and Security*, pages 499–505, 10-13 April 2007.

[70] J. McLean. The specification and modeling of computer security. *IEEE Computer*, 23(1):9–16, January 1990.

[71] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography.* Boca Raton:, CRC Press, 1996.

[72] A. Moreno, D. Sanchez, and D. Isern. Security measures in a medical multi-agent system. *Frontiers in Artificial Intelligence and Applications*, 100:244–255, 2003.

[73] MySpace. Myspace. *http://www.myspace.com/*, 2008.

[74] X-W. Nie, D-G. Feng, J-J. Che, and X-P Wang. Design and implementation of security operating system based on trusted computing. *2006 International Conference on Machine Learning and Cybernetics*, pages 2776–2781, Aug. 2006.

[75] S. Osborn. Integrating role graphs: A tool for security integration. *Data and Knowledge Engineering*, 43:317–333, 2002.

[76] M. Parashar and S. Hariri. Autonomic Computing: An Overview. *In Hot Topics, Lecture Notes in Computer Science, Springer Verlag*, 3566:247–259, 2005.

[77] B. Parducci, H. Lockhart, R. Levinson, and J. Bryce Clark. Representing and evaluating access control policies. 2007.

[78] C.P. Pfleeger and S.L. Pfleeger. *Security in Computing.* 3^{rd} ed., Pearson Education, Prentice Hall NJ, 2003.

[79] National Post. Credit card information stolen from winners, http://www.canada.com/nationalpost/story.html. Jan. 2007.

[80] I. Ray and N. Narasimhamurthi. A cryptographic solution to implement access control in a hierarchy and more. *In Proceedings. 7th ACM Symposium on Access Control Models and Technologies, Monterey, CA*, pages 65–73, 2002.

[81] R. Rivest, A. Shamir, and L. Adleman. A method of obtaining digital signatures and public key cryptosystems. *ACM Communications*, 21(2):120–126, 1978.

[82] W. Rjaibi. An introduction to multilevel secure relational database management systems. *In Proceedings. 24th Conference of the Center for Advanced Studies on Collaborative Research*, pages 232–241, 2004.

[83] W. Rjaibi and P. Bird. A multi-purpose implementation of mandatory access control in relational database management systems. *In Proceedings. 30th VLDB Conference, Toronto, Canada*, pages 1010–1020, 2004.

[84] R. Sandhu. Cryptographic implementation of tree hierarchy for access control. *Information Processing Letters*, 27:1–100, January 1988.

[85] R.S. Sandhu. Lattice-based access control models. *IEEE Computer*, 26(11):9–19, Nov. 1993.

[86] J. Schneider. The time-dependent travelling salesman problem. *Physica A*, (314):151–155, 2002.

[87] E.E. Schultz. Where have the worms and viruses gone? - new trends in malware. *Computer Fraud and Security*, pages 4–8, July 2006.

[88] V. Shen and T. Chen. A novel key management scheme based on discrete logarithms and polynomial interpolations. *Computers and Security*, 21(2):164–171, 2002.

[89] A. Spalka, A.B. Cremers, and L. Hartmut. Protecting confidentiality against trojan horse programs in discretionary access control system. *Lecture Notes In Computer Science, In Proceedings of the 5th Australasian Conference on Information Security and Privacy*, 1841:1–17, 2000.

[90] J. Staddon. Dynamic inference control. *In Proceedings. 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge discovery, San Diego, CA, USA*, pages 94–100, 2003.

[91] R. Sterritt, M. Parashar, and R. Tianfield, H. amd Unland. A concise introduction to autonomic computing. *Advanced Engineering Informatics*, (19):181–187, 2005.

[92] A.S. Tanenbaum and M. Van Steen. *Distributed Systems: Principles and Paradigms.* Prentice Hall, Upper Saddle River, NJ 07458, 2007.

[93] W-G. Tzeng. A time-bound cryptographic key assignment scheme for access control in a hierarchy. *IEEE Transactions on Knowledge and Data Engineering*, 14(1):182–188, 2002.

[94] M. Verma. Xml security: Control information access with xacml. *http://www.ibm.com/developerworks/xml/library/x-xacml/*, October 2004.

[95] S-Y. Wang and C-S. Laih. Merging: An efficient solution for time-bound hierarchical key assignment scheme. *IEEE Transactions on Dependable and Secure Computing*, 3(1):91–100, 2006.

[96] R. West. The psychology of security: Why do good users make bad decisions. *Communications of the ACM*, 51(4):34–41, April 2008.

[97] Wikipedia. Triple des. *http://en.wikipedia.org/wiki/Triple-DES*, June 2008.

[98] D. Woodruff and J. Staddon. Information flow: Private inference control. *In Proceedings. 11th ACM conference on Computer and communications security, Washigton DC, USA*, pages 188–197, 2004.

[99] Dai Y-S. Autonomic computing and reliability improvement. *In Proceedings, 8th IEEE Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, pages 204–206, 2005.

[100] C. Yang and C. Li. Access control in a hierarchy using one-way functions. *Elsevier: Computers and Security*, 23:659–644, 2004.

[101] X. Yi. Security of chien's efficient time-bound hierarchical key assignment scheme. *IEEE Transactions on Knowledge and Data Engineering*, 17(9):1298–1299, September 2005.

[102] X. Yi and Y. Ye. Security of tzeng's time-bound key assignment scheme for access control in a hierarchy. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):1054–1055, July. 2003.

[103] YouTube. Youtube. *http://www.youtube.com/*, 2008.

[104] W. Yu, Y. Sun, and R. Liu. Optimizing the rekeying cost for contributory group key agreement schemes. *IEEE Transactions on Dependable and Secure Computing*, 4(3):228–242, July-Sept. 2007.

[105] Z-L. Zhang, F. Hong, and J-G. Liao. Modeling chinese wall policy using colored petri nets. *In Proceedings of the 6th IEEE International Conference on Computer and Information Technology (CIT'06)*, pages 162–167, Sept. 2006.

[106] S. Zhu, S. Setia, and S. Jajodia. Performance optimizations for group key management schemes. *Proc. of* 23^{rd} *International Conference on Distributed Computing Systems (ICDCS '03)*, pages 163–171, 2003.

[107] X. Zou and B. Ramamurthy. A gcd attack resistant crthacs for secure group communications. *In Proceedings. International Conference on Information Technology: Coding and Computing (ITCC'04)*, 2:153–154, 2004.

Appendix A

Cryptographic Schemes: Comparison Tables

This appendix presents tables summarizing our comparisons and discussions on the CKM schemes that we presented in Chapter 2. Table A.1 summarizes the key management schemes on the basis of the key management model on which they are designed and the technique used by the key generation functions. Table A.2 summarizes the schemes in terms of suitability to hierarchy change and Table A.3 in terms of the different hierarchy types supported by each of the schemes.

Table A.1: A Summary of KM models of presented CKM schemes

Scheme	***Approach***	***Technique***
Ad-Hoc AT [2]	Dependent	Random Integers/Modulus
CAT [2]	Dependent	Primes/Modulus
Mackinnon et al. [66]	Dependent	Optimal assignment of Prime chains / Modulus
Sandhu [84]	Dependent	Chaining (Lower Key obtained from higher key)
Yang and Li [100]	Dependent	Chaining
Shen and Chen [88]	Dependent	polynomial interpolation
Ray et al.	Dependent	asymmetric encryption
Atallah et al.	Dependent	Chaining
Time-bound Schemes	Dependent	Chaining
Crampton et al. [25]	Dependent/Independent	Any
Ateniese at al. [6] and Blaze et al. [14]	Dependent/Independent	Master and Secondary Key Proxy Re-encryption

Table A.2: Suitability to Key Updates

Scheme	***Key Updates***
Ad-Hoc AT [2]	Change One - Change All
CAT [2]	Change One - Change All
Mackinnon et al. [66]	Change One - Change All
Sandhu [84]	Change One - Change All
Yang and Li [100]	Change One - Change All
Shen and Chen [88]	polynomial interpolation
Ray et al. [80]	Change One - Change All
Atallah et al.	Change One - Change one and only access keys
Time-bound Schemes	No change needed
Crampton et al. [25]	Lazy Re-encryption: Any
Ateniese at al. [6] and Blaze et al. [14]	Update Master and Secondary Key

Table A.3: Hierarchy Styles supported by CKM schemes presented and Vulnerability to Collusion attacks. *(**DG: Directed Graph)*

Scheme	***Style***	***Vulnerable to Collusion?***
Ad-Hoc AT [2]	DG	Yes
CAT [2]	DG	No
Mackinnon et al. [66]	DG	No
Sandhu [84]	Tree	No
Yang and Li [100]	Tree	No
Shen and Chen [88]	DG	No
Ray et al. [80]	DG	No
Atallah et al.	DG	No
Crampton et al [25]	Any	No Usually
Time-bound Schemes	No change needed	Yes for most and No for newer schemes
Crampton et al. [25]	Lazy Re-encryption: Any	No
Ateniese at al. [6] and Blaze et al. [14]	Update Master and Secondary Key	No

Appendix B

Hierarchies used in Experiments

This appendix presents diagrams of the hierarchies that were used to conduct the experiments. We observed that in wider hierarchies, key generation was faster than in deeper hierarchies because of the exponent generation algorithm that we use (see Algorithm 1). Data encryption times remain relatively the same irrespective of the structure of the hierarchy.

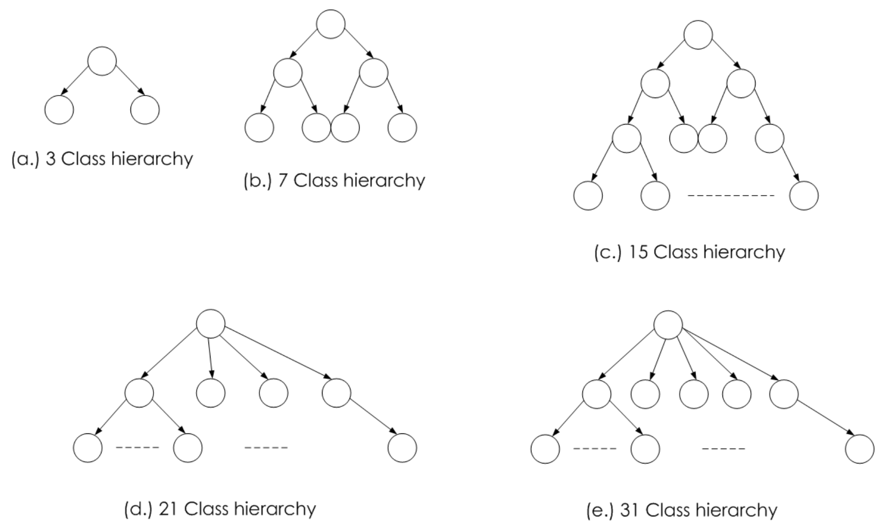
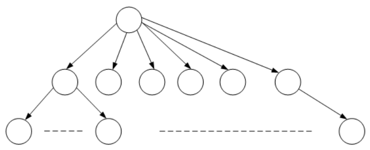
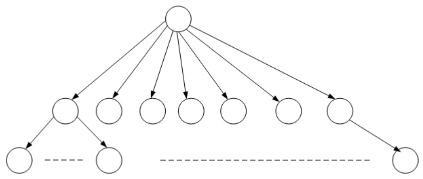
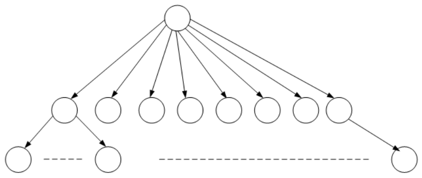
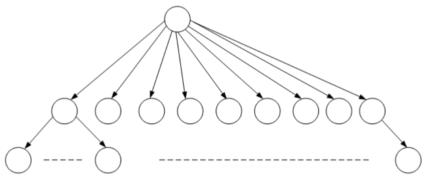


Figure B.1: Hierarchies of size 3 to 31


(f.) 43 Class hierarchy


(g.) 57 Class hierarchy


(h.) 73 Class hierarchy


(h.) 91 Class hierarchy

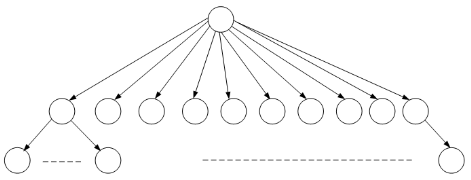

(h.) 111 Class hierarchy

Figure B.2: Hierarchies of size 43 to 111)