Rights violation detection in multi-level digital rights management systems

Amit Sachan

2012

Amit Sachan. (2012). Rights violation detection in multi-level digital rights management systems. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/48097>

<https://doi.org/10.32657/10356/48097>

RIGHTS VIOLATION DETECTION IN MULTI-LEVEL DIGITAL RIGHTS MANAGEMENT SYSTEMS

AMIT SACHAN

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Doctor of Philosophy

2012

Acknowledgments

It is a great pleasure to express my thanks and gratitude to many people who have made my PhD a memorable and valuable experience. First and foremost I wish to express my sincere gratitude to my PhD supervisor Dr. Sabu Emmanuel for his invaluable guidance and encouragement throughout my PhD. It has been always very easy to reach to him and discuss ideas with him. The discussions with him and the feedback provided by him have proved very valuable throughout my PhD. It was a really wonderful experience to work with him.

I take this opportunity to thank to Prof. Mohan Kankanhalli and Dr. Amitabha Das for their invaluable guidance and suggestions provided during my PhD. The suggestions provided by them helped a lot to improve my work.

I would like to thank all my friends who made my stay at NTU a memorable experience. In particular, I want to thank A. V. Subramanyam, Manish Narwaria, Dilip Prasad, Amit Singh, Raj Kumar Gupta, Zhang Peng and Dr. Tony Thomas for their help and support during my PhD. I would also like to thank all the members of our DRM team and lab staff at CEMNET lab for all their help, when it mattered most.

Last but not the least, I wish to thank my parents and brother for their tremendous support, and encouragement. I would like to especially thank to my parents, who always allowed me to freely choose my career path and always supported my decisions.

Contents

Acknowledgments	i
List of Figures	viii
List of Tables	x
Abstract	xi
1 Introduction	1
1.1 Motivation	1
1.2 Existing DRM Technology	4
1.2.1 DRM Technology in Two-level Systems	4
1.2.2 DRM Technology in Multi-level Systems	6
1.3 Problem Statement	7
1.3.1 Content Distribution Through Multiple Untrusted Distributors	7
1.3.2 Redistribution Rights Violation Detection	8
1.3.3 Usage Rights Violation Detection	11
1.3.4 Applications of the Proposed Technology	13
1.4 Major Contributions	13
1.5 Thesis Organization	15
2 Background Theory and Literature Review	18
2.1 DRM Architectures in Literature	19
2.2 Commercial DRM Systems and Standards	23
2.2.1 Open Mobile Alliance (OMA) DRM	23
2.2.1.1 OMA DRM 1.0	23
2.2.1.2 OMA DRM 2.0	24
2.2.2 Marlin DRM	25
2.2.3 Microsoft's Windows Media DRM	25
2.2.4 FairPlay DRM	26
2.3 Digital Licensing in DRM	26
2.4 Revocation Issues in DRM	29
2.5 Violation Detection in DRM	30

2.5.1	Unauthorized Distribution, Consumption and Modification . .	30
2.5.2	Improper Use of Permissions and Constraints	32
2.6	Conclusion	32
3	DRM Architecture for Multi-level Content Distribution Networks	34
3.1	Requirements in the Architecture	34
3.1.1	Requirements of the Owner	34
3.1.2	Requirements of the Distributors	35
3.1.3	Requirements of the Consumers	36
3.2	Entities in the DRM Architecture	37
3.2.1	The Distribution Chain Part	37
3.2.2	Administration Part	38
3.3	Various Functions in the Architecture	39
3.3.1	Content Distribution	39
3.3.2	Content Packaging	39
3.3.3	Super-distribution of Contents	42
3.3.4	Digital Licensing	43
3.3.4.1	License Creation	43
3.3.4.2	License Distribution	45
3.3.5	Content Consumption	46
3.3.5.1	Usage License Acquisition	46
3.3.5.2	License Execution	46
3.4	DRM System Violation Detection	47
3.5	Privacy Concerns	49
3.6	Related Works and Discussions	50
3.7	Discussions on Security and Rights Management	53
3.7.1	Security in the Proposed Architecture	53
3.7.2	Licensing and Rights Management	54
3.8	Conclusion	55
4	Instance-based Licenses Validation for Digital Rights Violation De- tection	56
4.1	Problem Definition	56
4.2	Related Work	61
4.3	Proposed Licenses Organization Based Method for Efficient Validation	63
4.3.1	Bit-vector Transform	63
4.3.1.1	Elementary Range Insertion	64
4.3.1.2	Bit-vector Modification	65

4.3.2	Validation in Transformed Space	67
4.3.2.1	Basis of Validation	67
4.3.2.2	Validation using Bit-vectors	68
4.3.3	Bit-vector Transform for Multiple Contents	70
4.3.4	Data Structure for Efficient Validation	70
4.4	Discussion	72
4.4.1	Special Cases	73
4.4.1.1	Multiple Constraint Ranges in a Received Redistribution License	73
4.4.1.2	Multiple Constraint Ranges in an Issued License	75
4.4.2	License Revocation Issues	76
4.4.2.1	Revocation of a Received Redistribution License	76
4.4.2.2	Revocation of a Constraint Range	76
4.4.3	Representation of Bit-Vectors	78
4.4.4	Design Considerations	78
4.4.4.1	Distributed Multi-server Deployment	79
4.4.4.2	Parallel Processing Issues	80
4.4.5	Applications of the Proposed Method	80
4.5	Performance Analysis	82
4.5.1	Mathematical Performance Analysis	82
4.5.2	Experimental Performance Analysis	84
4.6	Conclusion	89
5	Aggregate Licenses Validation for Digital Rights Violation Detection	92
5.1	Problem Definition	92
5.1.1	A Method for Aggregate Validation	94
5.1.1.1	Dynamic Update of Validation Inequalities	98
5.1.2	Requirement of Efficient Aggregate Validation	99
5.1.3	Our Contributions	100
5.2	Related Work	101
5.3	Validation Tree for Efficient Aggregate Validation	102
5.3.1	Validation Tree Based Data Organization	102
5.3.1.1	Generation of Validation Tree	102
5.3.1.2	Validation Using Validation Tree	104
5.3.1.3	Selective Pruning of Validation Tree	107
5.3.2	Analysis of the Validation Algorithm	110

5.3.2.1	Proof of Correctness	110
5.3.2.2	Compactness of Validation Tree	113
5.3.2.3	Efficient Validation Time Performance	114
5.3.2.4	Insertion of Records	114
5.4	Efficient Aggregate Validation By Deriving Relationship Between Val- idation Inequalities	115
5.4.1	Relationship Between Validation Inequalities	115
5.4.2	Proposed Data Structure	117
5.4.2.1	Bit-vector Representation of Log Records	118
5.4.2.2	Bit-vector Nodes	118
5.4.2.3	Two-dimensional Array Based Data Structure	119
5.4.3	Efficient Validation Using Proposed Data Structure	121
5.4.3.1	Calculation of the First Term	121
5.4.3.2	Subset Search Using Bit-vectors	122
5.4.3.3	Calculation of the Second Term	124
5.4.4	Theoretical Performance Analysis	124
5.4.4.1	Insertion Time Complexity	125
5.4.4.2	Validation Time Complexity	126
5.4.4.3	Storage Space Complexity	127
5.5	A Graph Theory Based Approach to Remove Redundant Validation Inequalities	127
5.5.1	Geometric Representation of Licenses	127
5.5.2	Redundant Validation Inequalities	128
5.5.3	Identification of Disconnected Groups	130
5.5.4	Validation Algorithm	132
5.5.4.1	Division of Validation Tree	133
5.5.4.2	Modification of Indexes	134
5.6	Experimental Performance Analysis	136
5.6.1	Property of Datasets	138
5.6.2	Compactness Analysis	138
5.6.3	Validation Time Performance	140
5.6.4	Insertion Time Performance	145
5.6.5	Storage Space Requirements	146
5.7	Conclusion	149

6	DRM Violations Detection Using Consumer Logs Analysis	151
6.1	Problem Definition	151
6.2	Preliminaries	154
6.2.1	Sequential Pattern Mining	154
6.2.2	Related Works	157
6.3	Log Transform for Sequential Pattern Mining	157
6.3.1	Format of Logs	158
6.3.2	Alert Logs Generation	158
6.3.2.1	Static State Model	158
6.3.2.2	Dynamic State Model	159
6.3.2.3	Alert Logs Database	161
6.4	Global and Semi-global Analysis	161
6.5	Proposed Efficient Algorithm	163
6.5.1	Log Data Classification	163
6.5.2	Global and Semi-global Violation Pattern Mining	166
6.5.2.1	Basis of Efficient Mining Using Bit-vectors	166
6.5.2.2	Violation Pattern Mining Algorithm	169
6.6	Discussion	173
6.6.1	Analysis of the Violation Patterns	173
6.6.2	Practical Issues with Violation Detection Using Logs	177
6.7	Performance Analysis	178
6.7.1	Mathematical Complexity Analysis	178
6.7.1.1	Separate Rules Databases Based Method	178
6.7.1.2	Notations	179
6.7.1.3	Time Complexity Using Separate Rule Databases	179
6.7.1.4	Time Complexity Using Our Proposed Approach	181
6.7.2	Experimental Performance Analysis	181
6.7.2.1	Violation Pattern Mining Time	182
6.7.2.2	Alert Logs Classification Time	184
6.7.2.3	Storage Space Required	184
6.8	Conclusion	185
7	Application of the Bit-vector Transform in Multimedia Social Networks	187
7.1	Introduction	187
7.2	Related Works	190
7.2.1	Access Control in Social Networks	190

7.2.2	Efficient Access Control	193
7.3	Model of Access Control	194
7.4	Proposed Efficient Access Control Mechanism	196
7.4.1	Access Rights Organization	196
7.4.2	Access Control Mechanism	200
7.4.3	Modification of Access Rights	201
7.4.4	Deletion of a Content	202
7.4.5	Efficient Data Representation	204
7.5	Performance Analysis	205
7.5.1	Mathematical Performance Analysis	206
7.5.2	Experimental Performance Analysis	208
7.5.3	Practical Significance of the Results	210
7.6	Conclusion	211
8	Conclusion and Recommendation for Future Work	212
8.1	Conclusion	212
8.1.1	Our Contributions	212
8.2	Recommendation for Future Work	214
8.3	Legal Mechanisms in DRM	217
	Publications	218
	Bibliography	220

List of Figures

1.1	DRM architectures	3
1.2	Functional architecture of two-level DRM system	5
1.3	Illustration of validation process	9
1.4	Summary of contributions	16
2.1	Layered architecture of DRM	21
3.1	Multiparty multilevel DRM architecture	37
3.2	The registration process	39
3.3	Collection of logs from the LS and consumers	48
4.1	(a). Multi-party multi-level DRM architecture (b). Validation process	59
4.2	Illustration of elementary ranges and bit-vectors	64
4.3	Elementary range insertion step	65
4.4	Bit-vector modification	66
4.5	(a)Insertion of L_{D_1} , (b)Insertion of L_{D_2}	67
4.6	Representation of an AVL tree	71
4.7	Operations on AVL tree	72
4.8	Constraint dimension for multi-ranged constraints	74
4.9	Representation of a bit-vector	79
4.10	Validation time complexity	85
4.11	Storage space complexity	87
4.12	Insertion time complexity	89
4.13	Time required to revoke a received redistribution license	90
5.1	Generation of validation tree	105
5.2	Pruning of validation tree	110
5.3	Structure of a bit-vector node	119
5.4	Example data structure (<i>arr</i>) for $N=5$	119
5.5	Representation of five redistribution licenses with two constraints . .	128
5.6	Graph and adjacency matrix for redistribution licenses in figure 5.5 .	131

5.7	Division of validation tree	134
5.8	Modification of indexes	136
5.9	Distinct sets of redistribution licenses	137
5.10	Average length of sets of redistribution licenses	137
5.11	Compactness analysis of the validation tree	139
5.12	Validation time performance of all three methods for dataset 2	141
5.13	Validation time performance of validation tree	142
5.14	Comparison of validation time performance of two-dimensional array based data structure and validation tree	143
5.15	Variation of number of groups	143
5.16	Validation time complexity after using graph theory based approach .	144
5.17	Insertion time performance	146
5.18	Insertion time vs validation tree division time	147
5.19	Storage space comparison	147
5.20	Storage space requirements for modified and validation tree based ap- proaches	148
5.21	Storage space requirements for modified and two-dimensional array based approaches	149
6.1	Violation pattern mining time	183
6.2	Alert logs classification time	184
6.3	Storage space required	185
7.1	Illustration of elementary ranges and bit-vectors	197
7.2	(a). Initial friendship(F) access right dimension. (b). Access right dimension after insertion of the end point 40 in the first content.(c). Access right dimension after insertion of the end point 100 in the first content.	198
7.3	Illustration of the content insertion process	199
7.4	(a). Friendship dimension (F), (b). Trust dimension (T), and (c). Age dimension (AG) after insertion of contents in example 7.10.	201
7.5	Insertion process in AVL tree	205
7.6	Access control time	208
7.7	Storage Space Required	209
7.8	Insertion time complexity	210

List of Tables

5.1	List of symbols	95
5.2	Issued licenses	95
5.3	Table of license records	104
5.4	Pre-processed log records	118
6.1	Table of <i>sequences</i>	155
6.2	Consumer log	158
6.3	Alert logs database (S)	161
6.4	Table of sub-databases	163
6.5	a_1 projected database	170
6.6	Different types of violations in DRM	174

Abstract

Over the past few years Internet has become one of the most convenient modes of multimedia content distribution. It has provided unique opportunity for multimedia content creators and distributors. However, it has also increased the fear of illegal content copy creation and distribution. Digital Rights Management (DRM) systems emerge as one of the possible solutions to prevent illegal distribution and usage.

Traditional DRM systems are two-level systems and comprise of owner and consumers as main entities. But a two-level DRM system may not provide sufficient business scalability to the owner. The owner may not be able to make proper business strategies for all regions and cultures. Thus, a multi-level DRM system containing owner, consumers, and multiple levels of distributors (e.g. distributors at national, regional and local levels) is required. However, inclusion of multiple levels of distributors in multi-level DRM systems presents several new concerns. These concerns include design of content packaging and distribution mechanism suitable for multi-level DRM system, design of suitable privacy preserving mechanisms, and design of suitable redistribution and usage rights distribution protocols. To take care of these concerns, in this thesis, we propose a multi-level DRM architecture, and present a cryptographic content packaging and distribution mechanism, privacy preserving mechanism, and redistribution and usage rights distribution mechanism.

Distributors are given redistribution rights using *redistribution licenses*. A *redistribution license* is a contract between owner and distributor that defines conditions on the content redistribution by the distributor. However, a malicious distributor may violate the conditions in the *redistribution licenses*. So, it is necessary to detect the violations of redistribution rights. Also, multi-level DRM systems are distributed in nature so efficient violation detection mechanisms may be required. For this purpose, we propose two new license validation techniques namely, instance based validation and aggregate validation. For efficient instance based validation, we propose bit-vector transform based licenses organization and an efficient data structure using bit-vector transform. For efficient aggregate validation, we propose a prefix tree based data structure, a graph theory based algorithm, and a data structure by determining the relationship between validation inequalities used for the aggregate validation.

Consumers are given usage rights using *usage licenses*. A *usage license* defines the conditions on usage of contents. It is executed using cryptographic protocols and a trusted DRM agent on the consumers' machines. However, due to various vulnerabilities such as operating system vulnerabilities, hardware/software vulnerabilities, a malicious consumer may illegally consume a content. Moreover, the malicious consumer can share the vulnerability information with others. This may cause huge losses to the system owner. So, it is necessary to detect the violations and vulnerabilities in DRM systems. In this thesis, we propose a violation and vulnerability detection method by analyzing the logs collected from the consumers. For log analysis, we propose a sequential pattern mining based algorithm called global and semi-global violation detection. Effectiveness and efficiency of the proposed algorithm are shown theoretically and experimentally.

We also demonstrate the application of our proposed bit-vector transform for efficient access control in multimedia social networks. The proposed method is found to be efficient as compared to existing access control mechanisms in social networks.

Chapter 1

Introduction

1.1 Motivation

Recent growth of Internet technologies has made the distribution of digital contents such as e-books, music, videos, etc. very convenient as compared to traditional means of content distribution. The traditional means of content distribution such as hard-copy distribution of books, distribution using cassettes, CDs, etc. require a distinct copy of content for each potential consumer. Whereas, for a digital content's distribution using the Internet, only a single copy needs to be uploaded over the Internet. The uploaded content can be downloaded by any number of users. This has provided a unique opportunity to the content owners/creators to generate revenue by distributing their contents in new and open markets.

Although digital contents can be easily distributed over the Internet, but the fear of illegal content usage and distribution has also increased. To protect the contents from illegal distribution and consumption, cryptographic techniques [32, 43, 54, 107] such as encryption and authentication may be used. However, the cryptographic techniques alone are not sufficient to prevent the illegal content distribution and consumption. A person who legally obtains a content can illegally share the encrypted multimedia content along with the keys corresponding to the content. Any illegitimate user can consume the content in the same manner as that by a legal consumer. So, a new technology in addition to the cryptographic techniques is also necessary.

Digital rights management (DRM) [18, 80, 148] emerged as a possible solution to prevent the illegal content distribution and usage. The DRM technology uses techniques such as licensing, secure software/hardware [171], secure storage [44, 144], etc. along with the cryptographic techniques to prevent the illegal content distribution and usage.

With the help of the DRM technology, the owner can securely distribute the contents to consumers, but the existing DRM technology limits the distribution capabilities of the owner. The existing DRM technology mainly supports two-level DRM architectures [122, 138, 162] (shown in Figure 1.1(a)) consisting of only the owner and consumers as main entities. In a two-level DRM architecture often it may not be feasible for the owner to make proper business strategies for all regions and cultures. To allow more innovative and scalable business models which have the flexibility of packaging multiple contents together in a regional and culturally sensitive manner it is necessary to have a more flexible and hierarchical distribution network. Hence, a multi-level DRM architecture, as shown in Figure 1.1(b), involving multiple levels of distributors and sub-distributors in addition to the owner and consumers is necessary. A local distributor can better explore the potentially unknown market to the owner and make strategies according to the market. In addition distributors can also help in handling different price structure of multimedia in different countries, and in case of price or demand fluctuation, the cost may be shared among the owner and distributors.

The requirement of multi-level DRM architectures has become even more prominent recently with the evolution of new applications such as the Internet protocol television (IPTV) distribution, multi-level video on demand (VOD) [51], multi-level movie rental services [95, 163], etc. These applications are distributed in nature and may require a multi-level distribution system for sufficient business flexibility. For example, in IPTV distribution systems the owner may require distributors at the national, regional, and local levels.

Figure 1.1: DRM architectures

Multi-level DRM systems involve the owner, multiple levels of distributors and consumers. The content distribution and consumption process in multi-level DRM systems is maintained by giving content redistribution rights to distributors and content usage rights to consumers respectively. Content redistribution rights allow the distributors to distribute the contents according to the conditions of distribution. For example, distribution of contents in certain region, up to a certain date, etc. Content usage rights allow the consumers to consume the contents according to the conditions of the content consumption. For example, consumption of contents in certain region, for certain number of counts, etc.

The redistribution and usage rights in the DRM systems are enforced by assuming that all the components in DRM systems behave in a reliable manner. However, DRM systems are vulnerable to attacks due to various possible vulnerabilities in the system [125, 135, 155]. Using vulnerabilities in the system, malicious distributors and consumers may violate the redistribution and usage rights given to them. In existing DRM systems, there is no provision to detect such violations and these violations may cause huge losses to the owner. So, the aim of this thesis is to enhance the

existing DRM technology by devising a multi-level DRM architecture and techniques to detect the violations in the system by malicious distributors and consumers.

1.2 Existing DRM Technology

There exist both two-level and multi-level DRM systems. In this section, we discuss about the basic DRM technology used in existing two-level and multi-level DRM systems.

1.2.1 DRM Technology in Two-level Systems

As shown in Figure 1.2, a two-level DRM system consists of the owner and consumers as main entities [42, 63, 101, 110, 115, 128, 138]. The main functionalities required in a two-level DRM system are secure delivery of contents, rights management, and secure consumption of the contents on the consumers' devices. The term 'secure delivery' refers to the condition that the contents must be encrypted and it must be ensured that the contents can be consumed by the legitimate users only. The term 'rights management' refers to the mutual agreement between the owner and consumer about the terms and conditions of the content usage. The terms and conditions for the content usage are defined in terms of the permissions and constraints, where permissions such as play, copy, print, etc. are the type of accesses allowed on the content, and constraints such as time allowed, region allowed, etc. are the restrictions associated with the content access. The term 'secure consumption' refers to the condition that a legitimate consumer should be able to consume the contents only according to the permissions and constraints in the licenses, and the consumer must not be able to alter the permissions and constraints in the licenses. The main technologies used to execute above mentioned functionalities include encryption [37, 106, 167], trusted DRM agent [128, 168], and digital licensing [102, 136, 140, 148].

To securely deliver a content, the owner initially encrypts a content K with an encryption algorithm $E_{EK}[\cdot]$ (where EK is the encryption key) to get the encrypted

Figure 1.2: Functional architecture of two-level DRM system

content $E_{EK}[K]$. Let the corresponding decryption algorithm be $D_{DK}[\cdot]$, where DK be the decryption key. The owner then uploads the encrypted content $E_{EK}[K]$ in the content server (CS) and the decryption key DK in the license server (LS).

A content acquired by a consumer can only be consumed using a trusted DRM agent at the consumer's device (if a DRM agent is not already there on a consumer's device, the owner can provide a link to consumer to download it). The functions of the trusted DRM agent are to acquire the licenses from the LS, securely decrypt the content (i.e. without revealing the decryption key to the consumer), and ensure that the content is consumed according to the permissions and constraints in the licenses. Secure acquisition of content is done using the public key infrastructure (PKI) system [98]. The PKI system generates a public/private key pair [89] for each DRM agent. The public keys and private keys have the property that a message encrypted with the public key can only be decrypted with the respective private key. The public key of the DRM agent is publicly available but the private key is kept secret only to the DRM agent.

A consumer who wants to consume the content, downloads the encrypted content from the CS. To consume the content the decryption key DK corresponding to the content is required. For this purpose, the trusted DRM agent at the consumer's device

contacts the LS. The LS sends DK in a license encrypted with the public key of the trusted DRM agent software at the consumer's device. The encryption of the license with the public key of the trusted DRM agent prevents the use of the same license for any other consumer as the license can be decrypted with the private key of the consumer's DRM agent. The license also contains the permissions and constraints associated with the usage of the content.

To consume the content, the DRM agent decrypts the license with its private key to obtain the DK and the permissions and constraints associated with the content usage. Then, the DRM agent securely decrypts the encrypted content with DK and allows the consumption of the content according to the permissions and constraints in the license.

1.2.2 DRM Technology in Multi-level Systems

A multi-level DRM system contains additional levels of distributors besides the owner and consumers. So, suitable mechanisms for the content distribution and rights management are required. The content delivery mechanism in multi-level DRM systems should be such that the contents can be securely delivered even through multiple distributors in a distribution chain. The rights management mechanism in multi-level DRM systems should be such that the owner and all the distributors in a distribution chain can enforce their rights requirements.

Some of the existing DRM systems such as Microsoft Windows media DRM [10], OMA DRM [14], Marlin DRM [65], etc. suggest the use of distributors in between the owner and consumers in their specifications. The distributors in these DRM systems are assumed to be known and trusted entities to the owner [128], and the rights management in the presence of distributors is usually done on the basis of trust and non-enforceable contracts. The distributors distribute on behalf of the owner and independently issue the licenses to the consumers. It is assumed that the distributors respect all the conditions on the content distribution imposed by the owner.

The consumers in the existing multi-level DRM systems are dealt in a similar manner as that in the two-level DRM systems. The consumers can acquire a content from a distributor and consume the content in the same way as described for the two-level DRM systems.

1.3 Problem Statement

From our study on the two-level and multi-level DRM technologies, we observe that the DRM is an essential component of the content distribution networks. However, still there are several critical issues in the existing DRM technology that can be addressed to improve the performance of the current DRM systems. In this thesis, we focus on the following issues mentioned in this section.

1.3.1 Content Distribution Through Multiple Untrusted Distributors

Most of the existing DRM systems assume that the distributors to be known and trusted. However, this may not be always true. In this thesis, we tackle the problem of designing a multi-level DRM system such that the content can be distributed from the owner to consumers through multiple untrusted distributors. Following are the main challenges associated with designing such an architecture.

- In a multi-level DRM architecture, the contents need to be distributed through untrusted distributors, so it is required to design the cryptographic mechanisms so that the untrusted distributors cannot illegally distribute the contents in a way that violate the owner's terms and conditions.
- In a multi-level DRM architecture, distributors should also be allowed to express their own rights requirements on the content distribution and consumption. However, the rights enforced by distributors must not violate the rights requirement of the owner. So, a mechanism to ensure that rights are correctly enforced by distributors is required.

- Another challenge is to make the architecture suitable for real world scenarios. For example, it may be required to provide the flexibility to the distributors in terms of adding their sub-distributors, and making their own different types of content packages consisting of several contents. So, the content encryption mechanism should also be able to provide these functionalities.
- Usage data may be collected from consumers for the violation detection and policy making purposes. The privacy of the consumers must be preserved during the collection and analysis of the usage data.

1.3.2 Redistribution Rights Violation Detection

The distributors in existing DRM systems are assumed to be trusted by the owner. So it is assumed that the distributors respect the rights requirement (e.g. distribute in a certain region only) of the owner. However, a malicious distributor may violate the rights requirement of the owner while distributing the contents to other distributors or consumers. The existing DRM systems have no provision to determine if a distributor has violated the rights requirement of the owner. To allow the owner to detect the violations by a distributor, we propose the use of two types of licenses namely, redistribution and usage licenses, and two types license validation mechanisms namely, instance-based validation and aggregate validation. In this section, we first discuss about the two types of licenses. Then we describe the problem of validation and the challenges involved in doing the validation.

(a). Redistribution and Usage Licenses: The distributors are given the redistribution rights using the redistribution licenses, and consumers are given the usage rights using the usage licenses. Initially the owner issues the redistribution licenses to its distributors. The distributors in turn using their received redistribution licenses can generate new different types of redistribution licenses to their sub-distributors and new usage licenses to consumers. These new licenses must be generated according to the permissions and constraints [143, 147, 169] in their received redistribution licenses.

Both redistribution (L_D) and usage licenses (L_U) for a content K are of the form: $\{K; P; I^1, I^2, \dots, I^M; A\}$, where P represents a permission, I^i represents the i^{th} ($1 \leq i \leq M$) instance-based constraint and A represents the aggregate constraint. The permissions such as play, copy, rip etc. are the actions that can be performed on the contents. The instance-based constraints in redistribution licenses are in the form of single/multiple range(s) of allowed values for distribution, such as allowed period of distribution, region allowed for distribution, etc. The instance-based constraints in usage licenses, such as expiry date of license, region allowed for play etc. may be in the form of a single/multiple value(s) or a range(s). The range(s)/value(s) of an instance-based constraint in the further generated redistribution and usage licenses using a redistribution license must be within the respective instance-based constraint range(s) in the redistribution license. The aggregate constraint decides the number of permission P counts that can be distributed or consumed using a redistribution license or usage license respectively. For the aggregate constraint, the sum of the aggregate constraint's counts in all the valid licenses generated using a redistribution license must not exceed the aggregate constraint's count in the redistribution license. As an example, consider a redistribution license L_D shown in Figure 1.3 with two instance-based constraints (Asia and a range [10,20]) and aggregate constraint equal to 100. Four new licenses are generated using it. The licenses L_{U_2} , and L_{U_4} are treated as invalid by the validation authority. L_{U_2} is invalid because UK is outside the Asia and L_{U_4} is invalid because the sum of the aggregate constraint's counts in the previously issued valid licenses (L_{U_1} and L_{U_3}) and L_{U_4} becomes 105, which exceeds the maximum allowed 100 using L_D .

Figure 1.3: Illustration of validation process

(b). License Validation: A malicious distributor may violate the permissions and constraints in the redistribution licenses during the generation of new licenses. Thus, as part of rights violation detection, it is then necessary to validate the newly generated licenses against the redistribution license that the distributor has received. So, a validation authority is maintained to do the validations of newly generated licenses. Every license generated by a distributor passes through the validation authority before it is being issued to other distributor or consumer. The validation authority verifies the permissions and constraints in a newly generated license against the permissions and constraints in the redistribution license used to generate the new license (for this purpose, the validation authority maintains a database of redistribution licenses possessed by the distributors). A license can only be issued if it satisfies all the conditions for instance-based and aggregate constraints.

For business flexibility reasons, distributors may need to acquire multiple redistribution licenses for the same content. Different pricing structures for different ranges of constraints may not be handled with a single redistribution license. For example, if the price of a content is different in Asia and Europe then a distributor would require different redistribution licenses for the two regions. In case of multiple redistribution licenses, the validation becomes complex for both the instance-based and aggregate constraints.

For the instance-based validation, the validation time grows proportionally with the number of redistribution licenses with the distributors as we discuss in detail in Chapter 4. Thus, the presence of large number of redistribution licenses may significantly grow the validation time. But all the validations (there may be very large number of validation requests per unit time in a large and distributed multi-level DRM system) must be performed in real time basis as new usage and redistribution licenses must be issued in real time. Thus, an instance-based validation mechanism that can do the instance-based validations efficiently in the presence of large number of redistribution licenses is required.

During the instance-based validation, a newly generated license can satisfy all the instance-based constraints in more than one redistribution licenses. Let the set of all the redistribution licenses for which a newly generated license satisfies all the instance-based constraints be given by S . For the aggregate validation, we can randomly select a redistribution license with sufficient number of remaining aggregate constraint counts (after the validation of previously issued licenses) from set S to validate the aggregate constraint's count in the issued license. But selecting a redistribution license randomly from the set S for the aggregate validation may cause potential loss to the distributors as we discuss in Chapter 5. To avoid a random selection, we propose a better aggregate validation approach using the validation inequalities in Chapter 5. The validation inequalities use the whole set of redistribution licenses S using which an issued license can be instance-based validated and the aggregate constraint value in the issued license. The validation inequalities are such that at least one inequality will not be satisfied in case of aggregate constraint violation.

By doing the aggregate validation using the validation inequalities derived in Chapter 5, the loss to the distributors can be reduced but the approach requires validation using an exponential number (to the number of redistribution licenses for a content) of validation inequalities and also each validation inequality may contain up to an exponential number of summation terms. This makes the validation process computationally intensive and necessitates to do the aggregate validation efficiently.

So, for the distributors' rights violation detection, the problem is to design efficient validation mechanisms for both the instance-based and aggregate validations. We propose new license organization techniques and data structures to handle the efficient license validation problem.

1.3.3 Usage Rights Violation Detection

DRM systems provide mechanisms to securely consume the contents at the consumers' devices using the usage licenses acquired from the owner or distributors. But a malicious consumer may violate the rights given to him (in the usage licenses) using

various possible vulnerabilities [125, 135, 155] in operating systems, DRM agent software, hardware associated with DRM systems, etc. Existing DRM technology focuses only on the mechanisms to securely consume the contents, but there is no provision to detect the violations in the systems and vulnerabilities causing the violations. These vulnerabilities and violations may cause huge losses to the owner. So, it is necessary to detect the violations, and determine and fix the vulnerabilities causing the violations.

To determine the violations, we generate the log files containing the consumers' usage records and collect these log files for the violation detection. However, a DRM system generally contains thousands of consumers and analysis of the logs from every consumer by an analyst may not be practically feasible. Also, the system owners are generally more interested in obtaining the violation statistics (e.g. most frequently occurring violations) in the complete system rather than the analysis of violation for each consumer individually. So, it would be much beneficial if we can get the representative sequences in the logs of different consumers. For this purpose, we propose the use of sequential pattern mining (SPM) [31, 93]. The advantage of the SPM algorithms is that the analyst gets only a small set of representative sequences (or patterns) of violations. The analyst can easily analyze these small set of violation patterns.

However, there are several challenges associated with the use of the SPM algorithms. First, the format of the logs obtained from consumers is not suitable to apply the SPM algorithms directly because the logs are not homogeneous i.e. logs obtained from different consumers correspond to different types of licenses with different permissions and constraints. Second, the existing SPM algorithms can find the violations only for the complete system. However, to obtain more useful information (e.g. vulnerability causing violation) it requires to obtain the violation information in different parts of the system e.g. the types of violations occurring only in certain region, operating system, or a combination of region and operating system etc. For

instance, if we find that a violation occurs most frequently in Windows XP then most likely the cause of violation will be associated with a vulnerability in Windows XP. To obtain such information, the existing SPM algorithms need to be applied multiple times, which may not be efficient. So, an efficient algorithm is required for this purpose. Thus, the problem is to design a mechanism to convert the logs obtained from consumers into a form suitable for the SPM algorithms, and design an algorithm for efficient violation detection.

1.3.4 Applications of the Proposed Technology

We find that our proposed bit-vector representation can also be useful for the efficient access control problem in multimedia social networks (MMSNs). The MMSNs are the type of social network over which multimedia contents such as images, videos, blogs, etc. can be shared. Contents shared by a person can be easily accessed by anybody else over the Internet. However, due to privacy, security, and legal concerns people often want to selectively share the contents only with their friends, family, colleagues, etc. Access control mechanisms play an important role in this situation. With the access control mechanisms one can decide the persons who can access a shared content and who cannot. But continuously growing content uploads and accesses, fine-grained access control requirements (e.g. different access control parameters for different parts in a picture), and specific access control requirements for multimedia contents can make the time complexity of the access control very large. So, it is important to design an efficient access control mechanism suitable for the MMSNs. We tackle the problem of making the bit-vector transform technique suitable for the access control in the MMSNs and making it compatible with other functionalities in the MMSNs such as access rights modification, content deletion, etc.

1.4 Major Contributions

The main contributions of this thesis include the design of a DRM architecture involving multiple untrusted distributors, license validation mechanisms for distribu-

tors' rights violation detection, violation detection mechanisms for consumers' rights violation detection and the application of the proposed bit-vector technique in social network systems. The details of the contributions are as follows:

- A multi-level DRM architecture is designed to distribute the contents with untrusted distributors. To manage untrusted distributors, a cryptographic algorithm for content packaging and distribution, and a license distribution mechanism are proposed. Also, a privacy preserving mechanism to preserve the privacy of the consumers is proposed.
- The instance-based validation problem is formulated. A bit-vector transform based license organization technique and a data structure using the bit-vector transform technique have been proposed for efficient instance-based validation. Also, the mechanisms for the revocation of the constraint ranges and licenses in the bit-vector transform domain have been proposed.
- A method to determine the violation of aggregate constraints in redistribution licenses has been proposed by deriving the validation inequalities. A prefix tree based data structure called validation tree is proposed to do the validation efficiently in the presence of exponentially large number of validation inequalities and exponentially large number of summation terms in these inequalities. In addition, a method by deriving the relationship between different validation inequalities to improve the validation time performance of the validation tree is proposed. Also, a graph theory based method to improve the validation time performance by removing the redundant validation inequalities is proposed.
- A sequential pattern mining (SPM) based method for detecting the violations of usage rights by analyzing the consumer logs is proposed. To make the logs

obtained from the consumers suitable for the SPM algorithms, a log conversion method is proposed. Also, to make the existing SPM algorithms more effective for violation detection, a method called global and semi-global violation detection is proposed.

- Application of our proposed bit-vector transform for efficient access control in multimedia social networks (MMSNs) is proposed. For the efficient access control purpose, users and contents in the MMSNs are modelled in the bit-vector transform domain. In addition, the mechanisms for different functionalities required in the MMSNs such as multimedia content deletion and access rights modification are provided using the data in the bit-vector transform domain.

Figure 1.4 summarizes and establishes the relationship among different contributions in this thesis.

1.5 Thesis Organization

Rest of this thesis is organized as follows: Chapter 2 gives the background theory and literature review in the areas of digital rights management and rights violation detection.

Chapter 3 presents our proposed multi-level DRM architecture to distribute the contents in DRM systems with multiple untrusted distributors. For the proposed architecture, we present a content packaging mechanism, a rights distribution mechanism, and a mechanism for preserving the privacy of the consumers.

Chapter 4 explains the instance-based license validation and presents the bit-vector transform based efficient instance-based license validation technique. The chapter also discusses the license revocation issues and applications of the proposed

Figure 1.4: Summary of contributions

license validation technique. The proposed license validation technique is evaluated theoretically and experimentally in terms of validation time, insertion time, storage space and revocation time.

Chapter 5 explains the aggregate license validation in detail. It presents a prefix tree based data structure called the validation tree and a two-dimensional array based data structure for efficient aggregate validation. The chapter also presents a graph theory based approach for improving the efficiency of validation tree based approach. The proposed data structures and techniques are evaluated theoretically and experimentally in terms of their compactness, efficiency of validation and insertion time.

Chapter 6 presents an efficient SPM based algorithm called global and semi-global violation detection to detect the violations by the consumers by analyzing the logs obtained from the consumers. The proposed technique is analyzed theoretically and experimentally to demonstrate its effectiveness and efficiency.

Chapter 7 discusses application of the bit-vector transform technique for efficient access control in the MMSNs. The chapter also presents algorithms for the functionalities required in the MMSNs such as multimedia content deletion and access rights modification are provided using the data in the bit-vector transform domain. The theoretical and experimental results demonstrate the efficiency of the proposed access control mechanism.

Chapter 8 discusses the conclusion drawn out of this thesis. It also points to some future research directions.

Chapter 2

Background Theory and Literature Review

Digital rights management (DRM) is a technology that has emerged as one of the major tools to combat the illegal content usage and distribution problems. Since the DRM technology first came into use, it has been evolved by the addition of new techniques in it. The focus of initial DRM systems was mainly on encryption [94], secure containers and unlocking the contents for those who pay for the contents [138]. Later the focus has been shifted to innovative business models, which can provide broader capabilities to the content owners. These new business models make use of wide variety of recent technologies such as rights expression languages [13, 16, 100], digital licensing [18, 53, 150], watermarking technologies [21, 27, 147], payment systems [68, 109], and rights usage monitoring [122, 129]. In addition, new issues such as revocation [61], privacy [60, 110, 170], inter-operability among different DRM systems [74], etc. have also been evolved in new DRM systems.

In this chapter, we first review the literature in DRM architecture and DRM business models. We then review the works discussing various issues in DRM systems that are relevant to this thesis. These include issues such as rights management, and revocation and violation detection. The literature related to efficient data structures and data mining algorithms for the efficient validation and violation detection purposes is discussed in the respective chapters later in this thesis.

2.1 DRM Architectures in Literature

There are three main goals in DRM architecture design namely, to ensure the secure distribution and consumption of the contents, to get more profit from the content distribution, and to protect the rights of the parties involved in the DRM system. Various existing DRM systems [8, 10] and DRM architectures [42, 63, 101, 110, 115, 128, 138] proposed in literature aim to achieve one or more of these goals.

Liu et al. in [128] discussed several basic two-level DRM architectures. A basic two-level DRM architecture generally consists of the following entities: content provider, license server and consumers. The content providers own the digital contents. They package and protect (using encryption algorithms) the contents for the secure content distribution. They send the protected content to be uploaded on the content server. The decryption keys and usage rules are sent to the license server. Consumers can download the content from the content server. Usage rules are sent to the consumers in the form of digital licenses. Using the licenses the consumers can consume the contents. These systems also support the super-distribution [142] of contents i.e. contents can be freely distributed in the encrypted format to the non-consumers too. However, non-consumers need to obtain the respective licenses for the contents from the LS to consume the contents.

Rosset et al. in [162] presented a DRM architecture to provide a better security mechanism than the basic two-level DRM architecture by including the authentication function. The additional security is because only an authenticated user can download the content and use the content. Authentication function is managed with the help of another entity called authentication entity. However, as the contents can only be downloaded by authenticated users, the super-distribution [142] is not supported and thus the scalability of the proposed mechanism remains a major issue. So, in our proposed architecture in Chapter 3, we follow a model of DRM in which consumers can be involved in the super-distribution of the content.

DRM architectures for the streaming type of media have also been proposed by many authors [14, 56, 79, 90]. Austerberry in [56] discusses various security and scalability aspects involved in audio and video streaming. In [79] Yin et al. proposed a DRM architecture for the streaming type of media. The authors addressed the quality of service (QoS), scalability, band width, and key management issues. To address the QoS, scalability and bandwidth issues, the authors proposed to divide the stream into two parts, base layer stream and enhancement layer stream. The base layer contains low bit rate minimum quality media, enhancement layer stream can be added to it to enhance the quality. For the key management, the authors divided the system into several clusters of users. The cluster leader for a cluster is responsible for the key management in the cluster.

Various authors proposed the concept of domains in DRM [33, 66, 101, 105, 119]. A domain is formed by a set of devices such that a legally acquired content can be freely moved among the devices in the domain. A domain of devices is managed by the domain manager of that domain [105]. Popescu et al. in [33] described a DRM architecture for home networks. The authors used the broadcast encryption and group authentication mechanisms to do the content encryption and device authentication respectively. The idea behind the broadcast encryption [61] is to allow a group of devices to establish a common secret by receiving the messages from the group controller. In group authentication mechanism, the identity of a device is not important as long as it can prove it as a member of a valid group. Sheppard et al. in [119] described how various business models can be supported using the domain licensing concept. They described it for social sharing, site licensing, household sharing, and club sharing scenarios. In addition domain expression language, to write the rights expressions in the case of domains.

In this thesis, we propose a rights management algorithm for the multi-level distribution scenario. The proposed rights management algorithms in Chapters 4 and 5 can also applied to the streaming media distribution scenario and domains scenario

Figure 2.1: Layered architecture of DRM

discussed in previous two paragraphs. The cluster leader and domain managers in the streaming media and domain type distribution scenarios can work in a similar manner as the distributors in the multi-level distribution scenario in this thesis [105].

Many authors have proposed layered architectures [64, 121, 122] of DRM. The objective of a layered architectures is to simplify the complex structure into several simple abstraction layers, with each layer providing service to the layer above it. In [122], the authors present a layered DRM architecture similar to the Open Systems Interconnection (OSI) framework. The model consists of five different layers. The layers from top to bottom (in a similar manner as in OSI framework) are application layer, negotiation layer, rights expression layer, upper digital rights enforcement layer, and lower digital rights enforcement layer.

Diehl in [64] proposed a four layer model for DRM systems with each layer having a different functionality. As shown in Figure 2.1, the layer based model proposed in [64] considers both the server side and client side. The four layers are trust layer, rights management layer, rights enforcement layer and content protection layer. The trust layer ensures that only compliant devices should be able to use licenses. Rights management layer deals with permissions and constraints on both server and client side. On the server side, the rights management layer receives the commercial rights

allocated to content. This layer translates the commercial rights into usage rights. On the client side, this layer checks whether the client is authorized to perform a requested action. The rights enforcement layer ensures that contents are used according to the rights. On the server side it verifies whether commercial rights (e.g. rights in redistribution licenses in multi-level DRM) are correctly translated to the usage rights. On client side, when a consumer requests certain action such as play, copy, etc. it verifies whether the consumer is authorized to perform the action. The content protection layer ensures that nobody can use the contents without proper usage rights.

The advantage of layered architecture is that each layer is independent and the entire system can be designed in several independent parts. Also, it helps in interpretability among different DRM systems. In our proposed multi-level DRM architecture in Chapter 3, we also considered independent functionality of different entities. For example, in our proposed architecture, the content distribution, license distribution and violation detection mechanisms are independent of each other. It helps our proposed validation and violation detection techniques in Chapters 4 to 6 to be used with existing DRM systems as we discuss in Section 2.2.

Iannella in [139] discussed the shift of DRM industry from the enforcement-centric view to more application oriented view with evolution of new business models. The author also emphasized the need of value chain to secure the contents and protect the rights of all the entities in new generation DRM systems evolving for new applications. Some of the typical application which are evolving over past few years are multi-level Internet protocol television (IPTV) distribution, multi-level video on demand (VOD) services [51], multi-level movie rental services [95, 163], etc. These applications are distributed in nature and may require multi-level distribution system to provide sufficient business flexibility. For example, in IPTV distribution systems the owner may require distributors at national, regional, and local levels. Several multi-level DRM architectures [68, 76, 152, 157] have also been proposed in the literature. These architectures provide functionality for distributors in addition to the

owner and consumers in the DRM system. However, these architectures do not scale well in terms of convenience of consumers in obtaining of contents, consumption of contents, and rights management, as we discuss in more detail in Section 3.6 in Chapter 3. Our proposed multi-level DRM architecture provides a more suitable way of content distribution, consumption and rights management. We discuss our proposed multi-level and other multi-level DRM architectures proposed in literature in more detail in Chapter 3.

2.2 Commercial DRM Systems and Standards

In this section, we discuss some of the popular commercial DRM systems and standards. Most of the existing DRM systems and standards are defined for a two-level system so we also discuss the compatibility of the proposed multi-level DRM architecture with the existing DRM systems and standards.

2.2.1 Open Mobile Alliance (OMA) DRM

OMA DRM [9, 14] is a DRM standard invented by open mobile alliance particularly for mobile devices. It is one of the widely used current DRM standards in mobile industry. Currently there are two different versions of OMA DRM namely, OMA DRM 1.0 and OMA DRM 2.0.

2.2.1.1 OMA DRM 1.0

OMA DRM version 1.0 [9] provides three basic DRM functionalities namely, forward lock, combined delivery and separate delivery without strong content protection. Forward lock is the method that allows a device to receive and consume a content, but does not allow forwarding of the content to other devices. It is mainly used in mobiles phones for mainly the forward-protection of ring tones and wall papers. Combined delivery is an extension of forward lock functionality. In combined delivery, licenses associated with the contents are attached to the content as well as forward lock is

enabled. This ensures the rights management along with the forward lock. In the separate delivery method, the contents and licenses are delivered separately. A content (in encrypted form) can be delivered via unsecure channels such as email and Bluetooth. A license for a content is delivered securely using cryptographic mechanisms. In this method, a person can forward a content to his/her friends; a friend who obtains the content need to separately obtain the licenses.

The content security in OMA DRM 1.0 version is based on the assumption that the mobile devices behave in a reliable manner. In OMA DRM 1.0 version, content is not encrypted in the forward-lock and combined delivery methods. Also, the symmetric content encryption key is not encrypted in the separate delivery mode. The media contents can be stolen if a mobile device is compromised or encryption key is revealed. So, OMA DRM 2.0 version was released for better security.

2.2.1.2 OMA DRM 2.0

OMA DRM version 2.0 [14] consists of four parts namely, public key infrastructure (PKI) system, rights object acquisition protocol (ROAP), DRM content format (DCF) and rights expression languages (REL). The PKI system [98] is the main security addition in OMA DRM version 2.0 in comparison to OMA DRM version 1.0. The licenses in OMA DRM version 2.0 are transferred using the PKI mechanism to ensure more security. However, commercially the OMA DRM version 2.0 is less popular as compared to the OMA DRM version 1.0 due to difficulty in the deployment of the PKI system and certification authority in the PKI system [9].

Both DRM version 1.0 and version 2.0 specify only a single licensing authority and a single license for content. The licensing and license validation mechanisms in our proposed multi-level DRM architecture are suitable for this specification in comparison with other multi-level DRM architectures proposed in literature as we discuss in Chapter 3. So, the proposed multi-level DRM architecture can be easily used along with existing OMA DRM standards.

2.2.2 Marlin DRM

Marlin DRM [65,151] is a general purpose DRM system founded in 2005 by Intertrust, Panasonic, Philips, Samsung and Sony. Marlin DRM is based on Octopus DRM architecture [65]. The main feature of Octopus architecture is user-centric rights management as compared to device-centric rights management in existing DRM systems i.e. contents in Octopus architecture are bound to users rather than devices. A user can use a purchased content on multiple devices and a device can belong to multiple users. The relationship among users and devices is mentioned using nodes and links in a directed graph. A link from a device pointing towards a user implies that the device is associated with the user. User-centric rights management provides the flexibility to add a device anytime depending upon users' convenience. The content protection and rights management on a device is done in a similar manner as in other existing DRM system (i.e. using DRM agent and licenses).

Our proposed multi-level DRM architecture and validation mechanism can also be used with Marlin DRM to distribute the contents in multi-level distribution scenario. The only difference will be, instead of generating the device specific licenses after the validation, user specific licenses need to be generated.

2.2.3 Microsoft's Windows Media DRM

Microsoft's windows media DRM (WMDRM) [10] is a two-level DRM system designed by Microsoft. It provides the basic functionalities of content encryption, certification, and licensing. The specifications also provide functionality of using it in the multi-level distribution scenario. However, the model described for the multi-level distribution scenario suffers the drawback that only a single entity can be involved in the rights management process. So, the owner loses control over the contents once the contents reach to the distributors. The distributor may distribute the contents in a manner that can violate the owner's rights requirements. Whereas, as discussed in

Chapter 3, our proposed DRM architecture provides the functionality of rights management in such a way that rights of owner and all the distributors in a distribution chain can be preserved.

2.2.4 FairPlay DRM

FairPlay [5, 118] is a DRM technology created by Apple Inc. particularly for the sell of music on Apple's products such as iTunes, iPad and iPhone. An online account for every user is created. FairPlay DRM allows a user to install at most five devices in his/her account.

When a user connects to the Apple to purchase a content, a random user key corresponding to the user is generated. The user key is used to encrypt the content decryption key and the encrypted content decryption is sent along with the encrypted content to the user. The user key is stored in the Apple's server and is sent to the user's device as well, which is stored in the key repository at the user's device. The content at the user's device can be consumed using the content key stored at the key repository. Whenever the user installs a new device, the new device connects to the Apple's server online and retrieves the user keys corresponding to the user at the new device.

Our proposed multi-level DRM architecture and validation algorithm can also be used with FairPlay DRM in a similar manner as described for its use with Marlin DRM in Section 2.2.2. After the validation of the issued licenses by a distributor, instead of sending the licenses to the consumers, the consumers' licenses can be stored in the consumers' online accounts as in FairPlay DRM.

2.3 Digital Licensing in DRM

Licenses are the documents that are generally used to specify the rights of a consumer over a content [102, 136, 140, 148]. Rights in the licenses are specified in terms of permissions and constraints [111], where permissions refer to the type of accesses

allowed over a content and constraints refer to restrictions associated with the content access. The rights are enforced with the trusted DRM agent at the consumers' device. The trusted DRM agent does not allow to consume a content if any of the constraint in the licenses is violated. For example, if a license states that a content can be viewed up to December 31, 2011 then the DRM agent does not allow the view of content after December 31, 2011.

Although, licenses are simple documents containing permissions and constraints, but there exist issues such as fairer rights mechanisms [21], portability of rights [74, 143, 172], and new mechanisms for the rights management in different scenarios [76, 85, 101, 152, 157]. These issues are addressed by various authors in the literature.

Arnab and Hutchison in [21] discussed how to allow fairer use of digital rights. The authors advocated for fairer mechanism that allow users to get better and balanced deal from the right holders. Licenses in present DRM systems have input only from the owner or right holders. The authors argue that in a fairer contract scenario consumers should also be able to give the desired rights. Consumers present their desired rights to the right holders, in turn right holder presents a set of licenses which closely resembles with the requested rights. Consumers can select a license from the licenses presented. We also considered this issue while designing licensing protocols for the proposed multi-level DRM architecture.

Several authors [74, 143, 172] have raised the issue of portability of contents and rights from one DRM system to another. The format of contents and rights expression may be different in different DRM systems. The content needs to be repackaged in the format that is suitable for DRM system for which we are transferring the content. Translation of rights is more complex. The set of permissions and constraints may be different in both DRM systems. Even if the same permissions and constraints exist, it may not be easy to find a mapping. The authors solved the problem by designing a common neutral rights expression language for multiple DRM systems. In [153], authors proposed a model in which the content and rights can be stored on

different devices in the distributed environment. The authors provided protocol of transfer of protected content and rights object from one device to the other, protocol for searching appropriate rights object in multiple devices in distributed environment, and protocols for RO consumption.

Chong et al. in [85] proposed a rights management scheme suitable for organizational scenarios. In the proposed scheme, rights to different persons in an organization may be allocated according to the role of the persons in the organization. For example, in an organization the manager might be able to read detailed data whereas other persons may only see the summary of the data.

Taban et al. in [101] have proposed a hierarchical group rights management scheme. In the scheme, the license server can issue rights for a group of contents to a group of users. Each group is managed with group rights. Although their architecture is suitable for some scenarios like educational institutes but it is difficult to use it as a general DRM architecture because of difficulty in maintaining the user groups.

In [157], the authors proposed onion policy administrative model (OPA). In the model, anybody can act as a potential distributor, and distribute the rights to others. The rights distributed by each level of distributors are verified at the consumer's device. The DRM agent at the consumer's device verifies that each right issued is compliant to the upper level rights. The scheme presented is technically sound but if any distributor in the distribution chain distributes the rights wrongly then the consumer will be penalized. So, if the rights can be validated at each level then it would be better for the consumers as proposed Chapter 3. Nair et al. [150] discussed types of DRM system modeling such as 'pay per use', 'use N times', and 'metered payment'. 'use N times' model is also suitable for the multi-level DRM architecture. In our proposed architecture, we realize it using the aggregate constraint in the licenses.

There exist some multi-level rights distribution models [76, 152, 157] in which distributors are also involved in rights distribution process. We discuss the rights distribution protocols in those models in detail in Chapter 3.

2.4 Revocation Issues in DRM

Revocation of licenses and certificates is another major security related issue in the DRM systems. It may be needed to revoke the certificates or licenses of the consumers, who are involved in violation. The OMA DRM 2.0 standard [14] suggests a way of revocation using a certificate revocation list (CRL). The identities of the revoked devices are mentioned in the CRL. The CRL is sent to each device. If the name of particular device/license is in CRL then DRM agent is stopped from playing the contents. Several authors modified the idea of revocation for several other scenarios as discussed below.

Popescu et al. in [33] proposed the idea of local and global revocation list for revocation of devices in home domains. Global revocation list is valid for the whole system and local revocation list is valid for a domain. A local revocation list is derived from global revocation list and it can be updated when domain manager goes online.

Commercial systems also have the functionality of license revocation. Windows media rights management [10] system revokes the licenses using a revocation agent at the consumer's device, which on getting revocation request from the service provider can revoke the certificates or licenses.

In multi-level DRM systems, the revocation issues exists not only for consumers but also for distributors. In a multi-level DRM system, the system owner may be interested in revoking the redistribution licenses or particular rights from the distributors. There exists no previous work that addresses the distributors' rights revocation issues in a multi-level DRM system. In this thesis, in Chapter 4, we propose redistribution license and constraint range revocation mechanisms to take care of revocation issues in multi-level DRM systems.

2.5 Violation Detection in DRM

Violation in DRM refers to the condition in which a distributor or consumer does not abide by the terms and conditions on content distribution and consumption enforced by the owner. The violations may happen due to various vulnerabilities [26, 125, 155] in operating system, DRM agent software, hardware associated with DRM, etc. The violation may lead to unauthorized distribution, consumption or modification of a multimedia content, or improper use of permissions and constraints in the licenses acquired by the distributors or consumers. In this section, we discuss both these issues.

2.5.1 Unauthorized Distribution, Consumption and Modification

Digital watermarking and fingerprinting techniques are mainly used to detect the unauthorized distribution, consumption or modification of a multimedia content. In digital watermarking techniques, visible or invisible information is added to the digital media contents. There are three main uses of digital watermarking technique for violation detection purpose viz. copy-detection, tamper-detection, and source-identification.

Copy-detection deals with the detection of copies of multimedia contents that may be illegally distributed or consumed. For copy-detection purpose, before the distribution of media contents, the owner inserts a robust watermark unique to the owner, distributor or consumer in a digital multimedia content. The robust watermark [81, 159] is a type of watermark that remains robust to various signal processing attacks such as rotation, compression, stretching, blurring, sharpening, etc. The watermark is then detected using watermark detection algorithm on the consumer's device. This in turn helps the consumers in identifying the copyrighted contents and helps the owner in identifying if the content is being illegally consumed by a consumer. Another application of copy-detection techniques is the identification of illegal distribution of

contents over the Internet and peer-to-peer (P2P) networks [161]. In this scenario, watermarking detection technique is used to detect or filter the copyrighted contents.

Source identification or digital fingerprinting is another application of robust watermarking technique, in which watermark information related to a consumer is embedded in the multimedia contents acquired by the consumer. If a copy of multimedia is found to be distributed illegally then the consumer may be traced back using the watermark information on the illegal copy. However, the process of embedding a watermark for fingerprinting purpose is very challenging. This is because to embed a watermark for fingerprinting purpose, the owner may need to generate a separate watermark signal for every consumer, which may be computationally time consuming [75]. So, several new types of fingerprinting techniques such as receiver-side fingerprinting [83], intermediate node fingerprinting [166] and joint fingerprinting and decryption [59] have been proposed.

In addition to copy detection and fingerprinting in basic DRM model containing owner and consumers only, several watermarking schemes [158] have been proposed for copy detection in the DRM systems containing multiple parties e.g. multi-level DRM architecture containing owner, multiple levels of distributors and consumers. This helps to detect the violation by distributors involved in the distribution process in addition to violations by consumers. Several works [147] in the literature also address the false implication concern of the consumers. The false implication concern is that, since the owner can generate a copy of the multimedia content with the same watermark as generated for a legitimate consumer and then distribute it to an illegal consumer and then later frame the legitimate consumer for the illegal distribution.

Tamper detection is another aspect of violation detection that is addressed by many researchers [126, 159]. For tamper detection purpose, fragile watermarks are used. Fragile watermarks are type of watermarks that can easily be destroyed on the application of common signal processing operations such as compression, rotation, blurring, sharpening, etc. So, a consumer after downloading a content can verify the

integrity of the content. If the correct watermark is present in the content then this implies that the content is not tampered.

2.5.2 Improper Use of Permissions and Constraints

Previously proposed violation detection techniques such as watermarking, fingerprinting, etc. focus only on the violations due to unauthorized distribution, usage or modifications in multimedia contents. However, an authorized distributor or consumer may also violate the system by violating the permissions and constraints granted in his licenses. For example, a distributor may be allowed to distribute a multimedia content only in Asia, but for business profitability reasons the distributor may be motivated to illegally distribute the content in other regions too. Similarly, a consumer by exploiting the vulnerabilities in DRM systems, may use content for a longer duration than allowed in licenses or more number of counts than initially allowed in the licenses. So, it is also necessary to detect the violations of redistribution rights by the distributors and usage rights by the consumers. To our knowledge, there exist no work that discusses the rights violation mechanism for determining the violation of rights given to a distributor or consumer in digital licenses.

In this thesis, we investigate the various ways in which a distributor or consumer can violate his rights and validation mechanisms to counter against these violations. These include license validation methods and consumers' log analysis methods. Details of these methods are presented in Chapters 4, 5 and 6.

2.6 Conclusion

In this chapter, we presented the background theory and literature relevant to this thesis. In particular, we presented the works related to DRM architectures and systems, digital licensing, license revocation, and violation detection areas. From the review of the literature in this chapter, we can conclude that most of the existing

DRM architectures and systems are only two-level systems, and existing DRM systems lack the mechanisms to detect the violations due to improper use of permissions and constraints. Next, in this thesis, we propose a multi-level DRM architecture suitable for the content distribution in multi-level content distribution scenarios. We also discuss the mechanisms to detect the violations due to improper use of permissions and constraints in Chapters 3 to 6.

Chapter 3

DRM Architecture for Multi-level Content Distribution Networks

In this chapter, we present our proposed multi-level DRM architecture involving the owner, multiple levels of distributors and consumers. We begin this chapter by discussing the requirements of the owner, distributors and consumers in a multi-level DRM system. Then, we present the description of entities and functionalities in our proposed DRM architecture. After that we discuss the violation detection and privacy issues in the proposed DRM architecture. Finally, we compare our proposed DRM architecture with the existing DRM systems and architectures, and analyze the security and rights management functionalities in the proposed architecture.

3.1 Requirements in the Architecture

The parties involved in a multi-level DRM architecture are the owner, distributors and consumers. Each of these parties has their own concerns and requirements that need to be satisfied by the architecture. In this section, we describe concerns and requirements of each party in a multi-level DRM architecture.

3.1.1 Requirements of the Owner

The main concerns and requirements of the owner are as follows:

- The first requirement of the owner is about the security of the contents. The distributors and consumers in a multi-level DRM architecture are not trusted

parties to the owner. So, it must be ensured that the contents remain secure when the contents are with the distributors and consumers.

- The second requirement of the owner is to regulate the content distribution and consumption process i.e. the owner should be able to define and enforce his/her own policies on the redistribution and usage of the contents.
- The third requirement is that the owner should be able to detect the violations by distributors and consumers. So, the architecture should have a provision to determine the violation of rights by the distributors or consumers.
- Another requirement of the owner is to support the super-distribution functionality in the architecture. The super-distribution functionality provides the flexibility to the consumers in the DRM system to freely transfer an encrypted copy of a content to another consumer, which might be useful for the business flexibility reasons. The consumer who receives the encrypted content does not require to download a copy again, but requires a new license to consume the content.

3.1.2 Requirements of the Distributors

The main concerns and requirements of the distributors are as follows:

- The distributors should be able to define their own policies on the distribution and usage of contents while distributing the contents to other distributors or consumers. However, these policies must satisfy the policies defined by the owner. For example, if the owner initially allows consumption of a content is Asia then the distributor should be able to allow consumption in countries in Asia.
- The distributors should be able to make combined package of two or more contents. This may be required for promotional purposes. For example, a dis-

tributor may wish to add local advertisements in the contents, which may help the distributors in generating revenue by applying their own business model.

- The distributors in a multi-level DRM architecture maintain their own content servers (CS) to distribute the contents. So, they are concerned about proving their distributorship to other parties, and protection of contents in their CS from being downloaded by consumers of other distributors. This concern is because a selfish distributor can redirect his/her consumers to other distributors' CS for downloading the contents without sharing the profit with them. To resolve distributors' concerns, a content packaging mechanism is required such that the contents in distributors' CS cannot be used by the owner and other distributors, and their distributorship can also be proved.

3.1.3 Requirements of the Consumers

The main concerns and requirements of the consumers are as follows:

- The first concern of the consumers is the ease of consumption of contents. In multi-level DRM architecture, content is distributed through distribution chains containing multiple distributors. The content may need stronger security requirements (e.g. to be encrypted multiple times) to fulfill the concerns of the owner and distributors. However, a consumer should be able to consume the contents with the same convenience as in a two-level DRM architecture.
- The second concern of the consumers is the convenience of obtaining and executing the licenses for the contents. The owner and all the distributors in a distribution chain in multi-level DRM architecture are concerned about enforcement of their policies. It may cause inconvenience to a consumer in terms of communication and computation cost to obtain and execute the content usage policies from the owner and all the distributors. So, a mechanism using which the consumers can conveniently acquire and execute their licenses.

Figure 3.1: Multiparty multilevel DRM architecture

- In DRM system, the owner may be collecting the usage data for the violation detection and policy making purposes. The privacy of the consumers must be preserved during the collection and analysis of the data.

Next, we propose a multi-level DRM architecture that can fulfill the above mentioned requirements of the owner, distributors and consumers.

3.2 Entities in the DRM Architecture

For the proposed DRM architecture, we first introduce various entities in the architectures. The functions performed by these entities are discussed in Section 3.3. Based on the function of different entities in the proposed DRM architecture, we divide them in two parts namely, a distribution chain part and an administrative part as shown in Figure 3.1.

3.2.1 The Distribution Chain Part

The distribution chain part deals with the content distribution process through different distribution chains in the architecture. A distribution chain in the architecture is

a chain of owner and intermediate distributors through which a content passes before reaching to a consumer. Depending upon the length of a distribution chain, parties involved may be the owner O , multiple levels of distributors $D_{i,j}$ and consumer C , where $D_{i,j}$ represents j^{th} distributor at the i^{th} level (shown in Figure 3.1). To distribute the contents to other distributors and consumers, the owner and distributors also maintain their own content servers (CS). The content servers are represented by $CS_{0,0}$ for the owner and $CS_{i,j}$ for the distributor $D_{i,j}$.

3.2.2 Administration Part

The administrative part is responsible for providing the supporting functionalities in the proposed architecture. These supporting functionalities include authentication of different parties in the architecture, registration, licensing and log collection. The administrative part consists of a certification authority (CA), a registration authority (RA), a license server (LS) and a log collection center (LCC).

The RA and CA are responsible for registration and associating unique identity with every party in the proposed DRM system. As shown in Figure 3.2, for the registration purpose, a distributor/consumer generates its public/private key pair and sends the public key along with its identity information such as, device ID, hardware ID, etc. to the RA. Upon obtaining this information, the RA requests digital certificate for the party from the CA. During the certificate request, the RA also provides public key information and identity information to CA. The CA then generates and signs the certificate and sends it back to the RA. The certificate is then served by the RA to the party concerned. Digital certificates are used for the authentication purpose in the architecture.

The LS signs and serves the licenses created by the owner/distributors to license requesting party. The LS also keeps record of permissions and constraints involved in each license issued. The LCC collects the logs from consumers and LS, and does the processing of the logs collected to make further business strategies and to detect the violation of permissions and constraints.

Figure 3.2: The registration process

3.3 Various Functions in the Architecture

In this section, we discuss in detail various functions, such as content distribution, content packaging, super-distribution, licensing, and content consumption.

3.3.1 Content Distribution

The owner is primarily responsible for distribution of content. The owner can appoint various distributors to facilitate the distribution process, and distributors are also allowed to appoint next level of distributors. The owner and distributors can distribute the contents through their own CS and they can also rely on the super-distribution technique. Consumers can download the content from their preferred distributor/owner's content server or get it through super-distribution.

3.3.2 Content Packaging

The proposed architecture supports the packaging of different types of media contents such as video, audio, text and image files. A typical content package consists of the content K and associated header H_K . Thus, the content package can be represented as: $H_K||K$. Content header H_K contains metadata about the content K , information to play the content, information about the compression algorithm, etc.

The second part of content packaging is the encryption of the content. The content encryption mechanism must be such that it resolve the owner's concern over security of content and distributors' concerns over the illegal download of content from their CS, as discussed in section 3.1. To resolve their concerns, we propose content

encryption scheme based on two content encryption keys. The first key, called *global encryption key* (GEK), is generated by the owner and it is unique for a content. The second key, called *local encryption key* (LEK), is different for the owner and different distributors. The objective of the GEK is to prevent unauthorized use of contents, and the objective of the LEK is to prevent the overloading of content servers of the owner and distributors by consumers of other distributors.

As the first step of the content encryption, the owner encrypts content K_m with GEK_m (GEK for the content K_m) without encrypting the format header H_k of the file. Thus, after the first step, the format of the file is preserved. Symbolically, the content K_m after encryption can be written as: $H_{K_m} \| E_{GEK_m}[K_m]$

In the next step, the owner encrypts initially encrypted content along with the format header again with LEK_{o-m} (i.e. LEK of the owner for the content K_m). Then a new header, called *package header*, H_{o-m} is appended to the content so that DRM agent can recognize it as a valid content encrypted with GEK and LEK. The package header H_{o-m} also provides the information about the content download URL, license acquisition URL and other metadata related to the owner. Thus obtaining the content as: $H_{o-m} \| E_{LEK_{o-m}}[H_{K_m} \| E_{GEK_m}[K_m]]$.

When a distributor purchases distribution rights for content from the owner or other distributor, the distributor is allowed to decrypt the content using LEK of the owner/distributor concerned. The distributor gets content only encrypted with GEK and follows process similar to the process followed by the owner in the second step of content encryption before uploading the content into his/her content server. Thus, the content at j^{th} distributor in the i^{th} level can be represented as: $H_{D_{i,j-m}} \| LEK_{D_{i,j-m}}[H_{K_m} \| E_{GEK_m}[K_m]]$, where $H_{D_{i,j-m}}$ is the new package header, and $LEK_{D_{i,j-m}}$ is the LEK of the j^{th} distributor in the i^{th} level for the content K_m .

Distributors are also allowed to combine two or more contents together to make a package. Packaging multiple contents provides additional flexibility to distributors while making strategies for a particular region as distributors can package several contents in regional and culturally sensitive manner. A typical package of two contents

K_m and K_n can be represented as: $H_{D_{i,j}-(m,n)} \| LEK_{D_{i,j}-(m,n)}[H_{K_m} \| E_{GEK_m}[K_m] \| H_{K_n} \| E_{GEK_n}[K_n]]$, where $LEK_{D_{i,j}-(m,n)}$ is the LEK of the j^{th} distributor in the i^{th} level for the package and $H_{D_{i,j}-(m,n)}$ is the header for the combined package. The header $H_{D_{i,j}-(m,n)}$ may contain additional information on the usage of two contents. For example, let K_m be a multimedia content to be viewed and K_n be the content corresponding to advertisements the distributor wants to insert. The distributor can specify the time instances of playing the advertisement in the package header $H_{D_{i,j}-(m,n)}$.

The scheme presented above also resolves distributors' concern about proving their distributorship as content cannot be used without the GEK, which can only be obtained from the LS. Involvement of the LS provides assurance about legal distributorship to other parties. However, the double encryption in the scheme presented above can raise consumers' concern over ease of usage of contents. But we can make use of the fact that aim of the LEK is only to protect the misuse of the CS. So, we can use a lightweight selective encryption algorithm such that it can distort viewing experience of consumers redirected by other distributors. For example, we can make use of group of pictures (GOP) structure for lightweight selective encryption of videos. GOP structure consists of three different types of frames namely I-frame, P-frame and B-frame. I-frames are intra-coded frames i.e. I-frames are coded without any reference to other frames. P-frames (forward predictive coded) can use either I-frame or previous P-frame for their coding. B-frames (bidirectional predictive coded) can use previous and/or next I-frame or P-frames for their coding. P-frames and B-frames can be intra-coded too, but for efficiency purpose generally predictive coding is used. Assuming that P-frames and B-frames are not intra-coded, information about I-frames will be required to decode other frames. In a video, encrypting 10 seconds of I-frames in every 1 minute video may be sufficient to disturb the viewing experience of consumers. This is because a consumer who has legally purchased the content from a distributor may not tolerate 10 seconds of degraded video in every 1 minute of video. Assuming 30 frames per second and size of GOP as 12 (i.e. one

I-frame every 12 frames), the selective encryption is needed for 25 frames out of 1800 frames per minute. It increases the complexity of decryption only by 1.38 percent as compared to single encryption for all the frames. The proposed solution does not affect security of the content as security will be at least that provided by the GEK (note that lightweight encryption is done only with LEK and its aim is only to protect the misuse of CS by another authorized distributor in the system).

3.3.3 Super-distribution of Contents

In the proposed architecture super-distribution can be supported in two different ways. Below we discuss both these ways of super-distribution.

In the first way of super-distribution, a consumer can transfer an encrypted content obtained from a distributor to his/her friend. The transferred content remains encrypted with both GEK and LEK. The consumer who receives the content can contact the respective distributor to obtain the license. However, this method will not work if the consumer who receives the content wants to purchase a license from another distributor. This is because the content is encrypted with LEK of the distributor from which content is downloaded originally. So, we also propose another method of super-distribution in which content can be freely distributed and a consumer who receives a content from his/her friend can obtain the license from any distributor of his/her choice.

In the second way of super-distribution, a user who wishes to super-distribute a content acquired by him/her can instruct the DRM agent to partially decrypt the content with LEK. The content obtained after the partial decryption can be transferred to any other user. The user can connect to the owner/distributor of his/her choice to obtain a license. The LEK is not required for consumption of content obtained through super-distribution.

Both these ways of super-distribution do not contradict the distributors' concern on illegal use of their CS. This is because the content downloaded by the CS of a

distributor without obtaining a license can only be super-distributed in the first way and consumption of the content requires a valid license from the same distributor.

3.3.4 Digital Licensing

A license is a digital document used to establish an agreement between two parties. In the agreement the license issuer allows the other party (license requesting party) to access the content. The content access is regulated with the help of permissions, constraints and content decryption keys. The permissions are the type of accesses that are allowed on a content e.g. play, copy, edit, reuse, and redistribute. The constraints are limitations associated with the permissions. Various types of constraints can be time based, count based, location based, etc. The content decryption keys are used to decrypt encrypted content and are available for a particular permission only if all the constraints associated with that permission are satisfied. For example if a license is issued with play permission with constraints of 10 count and validity of 30 days. The decryption keys in the license will not be available for play permission after 30 days even if less than 10 counts are used.

3.3.4.1 License Creation

License is created by the owner and distributors for other distributors and consumers. License contains the following entries: Name of license issuing party, ID of content(s), Permissions, Constraints, and Keys required for taking appropriate action. In this architecture two types of licenses can be created by the owner/distributors. The first type of license is redistribution license and the other is usage license. Both the types of licenses are described below:

(i) Redistribution Licenses: The redistribution licenses (L_D) are created by the owner or a distributor for another distributor lower in the distribution chain. The redistribution licenses contain LEK of the party which generates license for a particular content package. The distribution of a content using a redistribution license is managed using permissions and constraints in the licenses.

Permissions (P_D) such as play, copy, print, etc. in a redistribution license correspond to the actions that can be performed by the usage licenses generated using the redistribution license. Constraints in a redistribution license are the limitations on the distribution of content. There are two types of constraints in a redistribution license viz. instance-based constraints ($I_D = \{I_D^1, I_D^2, \dots, I_D^d\}$, if there are total d instance-based constraints) and an aggregate constraint (A_D). Instance-based constraints are in the form of single/multiple range(s) of allowed values for distribution. Examples of instance-based constraints are time period allowed for distribution (e.g. January 1, 2010 to February 28, 2010), region allowed for distribution (e.g. Asia, Europe), etc. Aggregate constraint is in the form of a single count value (e.g. 50,000), it determines the total counts that can be distributed in further redistribution and usage licenses generated using the original redistribution license. In terms of permissions and constraints, a redistribution license can be represented as:

$$L_D = \langle c; P_D; I_D = \{I_D^1, I_D^2, \dots, I_D^d\}; A_D \rangle \quad (3.1)$$

The enforcement of the redistribution licenses is done with the help of license server, which keeps record of redistribution and usage licenses issued by owner/distributor.

(ii) Usage Licenses: The usage licenses (L_U) are created jointly by the owner/distributor concerned and the LS. Involvement of the LS in license creation is necessary as distributors don't know the GEK. Usage licenses contain both the GEK and LEK of the owner/distributor for a particular content.

Similar to the redistribution licenses, the terms permissions, instance-based constraints, and aggregate constraints are also defined for usage licenses. Permissions (P_U) in a usage license are the actions, such as play, copy, print, etc. that can be performed on the content using the usage license. Instance-based constraints ($I_U = \{I_U^1, I_U^2, \dots, I_U^d\}$, where I_U^j is the j^{th} instance based constraint) in a usage license are in the form of single/multiple range(s) of allowed values for usage. Example

of instance-based constraints in usage licenses are time period allowed for consumption, region allowed for consumption, etc. Aggregate constraint (A^U) is the form of single count value (e.g. 30) and it determines the number of times a content can be consumed. In terms of permissions and constraints, a usage license can be represented as:

$$L_U = \langle c; P_U; I_U = \{I_U^1, I_U^2, \dots, I_U^d\}; A_U \rangle \quad (3.2)$$

The enforcement of the usage licenses is done with the help of trusted DRM agent at consumer's device.

3.3.4.2 License Distribution

The LS does the distribution of both types of licenses. In the proposed DRM architecture, the licenses can be distributed in two ways namely, ready-made license distribution and on-demand license distribution.

In the ready-made license distribution method certain predefined licenses for all the contents are already created and stored at the LS. A distributor/consumer selects preferred type of license on the owner/distributor's website. The owner/distributor then redirects distributors/consumers to the LS. The license can be acquired through the LS.

In the on-demand license distribution, the owner/distributor creates a license for each license request made by a distributor/consumer. For this type of license distributors/consumers can negotiate for permissions and constraints with the owner/distributor. The owner/distributor creates the license with requested permissions. After creating the license, the license generating party sends it to the LS and redirects requesting party to the LS.

In both the methods licenses are validated by the LS for their correctness and then served by the LS to requesting party through a secure channel. Details about the

license validations will be presented in Chapters 4 and 5. Both these license distribution methods have their own advantages. The ready-made license distribution method has less computational and communication complexity since the owner/distributors do not need to create a license for each license request. Whereas, the on-demand license distribution method provides flexibility to distributors/consumers in choosing permissions and constraints.

3.3.5 Content Consumption

Content consumption by consumers is a two step process. The first step is usage license acquisition and the second step is license execution to use the content. In this section we describe both these steps in detail.

3.3.5.1 Usage License Acquisition

Acquisition of usage licenses is required to use the contents. In order to acquire a usage license, a consumer needs to connect to a higher party (owner/distributor) in the distribution chain using license acquisition URL, which is in the header of the content. After getting a license acquisition request from the consumer the higher party first checks registration status of the user and if the consumer is not registered then the consumer is redirected to the RA to get registered. If the consumer is already registered then the higher party asks to the consumer to select the required permissions and constraints. Then it redirects the consumer to the LS for acquiring the license.

3.3.5.2 License Execution

A trusted DRM agent at the consumer's device is required for the usage of the contents. The DRM agent first decrypts the license using its private key. After that the DRM agent verifies the signature of the license server on the license. If the signature is verified correctly then the DRM agent adds the license in the pool of licenses, which is kept in a secure database. When the consumer needs to play the content, the

DRM agent decrypts the content using the LEK and GEK in the license. The DRM agent also maintains a registry, which contains the remaining permissions of active licenses. When the consumer uses a media, the DRM agent reads the license, evaluates the rights and adjusts the remaining permissions and constraints. When any of the permissions or constraints is not satisfied, the DRM agent must not consume the content.

3.4 DRM System Violation Detection

DRM systems are vulnerable to attacks due to the programming, operating system, or hardware based loopholes [70, 71]. For example, system date change in some older DRM systems may allow consumers to use the contents for longer period, or, modification in system registry may allow consumers to bypass count restriction. In a multi-level DRM system distributors can also violate the system by not restricting themselves to the permissions and constraints in their redistribution license. For instance, suppose a redistribution license allows the distributor to sell 1000 copies of content in Asia within 1 month. Now, if the distributor sells more than 1000 copies, or sells any copy outside Asia, or sells a copy after 1 month using the license then it would be a violation. Below, we present a framework for detection of violation by consumers and distributors.

Log files created at the consumers' device and the LS can be useful to detect violation. The log files at consumers' device contain data related to usage history. These log files are automatically created or appended by the DRM agent when certain actions such as play, copy, etc. are carried out on the content. The log files at the LS contain permission and constraints granted by the owner/distributors to other distributors/consumers. These log files are appended when the licenses are served by the LS. If a consumer violates the system then there would be some mismatch between entries in the log files created at the consumer's device and the permissions and constraints granted to the consumer. So, in order to detect the violations, we need

Figure 3.3: Collection of logs from the LS and consumers

to compare the log files collected from the consumer's device with the log files at the LS related to the consumer. We also assume that the log files can be securely stored at consumers' devices, and any modification in the log files can be detected [44, 86]. If a distributor is involved in violation then there would be mismatch between the permissions and constraints granted to the distributor and granted by the distributor to other parties. The violations by distributors are detected by the validation mechanisms discussed in Chapters 4 and 5.

As shown in Figure 3.3, logs from the LS and consumers are collected at the LCC for analysis purpose. Log analysis reports violation by an individual when at least one constraint with any permission is not satisfied. For example, if a consumer is allowed to play a content 10 times. But the logs collected from his/her device show that content are played for more than 10 times then system should report violation. In the above example the violation is detected on the basis of the count based constraint, where the operator used for the count based constraints is 'summation'. Similarly, for detection of violation based on other types of constraints require other types of operators. For the location based constraint, the operator required would be 'subset'. i.e. for distributors, locations allowed in further licenses must be a subset of the locations in the parent licenses, and for consumers, location of content usage must be a subset of locations allowed in the usage licenses. For the time based constraint, the operator required might be 'less than'(some times 'greater than' also). i.e. time of

use of a permission must be less than the time permitted by the time based constraint associated with the permission. So, to detect violation we need to select suitable data analysis operations based on operators associated with different types of attributes.

Although, using the method presented in the previous paragraph, violations can be detected. But the system owners are also interested in the reasons for violations and vulnerabilities that lead to a violation. This type of analysis might not be feasible with the analysis presented in the previous paragraph. So, we present a sequential pattern mining based violation analysis method in Chapter 6.

3.5 Privacy Concerns

In DRM systems, data is collected for different purposes such as making future business strategies and violation detection. Data collected is very sensitive in DRM systems as it may disclose some facts about consumers that consumers do not want to disclose. Consider the following case: a cancer patient may not want to disclose that he is a patient of cancer, but for his/her own benefit he purchases and views some documentaries on cancer. From the logs collected, it can be determined that the consumer is watching the documentaries on cancer and it may be concluded that he may be a patient of cancer. Thus, there is a likelihood that the privacy of the consumer is breached.

In the DRM system the owner/distributors are responsible for protecting privacy for their consumers. So usage data should be collected and processed by the owner/distributors for their own consumers. However, all distributors may not have enough resources to collect and process data. So, data is collected and processed by the LCC, which is under the control of the owner. Thus, it imposes a challenge on distributors to preserve privacy of their consumers. For this purpose, we use a temporary ID based scheme. In the scheme, distributors provide a temporary ID to their consumers. The temporary ID is used in usage logs in place of real ID. Logs at the LS should also contain only temporary ID and associated distributor as identifier.

The LCC analyzes the logs for violation detection, if any violation is detected then the LCC may take help of the logs from the LS to identify distributor associated with temporary ID of consumer. Then the distributor concerned may be contacted to identify violators.

The scheme presented above is privacy preserving as the LCC cannot correlate the data collected with consumers' real ID. The data at the LS also does not contain real ID of the consumer. Only the concerned distributor/owner knows about the real ID and identity of consumers. So, identity of consumers can only be disclosed with the help of distributors.

3.6 Related Works and Discussions

In this section, we discuss and compare the existing works in multi-level DRM systems and architectures with our proposed architecture.

Some of the popular commercially used DRM systems such as Microsoft Windows media DRM [10] specify the use of distributors in between the owner and consumers. But the distributors are assumed to be known and trusted entities to the owner [128], and the content distribution is usually done on the basis of trust and non-enforceable contracts. This is because these DRM systems are primarily designed for the two-level distribution scenario and only a single entity can enforce its rights requirements. Distributors in these DRM systems distribute on behalf of the owner and independently issue the licenses to the consumers. This type of distribution scenario suffers from the drawback that once a content is distributed to a distributor, the owner loses its control over the content distribution and cannot enforce his/her rights requirements [76]. A distributor may distribute the contents in a way that could violate the rights given to the distributor (e.g. no. of copies to be sold, region allowed for distribution, etc.).

Durfee and Franklin in [68] proposed a DRM architecture for the content distribution in multi-level distribution scenario. The authors discussed the problem of

non-disclosure of terms and conditions in a contract (or license) with a distributor in the next level of licenses issued by the distributor. The authors proposed a cryptographic technique using zero knowledge proofs for this purpose. But the proposed architecture suffers several drawbacks. First, very high complexity of zero knowledge proof allows the distributors to distribute only fixed set of licenses to the consumers. Second, the zero knowledge proofs may not be suitable for the cases of complex constraints in the licenses of such multi-range instance based constraints and aggregate constraints as discussed in Chapters 4 and 5. But our proposed architecture and validation mechanisms can support both fixed sets of licenses and on-demand licenses, and complex constraints in the licenses. Third, as suggested by Palmer et al. in [39, 40], the proposed method in [68], a malicious distributor can modify the licenses and prove them their own. In our proposed DRM architecture, a distributor also need not disclose the terms and conditions in his/her redistribution license in the next level of licenses.

The authors in [152] proposed a multi-level DRM architecture. In the proposed architecture, the rights on a content are initially granted by the owner, and the distributors cannot enforce their own additional rights requirements. However, distributors are provided with the flexibility of making a content package from two or more content providers. The authors also mentioned an approach of the rights enforcement in such cases.

In [157], a model called onion policy administrative model (OPA) is described. In the model, the owner appoints his/her delegate distributors to assign rights. The rights issued by each level of distributors are verified at the client device. The DRM agent at the consumer's device verifies that each right issued is compliant to the upper level rights. The scheme presented is technically sound. But in case some distributor in the distribution chain has distributed rights wrongly then the consumer will be penalized. This is unacceptable for most practical scenarios. So, if the rights can be verified at each level then it would be better for the consumers.

Li et al. in [76], discussed the requirement of enforcement of rights of owner and all the distributors in a distribution chain. The authors proposed a method which uses multiple licenses for this purpose. Initially, the owner inserts his/her rights requirements in the form of a license called prime license. The prime license cannot be separated from the content. The distributors describe their rights requirements in secondary licenses. In order to consume a content, the consumers need to obtain licenses from all the distributors in the distribution chain. During the license acquisition each party authenticates the consumers. The DRM agents at the consumers' devices extract the common rights from all the licenses and consume the content according to the common rights. However, this method in [76] suffers from the following drawbacks. First, this method is not scalable for the consumers because if there are multiple distributors in a distribution chain then a consumer needs to obtain separate licenses and get authenticated by the owner and all the distributors. Also, all the licenses obtained must be evaluated every time the content is consumed. In our proposed method, a consumer needs to contact only a single distributor (last distributor in the distribution chain) and needs to obtain only a single license, irrespective of the number of distributors in the distribution chain. This is because the compliance of the redistribution license with last distributors in a distribution chain with the redistribution licenses with previous distributors is already verified by the validation authority. So, if the single issued license satisfies the permission and constraints in the redistribution license with the last distributor, it will also satisfy the rights requirement of other distributors in the distribution chain. Another drawback of the method in [76] is that all the licenses obtained are evaluated to obtain the common policy on the consumers' devices so if a distributor in the distribution chain distributes the rights wrongly then the consumer will be affected (in case the rights in different licenses are contrasting). This is not desirable in practical scenarios. In our proposed method, the validations are performed by the validation authority before a license is issued to the consumer so a consumer always receives a valid license.

Digital media project [104] is another project that defines the use of distributors in present day business scenario and works towards the protection of rights of all the entities involved in a distribution chain including owner, distributors, and consumers. But the technical details of the system are not publicly available so it is not possible for us to analyze the details about content packaging and rights management.

3.7 Discussions on Security and Rights Management

In this section, we discuss the performance of our proposed architecture in terms of security and rights management.

3.7.1 Security in the Proposed Architecture

The proposed architecture has the following security features.

- (1) The proposed architecture provides a secure way of distribution of contents with a distribution chain. In the proposed architecture, a content with a distributor always remain encrypted with the *GEK* of the owner. The distributors are only allowed to do the encryption and packaging operation on encrypted content with *GEK*. The proposed architecture provides the same level of security as that of a two-level DRM architecture during the content consumption. Since every contents always remain encrypted with *GEK*, the security of the contents at the consumers' devices is at least as that in two-level DRM architectures.
- (2) The proposed architecture also deals with the security concern of the distributors. The security concern of the distributors is to avoid illegal use of their content servers by the consumers of other distributors. For this concern, we provide the flexibility of distributors in choosing their own *LEKs*. A content uploaded at the CS of a distributor remains always encrypted with the *LEK* of the distributor. So, the consumers belonging to other distributors may not be able to use the content.

- (3) In addition, the proposed architecture provides violation detection functionalities for the detection of violation by the distributors and consumers. The violation by distributors are detected using validation mechanisms in Chapters 4 and 5. Violations by the consumers are detected by analyzing their usage logs as we discuss in detail in Chapter 6.

3.7.2 Licensing and Rights Management

The proposed architecture has the following licensing and rights management features.

- (1) The licensing and rights management mechanism in our proposed architecture allows the owner and distributors to flexibly express their rights requirements. The validation authority takes care whether the rights issued by a distributor are compatible with the redistribution rights granted to the distributor in his/her redistribution licenses. Also, the proposed mechanism allows the consumers to conveniently obtain the licenses as the consumers need to contact only a single distributor and a single license need to be acquired, as compared to the other multi-level DRM architectures [76, 152, 157] discussed in Section 3.6, in which multiple licenses may need to be acquired.
- (2) The licensing mechanism proposed for the architecture is also compatible with existing DRM standards such as OMA DRM standard and Marlin DRM standard. This is because these mechanisms specify the use of only a single license to consume a media content and in our proposed architecture consumers always get a single license irrespective of the number of distributors in a distribution chain. This is in contrast to most of the other multi-level DRM architectures [76, 152, 157] proposed in literature.
- (3) The proposed architecture also supports redistribution licenses and constraint ranges revocation functionality in case of violations by distributors, which we

discuss in detail in Chapter 4. Whereas, in existing multi-level DRM architectures [68, 76, 152, 157], once the rights are issued to the distributors it is not possible to revoke the redistribution rights.

3.8 Conclusion

In this chapter we presented a multiparty, multilevel DRM architecture for the content distribution in an environment in which the distributors and consumers may not be trusted. The architecture takes care of all the parties involved in terms of content packaging and distribution, and license creation and distribution. The architecture also provides additional security by using log files based method for violation detection. In the architecture, a privacy preserving mechanism is discussed to ensure that consumers' privacy does not breach because of usage data collection.

Next, in Chapters 4 to 6, we discuss the violation detection aspects in the proposed architecture in more detail. Chapters 4 and 5 discuss the validation mechanisms as a means for violation detection for the distributors. Chapter 6 discusses the violation detection for the consumers.

Chapter 4

Instance-based Licenses Validation for Digital Rights Violation Detection

In this chapter, we discuss about the instance-based validation for distributors' violation detection. This involves the discussion on instance-based validation and an efficient mechanism for doing the instance-based validation. We start the chapter by mathematically defining the problem of instance-based validation. After that we discuss the related works in the areas related to the efficient instance-based validation. Then, we present our proposed bit-vector transform based efficient validation mechanism. Finally, we conclude the chapter by comparing the performance of our proposed approach with other existing approaches.

4.1 Problem Definition

As discussed in chapter 3, content distribution and usage in the proposed multi-level DRM architecture is maintained with the redistribution licenses (L_D) and usage licenses (L_U). As shown in Figure 4.1 (a), initially the owner issues the redistribution licenses to its distributors. The distributors in turn using their received redistribution licenses can generate new different types of redistribution licenses to their sub-distributors and new usage licenses to the consumers according to the permissions and constraints [143, 147, 169] in their received redistribution licenses.

The permissions (P_D) such as play, copy, print, etc. in a redistribution license correspond to the actions that can be performed by the usage licenses generated

using the redistribution license. The constraints in a redistribution license are the limitations on the distribution of content. There are two types of constraints in a redistribution license viz. instance-based constraints ($I_D = \{I_D^1, I_D^2, \dots, I_D^d\}$, if there are total d instance-based constraints) and an aggregate constraint (A_D). Instance-based constraints are in the form of single/multiple range(s) of allowed values for distribution. Examples of instance-based constraints are time period allowed for distribution (e.g. January 1, 2010 to February 28, 2010), region allowed for distribution (e.g. Asia, Europe), etc. Aggregate constraint is in the form of a single count value (e.g. 50,000), it determines the total counts that can be distributed in further redistribution and usage licenses generated using the original redistribution license. A redistribution license can be represented as:

$$L_D = \langle c; P_D; I_D = \{I_D^1, I_D^2, \dots, I_D^d\}; A_D \rangle \quad (4.1)$$

Similar to the redistribution licenses, the terms permissions, instance-based constraints, and aggregate constraints are also defined for the usage licenses. The permissions (P_U) in a usage license are the actions, such as play, copy, print, etc. that can be performed on the content using the usage license. The instance-based constraints ($I_U = \{I_U^1, I_U^2, \dots, I_U^d\}$, where I_U^j is the j^{th} instance based constraint) in a usage license are in the form of single/multiple range(s) of allowed values for usage. Example of the instance-based constraints in usage licenses are time period allowed for consumption, region allowed for consumption, etc. Aggregate constraint (A_U) is the form of single count value (e.g. 30) and it determines the number of times a content can be consumed. A usage license can be represented as:

$$L_U = \langle c; P_U; I_U = \{I_U^1, I_U^2, \dots, I_U^d\}; A_U \rangle \quad (4.2)$$

The range(s) of instance-based constraints in newly generated redistribution and usage licenses must be within the range(s) of respective instance-based constraint in

the redistribution license used to generate them. In addition, the sum of aggregate constraint value in the redistribution licenses and usage licenses generated using a received redistribution license must be less than the value of aggregate constraint count in the given redistribution license. Let the range of the j^{th} instance-based constraint in L_D be given by $[x_D^j, y_D^j]$. Now consider the generation of the k^{th} usage license (L_{U_k}) using L_D . If the j^{th} instance-based constraint in the issued license is a range type constraint with the range $[x_{U_k}^j, y_{U_k}^j]$ then the condition $[x_{U_k}^j, y_{U_k}^j] \subseteq [x_D^j, y_D^j]$ must be true. Else if, the j^{th} constraint is a single valued constraint with the value $v_{U_k}^j$ then the condition $v_{U_k}^j \in [x_D^j, y_D^j]$ must be true. For aggregate constraints, consider m_1 number of usage licenses and m_2 number of redistribution licenses being already issued using the redistribution license L_D with the j^{th} ($1 \leq j \leq m_1$) usage license having the aggregate constraint's value A_{U_j} and the k^{th} ($1 \leq k \leq m_2$) usage license having the aggregate constraint's value A_{D_k} . Now, the distributor wants to generate the $(m_1+1)^{th}$ usage license with the aggregate constraint's value equal to $A_{U_{m_1+1}}$ (note that similar condition also holds for the generation of new redistribution license). Then the following condition must hold.

$$\sum_{j=1}^{m_1} A_{U_j} + \sum_{k=1}^{m_2} A_{D_k} + A_{U_{m_1+1}} \leq A_D \quad (4.3)$$

Distributors in existing DRM systems are assumed to be trusted entities and they can independently (from the owner) distribute the contents, and permissions and constraints associated with the contents. The content distribution in currently deployed systems (such as Microsoft WMRM, Intertrust Marlin DRM, OMA DRM, etc.) is usually done on the basis of trust and non-enforceable contracts with the distributors. It is assumed that distributors are trusted and they behave in an expected manner. But a malicious distributor may violate the permissions and constraints in the received redistribution licenses during the generation of new licenses. Thus, as part of rights violation detection, it is then necessary to validate the newly generated licenses

against the redistribution license that the distributor has received. As shown in Figure 4.1(b), a centralized¹ validation authority is maintained to do the validations of newly generated licenses. Every license generated by a distributor passes through the validation authority before it is being issued to other distributor or consumer. The validation authority validates the issued license for both the instance-based and aggregate constraints. For this purpose, the validation authority maintains a database of redistribution licenses possessed by the distributors. A license can only be issued if it satisfies all the conditions for the instance-based and aggregate constraints mentioned in the previous paragraph.

Figure 4.1: (a). Multi-party multi-level DRM architecture (b). Validation process

In practical scenarios, each distributor may be distributing a large number of contents and also for business flexibility reasons a distributor may require multiple redistribution licenses for each content. Different pricing structures for different ranges of constraints may not be handled with a single redistribution license. For example, if the price of a content is different in Asia and Europe then a distributor would require different redistribution licenses for both the regions.

In case of multiple received redistribution licenses, a newly generated license is valid based on the instance-based constraints if it satisfies all the instance-based

¹The term “centralized” indicates that the validation authority is owned by a single entity. However, its physical realization can be multi-server based. Issues related to multi-server deployment are discussed in Section 4.4.

constraints in at least one received redistribution license. Let a distributor have N number of received redistribution licenses $L_{D_1}, L_{D_2}, \dots, L_{D_N}$, each having d number of instance-based constraints. A new usage license L_U is generated. L_U will be valid if

$$\exists k : I_U^j \subseteq I_{D_k}^j \quad \forall j \leq d \quad (4.4)$$

Thus, for doing the instance-based validation, we need to find at least one redistribution license for which the issued license satisfies all the instance-based constraint. If no such redistribution license is found then the issued license will be treated as invalid.

The validation time for the instance-based validation grows proportionally with the number of redistribution licenses with the distributors. Thus, the presence of large number of redistribution licenses may significantly grow the validation time. But all the validations (there may be very large number of validation requests per unit time in a large and distributed multi-level DRM system) must be performed in real time basis as new usage and redistribution licenses must be issued in real time. Thus, a validation mechanism that can do the validations efficiently in the presence of large number of redistribution licenses is required. So, in this chapter we are dealing with the problem of efficient license validation. The problem and procedure of aggregate validation in case of multiple redistribution licenses is discussed in Chapter 5.

Similar to the efficient instance-based license validation problem, the interval query problem [108, 156], and packet classification problem [24, 45, 124, 173] have been researched in the past. But as discussed in Sections 4.2 and 4.4, the problem of license validation is more complex because the works in other areas are applicable only to the single-range type constraints in redistribution licenses and single value type constraints in usage licenses. However, in our case multi-ranged/ multi-valued constraints are also possible. Furthermore, works in other areas may not be able to perform the important DRM related functions such as license revocation and constraint range revocation.

We propose a license organization technique called bit-vector transform to do the validations efficiently. The proposed technique organizes redistribution licenses received by the distributors at the validation authority so that the validation can be performed efficiently. Mathematical analysis and experimental results show that our proposed technique can do the validation efficiently. To the best of our knowledge, the work presented is the first for the instance-based license validation and license organization for the DRM systems.

4.2 Related Work

In this chapter, we are proposing an efficient license validation technique. There are other problems such as the interval query problem [108, 156] and packet classification problem [45, 124, 173], which are similar to the license validation problem have been researched in the past. In the interval query problem, a set of n intervals is given. The problem is to find all the intervals that intersect a given point. Several data structures such as interval tree, segment tree [108, 156] etc. are proposed to solve the problem efficiently. The packet classification problem in computer networks security is generalized version of the interval query problem in multiple dimensions. In the packet classification problem, a server maintains a number of rules to classify the incoming packets as normal or malicious. The rules and packets consist of multiple attributes where each attribute is in the form of a range in a rule and in the form of single values in incoming packets. Each incoming packet is verified with the rules to find the matching rules. An incoming packet may satisfy multiple rules; the best matching rule (based on the pre-defined priority of rules) is selected to classify the packet.

We surveyed various algorithms used in the packet classification problem [24, 45, 123, 124, 173]. Efficient packet classification problem is mainly handled by using efficient data structures. In [24, 124, 173], the authors proposed the use of multidimensional data structures. But implementation of multidimensional data structures

is often limited only to few dimensions and applicable to the static cases. This is because multidimensional data structures have complex and rigid structure, which often leads to exponential growth of either storage space and/or construction time [24] with increase in the number of dimensions. This limits their use in practical scenarios. In [45, 77] for packet classification problem, authors used several one dimensional efficient data structures such as interval tree, segment tree, etc. and their variants [77, 108, 156]. Using these data structures, the results are obtained for each dimension independently and then the results obtained for every dimension can be combined to classify the packets.

However, the problem in our case is more complex than the packet classification problem. This is because in our case redistribution licenses may contain multiple ranges and usage licenses may contain range type constraints. These cases may not be directly and efficiently handled with the algorithms proposed for the packet classification problem. In order to use the packet classification algorithms, we need to divide every redistribution license containing multi-range constraints into several redistribution licenses such that every new redistribution license after the division contains only a single range for every constraint. A redistribution license with d constraints with n_i ($n_i \geq 1$) number of ranges for the i^{th} constraint needs to be divided into $n_1 * n_2 * \dots * n_d$ redistribution licenses so that all the redistribution licenses after the division contain a single range for every constraint. Thus, after considering the division of all the redistribution licenses, the validation time using packet classification algorithms will become significantly high. Whereas, as we discuss in Section 4.5.2, our proposed algorithm requires almost same time in contrast to data structures used in packet classification problem after considering multiple ranges per constraint. Furthermore, DRM related operations such as constraint range revocation and license revocation can be easily performed with our proposed data structure.

In this chapter, we are presenting a validation algorithm by proposing bit-vector transform based logic in Section 4.3. The logic used in bit-vector transform can also be

implemented using boolean data structures such as binary decision diagrams (BDDs) [134, 145]. However, as we discuss in Section 4.5.2, our proposed data structure and operations on the proposed data structure are much more suitable for implementing the bit-vector transform based logic as compared to BDD based implementation.

4.3 Proposed Licenses Organization Based Method for Efficient Validation

In this section, we present our proposed bit-vector transform based license organization method for efficient instance-based license validation. The proposed method works by organizing the received redistribution license data in the form suitable for bit-vector transform. For the sake of simplicity, in this section we present the validation in the case of single range type constraints in redistribution licenses and single valued/ range type constraints in usage licenses. More general cases of multi-range type constraints in redistribution licenses and range type constraints in usage licenses are discussed in Section 4.4.

4.3.1 Bit-vector Transform

The bit-vector transform converts the received redistribution licenses with the distributors into a d dimensional hyperspace, where d is the number of instance-based constraints. As shown in Figure 4.2, each dimension is divided into non overlapping *elementary ranges* based on the respective constraint ranges in the received redistribution licenses. Each elementary range represents a bit-vector. If there are N received redistribution licenses then the length of each bit-vector is N bits. The n^{th} bit in a bit-vector corresponds to the n^{th} received redistribution license. The n^{th} bit in the bit-vector corresponding to an elementary range is equal to 1 if the n^{th} redistribution license contains that elementary range. Else, it is 0.

Initially, we assume that each constraint dimension contains a single elementary range from 0 to ∞ with no associated bit-vector with the elementary range. When a

Figure 4.2: Illustration of elementary ranges and bit-vectors

new redistribution license is acquired by a distributor, it needs to be transformed into the bit-vector transform domain. Two steps are required to transform a new received redistribution license into the bit-vector transform domain. The first step is the elementary range insertion and the second step is the bit-vectors modification. During the elementary range insertion step constraint ranges in the redistribution licenses are inserted along the constraint dimensions. During the bit-vector modification step, the existing bit-vectors are modified to obtain new bit-vectors. Next, we present details of both these steps.

4.3.1.1 Elementary Range Insertion

Let the j^{th} constraint range in the n^{th} received redistribution license (assuming $n - 1$ received redistribution licenses were already inserted) be given by $[a_{D_n}^j, b_{D_n}^j]$. For the elementary range insertion, we call $Insert(a_{D_n}^j)$ and $Insert(b_{D_n}^j)$.

Algorithm: $Insert(x^j)$

- (i) Search x^j along the j^{th} constraint dimension.
- (ii) If x^j is already present along the j^{th} constraint dimension then Exit.
- (iii) Insert x^j at the appropriate position in the numerical order along the j^{th} constraint dimension. It will divide an existing elementary range into two elementary ranges.
- (iv) Assign the same bit-vector as the previous elementary range to both the newly generated elementary ranges.

Figure 4.3 illustrates the steps required for elementary range insertion for the n^{th} received redistribution license. It is assumed that the point $a_{D_n}^j$ was not present and $b_{D_n}^j$ was already present along j^{th} constraint dimension before the insertion of the n^{th} received redistribution license. Since $a_{D_n}^j$ was not present along the j^{th} constraint dimension so insertion of $a_{D_n}^j$ divides an existing elementary range in to two elementary ranges, both the elementary ranges contain the same bit-vector. Also, since $b_{D_n}^j$ was already present along j^{th} constraint dimension so no action is taken.

Figure 4.3: Elementary range insertion step

4.3.1.2 Bit-vector Modification

In this step, as shown in Figure 4.4, each $n - 1$ length bit-vector is modified to an n length bit-vector. Call $Modify(a_{D_n}^j, b_{D_n}^j)$ to modify the bit-vectors along the j^{th} constraint dimension.

Algorithm: $Modify(a^j, b^j)$

- (i) Append a bit (to the LHS) 0 to the bit-vectors for all the elementary ranges between 0 and a^j .
- (ii) Append a bit (to the LHS) 1 to the bit-vectors for all the elementary ranges between a^j and b^j .
- (iii) Append a bit (to the LHS) 0 to the bit-vectors for all the elementary ranges between b^j and ∞ .

Figure 4.4: Bit-vector modification

Example 1: An illustration to insert the two licenses with two instance-based constraint ranges: L_{D_1} with instance-based constraints' ranges as $\{I^1=(100, 200], I^2=(0, 10]\}$ and L_{D_2} with instance-based constraints' ranges as $\{I^1=(0, 200], I^2=(5, 15]\}$ is shown in Figures 4.5 (a) and (b) respectively.

Initially, before the insertion of any redistribution licenses both constraint dimensions I^1 and I^2 are assumed to have only a single elementary range from 0 to ∞ . On the insertion of L_{D_1} , for the constraint I_1 , both the points 100 and 200 are inserted using $Insert(x^j)$ with $x^j=100$ and 200 respectively. Then using $Modify(a^j, b^j)$ with $a^j=100$ and $b^j=200$, bit-vector 1 is inserted in the elementary range 100 and 200, and bit-vector 0 is inserted in all other elementary ranges. Next, on the insertion of L_{D_2} , for the constraint I^1 , we need not insert any new points along I^1 as both 0 and 200 are already present along I^1 . During bit-vector modification step, the bit-vectors in the elementary ranges between 0 and 200 are modified by appending the bit 1 to the LHS. For all other bit-vectors, the bit 0 is appended.

The proposed bit-vector transform gains efficiency by using the fact that most of the constraints have a fixed maximum number of elementary ranges possible in practical scenarios. So, the constraints' ranges in many received redistribution licenses may share some common end points. If an end point is common in one or more received redistribution licenses it needs to be stored only once. For example, during the insertion of L_{D_2} in Figure 4.5 (a), we need not insert the point 200 again along

Figure 4.5: (a) Insertion of L_{D_1} , (b) Insertion of L_{D_2}

the constraint I^1 dimension as it was already present. Secondly, the points along the constraint dimensions are present in the increasing order of their magnitudes. So, an efficient data structure can be used to represent each constraint dimension as we discuss in Section 4.3.4.

4.3.2 Validation in Transformed Space

In this section, we discuss the process of validation of the issued licenses in the bit-vector transformed space. First, we discuss the basis of validation. Then using the basis, we discuss the process of validation using the bit-vectors.

4.3.2.1 Basis of Validation

As discussed previously, the validation task is to determine the set of redistribution licenses for which an issued license satisfies all the instance-based constraints. Let the set of redistribution licenses be represented as S for an issued license L_U . To determine the set S , we divide the problem into several similar sub-problems, one for each constraint. In the sub-problem for the j^{th} constraint, we determine the set of received redistribution licenses S_j such that the j^{th} constraint in L_U is a subset of j^{th} constraint range in each received redistribution license in S_j i.e.

$$S_j = \{L_{D_k} : I_U^j \subseteq I_{D_k}^j \forall k \leq N\} \quad (4.5)$$

The set S can be obtained by performing the intersection operations over the sets obtained for all d constraints.

$$S = S_1 \cap S_2 \cap \dots \cap S_d. \quad (4.6)$$

This basis of validation can be easily and efficiently applied using the bit-vectors, as presented in the next sub-section.

4.3.2.2 Validation using Bit-vectors

The process of validation using bit-vectors involves obtaining separate bit-vectors for each constraint dimension and then obtaining the resultant bit-vector using these bit-vectors. An issued license (let L_U) may contain both single valued and range type constraints. The process of obtaining the bit-vector is different for the single valued and range type constraints. Therefore, we separately discuss the process of obtaining the bit-vector for the single valued and range type constraints. Then we discuss the process to get the resultant bit-vector.

4.3.2.2.1 Single Valued Constraints Let the j^{th} constraint in L_U be a single valued constraint with value v^j . First, we find the elementary range along the j^{th} constraint dimension to which the constraint value v^j in the issued license belongs. Let the constraint value v^j in the issued license belongs to the elementary range with bit-vector equal to B^j . Then the bit-vector for the j^{th} constraint is given by B^j .

For example, consider the constraint I^2 (i.e. $j=2$) in Figure 4.5 (b). If the value of the constraint I^2 in an issued license is 12 then the corresponding bit-vector B^2 will be 10.

4.3.2.2.2 Range Type Constraints Let the j^{th} constraint in L^U be a range type constraint with range $(a_U^j, b_U^j]$. Let the range of the j^{th} constraint in the n^{th} received redistribution license L_{D_n} be given by $(a_{D_n}^j, b_{D_n}^j]$. If L^U can be instance-based validated using the redistribution license L_{D_n} then the range of each range type

constraint in L_U must be within the range of the same constraint in L_{D_n} . So, for the j^{th} instance-based constraint $(a_U^j, b_U^j]$ must be within $(a_{D_n}^j, b_{D_n}^j]$. Since, the n^{th} bit (corresponding to L_{D_n}) is 1 for all bit-vectors between $a_{D_n}^j$ and $b_{D_n}^j$. Therefore, the n^{th} bit will also be 1 at the points a_U^j and b_U^j . The same is true for all the values of n . So, the bit-vector, B^j , in this case is obtained by taking the *AND* operation between the bit-vectors at the end points a_U^j and b_U^j .

For example, consider the constraint I^2 (i.e. $j=2$) in Figure 4.5 (b). If the range of the constraint I^2 in an issued license is $(4, 12]$ then the corresponding bit-vector will be obtained by performing an *AND* operation between bit-vectors obtained at 8 and 12 i.e. between bit-vectors 01 and 10. So, the bit-vector B^2 will be 00.

4.3.2.2.3 Calculation of the Resultant Bit-vector In the bit-vector transform domain each bit-vector has a unique bit position for each particular redistribution license. So, performing the logical *AND* operations among them for all d values of j is equivalent to the intersection operation in Equation 4.6. The resultant bit-vector (B) after the *AND* operations is given as:

$$B = B^1 \wedge B^2 \wedge \dots \wedge B^d \quad (4.7)$$

The set of received redistribution licenses to which the issued license can be instance-based validated is the set of licenses formed by the licenses represented by the bits= 1 in the bit-vector B . For instance, a new license with the constraint values 156 and 12 is issued using the redistribution licenses L_{D_1} and L_{D_2} (in example 1 in Section 4.3.1). In this case, we can observe from Figures 4.5(b) that 156 is within the elementary range $(100, 200]$ and 12 is within the elementary range $(10, 15]$ respectively. Thus $B^1 = '11'$ and $B^2 = '10'$ and $B = B^1 \wedge B^2 = '10'$. The value equal to 1 of the bit corresponding to the license L_{D_2} shows that the issued license satisfies all the instance-based constraints only in L_{D_2} .

4.3.3 Bit-vector Transform for Multiple Contents

In previous sub-sections, we discussed the bit-vector transform for a single content with multiple received redistribution licenses. The proposed method can also be extended to accommodate multiple contents, each with multiple received redistribution licenses. To accommodate the multiple contents, we use a separate dimension to identify the contents with distributor. The dimension is divided into the number of elementary ranges equal to the number of different contents with the distributor. So, if there are M contents with the distributor then M elementary ranges would be present. If the n^{th} received redistribution license is obtained for the m^{th} content then we need to append the n^{th} bit equal to 1 in the elementary range corresponding to the m^{th} content. For other elementary ranges, we append a bit equal to 0. A separate dimension for contents also helps in situations in which a single license is associated with multiple contents (e.g. on a package of contents).

4.3.4 Data Structure for Efficient Validation

In the bit-vector transform domain elementary ranges along each constraint dimension are present in the increasing order of magnitude. Thus, each constraint dimension can be represented using a search efficient data structure. However, the data structure must support efficient insertion, deletion and searching functionality. Some of the most appropriate data structures for this purpose include red-black trees, Splay trees and Adelson-Velsky Landis (AVL) trees [20, 41]. Red-black trees are more efficient than AVL trees in the insertion and deletion time, but perform relatively slower as compared to AVL trees in terms of searching, which is the most frequent operation during the validation process [20, 156]. Splay trees are suitable for application in which same elements need to be accessed more frequently and in the worst case the search time can be linear [156]. However, AVL trees always perform in logarithmic time. So, we use an AVL tree to represent the constraint dimensions.

An AVL tree has a property that right subtree of a node stores all the nodes with value greater than the value in the node and left subtree stores the nodes with value less than the value in the node. Thus, in Figure 4.6, if each A, B,..., and F represents a numerical value then $A < B < C \dots < F$ is true. We use two types of nodes: internal nodes and leaf nodes. Internal nodes store end points of constraint ranges in received redistribution licenses and leaf nodes store elementary ranges and the bit-vectors corresponding to the respective elementary ranges. The readers may refer to [20] to get more details about AVL trees.

Figure 4.6: Representation of an AVL tree

A separate AVL tree is designed for each constraint dimension and let it be represented by AVL_j for the j^{th} constraint dimension. Internal nodes in the AVL_j store the start or end points of the j^{th} constraint range in received redistribution licenses. Each leaf node represents a unique elementary range determined by the values stored in the internal nodes and stores the bit vector corresponding to the elementary range. For illustration, AVL tree representation for the constraint dimensions I^2 in Figures 4.5(a) and (b) is as shown in Figure 4.7(a) and (c) respectively. As shown in the Figure 4.7, the end points of constraint ranges are stored in internal nodes (represented by circles) and bit-vectors are stored in the leaf nodes (represented by rectangles). To insert a new redistribution license two steps are required. In the first step, AVL tree for each constraint dimension is modified by inserting the respective constraint

Figure 4.7: Operations on AVL tree

range in it. In the second step, the bit-vector corresponding to each elementary range is modified by traversing the modified AVL trees.

Let the j^{th} constraint range in n^{th} received redistribution license be given by $[a_{D_n}^j, b_{D_n}^j]$. In the first step, both $a_{D_n}^j$ and $b_{D_n}^j$ are inserted in the AVL_j using the normal insertion process in AVL trees [41]. Figure 4.7(a) represents the AVL tree after the insertion of received redistribution license L_{D_1} (see Example 1 in Section 4.3.1). Figure 4.7 (b) represents the AVL tree after the constraint range insertion step during the insertion of L_{D_2} .

During the bit-vector modification step, an in-order traversal (nodes are processed recursively by processing the left sub-tree, then processing the root, and finally the right sub-tree) in AVL_j is performed. During the in-order traversal, bit-vectors for all the constraint ranges between $a_{D_n}^j$ and $b_{D_n}^j$ are modified by appending a bit equal to 1. Rest of the bit-vectors are modified by appending a bit equal to 0. Figure 4.7 (c) represent the bit vector modification step after the insertion of L_{D_2} .

During the validation, a search is made in each AVL tree to obtain the bit-vector for the respective constraint dimension. Then, equation 4.7 is applied to obtain the resultant bit-vector.

4.4 Discussion

In this section, we discuss the special cases in our algorithm, license revocation issues in multi-level DRM systems, design considerations and applications of the proposed technique.

4.4.1 Special Cases

We discuss two special cases in this section. We first discuss the case in which a constraint in a received redistribution license has multiple ranges. Then we discuss the case in which an issued license has constraints with multiple ranges. Other special cases such as absence of a constraint range from a received or issued license can easily be handled by taking the absent constraint range as the whole range from 0 to ∞ .

4.4.1.1 Multiple Constraint Ranges in a Received Redistribution License

In the redistribution licenses, although, practically most of the constraints can be modeled with a single range (e.g. validity period, range of the number of counts in further redistribution licenses), but it may not be possible to represent some constraints in some cases using a single range. For example, consider the constraint region allowed for distribution, which can be represented in the numerical form using a similar convention used for the postal or phone codes. Now, let a distributor to acquire a redistribution license with the constraint region allowed as Asia and Europe. If both these regions are disjoint (according to our convention) then the constraint cannot be represented with a single range and we require multiple ranges for it.

Let the j^{th} constraint in the n^{th} received redistribution license contains multiple ranges. In the simplest way, the insertion along the j^{th} constraint dimension can be performed by inserting the end points of all the ranges present in the constraint. Then by setting the n^{th} bit equal to 1 for the elementary ranges that are inside any of the range present in the j^{th} constraint. In this case, although, the validation process works correctly for the issued licenses with the j^{th} constraint as a single valued constraint, but it may not work correctly for the issued licenses that contain the j^{th} constraint as a range type constraint. This is because the validation process in case of a range type constraint does the validation on the basis of the bit-vectors obtained only at the end points of the range (see Section 4.3.2). It does not take care of an invalid range(s) in between the valid ranges at the end points. For example, in the previous example if

Africa is in between Asia and Europe then according to the convention chosen then a new license issued by the distributor with constraint region allowed as Asia, Africa and Europe will be treated as a valid license by the validation authority (though it is invalid). This is because the end points of the constraint range in the issued license fall within Asia and Europe, which are the valid regions in the redistribution license. So, we have to eliminate the cases in which the n^{th} bit becomes 1 even if a constraint range in the issued license falls in two different constraint ranges (of the same constraint) present in the n^{th} redistribution license.

Figure 4.8: Constraint dimension for multi-ranged constraints

To handle such cases, we extend the currently proposed bit-vector transform based structure. If there are maximum max^j allowed constraint ranges for the j^{th} constraint then the structure is extended by including $max^j - 1$ additional bit-vectors with each elementary range along the j^{th} constraint dimension. Each bit-vector corresponds to a particular range in the order, out of the total max^j possible ranges. Figure 4.8 shows a constraint dimension for a constraint allowed to contain maximum 3 ranges.

Similar to the insertion process for a single ranged constraint's dimension, two steps are required for the insertion along a multiple ranged constraint's dimension. In the first step, all the end points for all the ranges present for a constraint are inserted using the same process of insertion in Section 4.3.1. However, during the bit-vector modification step, bit-vectors are modified selectively according to the order of

a constraint range in a received redistribution license. The k^{th} range only modifies the k^{th} bit-vector (out of max^j) according to the bit-vector modification step in Section 4.3.1. For example, if two constraint ranges, $(a_1^j, b_1^j]$ and $(a_2^j, b_2^j]$ are present for the j^{th} constraint with $max^j=3$. Initially a_1^j , b_1^j , a_2^j , and b_2^j are inserted along the j^{th} constraint dimension. Then $Modify(a_1^j, b_1^j)$ and $Modify(a_2^j, b_2^j)$ are called to modify bit-vector1 and bit-vector2 respectively. Bit-vector3 is modified by appending all bits equal to 0 to all the bit-vectors.

During the validation, all max^j bit-vectors are initially obtained at both the end points in an issued license. Let the bit-vectors corresponding to the start and end points be represented by $\{B_{start}^1, B_{start}^2, \dots, B_{start}^{max^j}\}$ and $\{B_{end}^1, B_{end}^2, \dots, B_{end}^{max^j}\}$ respectively. To obtain the resultant bit-vector, the following operation is performed: $(B_{start}^1 \wedge B_{end}^1) \vee (B_{start}^2 \wedge B_{end}^2) \vee \dots \vee (B_{start}^{max^j} \wedge B_{end}^{max^j})$, where ' $\vee$ ' and ' $\wedge$ ' are bit-wise *OR* and *AND* operations respectively. The operation ensures that the n^{th} bit in the resultant bit-vector along the j^{th} constraint dimension be 1 if both the end points in the issued license fall within the same range for the j^{th} constraint in the n^{th} redistribution license.

The modified data structure for multi-ranged constraints also does the validation efficiently as during the validation the search along each dimension needs to be performed only once irrespective of the maximum possible ranges for the constraint (e.g. max^j for the j^{th} constraint). We only need to perform a small number of additional *AND* and *OR* operations. Thus, as shown in Section 4.5, the relative performance of our method becomes better as compared to other methods for this case.

4.4.1.2 Multiple Constraint Ranges in an Issued License

Let the j^{th} constraint in an issued license contains multiple ranges. This case can be handled by obtaining the bit-vectors at the end points of all the ranges. The final bit-vector for the j^{th} constraint will be the bit-vector obtained by taking the *AND* operation between the bit-vectors at all the end points. Rest of the validation process remains the same.

4.4.2 License Revocation Issues

License revocation [33, 61] is one of the major security requirements in DRM architectures. In the proposed multi-level DRM architecture, distributors distribute the contents on behalf of the owner. Thus, they may be involved in violation of rights given to them. If a distributor is found guilty of violation of rights then it may become necessary to revoke some or all redistribution licenses with the distributor. In another scenario, in a distributed multi-level DRM architecture, it may become sometimes mandatory to remove a particular constraint range from the redistribution licenses. For example, a distributor may be suspended from distribution in a certain country 'X'. In this case it is necessary to remove the country 'X' from all the redistribution licenses. Since in our method, the licenses are organized into a different structures, it may present a concern whether these important revocation issues can be handled with our license organization. Both these revocation issues can be performed using the bit-vector based license organization structure.

4.4.2.1 Revocation of a Received Redistribution License

If we need to revoke the n^{th} received redistribution license out of the total N number of received redistribution licenses then essentially we need to perform an *AND* operation for all the bit-vectors along any one constraint dimension with an N length bit-vector consisting of all bits 1 except the n^{th} bit which is 0. By doing this, it is ensured that the n^{th} received redistribution license cannot be used to issue further licenses.

4.4.2.2 Revocation of a Constraint Range

If we have to revoke a constraint range $[a^j, b^j]$ for the j^{th} constraint, we just need to replace all the bit-vectors between a^j and b^j with a bit-vector having all bits equal to 0. This will cause resultant bit-vector 0 if the j^{th} constraint value falls within a^j and b^j . However, if the j^{th} constraint in the issued license is a range type constraint then this method will not work. For instance, consider a redistribution license L_{D_n} and a usage

license L_U with the j^{th} constraint ranges $[a_{D_n}^j, b_{D_n}^j]$ and $[a_U^j, b_U^j]$ respectively. If these constraint ranges are such that $[a^j, b^j] \subset [a_U^j, b_U^j] \subseteq [a_{D_n}^j, b_{D_n}^j]$ then the issued license is invalid as it contains a revoked constraint range. But according to the validation process for the range type constraints, it will remain valid as the bit-vector for this constraint will be obtained by performing the *AND* operation between bit-vector at a_U^j and b_U^j ; and both points will have n^{th} bit equal to 1.

We handle the revocation in range type constraints using the fact that for an issued license to be valid, any constraint range in the issued license must not overlap with any of the revoked constraint range(s) for the same constraint. For that, we associate an additional structure with the constraints which have any revoked range(s). The structure consists of two dimensions, one for storing the start points and another for storing the end points of the revoked constraint ranges (let $start^j$ and end^j , respectively, for the j^{th} dimension). Initially, when the first range is revoked for the j^{th} constraint $start_j$ and end_j are created. Both, $start^j$ and end^j are initially assumed to contain a single elementary range from 0 to ∞ and no bit-vectors. To revoke the range $[a^j, b^j]$, we call *Constraint_revoke*($start^j, a^j$) and *Constraint_revoke*(end^j, b^j).

Algorithm: *Constraint_revoke*($dimension^j, x^j$)

- (1) Search x^j along $dimension^j$.
- (2) If x^j is not present along $dimension^j$ then insert x^j and assign the same bit-vector as that of previous elementary range to both elementary ranges created after division.
- (3) Append a bit (to the LHS) equal to 0 to all the bit-vectors before x^j in the numerical order and a bit equal to 1 to all the bit-vectors after x^j in the numerical order.

Now, when a new license is issued, before doing the validation, a revocation check is performed for the constraints which have revoked ranges. Let the j^{th} constraint has

some revoked ranges and in an issued license the j^{th} constraint be given by $[a_U^j, b_U^j]$. Then first we retrieve the bit-vectors corresponding to a_U^j from $start^j$ and b_U^j from end^j . After that we check if both the bit-vectors are equal. Both the bit-vectors can be equal if and only if both a_U^j and b_U^j fall outside of all the revoked ranges and there is no revoked range in between a_U^j and b_U^j . Otherwise, they must differ by at least one bit (corresponding to the revoked range(s) with which the license is overlapping). Thus, if both the bit-vectors are not equal, it implies that the constraint in the issued license contains a revoked range (or it is completely within at least one revoked range) and the issued license is invalid. We don't need to do the further validation. Else, the normal validation process is followed. If the j^{th} constraint in an issued license is single valued with value v_U^j then we obtain the bit-vectors for v_U^j from both $start^j$ and end^j and follow the rest of the process same as discussed for the range type constraints.

4.4.3 Representation of Bit-Vectors

In computer bit-vectors are represented using the integer format. Each bit in a bit-vector represents a particular received redistribution license. If the integers in a computer are represented using b bits then each bit-vector can only contain up to b bits. However, the number of received redistribution licenses (N) acquired by a distributor can be more than b . Thus, a representation that can accommodate any number of received redistribution licenses is required. We use a linked list representation of bit-vectors for this purpose. Each element in a linked list contains a bit-vector of b bits and a pointer to the next b bits in the bit-vector. The last element of the linked list points to null. Thus, as shown in Figure 4.9, if there are N received redistribution licenses then each bit-vector can be represented using $\lceil (N/b) \rceil$ integers, where $\lceil \cdot \rceil$ is the ceiling operator.

4.4.4 Design Considerations

In this chapter, we described all the algorithms for single centralized validation server. However, there may be concerns on availability of information in case the single

Figure 4.9: Representation of a bit-vector

centralized server fails or remains busy (e.g. in case the validation authority receives too many validation requests to be handled by the single server or server comes under distributed denial of service attack [91, 141]). The owner may require a distributed multi-server deployment or parallel processing of the validation algorithm in such cases [114, 127, 174]. So, in this section, we discuss the suitability of the proposed algorithm in the distributed multi-server deployment and parallel processing.

4.4.4.1 Distributed Multi-server Deployment

All the algorithms in this chapter are discussed for the centralized single-server deployment, but these algorithms can easily be used in distributed multi-server deployment. Distributed multi-server deployment of the proposed validation algorithm may raise the concern on ease and overhead of information synchronization between multiple servers. But in our system information can be synchronized with very little overhead between multiple servers. This is because information synchronization in our system is required only during the acquisition of a new redistribution license by a distributor, revocation of a redistribution license, and revocation of a constraint range. But all these operations are much less frequent as compared to the validation of issued licenses. A redistribution license generally has aggregate constraint value to issue thousands of licenses so the number of licenses issued is much more than the number of the redistribution licenses inserted. Thus, the overhead due to the information synchronization is very small while using multi-server deployment. So, the proposed algorithms can be easily used in the distributed multi-server deployment.

Design of an efficient validation algorithm remains an important component of the system design in a multi-server deployment too [96, 112]. An efficient algorithm can significantly reduce the resources required in terms of hardware, energy consumption, etc. both in single-server and multi-server deployments. In particular, as discussed in Section 4.5, the proposed algorithm is about 50 times faster as compared to the direct way of validation and overheads due to storage space and insertion time are very less. So, the proposed efficient algorithm would be much useful for better resource management in both the single-server and multi-server deployments.

4.4.4.2 Parallel Processing Issues

The proposed validation algorithm is also suitable for the parallel processing. In the proposed validation algorithm, to perform the validation, we search bit-vectors independently for each of d constraint dimension, and then perform $d - 1$ AND operations between them. To parallelize the system, we can perform search operation among each dimension independently. In addition, the AND operations can also be parallelized as AND operations follow the associative property (i.e. $(x \wedge y) \wedge z = x \wedge (y \wedge z)$), where $\wedge$ is the symbol for bitwise AND operation. So, the resultant of the AND operations (equation 4.7) can be calculated by first obtaining AND operation between bit-vectors for different pairs of dimensions, and then by using these partial results to calculate the resultant of the AND operations. For example, let there be total four dimensions, and B^1 , B^2 , B^3 and B^4 be the bit-vectors obtained for these dimensions. Then, first $B^1 \wedge B^2$ and $B^3 \wedge B^4$ can be calculated separately. The partial results can then be used to calculate $B^1 \wedge B^2 \wedge B^3 \wedge B^4$.

4.4.5 Applications of the Proposed Method

The proposed validation technique is applicable to the multi-level content distribution systems where the distributors may not be trusted by the owner, such as multi-level Internet protocol television (IPTV) distribution [1, 7, 19, 73, 97], multi-level video on demand (VOD) services [51], multi-level movie rental services [95, 163], etc. In IPTV

distribution systems digital media is often distributed using multiple levels of distributors at national, regional, and local levels [1, 7, 97]. The IPTV distribution systems distribute both download and streaming types of media in the form of following three services: live television, time shifted program telecast, and video on demand (VOD) [73]. Similar to MPML DRM architecture, the owner initially sends the media contents (or live feed in case of live television) to its distributors, the distributors in turn can distribute the contents to their sub-distributors and consumers.

In IPTV distribution scenario, the distributors can distribute the contents using redistribution licenses and consumers can consume the contents using usage licenses. Typical permissions in IPTV distribution scenario are play, download, etc. Typical constraints in the usage licenses can be usage counts, duration of use (e.g. a consumer may subscribe a live TV program for one month or may use a video obtained using VOD service for 3 days), expiry date, region allowed, etc. Respective constraints in the redistribution licenses can be total allowed counts to be distributed, constraints on allowing usage duration in usage licenses, total period of distribution, region allowed for distribution, etc. The distributors can generate new usage licenses for the consumers but these newly generated usage licenses must be in accordance with the redistribution licenses they received from the owner or other distributors. So, the validation of each newly generated license is required. Since IPTV services are distributed in nature and have a large number of contents and consumers, and also each consumer may request multiple services so it is required to do the validations efficiently. Our proposed approach of efficient license validation can be much helpful in such scenarios.

Similar distribution scenarios exist in multi-level VOD and multi-level movie rental services. The proposed validation technique can be applied in a similar manner to multi-level VOD and multi-level movie rental services.

4.5 Performance Analysis

In this section, we evaluate the mathematical and experimental performance of our proposed method in terms of validation time complexity, storage space complexity, redistribution license insertion and revocation complexity.

4.5.1 Mathematical Performance Analysis

To analyze the performance mathematically, we assume that d constraints are present and the j^{th} constraint dimension has r^j elementary ranges after the insertion of N received redistribution licenses. r^j elementary ranges implies $r^j + 1$ points along the j^{th} constraint dimension. Thus, in the corresponding AVL tree j^{th} constraint dimension can be represented using $r^j + 1$ internal nodes [38]. We analyze the performance for both the cases of single range and multi-ranged constraints.

a. Validation Time Complexity. The validation process consists of two steps. The first step is to obtain the individual bit-vectors for each constraint dimension. The second step is to perform the *AND* operation among the individual bit-vectors. Thus, overall, we need to search start and end points of each constraint range in the AVL trees and perform $d + d - 1$ *AND* operations (d in the first step and $d - 1$ in the second step). The complexity of the search step for the j^{th} constraint dimension is of $O(2 * \log_2(r^j))$ (if there are n nodes present in an AVL tree, then the number of comparisons required are of $O(\log_2(n))$ [25, 48]). As discussed in Section 4.4.3, if N received redistribution licenses are present then each bit-vector can be represented using $\lceil N/b \rceil$ integers, where b is the number of bits using which integers can be represented in a computer. So, the number of the *AND* operations required are $(2 * d - 1) * \lceil N/b \rceil$. Therefore, the overall time required is $c_1 * \sum_{j=1}^d 2 * \log_2(r^j) + c_2 * (2 * d - 1) * \lceil N/b \rceil$, where, c_1 and c_2 are arbitrary system dependent constants. We computed the complexities after assuming all the range type constraints in the issued

licenses. For single valued constraints, we require only one search operation and $d - 1$ *AND* operations. Thus, the time required will be $c_1 * \sum_{j=1}^d \log_2(r^j) + c_2 * (d - 1) * \lceil N/b \rceil$.

For the case of multi-ranged constraints, if there are $max^j (max^j \geq 1)$ possible ranges for the j^{th} constraint then we require $max^j - 1$ *OR* operations and max^j *AND* operations. Thus, the complexity in this case would be $c_1 * \sum_{j=1}^d 2 * \log_2(r^j) + c_2 * \sum_{j=1}^d (2 * max^j - 1) * \lceil N/b \rceil + c_2 * (d - 1) * \lceil N/b \rceil$.

b. Storage Space Complexity. If there are r^j elementary ranges along the j^{th} constraint dimension then r^j bit-vectors would be present. Each bit vector can be represented using $\lceil N/b \rceil$ integers. Thus, $\sum_{j=1}^d r^j * (b * \lceil (N/b) \rceil)$ number of bits would be required to store the bit-vectors. If there are r^j elementary ranges then there will be $r^j - 1$ end points along the j^{th} constraint dimension. It requires $\sum_{j=1}^d r^j * (b)$ bits to store the end points. Further, an additional storage space equal to $2 * \sum_{j=1}^d r^j * (b)$ is required to store the pointers, while storing the j^{th} dimension using an AVL tree. Thus, the total space required is $\sum_{j=1}^d r^j * (b * \lceil (N/b) \rceil) + 3 * \sum_{j=1}^d r^j * b$ bits. For the case of multi-ranged constraints, the storage space required is $\sum_{j=1}^d r^j * max^j * (b * \lceil (N/b) \rceil) + 3 * \sum_{j=1}^d r^j * b$ bits.

c. Redistribution License Insertion Complexity. Let the j^{th} constraint range in a received redistribution license be given by $[a^j, b^j]$ and the number of elementary ranges between a^j and b^j be $r^{j'}$. Then the time required for constraint range insertion (see Section 4.3) is given by $2 * c_1 * \sum_{j=1}^d \log_2(r^j)$. The time required for the bit-vector modification step is $c_2 * \sum_{j=1}^d (r^{j'})$. Thus, the total time required is $2 * c_1 * \sum_{j=1}^d \log_2(r^j) + c_2 * \sum_{j=1}^d (r^{j'})$. In the case of multi-ranged constraints, the worst case insertion time required is $2 * c_1 * \sum_{j=1}^d max^j * \log_2(r^j) + c_2 * \sum_{j=1}^d max^j * (r^{j'})$.

d. Redistribution License Revocation Complexity. For revocation of a received redistribution license (say n^{th}), we need to select one constraint dimension

(say j^{th}). In each bit-vector along the constraint dimension, we first need to traverse the linked list to find the element containing n^{th} received redistribution license. To change the bit corresponding to the n^{th} redistribution license, we need to traverse $(N - n)/b$ elements in each linked list. Then we need one more operation to change the bit corresponding to the n^{th} received redistribution license to 0. In the case of multi-ranged constraints, the revocation complexity will change if a constraint having multiple possible ranges is selected. In this case we need to change the n^{th} bit in all max^j bit-vectors for each elementary range.

4.5.2 Experimental Performance Analysis

In this section, we present experimental results of our proposed approach in terms of validation time complexity, storage space complexity, license insertion complexity and revocation complexity for different values of number of received redistribution licenses (N). We have written all programs in Java and performed all the experiments on Intel(R) Xeon(R) 3.20 GHZ 32-bit CPU with 6 GB RAM. For each value of N , we perform the experiment for 1,000 times and then sum the time required. The value of b is 32 in our case as the bit-vectors are represented using 32 bits integers. To our knowledge, any standard benchmark data related to licenses in multi-level DRM is not available. So, for the experiments we generate our own simulation data.

For the experiment purpose, we assume that there are 100 contents and up to 10,000 received redistribution licenses for these 100 contents. We randomly choose between 3 to 7 permissions and 5 associated constraints with each permission in each license. We assume that the issued licenses contain both range type and single valued constraints with probabilities 0.4 and 0.6 respectively. We compare the performance for the validation time and storage space requirements of our method with the direct method and a modified direct method. In the direct method each constraint's value/range in an issued license is compared with the respective constraint's range in all the received redistribution licenses one by one. In the modified approach, the comparison with a received redistribution license is terminated as soon as we find a constraint

that is not satisfied. For the received redistribution license insertion and revocation, we compare the performance in terms of overhead as compared to the time required for the validation of a single license. In addition, as mentioned in Section 4.2, we also show the validation time and insertion time performance of the proposed validation and insertion algorithms with binary decision diagrams (BDD). BDDs are readymade data structures used in hardware validation and verification problems. To implement any Boolean logic with BDDs, we only need to program the BDD variables. Similar to the verification and validation logic in hardware, the validation logic in DRM can also be expressed in terms of Boolean variables and programmed with BDDs. More details about the BDD based implementation are discussed in Appendix.

Figure 4.10: Validation time complexity

Figure 4.10 compares the validation time performance of our proposed method against the direct approach (scaled down by a factor of 1/10) and modified approach for both single-ranged as well as multi-ranged constraints' cases. For multi-ranged constraints' case, we assume that 2 out of 5 constraints are multi-ranged constraints and the number of ranges for these constraints can be 1, 2, or 3 with the probabilities 0.4, 0.4, and 0.2 respectively. It is clear from Figure 4.10 that our method performs

much better than both the direct method and modified method for all values of N for both single-ranged and multi-ranged constraints' cases. The relative performance of our method becomes better as N increases. In particular the gain as compared to the direct and modified approaches is about 25 and 6 times respectively for $N=10,000$ (the graph for the direct approach is scaled down by 10) for the single-ranged constraints' case. The reason for the performance enhancement is the finite maximum possible values of elementary ranges and use of efficient data structure in bit-vector transform domain. Due to this the first term in the validation time expression (in Section 4.5.1) becomes constant once r^j reaches near its maximum. Another reason for the performance enhancement is the use of bit-vectors and efficiency of operations over bit-vectors in our method. Due to this the second term in the expression of validation time grows much slowly. Also, we observe that although the validation time performance of BDD based implementation is better than the direct method, but it takes relatively more time for the validation as compared to the modified method and implementation using our proposed data structure. The BDD based implementation takes relatively more time for validation because to perform each AND operation in BDD based implementation, $O(N)$ time is required. Whereas, in our proposed data structure, it requires only $\lceil N/b \rceil$ time for each AND operation. Note that the experiments for BDD based implementations are performed for up to $N=8,000$ as BDD based implementation has very high memory requirements due to complex structure of BDDs. Even after allocating 5 GB memory to java virtual machine (JVM), we could perform experiments up to $N=8,000$. Whereas, for our proposed data structure, direct and modified method, to perform experiments up to $N=10,000$, we require only a maximum of 64 MB memory.

We also observe that our method performs better after considering multi-ranged constraints. The validation time for the modified method increases by about 25% whereas for our method it increases by only 10% for most of the values of N . This is because as discussed in Section 4.4, during the validation, we need to search along

each dimension only once irrespective of the number of maximum ranges possible. And the increase in the validation time is only due to the additional logical *AND* and *OR* operations.

Figure 4.11 compares the performance in terms of storage space required for our proposed method against the direct approach for both single-ranged as well as multi-ranged constraints' cases. It can be observed from the Figure 4.11 that our method takes only about 50% more space as compared to storage space required for direct method for both single-ranged as well as multi-ranged constraints' cases. The relative performance of our method is same for both the cases as the increase in the space required to accommodate additional ranges is linear for both direct method and our proposed method.

We can also observe that although storage space required for our method is more than the direct approach, but it remains less than 50 MB, which is very small as compared to gigabytes and terabytes of storage space available in modern computers. So, the overhead due additional storage space remains very small practically.

Figure 4.11: Storage space complexity

Figure 4.12 compares the received redistribution license insertion complexity for

both the cases of single-ranged and multi-ranged constraints. The insertion time initially increases rapidly up to approximately $N=300$ but becomes constant as N grows further. This is because the number of elementary ranges for each constraint dimension reaches near to its maximum after $N=300$ and increase in the number of elementary ranges is very less after $N=300$. It is also evident from the expression of received redistribution license insertion complexity in Section 4.5.1. The values of r_j becomes constant after $N=300$ and small variations in the graph are due to the variation in the values of $r^{j'}$. For the multi-ranged constraints' case, the slope of the curve (for $N < 300$) is more due to more number of ranges to be inserted.

As shown in Figure 4.12, the insertion time with BDD based implementation is much higher as compared to the insertion time with the proposed data structure (the graphs for insertion time required in BDD based implementation is scaled by $1/2,000$ times). The higher insertion time with BDD based implementation is because it requires $O(N)$ time to perform every bit-vector modification required whereas in proposed data only $O(1)$ time is required (as we need to modify only a single bit in our proposed data structure).

Also, the insertion time of the proposed method remains comparable with the validation time. Each redistribution license generally has sufficient aggregate constraint counts to generate thousands of licenses. So, the overhead due the insertion of redistribution licenses remains very small practically. However, same is not true for BDD based implementation because the insertion time with BDD based method is much higher (about 10,000 times for $N=8,000$) as compared to the validation time.

Figure 4.13 illustrates the license revocation performance of our proposed method. It is evident from the Figure 4.13 that the time required for the redistribution license revocation remains comparable with the time required for the validation of a single issued license. Also, the revocation of redistribution licenses is very less frequent event as compared to validation of issued licenses so the overhead due to revocation of licenses (in case it is required) is very small.

Figure 4.12: Insertion time complexity

4.6 Conclusion

In this chapter, we presented the problem of license validation in multi-level DRM system in the case of multiple redistribution licenses. Since the validation needs to be done in real time, we proposed a bit-vector transform method and a data structure using bit-vector transform to improve the validation time performance. The proposed method and data structure significantly reduce the validation time. The proposed method is scalable for both the cases of single/multiple constraints' ranges in redistribution licenses. The proposed license validation method is also able to perform the basic DRM security related functions such as license revocation and constraint range revocation. We performed the mathematical and experimental analysis for the proposed bit-vector transform based validation method. Both mathematical and experimental results show that our proposed method performs much better than other existing methods of the validation. In addition, we evaluated the performance of our proposed method in terms of the storage space complexity, license insertion and revocation complexities. The proposed method also performs well in terms of all these parameters. Thus, our proposed bit-vector based license organization method

Figure 4.13: Time required to revoke a received redistribution license

turns out to be a good choice for the representation of licenses.

Appendix: Details of BDD based Implementation

In BDD based implementation, we considered a separate BDD variable for each redistribution license. So, if there are total N redistribution licenses there will be total N BDD variables. The elementary ranges are inserted into the AVL tree in the same way as in our implementation in the chapter. The bit-vectors (each bit-vector acts as a BDD object) for the elementary ranges are generated using BDD variables for the redistribution licenses.

Let the BDD variable for a newly received (say n^{th}) redistribution license be given by b^n and BDD object corresponding to the bit-vector corresponding to an elementary range R be given by B^R . If the n^{th} redistribution license contains the elementary range R then new BDD object corresponding to the bit-vector $B^{R'}$ is calculated by performing a *BDD AND*³ operation between BDD corresponding to

³note that *BDD AND* operation between two BDD objects is different from normal AND

B^R and b^n . So, the n^{th} variable in the BDD corresponding to an elementary range is 1 if the elementary range is present in the n^{th} redistribution license else it is a don't care bit (denoted by *).

For validation purpose, we obtain the BDD objects from elementary ranges using the same procedure of obtaining bit-vectors in the chapter. Then, equivalent to AND operation between two bit-vectors is obtained by performing *BDD RESTRICT* operator twice. For example, let two BDD objects be $B^1=1*1*1$ and $B^2=1***1$ (we consider these equivalent to bit-vectors 10101 and 10001 respectively). In our implementation, we first apply *BDD RESTRICT*(B^1, B^2) operation, the resultant BDD object is given by $**1**$. Then again a *BDD RESTRICT* operation is carried out between B^1 and resultant BDD object, which results in BDD object $1***1$. The resultant BDD object is equivalent to bit-vector 10001, which is the result of AND operation between 10101 and 10001. For validation, we use the fact that if final resultant BDD object contains any bit equal to 1 then the issued license will be a valid license.

For the validation using multi-ranged constraints in redistribution licenses and range type constraints in usage licenses, we perform *BDD AND* operation (which is equal to normal OR operation between the bit-vectors) between the max^j BDD objects obtained for each elementary ranges, where max^j is the number of ranges for the j^{th} constraint.

operation between two bit-vectors. We represent all BDD operations in italics to differentiate them from normal logic operations. More details about BDD operations can be found at <http://javaddlib.sourceforge.net/jbdd/>.

Chapter 5

Aggregate Licenses Validation for Digital Rights Violation Detection

In this chapter, we discuss about the problem of aggregate licenses validation for the distributors' violation detection. For this purpose, we first define the problem of efficient aggregate validation. Aggregate validation requires the validation using exponential number of validation inequalities and each inequality may contain up to an exponential number of summation terms. These factors make the validation process computationally intensive. In this chapter, we propose data organization and graph theory based algorithms for efficient aggregate validation. We also analyze the proposed algorithms theoretically and experimentally and compare their performance against the existing approaches. The proposed algorithms significantly improve the performance of the existing approaches.

5.1 Problem Definition

The problem of aggregate validation is to determine whether aggregate constraints in the redistribution licenses are not violated. For example, if a redistribution license for a content is allowed to generate 10,000 aggregate constraint counts then the sum of aggregate constraint's count in further issued licenses using the redistribution license must be less than 10,000. The problem of aggregate validation remains simple if there is only one redistribution license for a content. However, as discussed in Chapter 4,

a distributor would require multiple redistribution licenses for a content in practical scenarios. The problem of the aggregate validation becomes complex when there exist multiple redistribution licenses for a content

In the case of multiple licenses, as shown in Example 5.1, if all the instance-based constraint ranges in one redistribution license overlap with the respective constraints in other redistribution licenses then the licenses are said to be overlapping.

Example 5.1 *Consider the redistribution licenses L_{D_1} , L_{D_2} , L_{D_3} acquired by a distributor to distribute a content K with play permission and two instance-based constraints (validity period of redistribution licenses T , and region allowed for redistribution R) and one aggregate constraint A in them.*

$L_{D_1} = \langle K; \text{Play}; I_{D_1} : T = [10/03/09, 20/03/09], R = [\text{Asia}, \text{Africa}]; A_{D_1} = 2000 \rangle$.

$L_{D_2} = \langle K; \text{Play}; I_{D_2} : T = [15/03/09, 25/03/09], R = [\text{Asia}]; A_{D_2} = 1000 \rangle$.

$L_{D_3} = \langle K; \text{Play}; I_{D_3} : T = [15/03/09, 30/03/09], R = [\text{Africa}]; A_{D_3} = 3000 \rangle$.

Among the redistribution licenses shown above L_{D_1} and L_{D_2} , and L_{D_1} and L_{D_3} are the overlapping pairs of licenses as for both the pairs both the instance-based constraint (T and R) ranges have an overlapping interval. L_{D_1} and L_{D_2} are overlapping for $T=[15/03/09, 20/03/09]$ and $R = [\text{Asia}]$, and L_{D_1} and L_{D_3} are overlapping for $T=[15/03/09, 20/03/09]$ and $R = [\text{Africa}]$. Similarly, if there are N redistribution licenses for a content, then there can be multiple combinations of the overlapping licenses.

In case of multiple overlapping redistribution licenses, a newly generated license can be instance-based validated using more than one redistribution licenses. For aggregate validation we need to select one redistribution license out of multiple redistribution licenses using which the new license can be instance-based validated.

However, selecting a redistribution license randomly out of the multiple redistribution licenses may cause potential loss to the distributors as illustrated in Example 5.2.

Example 5.2 *Let the distributor possessing the three redistribution licenses L_{D_1} , L_{D_2} , L_{D_3} in example 5.1, generates the following license.*

$$L_{U_1} = \langle K; \text{Play}; I_{U_1} : T = [15/03/09, 19/03/09], R = [\text{Asia}]; A_{U_1} = 800 \rangle.$$

The license L_{U_1} can be instance-based validated using two redistribution licenses viz. L_{D_1} and L_{D_2} . Now, let the validation authority randomly picks L_{D_2} for the aggregate validation purpose then the remaining counts for play permission in L_{D_2} will become 200 (1000-800) after the validation. Suppose the distributor generates another license L_{U_2} :

$$L_{U_2} = \langle K; \text{Play}; I_{U_2} : T = [21/03/09, 24/03/09], R = [\text{Asia}]; A_{U_2} = 400 \rangle.$$

The license L_{U_2} can only be instance-based validated using the redistribution license L_{D_2} . The validation authority will now consider this redistribution license as invalid because the redistribution license L_{D_2} cannot be used to generate more than remaining 200 counts after the validation of L_{U_1} . In this case, a better solution would be to validate L_{U_1} using L_{D_1} , and L_{U_2} using L_{D_2} . This solution results in both L_{U_1} and L_{U_2} as valid based on the aggregate constraints. Thus, the challenge is to do the aggregate validation such that the distributors can use their redistribution licenses in an intelligent way.

5.1.1 A Method for Aggregate Validation

In order to do the aggregate validation in a better way, in this section, we discuss a method that uses the sets of received redistribution licenses for doing the aggregate validation. For this purpose, we first define the symbols used in this Chapter in Table 5.1.

An issued license is said to belong to a set S of redistribution licenses if it can be instance-based validated using all the licenses in the set S . For example, in Example

Table 5.1: List of symbols

Symbol	Explanation
S^N	The set of all N redistribution licenses present with distributor for a content i.e. $S^N = \{L_{D_1}, L_{D_2}, \dots, L_{D_N}\}$.
$SB^r[S]$	The r^{th} subset of the set S of redistribution licenses. Thus if set S contains k redistribution licenses then $r \leq 2^k - 1$, and $k \leq N$.
$E^i[S]$	i^{th} redistribution license in the set S of redistribution licenses.
$C[S]$	Sum of the aggregate constraint counts in all previously issued licenses which belong to the set S of redistribution licenses.
$A(x)$	Aggregate constraint count in a redistribution license x .
$A[S]$	Sum of aggregate constraint's counts in licenses in the set S .
$C\langle S \rangle$	LHS of the validation inequality for the set S i.e. summation of the counts corresponding to all subsets of S .
$C\langle [L_{D_{k_p}}] S \rangle$	Summation of counts for all the sets made by union of the set $[L_{D_{k_p}}]$ and all the subsets of S including $\emptyset$.

Table 5.2: Issued licenses

Issued Licenses	Set (S)	Set Counts (C)
L_{U_1}	$\{L_{D_1}, L_{D_2}\}$	800
L_{U_2}	$\{L_{D_2}\}$	400
L_{U_3}	$\{L_{D_1}, L_{D_2}\}$	30
L_{U_4}	$\{L_{D_1}, L_{D_3}\}$	30
L_{U_5}	$\{L_{D_2}\}$	800
L_{U_6}	$\{L_{D_1}, L_{D_3}\}$	20

5.2, the issued licenses L_{U_1} can be instance-based validated using the redistribution licenses L_{D_1} , and L_{D_2} . Thus, the set of redistribution licenses to which L_{U_1} belongs is $\{L_{D_1}, L_{D_2}\}$.

$C[S]$ denotes the sum of aggregate constraint counts in all the previously issued licenses that belongs to the set S of redistribution licenses. For example, in Table 5.2 let the licenses be issued using L_{D_1} , L_{D_2} and L_{D_3} . After L_{U_6} being issued, the value of $C[\{L_{D_1}, L_{D_2}\}]$, $C[\{L_{D_2}\}]$, and $C[\{L_{D_1}, L_{D_3}\}]$ will be 830, 1200, and 50 respectively.

Using the definition of $C[S]$ and symbols ($SB^r[S]$, $E^i[S]$, and $A[S]$) defined in Table 5.1, we derive the validation inequalities to do the validation in a better way. For deriving the first inequality, we use the fact that the sum of the aggregate constraint counts in all previously issued licenses must not exceed the sum of the aggregate constraint counts in all redistribution licenses. Further, each valid issued license

belongs to one and only one set of redistribution licenses out of the total $2^N - 1$ possible sets (due to N redistribution licenses). In an inequality form we can write:

$$\sum_{r=1}^{2^N-1} C[SB^r[S^N]] \leq \sum_{i=1}^N A(E^i[S^N]). \quad (5.1)$$

The LHS of the inequality in Equation 5.1 represents the sum of counts in the issued licenses that belongs to the set formed by any possible combination of all N redistribution licenses. Each possible combination can be represented by a subset of S^N (see Table 5.1). The RHS denotes the summation of the maximum allowed aggregate constraint counts in the licenses that are present in S^N . Since, S^N contains all N redistribution licenses therefore the RHS denotes the sum of aggregate constraint counts in all N licenses. In case of redistribution licenses used in example 5.1, the sum will be 6000.

Although, the inequality in Equation 5.1 can limit the counts issued in total using all the redistribution licenses but it cannot prevent the violation of individual licenses or set of licenses, which are proper subsets of the set S^N , as shown in Example 5.3.

Example 5.3 Consider the redistribution licenses L_{D_1} , L_{D_2} , and L_{D_3} as in example 5.1. The above inequality ensures only the sum of all the play counts in the licenses issued must be less than 6000 i.e. $C[\{L_{D_1}\}] + C[\{L_{D_2}\}] + C[\{L_{D_3}\}] + C[\{L_{D_1}, L_{D_2}\}] + C[\{L_{D_1}, L_{D_3}\}] + C[\{L_{D_2}, L_{D_3}\}] + C[\{L_{D_1}, L_{D_2}, L_{D_3}\}] \leq 6000$. But this inequality may not be able to prevent the violation due to issuing of excess counts for any individual received redistribution licenses or combination of received redistribution licenses. In this case the inequality in Equation 5.1 can be satisfied even if the counts generated for play permission using only the license L_{D_1} becomes more than 2000 i.e. $C[\{L_{D_1}\}] > 2000$, or counts using only the licenses L_{D_1} , and L_{D_3} becomes more than 5000 i.e. $C[\{L_{D_1}\}] + C[\{L_{D_3}\}] + C[\{L_{D_1}, L_{D_3}\}] > 5000$, but both these conditions are invalid. In Theorem 5.1, we formally prove that all these conditions are invalid and indicate violations.

Theorem 5.1 *Consider a set S ($S \subset S^N$) of redistribution licenses containing m ($1 \leq m < N$) redistribution licenses. The following condition on S in Equation 5.2 indicates a violation.*

$$\sum_{l=1}^{2^m-1} C[SB^l[S]] > \sum_{i=1}^m A(E^i[S]). \quad (5.2)$$

Proof: The LHS of the inequality in Equation 5.2 represents the sum of counts in the issued licenses that belong to the set formed by any possible combination of m redistribution licenses in the set S i.e. any subset of the set S .

Consider a subset S_1 of S . All the issued licenses that belong to S_1 can be considered to be issued using a redistribution license in set S_1 only. For example, the license L_{U_1} in example 5.2 belongs to the set $\{L_{D_1}, L_{D_2}\}$ so L_{U_1} can be considered to be issued using either L_{D_1} or L_{D_2} only (but not using L_{D_3}). Since S_1 is a subset of S so it may be inferred that all the issued licenses that belong to S_1 can be considered to be issued using a redistribution license that is in the set S . Generalizing it over all possible sets S_1 , all the issued licenses that belong to any subset of the set S can be issued using a redistribution license in the set S only. This implies that the LHS of the inequality in Equation 5.2 (which represents the sum of counts belonging to any subset of the set S) must not be more than the maximum possible counts that can be issued using the redistribution licenses in the set S .

But the maximum possible counts that can be issued using only the redistribution licenses in the set S is equal to the sum of aggregate constraints counts in the redistribution licenses in the set S , which is equal to the RHS of the inequality in Equation 5.2. So, if the LHS becomes greater than the RHS then it will correspond to a violation. ■

Since the set S in Theorem 5.1 can be formed in $2^N - 2$ ways (excluding the null set) so the violation can happen in $2^N - 2$ ways (i.e. all the proper subsets of the set S^N may cause violation). So, we require an inequality (in addition to inequality in

Equation 5.1) for every possible proper subset of the set S^N . For the r^{th} subset of the set S^N , $SB^r[S^N]$, the validation inequality can be given as:

$$\sum_{l=1}^{2^m-1} C[SB^l[SB^r[S^N]]] \leq \sum_{i=1}^m A(E^i[SB^r[S^N]]),$$

where $m = |SB^r[S^N]|$, and $1 \leq m \leq N$. (5.3)

The inequality in Equation 5.3 is of the similar form to the inequality in Equation 5.1 and can be obtained by replacing S^N by $SB^r[S^N]$. In Equation 5.3, m is the cardinality of $SB^r[S^N]$ or number of redistribution licenses in the set $SB^r[S^N]$. The LHS of the inequality in Equation 5.3 represents the sum of the counts in the issued licenses that belong to any of the subset of $SB^r[S^N]$, and the RHS represents the sum of the aggregate constraint counts in all the redistribution licenses in $SB^r[S^N]$. Similar inequalities for all the values of r ($1 \leq r \leq 2^N - 2$) ensure that in case of any violation at least one inequality will not be satisfied.

5.1.1.1 Dynamic Update of Validation Inequalities

A distributor can acquire a redistribution license any time. Thus, the validation inequalities are also need to be updated.

We assume that N received redistribution licenses for a content were already present and $(N + 1)^{th}$ redistribution license, $L_{D_{r_{N+1}}}$, is just acquired. To derive the new inequalities, we define the following new sets that can be created using the subsets of the set S^N and $(N + 1)^{th}$ received redistribution license.

$$S^r = SB^r[S^N] \bigcup [L_{D_{r_{N+1}}}] \forall r \leq 2^N - 1. \quad (5.4)$$

The new inequality for each value of $r \leq 2^N - 1$ can be written as:

$$\sum_{l=1}^{2^k-1} C[SB^l[S^r]] \leq \sum_{i=1}^k A(E^i[S^r]).$$

where, $k = |S^r|$, $1 \leq k \leq N + 1$. (5.5)

In addition to the above inequalities, one inequality for the set containing the single license $L_{D_{r_{N+1}}}$ will be formed.

$$C[L_{D_{r_{N+1}}}] \leq A(L_{D_{r_{N+1}}}). \quad (5.6)$$

So, the total number of inequalities will be $2^N - 1 + 2^N - 1 + 1 = 2^{N+1} - 1$ including the previous $2^N - 1$ inequalities (Equations 5.1 and 5.3), which is the correct number of inequalities for $N + 1$ redistribution licenses.

5.1.2 Requirement of Efficient Aggregate Validation

If N number of redistribution licenses are with a distributor then there are $2^N - 2$ proper subsets for the set of all N redistribution licenses. Hence, inequality in Equation 5.3 needs to be applied for all $2^N - 2$ subsets. Thus, in total $2^N - 1$ validation inequalities are possible (including Equation 5.1). Let a newly generated license can be instance-based validated using k number of redistribution licenses out of the total N redistribution licenses. Then, in this case the newly generated license belongs to a set of k redistribution licenses. The set formed due to k redistribution licenses will be a subset of $2^{(N-k)}$ sets thus it will be present in $2^{(N-k)}$ validation inequalities. So, if a newly issued license can be instance-based validated using any k redistribution licenses out of the total N redistribution licenses, we need to do the validation using $2^{(N-k)}$ validation inequalities. Validation using such a large number of validation inequalities every time a new license is issued is computationally intensive process. Also, the violation of aggregate constraints is not a frequent event as the received redistribution licenses generally have sufficient aggregate constraint counts to generate thousands of licenses. So, instead of doing the validation every time a new license is issued, we collect the logs of the sets of redistribution licenses (to which the issued licenses belong) and aggregate constraint counts in the issued licenses. And do the aggregate validation periodically (may be once everyday) using the logs collected. We refer each entry corresponding to an issued license as a *record* further in this chapter.

During the offline aggregate validation the validation process, in the first step different *set counts* can be aggregated and in the second step the aggregated *set counts* can be applied to the validation inequalities. In the worst case, there are $2^N - 1$ sets of redistribution licenses possible for N redistribution licenses. If there are M number of records then it would take $O(M * 2^N)$ time for the first step. A set of redistribution licence containing k redistribution licenses can be selected in $\binom{N}{k}$ ways therefore there will be $\binom{N}{k}$ validation inequalities for sets containing k redistribution licenses. Each such validation inequality contains $2^k - 1$ summation terms. So, total $\sum_{k=1}^N \binom{N}{k} (2^k - 1) \approx 3^N$ summations are needed for the second step. The time complexities for *set counts* these steps may be quite high for the practical purposes. Thus, an efficient method is required to reduce the total validation time.

5.1.3 Our Contributions

For efficient aggregate validation, we made three different contributions. Our first contribution is, design of an efficient data organization mechanism. In the proposed mechanism, we organize the log data containing log records in a prefix tree based structure called validation tree. It helps to do the efficient validation using the validation inequalities. Our second contribution is, the derivation of a relationship between different validation inequalities and a two-dimensional array based data structure design for more efficient validation using the derived relationship. The proposed algorithm achieves better validation time performance as compared to the validation tree based approach. Our third contribution is, design of a graph theory based algorithm to improve the performance of validation tree and two-dimensional array based data structure. The proposed algorithm uses graph-theoretic concepts to determine the disconnected groups of redistribution licenses, which in turn helps to improve the validation time by identifying and removing redundant validation inequalities.

5.2 Related Work

The problem in our case is to aggregate the *set counts* and use the aggregated *set counts* in aggregate validation inequalities (refer to Equations 5.1 and 5.3 in Section 5.1). The term *set count* refers to the count associated with a set in a given database containing sets and their counts. For example, consider the row corresponding to the license L_{U_1} in Table 5.2, the set corresponding to L_{U_1} is $\{L_{D_1}, L_{D_2}\}$ and the count for the set $\{L_{D_1}, L_{D_2}\}$ is 800. There are a number of methods to aggregate the *set counts* especially in the area of frequent pattern mining [103, 130, 154]. The most popular algorithms to aggregate the *set counts* are *Apriori* and *FP-Tree* algorithms.

Apriori algorithm was proposed by Agrawal et al. in [130]. Various improvements over the originally proposed Apriori algorithms are discussed by many authors [50, 58, 88]. The purpose of Apriori algorithm and its variants is to find all the sets whose count is above a certain threshold. They use the anti-monotone property of the *set counts* i.e. if a set $\{a\}$ is not frequent then $\{a, b\}$ cannot be frequent. This property is not applicable in our case as the counts for different sets are aggregated independently in our case in contrast to frequent pattern mining case where counts for a set are also aggregated towards all its subsets. Furthermore, on the basis of threshold, sets cannot be removed in our case. The aggregate validation process requires validation using all the non zero *set counts*.

Problem with Apriori based algorithms is that they require repetitive scan of transaction records to compute the count values for different length itemsets. FP-tree [52, 88] removes this drawback by using a prefix tree based data structure. However, both the algorithms aggregate the counts for different itemsets after removing non-frequent itemsets, which is not applicable in our case. The problem in our case is much different than frequent pattern mining problem. Firstly, it is because all *set counts* are independent so the anti-monotone property that is used in frequent pattern mining problems cannot be used. Secondly, unlike the frequent pattern mining problem in which only the sets with frequent items are need to counted, we need to aggregate

the counts for all the sets. Thirdly, to do the validation using aggregate validation inequalities, in addition to obtaining the *set counts*, we also need to apply them in the validation inequalities. Thus, a different process than the frequent pattern mining needs to be applied.

5.3 Validation Tree for Efficient Aggregate Validation

In this section, we present our proposed validation tree based data organization and license validation mechanism, and mathematical analysis of the validation tree.

5.3.1 Validation Tree Based Data Organization

To do the validation efficiently, we observe that the LHS of each validation inequality (Equations 5.1 and 5.3) for a set S aggregates the *set counts* of all the sets that are subset of S . Thus, if we can efficiently aggregate the *set counts* of all the subsets of a set by scanning the log records then instead of two steps discussed in Section 5.1.2, the aggregate validation can be done in a single step. However, scanning all the log records for finding the subsets of each set is costly operation. The efficiency can be improved if we can represent the log records such that the subsets of each set of redistribution licenses can be efficiently found. We propose validation tree, a prefix tree based structure, for this purpose. It helps to find subsets efficiently in two ways. First, it helps to represent the records in very compact manner. In the Validation Tree each set of redistribution licenses need to be stored only once even if it is present multiple times in the logs. Second, as we discuss in Sections 5.3.1.2 and 5.3.1.3, the properties of sets can be used to do the validation efficiently and selectively prune the unnecessary nodes.

5.3.1.1 Generation of Validation Tree

Initially, as shown in Figure 5.1, a *root* node is created. The tree is then expanded by using the log records (or the collection of individual record for each issued license

in the logs generated during the online instance-based validation). Each node in the validation tree stores the following fields: name of a redistribution license (L), a count value (C), and links to the child nodes. The count value C determines the count associated with the set formed by the redistribution license in the node and all its prefix nodes (nodes in the path from the *root* node to the current node) in the same branch. Links to the child nodes connect a node to its child nodes. If there is no child node of a node then there will be a single link and it will point to null. We use the convention that the redistribution licenses are indexed in the order they were acquired by a distributor i.e. if L_{D_j} is acquired before L_{D_k} then $j < k$. Child nodes of a node are ordered in the increasing order of their indexes. Without loss of generality⁶, we also assume that a redistribution license can act as a prefix only to the redistribution licenses having an index greater than the index of redistribution license i.e. if L_{D_j} can act as prefix to L_{D_k} if $j < k$, as shown in Figure 5.1.

As discussed previously, the records are in the form of set of redistribution licenses and a count value associated with the record. The redistribution licenses in each set are sorted in increasing order according to their indexes. Let the set of redistribution licenses in the record that needs to be inserted be given by: $R=[r, R']$ and the count value in the record be given by *count*, where r is the first redistribution license and R' be the set of remaining redistribution licenses. Let the *root* node (with $L=0$, and $C=0$ as fields in it) be described initially as T . Algorithm *Insert*($T, R, count$) is used to insert the record R .

In the algorithm, initially the *root* node is set as the *current* node. The algorithm reads the redistribution licenses in the *current* record one by one. Let the currently read redistribution license be given by r . If a node containing the *current* redistribution license r is found among the child nodes of the current node then that node is set as the *current* node. If no such node is found then a child node to the *current* node

⁶One may choose any possible ordering on prefixes as the validation process (using Equations 5.1 and 5.3) only depends on the complete set of redistribution licenses but not on the ordering of the redistribution licenses in the set.

Algorithm 1: $Insert(T, R, count)$ **Input:** $T, R=[r, R'], count$ **Output:** Expanded prefix tree based structure

- (1) Sequentially traverse the child nodes of T until all child nodes are traversed or we find a child T' such that $T.L \geq r$.
/* Note: The symbols $T.L$ and $T.C$ refer to index of the redistribution license (L) in the node T and count value (C) stored in the node T respectively. */
- (2) If $T'.L=r$ then go to step 4.
- (3) Else add a node T' such that $T'.L=r$ and $T'.C=0$ as the child node of T (in the order).
- (4) If $R'=\text{null}$ set then $T'.C = T'.C + count$.
- (5) Else, call $Insert(T', R', count)$.

Table 5.3: Table of license records

Serial Number	Set of redistribution licenses	Count
1	$\{L_{D_1}\}$	8
2	$\{L_{D_1}, L_{D_2}\}$	4
3	$\{L_{D_1}, L_{D_2}, L_{D_4}\}$	14
4	$\{L_{D_5}\}$	13
5	$\{L_{D_3}, L_{D_5}\}$	9
6	$\{L_{D_1}, L_{D_2}\}$	7
7	$\{L_{D_1}, L_{D_4}\}$	12
8	$\{L_{D_2}, L_{D_4}\}$	16

containing the redistribution license r is inserted and the node is set as the *current* node. The process continues until all the redistribution licenses in the *current* record are finished. When all the redistribution licenses in the record are finished then the count C corresponding to the *current* node is updated by adding the *count* in the record to it. Figure 5.1 illustrates the validation tree designed based on the Algorithm 1 for the log records in Table 5.3.

5.3.1.2 Validation Using Validation Tree

To do the validation efficiently, we first use the fact that if a set S_1 is not a subset of another set S then any superset S_2 of S_1 also cannot be a subset of the set S .

Figure 5.1: Generation of validation tree

Thus, in a prefix tree based structure, if the set of redistribution licenses formed by the redistribution license at a node N_1 and all its prefix nodes is not a subset of a set S then any other node having the node N_1 as a prefix node cannot be a subset of the set S . Thus, we need not travel the child nodes of the node N_1 . Another fact we use in the algorithm that a set containing n redistribution licenses cannot be a subset of any other set containing less than n redistribution licenses. Thus, if we keep track of the number of redistribution licenses for a set found already (as done with the variable *licNumber* in Algorithms 2 and 3) then many traversals, especially for the validation inequalities for the sets containing small number of redistribution licenses, can be eliminated.

As discussed in Section 5.1, for the validation purpose, we need a validation inequality for every possible set of redistribution licenses. If N redistribution licenses are present then we can map each set corresponding to a validation inequality into a bit-vector containing N bits, where each bit in a bit-vector corresponds to a particular redistribution license. A bit from LSB to MSB corresponds to a redistribution license with index from lower to higher. If a redistribution license is present in a set then the bit corresponding to it is 1 else it is 0. For example, let there be 10 redistribution licenses then the bit-vector for the set $\{L_{D_1}, L_{D_3}, L_{D_9}, L_{D_{10}}\}$ is represented using the

bit-vector 1100000101. So, if we consider the bit-vectors in the integer format then all the sets can be represented using the values from 1 to $2^N - 1$. Thus, every integer between 1 to $2^N - 1$ can be used to represent a specific validation inequality.

The validation tree is used to compute the LHS of the inequalities in Equations 5.1 and 5.3. Whereas, the RHS (let A for the present validation inequality) is calculated directly from the redistribution licenses. Let us assume the example in the previous paragraph, the value of A for the validation inequality for the set $\{L_{D_1}, L_{D_3}, L_{D_9}, L_{D_{10}}\}$ be given by the sum of the aggregate constraint's counts in the redistribution licenses L_{D_1} , L_{D_3} , L_{D_9} , and $L_{D_{10}}$. In the equation form, $A=A(L_{D_1})+A(L_{D_3})+A(L_{D_9})+A(L_{D_{10}})$.

Let us assume T represents the *root* node initially, and B represents the bit-vector equivalence of the current set of redistribution licenses considered for the validation inequality. The algorithm $Validation(T)$ evokes the validation process and does the validation using all possible validation inequalities. In the first step, it calculates the RHS of each validation inequality (using the process in previous paragraph). In the second step, it calls Algorithm 3, $VaLHS(T, B, licNumber)$, to calculate the LHS of the same validation inequality. Finally, the algorithm compares the RHS and LHS to determine if the inequality is valid or not.

Algorithm $VaLHS(T, B, licNumber)$ traverse the validation tree for the set of redistribution license determined by the bit-vector B . The parameter $licNumber$ is the number of redistribution licenses in the set corresponding to the current validation inequality. This parameter is used to use the fact that a set containing n number of redistribution licenses cannot be a subset of another set with less than n redistribution licenses. For illustration, consider the validation of the validation inequality for the set $\{L_{D_1}, L_{D_2}, L_{D_3}\}$ for the validation tree shown in Figure 5.1. The bit vector for this set (considering 5 redistribution licenses are present) will be 00111. Initially, the value of $licNumber$ will be 3. Let C_v denotes the LHS of the current validation inequality for the set $\{L_{D_1}, L_{D_2}, L_{D_3}\}$, it is initialized to 0 for every validation inequality.

Algorithm 2: Validation(T)

 Temporary Variables: $A_v=0$, $C_v=0$
for $i=1$ to $2^N - 1$ **do** $licNumber=0$. **for** $j=1$ to N **do** **if** $(1 \ll (j-1) \text{ AND } i) \neq 0$ **then**

/* Note: $\ll$ is logical left shift operator and left
 shift operation on 1 by $j-1$ results in a bit-vector
 with the j^{th} bit equal to 1 and rest of the bits equal
 to 0.

*/

 $A_v = A_v + A(j)$. $licNumber = licNumber + 1$; Call $C_v = ValHS(T, i, licNumber)$. **if** $C_v \leq A_v$ **then**

| Declare(Valid Inequality).

else

| Declare(Invalid Inequality).

Initially, the traversal is started from the *root* node. During the traversal, we add the *counts* in the nodes $root \rightarrow L_{D_1}$, $root \rightarrow L_{D_2}$, $root \rightarrow L_{D_3}$ to C_v . Next, we traverse the nodes $root \rightarrow L_{D_1}$, and $root \rightarrow L_{D_2}$. Note that, although, $\{L_{D_3}\}$ is a subset of $\{L_{D_1}, L_{D_2}, L_{D_3}\}$ but the algorithm will not traverse the node $root \rightarrow L_{D_3}$. This is because the value of *licNumber* becomes 0 after three subsets of the set $[L_{D_1}, L_{D_2}, L_{D_3}]$ are found as child nodes of the *root* node. All the child nodes of the node $root \rightarrow L_{D_3}$ will always have an index greater than 3 (index of L_{D_3}) and cannot be subset of the set $\{L_{D_1}, L_{D_2}, L_{D_3}\}$. During the traversal of the node $root \rightarrow L_{D_1}$, we add the *counts* in the node $root \rightarrow L_{D_1} \rightarrow L_{D_2}$ to C_v . Next, using the similar procedure the nodes $root \rightarrow L_{D_2}$ and $root \rightarrow L_{D_1} \rightarrow L_{D_2}$ are traversed. The final value of C_v for this case is 19.

5.3.1.3 Selective Pruning of Validation Tree

Selective pruning is the process of rearranging the validation inequalities in such a way that after the evaluation of some validation inequalities, a part of the validation tree can be temporarily pruned. This reduces the search space for the rest of the validation inequalities.

Algorithm 3: $ValHS(T, B, licNumber)$

Subroutine: $Process(T, B, licNumber)$ {
foreach *child of T (sequentially)* **do**
 if ($licNumber > 0$) **then**
 Let the current child be denoted by T' .
 Temporary variables: i and j .
 i =index of the redistribution license stored in the node T'
 j =a bit-vector formed by making i^{th} bit (from the LSB) as 1 and rest of
 bits as 0 i.e. $j=1 \ll (i-1)$.
 if ($B \text{ AND } j \neq 0$) **then**
 $C_v = C_v + T'.C$.
 $licNumber = licNumber - 1$.
 if ($licNumber > 0$) **then**
 Call $Process(T', B, licNumber)$.
 } **Return**(C_v).
 }

The pruning is based on the observation that for a validation inequality, we have to traverse the child nodes of a node sequentially irrespective of the redistribution licenses present in the validation inequality. For example, while evaluating the validation inequality for the set $\{L_{D_2}, L_{D_3}, L_{D_4}\}$ using the validation tree in Figure 5.1, the traversal is started from the node $root \rightarrow L_{D_1}$. But, neither $\{L_{D_1}\}$ nor the set formed by any node in the subtree followed by $root \rightarrow L_{D_1}$ can be a subset of the set $\{L_{D_2}, L_{D_3}, L_{D_4}\}$ (as set formed by any of the node in the subtree followed by $root \rightarrow L_{D_1}$ will always contain L_{D_1}). Thus, for this example, if we can avoid the traversal of $root \rightarrow L_{D_1}$ for all 2^{N-1} validation inequalities that do not contain L_{D_1} then a significant amount of processing can be saved.

To eliminate the unnecessary traversals, we rearrange the validation inequalities such that one by one child nodes of the $root$ node can be deleted after validation using some inequalities. To delete the node $root \rightarrow L_{D_i}$, we divide the validation inequalities in two categories. First, the validation inequalities whose set contain L_{D_i} and second the validation inequalities whose set doesn't contain L_{D_i} . For the validation inequalities whose set doesn't contain L_{D_i} , the node $root \rightarrow L_{D_i}$ and all the nodes in its subtree are redundant. Thus, if we first do the validation using the validation

inequalities whose set contain L_{D_i} then after validation using all such inequalities $root \rightarrow L_{D_i}$ (and hence its subtree) can be removed from the validation tree. The process is recursively applied for removing the subtree corresponding to each redistribution license.

Procedure of Selective Pruning: To employ the process of selective pruning efficiently, temporarily during the validation process, we modify the validation tree by replacing the name of the redistribution licenses in all the nodes (j by $N + 1 - j$). Let the modified redistribution licenses be represented by L' so $L'_{D_j} = L_{D_{N+1-j}}$. The validation tree shown in Figure 5.1 will be changed to the validation tree as in Figure 5.2(a). Algorithm 4 is then used in place of Algorithm 2.

Algorithm 4: Modified_Validation(T)

Temporary Variables: $A_v=0$, $C_v=0$, $k=N$.

```

while  $k > 0$  do
    for  $i=2^{(k-1)}$  to  $2^k - 1$  do
         $licNumber=0$ .
        for  $j=1$  to  $k$  do
            if  $(1 < (j-1) \text{ AND } i) \neq 0$  then
                 $A_v = A_v + A(N + 1 - j)$ .
                 $licNumber = licNumber + 1$ ;
            Call  $C_v = ValHS(T, i, licNumber)$ .
            if  $C_v \leq A_v$  then
                Declare(Valid Inequality).
            else
                Declare(Invalid Inequality).
        if root has a child  $T'$  such that  $T'.L' = k$  then
            Remove  $T'$  from the validation tree.
         $k=k-1$ .
    
```

In Algorithm 4, all $2^N - 1$ validation inequalities are divided recursively in to N parts using the parameter k . Initially, for $k=N$, validation inequalities corresponding to $i=2^{N-1}$ to $2^N - 1$ are selected. Note that the N^{th} bit in all these values of i is 1. Thus, the validation inequalities corresponding to all these values of i contain L'_{D_N} in their sets. It can also be observed that the bit-vectors corresponding to $i=1$ to

Figure 5.2: Pruning of validation tree

$2^{N-1} - 1$ have the N^{th} bit equal to 0. So, no value of i in the range $i=1$ to $2^{N-1} - 1$ corresponds to a set containing L'_{D_N} . Hence, the node $root \rightarrow L'_{D_N}$ can be safely removed for the validation inequalities corresponding to $i=1$ to $2^{N-1} - 1$ (after doing the validation with the validation inequalities corresponding to $i=2^{N-1}$ to $2^N - 1$). Similarly, remaining values of i can be divided into two parts $i=1$ to $2^{N-2} - 1$ and $i=2^{N-2}$ to $2^{N-1} - 1$ and the same process can be applied to remove the node $root \rightarrow L'_{D_{N-1}}$. The process is applied recursively until k becomes equal to 0. Figure 5.2(b) and 5.2(c) represent the validation tree after the removal of $root \rightarrow L'_{D_5}$ and $root \rightarrow L'_{D_4}$, respectively, from the validation tree in Figure 5.2(a).

5.3.2 Analysis of the Validation Algorithm

In this section, we theoretically analyze various aspects such as correctness, compactness, reasons for efficient performance and insertion time performance of our method. The experimental analysis is performed in Section 5.6.

5.3.2.1 Proof of Correctness

To prove the correctness of the proposed algorithm we need to prove that the validation algorithm terminates and all the validation inequalities are correctly evaluated

using the algorithm.

First, we prove that the validation algorithms (Algorithms 2 and 4) terminate after finite number of iterations. Algorithms 2 and 4 contain two *for* loops, both are finite, but in each iteration the external *for* loop calls algorithm 3. Thus, if Algorithm 3 terminates for each value of i in Algorithms 2 and 4 then we can say that the Algorithm 2 and 4 will terminate after finite number of iterations. Algorithm 3 is recursively applied for the child nodes of a node that satisfy a certain criteria based on *if* conditions. *if* conditions are applied separately for each child so for each child the process will terminate. Also, the size of the validation tree is finite given there can exist only a finite sets of redistribution licenses (hence nodes in the validation tree). Thus, the recursive process will finally terminate. This proves the first condition for correctness.

Now, we prove that all the inequalities are correctly validated using algorithms discussed. For this purpose, we consider a general set of received redistribution licenses S and validation inequality for it.

$$\sum_{l=1}^{2^m-1} C[T^l[S]] \leq \sum_{i=1}^m A(E^i[S]), \quad (5.7)$$

where $m = |[S]|$, and $1 \leq m \leq N$.

We need to separately prove that both LHS and RHS are evaluated correctly using the validation inequalities. LHS in the inequality does the summation of *set counts* in all the sets that belong to any of the subsets of the S . For LHS, first, we need to prove that the information is completely and correctly inserted in the validation tree. Second, we need to prove that the *set counts* for all the subsets of the set S are added during the traversal of validation tree. And finally the selective pruning process does not affect the algorithm. RHS does summation of aggregate constraint counts in all redistribution licenses in the set S . For RHS, we need to prove that the aggregate constraint counts are correctly added.

In validation tree, we insert all the sets of redistribution licenses, which have finite non zero *set count* values. Rest of the sets have the *set count* values equal to 0 thus their absence in the validation tree does not affect the validation process. So, the information inserted is complete for the validation purpose. During the insertion process, we insert a set of redistribution licenses in such a way that path in the validation tree is determined by the redistribution licenses in the set and count value is stored in the final node. Also, during the validation process, we assume that set of redistribution licenses at a node is determined by the redistribution license at the node and redistribution licenses in all its prefix nodes. This assumption is in accordance with the insertion process in the validation tree as during the insertion process count is inserted only in the final node. Thus, the count in each node will correspond to the set of redistribution licenses. Hence, the information inserted is correct for the validation purpose.

Now, we prove that *set counts* for all the subsets of a set are added during the validation process. The traversal of a node in the validation tree is decided by three *if* statements in Algorithm 3 (we refer them as outer, middle and inner *if* statements further). In the absence of all three *if* statements, all nodes in the validation tree are traversed recursively. However, it will lead to wrong result due to addition of *set counts* present in all nodes in the validation tree. The middle *if* statement takes care of this concern and only allows to add the *set count* when the redistribution license stored in the node is one in the set S (which is dictated by B). Further, it also terminates the traversals of child nodes of the nodes that do not contain one of the redistribution licenses in the set S . But, if the set formed by a node is not a subset of the set S then all the nodes having the node as prefix cannot be subset of S . So, the algorithm is correct if we only consider middle *if* statement. The inner and outer *if* statements are conditioned using the parameter *licNumber*. *licNumber* initially contains the value equal to the number of redistribution licenses in S . Value stored in *licNumber* is decremented recursively as a node containing a redistribution license

in S is found. Since, any subset of S cannot contain more redistribution licenses than in the set S . Thus, when *licNumber* number becomes 0 for a node then neither any of its child node nor any node at the same level followed by the node can be a subset of the set S . All such traversals can be terminated safely. Finally, for the LHS, we prove that the selective pruning process will not affect the result. The algorithm for validation after selective pruning (Algorithm 4) is same as that of Algorithm 2 except the parameter k is used instead of N . Now, the values of i are different for different values of k with minimum value $i=1$ for $k=1$ and maximum value $i=2^N - 1$ for $k=N$ and $\sum_{k=1}^N 2^{k-1} = 2^N - 1$. All three conditions together indicate that both the algorithms validate the same validation inequalities. Further, as already illustrated in Section 5.3.1.3, the selective pruning does not affect evaluation of any validation inequality. Thus, the process of selective pruning will not affect the correctness of the validation process.

For RHS, the aggregate constraint count for a redistribution license is added only when the redistribution license is in the set corresponding to bits equal to 1 in i (which is decided by left shift and AND operation in Algorithm 2). This is in accordance with the validation inequalities. Thus, both LHS and RHS are correctly evaluated using the proposed algorithm.

5.3.2.2 Compactness of Validation Tree

In this section, we show the compactness of our proposed validation tree over the data represented in the form of log records. First, considering the worst case when each record in the logs contains a different set. In validation tree, each record will contribute at most one path with the path length (excluding the *root* node) equal to the number of redistribution licenses in the record. Thus, the maximum number of nodes can never exceed the total number of redistribution licenses in all the sets of redistribution licenses in the logs. However, prefixes are often common in many records. So, a lot of paths are shared. For example, two records $\{L_{D_1}, L_{D_2}, \dots, L_{D_k}, \dots, L_{D_n}\}$ and

$\{L_{D_1}, L_{D_2}, \dots, L_{D_k}\}$ will share a common path $root \rightarrow L_{D_1} \rightarrow L_{D_2} \rightarrow \dots \rightarrow L_{D_k}$. Thus, the number of nodes in the validation tree would be much smaller than the total number of redistribution licenses in all the sets of redistribution licenses in the logs. Also, in practice, many records in the logs may contain the same set of redistribution licenses. It further improves the compactness of the validation tree. The compact representation helps in reducing both validation time and memory (storage space) requirements.

5.3.2.3 Efficient Validation Time Performance

Efficient validation time performance of the validation tree is mainly because of two factors. First, the compactness of representation of data in validation tree, as discussed in Section 5.3.2.2. Due to the compact representation of data, the search space for doing the validation is reduced. Second, use of an efficient subset search algorithm and pruning strategy in our algorithm, these help by eliminating the redundant traversals in the validation tree.

5.3.2.4 Insertion of Records

In this section, we show the complexity of records insertion in our proposed validation tree.

Theorem 5.2 *If the index of the last redistribution license in a record be k_r then at most k_r node traversals are sufficient to insert the record.*

Proof: Let there be r redistribution licenses in the record R : $R = \{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_r}}\}$. Then according to Algorithm 5, at most k_1 child nodes of the *root* node will be traversed to find either the node containing $L_{D_{k_1}}$ or a node having index greater than k_1 . If no node containing $L_{D_{k_1}}$ is found then a new node containing $L_{D_{k_1}}$ is created and rest of the redistribution licenses in the record can be inserted directly in the same branch. Else, if we find a node $L_{D_{k_1}}$ then to search for $L_{D_{k_2}}$ we need at most

k_2-k_1 traversals among the child nodes of $root \rightarrow L_{D_{k_1}}$. So, the maximum number of traversals required will be $k_1+k_2-k_1+\dots+k_r-k_{r-1}=k_r$. ■

Now, $k_r \leq N$ so if M records are present then the worst case insertion complexity will be $O(M*N)$. In practical scenario, the number of traversals required will be even lower as not all the nodes contain all the possible child nodes. So, in general, if we have to search the node containing $L_{D_{k_p}}$ among the child nodes of $L_{D_{k_{p-1}}}$ then we may not always need k_p-k_{p-1} comparisons. The insertion time is insignificant as compared to validation complexity of both validation directly using the original logs and our approach. Also, the insertion time performance of our method is much better than an alternative strategy called modified approach for efficient validation as we discuss using experimental results in Section 5.6.

5.4 Efficient Aggregate Validation By Deriving Relationship Between Validation Inequalities

In this section, we propose an efficient validation algorithm by deriving a relationship between different validation inequalities, and using the derived relationship as a basis to design a data structure for efficient validation.

5.4.1 Relationship Between Validation Inequalities

Given the log records in the form of Table 5.2, the task of validation is to determine whether the validation inequalities in Equations 5.1 and 5.3 are satisfied for S^N and $SB^r[S^N]$ (for all $2^N - 2$ possible values of r) respectively, where $SB^r[S^N]$ is the r^{th} subset of S^N .

Let us define the symbol $C\langle S \rangle$, $C\langle S \rangle$ represents the LHS of validation inequality in Equation 5.3 which is equal to summation of all the counts that are subset of the set S . Let a general set $S (\subseteq S^N)$ of redistribution licenses be $S = \{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_r}}\}$, where $1 \leq k_1 < \dots < k_r \leq N$. The validation inequality for the set S is given by

$$C\langle S \rangle \leq A[S] \text{ or } C\langle \{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_r}}\} \rangle \leq A[\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}].$$

Let us also define the symbol $C\langle [L_{D_{k_p}}] | S \rangle$, where $L_{D_{k_p}} \cap S = \emptyset$. The symbol $C\langle [L_{D_{k_p}}] | S \rangle$ represents the summation of counts for all the sets made by union of the set $[L_{D_{k_p}}]$ and all the subsets of S including $\emptyset$. For example, for the set $S = \{L_{D_1}, L_{D_5}\}$, all the subsets are $\{\emptyset\}$, $\{L_{D_1}\}$, $\{L_{D_5}\}$ and $\{L_{D_1}, L_{D_5}\}$. Thus, $C\langle \{L_{D_3}\} | S \rangle = C[\{L_{D_3}\}] + [C\{L_{D_1}, L_{D_3}\}] + C[\{L_{D_3}, L_{D_5}\}] + C[\{L_{D_1}, L_{D_3}, L_{D_5}\}]$. If a set S has n elements then $C\langle [L_{D_{k_p}}] | S \rangle$ contains 2^n summation terms.

Using the definition of $C\langle [L_{D_{k_p}}] | S \rangle$, we derive the following relationships between the LHS and RHS of the validation inequalities for the sets $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\}$ and $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$ in Theorem 5.3, where $1 \leq k_1 < \dots < k_{r-1} < k_r \leq N$.

Theorem 5.3 *The LHS and RHS of the validation inequality for a general set $S = \{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$ can be represented using the relationships as in Equations 5.8 and 5.9:*

$$\begin{aligned} LHS = C\langle \{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\} \rangle &= C\langle \{L_{D_{k_r}}\} | \{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\} \rangle + \\ &C\langle \{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\} \rangle \end{aligned} \quad (5.8)$$

$$RHS = A[\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}] = A[\{L_{D_{k_r}}\}] + A[\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\}] \quad (5.9)$$

Proof: The proof for Equation 5.8 is based on the fact that the summation terms in the LHS can be divided into two types of terms. First, the terms that must contain the redistribution license $L_{D_{k_r}}$ and second the terms that must not contain $L_{D_{k_r}}$. The first term in the RHS of Equation 5.8 does the summation of all

the terms that must contain $L_{D_{k_r}}$ and the second term does the summation of all the terms that must not contain $L_{D_{k_r}}$. To prove Equation 5.9, we can represent $A[\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}]$ as $A[\{L_{D_{k_r}}\}] + A[\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_{r-1}}}\}]$, which is equal to the RHS of Equation 5.9. ■

From Theorem 5.3, we can infer that if we have already evaluated the LHS and RHS of the validation inequality for set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\}$ then its LHS and RHS can be used to efficiently compute the LHS and RHS for the validation inequality for all the sets $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$ such that $k_{r-1} < k_r \leq N$.

Calculation of RHS of the validation inequality for the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$ is straightforward and it only requires the addition of $A[\{L_{D_{k_r}}\}]$ to the RHS of validation inequality for the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\}$. But to calculate the LHS, we need to evaluate the term $C\langle\{L_{D_{k_r}}\}|\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\}\rangle$, which contains 2^{r-1} terms (using the definition of $C\langle[L_{D_{k_p}}]|S\rangle$) out of the total $2^r - 1$ terms in $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$. However, properties 1 and 2 can be used to efficiently evaluate the summation terms in $C\langle\{L_{D_{k_r}}\}|\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\}\rangle$.

Property 1: The set of redistribution licenses in all these terms must contain $L_{D_{k_r}}$ and the index of all other redistribution licenses in each term in $C\langle\{L_{D_{k_r}}\}|\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\}\rangle$ always remains less than k_r .

Property 2: If a set contains n elements then any of its subsets (including itself) always contain less than or equal to n elements. Thus, any term in $C\langle\{L_{D_{k_r}}\}|\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\}\rangle$ does not contain more than r redistribution licenses including $L_{D_{k_r}}$.

Next, we propose a data structure so that the properties 1 and 2 can be used for efficient validation.

5.4.2 Proposed Data Structure

To discuss the proposed data structure, we first discuss bit-vector representation of log records, and bit-vector nodes. Then, we discuss the proposed two dimensional array based data structure, and insertion of log records in the proposed data structure.

5.4.2.1 Bit-vector Representation of Log Records

For efficient validation, we first pre-process the log records of issued licenses as in Table 5.3 to obtain the bit-vector representation. During the pre-processing, if there are total N redistribution licenses then we convert each set of redistribution licenses to an N bits bit-vector. Each bit in a bit-vector corresponds to a particular redistribution license (in increasing order from LSB to MSB). For example, for the log records in Table 5.3, let there be total 5 redistribution licenses then each bit-vector will consist of 5 bits. The bit-vector corresponding to the set $\{L_{D_1}, L_{D_3}\}$ will be 00101.

During the pre-processing step, we also compute the total number of redistribution licenses (*total*) and highest indexed redistribution license (*highest*) in the set corresponding to each log record in Table 5.3. For example, the values of *total* and *highest* for the set $\{L_{D_1}, L_{D_4}\}$ are 2 and 4 respectively. Table 5.4 shows the bit-vector representation, and values of *total* and *highest* for the log records in Table 5.3.

Table 5.4: Pre-processed log records

License	Bit-vector (B)	<i>Counts</i> (C)	<i>total</i>	<i>highest</i>
L_{U_1}	00001	8	1	1
L_{U_2}	00011	4	2	2
L_{U_3}	01011	14	3	4
L_{U_4}	10000	13	1	5
L_{U_5}	10100	9	2	5
L_{U_6}	00011	7	2	2
L_{U_7}	01001	12	2	4
L_{U_8}	01010	16	2	4

5.4.2.2 Bit-vector Nodes

Bit-vector nodes are the basic elements of the proposed data structure. As shown in Figure 5.3, each bit-vector node consists of three fields: a bit-vector (*bitVec*) consisting of N bits, a count value (*count*), and a pointer (*next*) pointing to other *Bit-vector node* (to the *null* initially). We describe the process of calculation of *bitVec*, *count*, and *next* in a *bit-vector node* during the process of insertion of the log records in the proposed data structure in Section 5.4.2.3.

Figure 5.3: Structure of a bit-vector node

5.4.2.3 Two-dimensional Array Based Data Structure

As shown in Figure 5.4, the proposed data structure is of the form of a two-dimensional array of pointers, arr of size $N \times N$, where each pointer points to a *bit-vector node* (to the *null* initially). The two dimensions in the proposed data structure corresponds to *highest* and *total* in Table 5.4. In the proposed data structure, we insert all the log records, which are in the form of Table 5.4, using the insertion process discussed next.

 Figure 5.4: Example data structure (arr) for $N=5$

5.4.2.3.1 Insertion of Log Records: For the insertion of log records, we first initialize a two-dimensional array arr consists of $N \times N$ array of pointers, with each pointer pointing to null. Algorithm 5 is used to insert the log records in the proposed data structure.

During the insertion process each log record in Table 5.4 is scanned one by one. Then, for each record, based on the values of *highest* and *total* for the log record, the respective position is searched in the array arr . For example, the log record

corresponding to L_{U_1} in Table 5.4 has *total* and *highest* values 1 and 1 respectively so we move to the position $arr[1][1]$ in the array arr .

Then, if the pointer at the searched position points to null then we create a new *bit-vector node* with the values *bitVec* and *count* same as the values of B and C respectively for the current log record in Table 5.4. For example, on the insertion of the log record corresponding to L_{U_1} in Table 5.4, a new *bit-vector node* with *bitVec*=00001 and *count*=8 is inserted at $arr[1][1]$ initially. Note that the *count* field in a *bit-vector node* can be changed on the insertion of other log records so the *count*=8 in the *bit-vector node* can be changed on the insertion of other log records.

Else if the pointer at the searched position does not point to null then we traverse *bit-vector node* pointed by the pointer at the element searched. If the bit-vector *bitVec* at the *bit-vector node* traversed is equal to bit-vector B of the current log record then the *count* in the *bit-vector node* is updated by adding the value C of the current log record to it. Else the *bit-vector node* pointed by the pointer *next* of the current *bit-vector node* is traversed. If the *bit-vector node* pointed by the current *bit-vector node* is not null then the same process is repeated for the *bit-vector node* pointed by the current *bit-vector node* (lines 4 to 10 in Algorithm 5). Else if the *bit-vector node* pointed by the current *bit-vector node* is null then a new *bit-vector node* with the *bitVec* and *count* same as the values of B and C respectively for the current log record is created (lines 11 to 14 in Algorithm 5), and inserted by updating the next pointer the current *bit-vector node* to the newly created *bit-vector node*.

Figure 5.4 represents the data structure obtained after the insertion of all the log records in Table 5.2. Note that the dark cells of arr in Figure 5.4 are redundant as the value of *total* for any log record cannot be more than the value of *highest*. These cells can be removed easily by using several one dimensional arrays of different sizes instead of a single $N \times N$ array. But for the ease of understanding and implementation we use two-dimensional $N \times N$ array. Using either of these does not affect the validation time and records insertion performance.

Algorithm 5: Insertion

Input: Log Records

```

1  foreach Log Record do
2      Variables:  $B, C$  highest, total for the log record.
3      Pointer:  $current = arr[highest][total]$ .
4      while  $current.next \neq null$  do
5          if  $current.next.bitVec = B$  then
6               $current.next.count = current.next.count + C$ .
7              Exit from while loop.
8          end
9           $current = current.next$ .
10     end
11     if  $current.next = null$  then
12         Create a new node,  $new$ , with  $new.bitVec = B$ ,  $new.count = C$  and
13          $new.next = null$ .
14          $current.next = new$ .
15     end
16 end
    
```

5.4.3 Efficient Validation Using Proposed Data Structure

The task of the validation is to determine whether all $2^N - 1$ validation inequalities in Section 5.1.1 are satisfied. As discussed in Section 5.4.1, the validation using each validation inequality (say for the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$) consists of two parts. The first part is to calculate the first term in the RHS of the Equation 5.8 using the data structure proposed in this section. The second part is to obtain the second term in the RHS of the Equation 5.8 from the previously validated validation inequality for the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\}$.

5.4.3.1 Calculation of the First Term

The first term in the RHS of the Equation 5.8 is $C\langle\{L_{D_{k_r}}\}|\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\}\rangle$. It represents summation of counts for all the sets such that the sets are subset of $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$ and contain $L_{D_{k_r}}$, and let the value of $C\langle\{L_{D_{k_r}}\}|\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\}\rangle$ before the traversal of data structure be 0.

According to property 1 in Section 5.4.1, all the terms in $C\langle\{L_{D_{k_r}}\}|\{L_{D_{k_1}}, \dots, L_{D_{k_{r-1}}}\}\rangle$ have the highest index equal to k_r . And according to property 2, any term in

$C(\{L_{D_{k_r}}\} | \{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\})$ does not contain more than r redistribution licenses. In our proposed data structure we have inserted the log records using the value of highest indexed redistribution license (*highest*), and total number of redistribution licenses (*total*). So, all the terms in $C(\{L_{D_{k_r}}\} | \{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\})$ must be present in the elements with *highest*= k_r , and *total*= 1 to r in array *arr*. Therefore, using the proposed data structure, instead of traversing all the records in Table 5.2 to calculate $C(\{L_{D_{k_r}}\} | \{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\})$, we only need to traverse the selective terms as determined by k_r and r .

To compute $C(\{L_{D_{k_r}}\} | \{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\})$, one by one, we traverse the linked lists of *bit-vector nodes* pointed by the elements with *highest*= k_r and *total*= 1 to r in array *arr*. Then for each *bit-vector node* traversed, we determine whether the set represented by the bit-vector *bitvec* is a subset of the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_r}}\}$. The procedure to determine whether the set represented by a bit-vector node is subset of the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_r}}\}$ is mentioned in Section 5.4.3.2.

If the set represented by a *bit-vector node* is a subset of $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$ then we update the value of $C(\{L_{D_{k_r}}\} | \{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\})$ by adding the value of the field *count* in the *bit-vector node* to it. After traversing all the linked list corresponding to *highest*= k_r , and *total*= 1 to r , we obtain the final value of $C(\{L_{D_{k_r}}\} | \{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\})$. Algorithm 6 shows the pseudo-code for the procedure of obtaining LHS and RHS of the validation inequality for the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$.

5.4.3.2 Subset Search Using Bit-vectors

Let the set corresponding to a validation inequality be $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$ and we need to determine whether the set represented by a bit-vector *bitvec* is a subset of the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$. In this section, we determine this using the bit-vector representation of the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$. For bit-vector

representation, we use the same convention in Section 5.4.2 described for the bit-vector representation of the sets in the log records. Let the bit-vector corresponding to the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$ be given by B .

Algorithm 6: *Traversal* $(B, lhs_prev, rhs_prev, r, k_r)$

Input:

B : Bit-vector corresponding to the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$.

lhs_prev : LHS of the validation inequality for the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$.

rhs_prev : RHS of the validation inequality for the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$.

r : Number of redistribution licenses in the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$.

k_r : Index of the highest indexed redistribution license in the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$.

Output:

lhs : LHS of the validation inequality for the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$.

rhs : RHS of the validation inequality for the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$.

Variable: A : array of size N containing aggregate constraint values in redistribution licenses (i^{th} element corresponds to i^{th} redistribution license).

$lhs = lhs_prev, rhs = rhs_prev + A[k_r]$.

for $i=1$ **to** $i \leq r$ **do**

Pointer: $current = arr[k_r][i].next$

while $current \neq \text{null}$ **do**

if $current.bitVec \text{ AND } B' = 0$ **then**

 /* AND operation determines whether set represented by
 $current.bitVec$ is a subset of the set represented by
 B (see Section 5.4.3.2). */

$lhs = lhs + current.count$.

end

$current = current.next$.

end

end

Return $\{lhs, rhs\}$.

The set corresponding to $bitvec$ will be a subset of $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$ if $B' \text{ AND } bitvec = 0$, where B' is complement of B . It is based on the fact that the set corresponding to $bitvec$ can only be a subset of $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$ if the set corresponding to $bitvec$ does not contain any elements that is not in the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$. In terms of bit-vectors, $bitvec$ must not have any bit equal to 1 at the positions for which bit is 0 in B . Thus, the AND operation between $bitvec$ and B' will only be 0 if the set corresponding to $bitvec$ is a subset of $\{L_{D_{k_1}},$

$$L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}.$$

5.4.3.3 Calculation of the Second Term

The second term in the RHS of the Equation 5.8 is $C\langle\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\}\rangle$. Since it comes from the validation inequality for the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$, so the validation inequality for the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$ must be evaluated before the validation inequality for the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$. Also, the condition must be true for all the pairs of sets $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\}$ and $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$.

We do the validation using the validation inequalities in the order as determined by Algorithm 7. In Algorithm 7, initially we start with the auxiliary validation inequality for the empty set ($\{\emptyset\}$). For the initial validation inequality, we assume that $C\langle\{\emptyset\}\rangle=0$ and $A\langle\{\emptyset\}\rangle=0$. Initially, we also assume that $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\}=\{\emptyset\}$. Then, one by one, we make a new set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$ by adding $L_{D_{k_r}}$ to the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\}$ for all the values of k_r : $k_{r-1} < k_r \leq N$; do the validation using validation inequality corresponding to the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$; and recursively follow the previous two steps for the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$ to generate the new sets by adding a redistribution licenses to the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}, L_{D_{k_r}}\}$. If there are total N redistribution licenses, then the recursive process generates all $2^N - 1$ sets of redistribution licenses that are subsets of the set S^N (see Table 5.1).

LHS and RHS for each validation inequality are calculated in Algorithm 7 by calling the Algorithm 6. Finally, for each validation inequality, we compare the LHS and RHS to determine whether it is valid or invalid.

5.4.4 Theoretical Performance Analysis

In this section, we analyze the theoretical performance of the proposed method in terms of insertion time complexity, validation time complexity, and memory requirements. For calculating these complexities, we assume that there are total R number

of log records present in Table 5.2 and these log records contain D number of distinct sets of redistribution licenses. For example, for Table 5.2, the value of R is 8, but the value of D is 7 as the sets $\{L_{D_1}, L_{D_2}\}$ is shared by two log records.

Algorithm 7: Validation

Parameters: $lhs, rhs=0$.

B : Bit-vector corresponding to the set $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\}$, initially $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\} = \{\emptyset\}$ so B contains all N bits 0 initially.

$total$: Number of redistribution licenses in $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\}$ i.e. $r-1$ with initial value=0.

$final$: index of the highest indexed redistribution license in $\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\}$ i.e. k_{r-1} with initial value=0.

Call *Set Generation*($B, lhs, rhs, total, final$).

Subroutine: *SetGeneration*($B, lhs, rhs, total, final$){

$total_new = total + 1$.

for $i = (final + 1)$ **to** $i = N$ **do**

 bit-vector: $B_new = B + 1 \ll (i-1)$.

 /* $1 \ll (i-1)$ denotes the left shift operation on integer 1 by $i-1$. */

$final_new = i$.

$\{lhs_new, rhs_new\} = Traversal(B_new, lhs, rhs, total_new, final_new)$.

if $lhs_new \leq rhs_new$ **then**

 Declare(Valid inequality).

else

 Declare(Invalid inequality).

end

if $i < N$ **then**

 Call *Set Generation*($B_new, lhs_new, rhs_new, total_new, final_new$)

end

end

}

5.4.4.1 Insertion Time Complexity

Insertion time required is equal to the time required to insert the log records in the data structure to be generated for efficient validation.

Insertion time complexity in the proposed data structure is contributed by two factors. First, the pre-processing of each log record to obtain the variables bit-vectors

(B), *total* and *highest*. Second, for each log record, traversing the linked list pointed by $arr[total, highest]$. In the worst case, the complexity would be of $O(N)$ for obtaining B , *total* and *highest*, and $O(L)$ for traversing the linked list, where L is the number of elements in the linked list pointed by *total* and *highest*. However, the average value of L (or average number of nodes per linked list) is practically small as compared to N . This is because the log records in the proposed method are divided into $(N)*(N+1)/2$ parts (see Figure 5.4). Thus, the average value of L is equal to $D/((N)*(N+1)/2)$, which is very small practically. For example, for $N=30$ we observe that all the log records contain about 500 distinct sets of redistribution licenses (so $D=500$). These log records are divided into $30*31/2=465$ parts so the value of L is approximately equal to 1. Thus, the complexity of insertion of each log record with the proposed data structure can be approximated to $O(N)$.

5.4.4.2 Validation Time Complexity

Validation time is the total time required to do the validation using all $2^N - 1$ validation inequalities. For each inequality, the validation task is the calculation of LHS and RHS according to equations 5.8 and 5.9 respectively.

For each validation inequality, the time complexity in calculating the RHS is $O(1)$ as in equation 5.9, we only need to compute $A[\{L_{D_{k_r}}\}]$ (rest of the terms come from a previously validated validation inequality). However, the time complexity in calculating the LHS depends on the time required to compute the term $C(\{L_{D_{k_r}}\}|\{L_{D_{k_1}}, L_{D_{k_2}}, \dots, L_{D_{k_{r-1}}}\})$ by traversing the proposed data structure.

The time required to traverse the data structure depends on the values of k_r and r for a validation inequality in Algorithm 6. The validation time complexity of validation for our proposed validation algorithm is given by the time required to traverse the linked lists determined by the parameters k_r and r . Thus, the time required for validating the validation inequality for a set containing r ($1 \leq r \leq N$) redistribution licenses is of $O(\sum_{i=1}^r L_{(k_r, i)})$, where $L_{(k_r, i)}$ is the number of elements in the linked list

corresponding to element $arr[k_r, i]$ in Figure 5.4. As discussed previously, the average value of L generally is very small so the validation time complexity can be approximated to $O(r)$. This is very significant gain practically as the validation inequality for a set containing r redistribution license contains a total of $2^r - 1$ summation terms, which is reduced to $O(r)$ under practical scenario (i.e. after assuming practically small value of L).

5.4.4.3 Storage Space Complexity

Storage space complexity is the storage space required to store array based data structure and linked lists of *bit-vector nodes* associated with it. The proposed data structure requires N^2 number of elements in array arr and total D number of *bit-vector nodes* in all the linked lists (as each distinct set is stored in different nodes). So, the storage space complexity is given by $O(N^2 + D)$.

5.5 A Graph Theory Based Approach to Remove Redundant Validation Inequalities

In this section, we present an approach using geometric representation of licenses to identify the redundant validation inequalities and a graph theory based method to remove the redundant validation inequalities for more efficient validation performance.

5.5.1 Geometric Representation of Licenses

If there are M number of instance-based constraints in the licenses then each redistribution /usage license can be represented as an M dimensional hyper-rectangle e.g. Figure 5.5 shows five redistribution licenses and two usage licenses represented in a 2-dimensional space. The set of redistribution licenses to which an issued license can be instance-based validated is given by the set of redistribution licenses whose hyper-rectangles completely contain the hyper-rectangle formed by the issued license. For example, in Figure 5.5, the hyper-rectangle formed by L_{U_1} is completely within

Figure 5.5: Representation of five redistribution licenses with two constraints

the hyper-rectangle formed by L_{D_4} only. Thus, the set of redistribution licenses to which L_{U_1} can be instance-based validated is given by $\{L_{D_4}\}$. However, L_{U_2} is not completely within any of the five redistribution licenses so it cannot be instance-based validated using any of the five redistribution licenses and it is treated as invalid.

5.5.2 Redundant Validation Inequalities

In this section, we first define the overlapping redistribution licenses and non-overlapping sets of redistribution licenses. Then with the help of Theorems 5.4 and 5.5, we present a method to identify the redundant validation inequalities.

Overlapping Redistribution Licenses. Two redistribution licenses, L_{D_j} and L_{D_k} , with instance-based constraints $\{I_{D_{j_1}}, I_{D_{j_2}}, \dots, I_{D_{j_M}}\}$ and $\{I_{D_{k_1}}, I_{D_{k_2}}, \dots, I_{D_{k_M}}\}$, are overlapping if $I_{D_{j_m}} \cap I_{D_{k_m}} \neq \emptyset, \forall m \leq M$, where M is the number of total instance-based constraints. Geometrically, two redistribution licenses L_{D_j} and L_{D_k} are overlapping if and only if all their constraint dimensions have an overlap. Thus, in Figure 5.5, licenses L_{D_1} and L_{D_2} are overlapping but L_{D_1} and L_{D_4} are non-overlapping.

Non-overlapping Sets. Two sets S_1 and S_2 of received redistribution licenses are said to be non-overlapping if S_1 and S_2 contain different licenses (i.e. $S_1 \cap S_2 = \emptyset$) and any of the constraint ranges in the license in the set S_1 does not overlap with any license in the set S_2 . For example, the sets $S_1 = \{L_{D_1}, L_{D_2}\}$ and $S_2 = \{L_{D_5}\}$ are non-overlapping in Figure 5.5 as neither L_{D_1} nor L_{D_2} (in set S_1) overlaps with L_{D_5} (in set S_2).

Theorem 5.4 *If all the redistribution licenses in a set S do not have a common overlapping region then count ($C[S]$) associated with that set will always be 0.*

Proof: If all the redistribution licenses in a set S do not have a common region then any valid issued license cannot be simultaneously within the hyper-rectangles formed by all the redistribution licenses in the set. Thus, the count value for the set will always be 0. For example, redistribution licenses L_{D_1} , L_{D_2} , and L_{D_3} in Figure 5.5 do not form a common region and hence the hyper-rectangle formed by any issued license cannot be inside the hyper-rectangles of L_{D_1} , L_{D_2} , and L_{D_3} simultaneously. So, $C[\{L_{D_1}, L_{D_2}, L_{D_3}\}]$ will always be 0. ■

Corollary 5.4.1. Set S of received redistribution licenses formed by taking at least one redistribution license from two or more non-overlapping sets of received redistribution licenses will always have *set count* equal to 0. This is because if we take at least one received redistribution licenses from two different non-overlapping sets of received redistribution licenses then a common region cannot be formed by all the received redistribution licenses in S .

Theorem 5.5 *If a set S containing n received redistribution licenses can be represented with m number of non-overlapping sets ($S_i \cap S_j = \emptyset$, $1 \leq i < j \leq m \leq n$) of received redistribution licenses such that $S_1 \cup S_2 \cup \dots \cup S_m = S$ then the validation inequality for the set S can be expressed as the sum of validation inequalities for the sets S_1 , S_2 , ... and S_m . Validation inequality for a set S_i is: $C\langle S_i \rangle \leq A[S_i]$.*

$$\begin{aligned} C\langle S \rangle &= C\langle S_1 \rangle + C\langle S_2 \rangle + C\langle S_3 \rangle + \dots + C\langle S_m \rangle \leq \\ A[S] &= A[S_1] + A[S_2] + A[S_3] + \dots + A[S_m] \end{aligned} \quad (5.10)$$

Thus, if the validation inequalities for the sets S_1 , S_2 , ... and S_m are already validated then we do not need to validate the validation inequality for the set S .

Proof: m number of non-overlapping sets of the received redistribution licenses are represented by $S_1, S_2, \dots, S_m$, where $2 \leq m \leq N$. Now consider a set formed by at least one received redistribution license each from m' ($2 \leq m' \leq m$) sets out of total m sets. As $S_i \cap S_j = \emptyset$ therefore using corollary 5.4.1, any such set formed will always have the *set count* value equal to 0. Thus, all the sets which are created by taking at least one received redistribution license each from any m' sets out of m sets will always have count value equal to 0. So, $C\langle S \rangle$ will be only due to the sets formed due to redistribution licenses within each of the m sets individually. Thus, $C\langle S \rangle$ can be written as: $C\langle S \rangle = C\langle S_1 \rangle + C\langle S_2 \rangle + C\langle S_3 \rangle + \dots + C\langle S_m \rangle$. Also, the sets $S_1, S_2, \dots$ and S_m are non-overlapping and $S_1 \cup S_2 \cup \dots \cup S_m = S$. Therefore, $A[S]$ can be written as: $A[S] = A[S_1] + A[S_2] + A[S_3] + \dots + A[S_m]$.

Thus, Equation 5.10 is correct and it can be obtained by summation of the validation inequalities for the individual sets $S_1, S_2, \dots$ and S_m . Hence if the validation inequalities for the sets $S_1, S_2, \dots$ and S_m are already evaluated then the validation inequality for the set S can be removed. ■

For example, the redistribution licenses in Figure 5.5 can be divided in two groups, with group 1 and group 2 containing the redistribution licenses $(L_{D_1}, L_{D_2}, L_{D_4})$ and (L_{D_3}, L_{D_5}) respectively. According to Theorem 5.5, if the validation inequalities for all subsets of the sets $\{L_{D_1}, L_{D_2}, L_{D_4}\}$ and $\{L_{D_3}, L_{D_5}\}$ are evaluated independently then we need not evaluate the validation inequalities for any set generated by taking at least one redistribution license from each group. So, the inequalities for the sets $\{L_{D_1}, L_{D_3}\}, \{L_{D_1}, L_{D_2}, L_{D_3}\}$, etc. need not be evaluated.

5.5.3 Identification of Disconnected Groups

To use the basis discussed in Section 5.5.2, we need to identify the disconnected groups of redistribution licenses. In this section, we present a graph theory based algorithm to find the disconnected groups of redistribution licenses.

We use a graph [62, 87] $G=(V, E)$ to identify the disconnected groups of redistribution licenses, where V is the set of vertices and E is the set of edges in G . As

Figure 5.6: Graph and adjacency matrix for redistribution licenses in figure 5.5

shown in Figure 5.6, each redistribution license is represented using a vertex in G . There is an edge between the vertices corresponding to the i^{th} and j^{th} redistribution licenses if they are overlapping. We represent the graph using an adjacency matrix Adj of size $N \times N$. The entry in the i^{th} row and j^{th} column, $Adj_{i,j}$, is 1 if the i^{th} and j^{th} redistribution licenses are overlapping else it is 0. Figure 5.6 shows the graph and matrix Adj for redistribution licenses in Figure 5.5.

We perform the depth first search (DFS) [62] in Algorithm 8 on G to identify the number of groups of redistribution licenses and redistribution licenses in each group. The DFS starts from the node corresponding to the first redistribution license. It finds all the nodes that are directly or indirectly (through other nodes) connected to the node corresponding to the first redistribution license. All such nodes form one group of redistribution licenses. Next, the algorithm sequentially searches for a redistribution license, which is not present in any previously formed groups. Then, the algorithm finds all the nodes that are directly or indirectly connected to the node corresponding to the searched redistribution license. The process continues until a group is allocated to every redistribution license. To perform DFS, two arrays viz. *Visited* and *Group* of size N and $N \times N$, respectively, are initialized with each element equal to 0 in Algorithm 8. Array *Visited* keeps the information about all the nodes traversed in the graph during the DFS. Array *Group* is modified by the algorithm and after the completion of algorithm it stores the information about the redistribution licenses in different groups of redistribution licenses. Each row in *Group*

corresponds to a group of redistribution licenses and stores the information about the redistribution licenses present in that group. If the j^{th} redistribution license is in the i^{th} group then $Group_{i,j}=1$ else it is 0. Please note that the number of rows in $Group$ are N as the maximum number of groups formed can be N (assuming no connected licenses). If there are g groups then only the first g rows give information about groups and redistribution licenses in each group. For example, the first two rows in array $Group$ for the graph in Figure 5.6 are given by $(1, 1, 0, 1, 0)$ and $(0, 0, 1, 0, 1)$ as the redistribution licenses in the first and second group are $(L_{D_1}, L_{D_2}, L_{D_4})$ and (L_{D_3}, L_{D_5}) respectively. Rest of the rows have all entries equal to 0. Algorithm 8 also calculates array $GroupSize$, which determines the number of redistribution licenses in each group.

Algorithm 8: Group Formation

```

int  $g=0$ .
array  $Visited$ : Size  $N$ . All elements are initialized to 0.
array  $Group$ : Size  $N \times N$ . All elements are initialized to 0.
array  $GroupSize$ : Size  $N$ . All elements are initialized to 0.
 $Adj$ : Adjacency matrix of the graph.
for  $i=1$  to  $i \leq N$  do
    if  $Visited[i]=0$  then
         $g=g+1$ .
        Call  $Depth\_first(i, g)$ .
Print(Number of groups= $g$ ).
Subroutine: $Depth\_first(i, k)$ 
{
     $Group[k][i]=1, Visited[i]=1$ 
     $GroupSize[k]=GroupSize[k]+1$ .
    for  $j=i+1$  to  $i \leq N$  do
        if  $Adj[i][j]=1$  and  $Visited[j]=0$  then
            Call  $Depth\_first(j, k)$ .
}
```

5.5.4 Validation Algorithm

The proposed graph theory based algorithm can be used to the validation more efficiently with both the validation tree and two-dimensional array based data structure. In this section, we present the use of the proposed graph theory based algorithm with

the validation tree. The proposed algorithm works by dividing the original validation tree in g parts and modifying the index of the nodes in each newly generated validation tree. A similar algorithm can also be designed for the use with two-dimensional array based data structure by dividing the original array based data structure into g two-dimensional array based data structure and modification of indexes of the two-dimensional data structures obtained after the division.

5.5.4.1 Division of Validation Tree

Using corollary 5.4.1, a set formed by taking at least one redistribution license from two or more groups out of the total g groups will have *set count* equal to 0. So, in the log records any such set cannot exist. This implies that in validation tree any branch will not contain the redistribution licenses from two or more groups out of the total g groups. For example, in Figure 5.5, any issued license cannot belong to the sets $\{L_{D_1}, L_{D_3}\}$ or $\{L_{D_1}, L_{D_3}, L_{D_5}\}$. So these will not be present in the logs and there cannot be branches $root \rightarrow L_{D_1} \rightarrow L_{D_3}$ or $root \rightarrow L_{D_1} \rightarrow L_{D_3} \rightarrow L_{D_5}$. Thus, the validation tree can be divided into g independent parts; each part contains the nodes corresponding to the redistribution licenses in a particular group. Algorithm 9 (by calling $Separation(root, g)$) is used to divide the original validation tree into g new validation trees. The algorithm checks for the group to which a child node of the *root* node belongs. If a child node belongs to the j^{th} group then link the child nodes to the j^{th} newly generated validation tree with root node as $root_j$. Figure 5.7 shows two validation trees obtained after the division of the validation tree in Figure 5.1.

Algorithm 9: $Separation(T, g)$

```

Initialize  $m$  number of root nodes.
Let the root of the  $i^{th}$  validation tree be defined as  $root_i$ .
foreach Child of T do
    Let current child be  $T'$ 
    if  $T'.index \in Group[j]$  then
        Link  $T'$  as child node of  $root_j$ .

```

Figure 5.7: Division of validation tree

5.5.4.2 Modification of Indexes

In this step, the indexes of the nodes in each new validation tree are modified. This is to ensure that if N_k ($=GroupSize[k]$, calculated in Algorithm 8) redistribution licenses are in the k^{th} group then the indexes of nodes in the k^{th} validation tree vary from 1 to N_k , which is required for the validation algorithm (Algorithm 2). Algorithm 10 is used to modify the indexes for each validation tree. To modify the indexes of the nodes in the k^{th} validation tree (with root node $root_k$), algorithm *Modification* ($root_k, A_k$) is called. The algorithm calculates and uses an array named $position_k$ to modify the indexes. If a redistribution license (say j^{th}) is in the k^{th} group then the value stored at $position_k[j]$ determines the new index of the j^{th} redistribution license. The original index is replaced by the new index in the last step in Algorithm 10. For instance, consider array $position_2$ for the second validation tree in Figure 5.7. According to Algorithm 10, it will be (0, 0, 1, 0, 2). So, the indexes 3 and 5 in the nodes of the second validation tree in Figure 5.7 are replaced by 1 and 2 respectively in Figure 5.8. The *validation trees* in Figure 5.7 after modification of indexes are shown in Figure 5.8.

Algorithm 10: *Modification*($root_k, A_k$)

```

array Group: Same as calculated in Algorithm 8.
array positionk: size  $N$ . All elements are initialized to 0.
 $p=1$ .
for  $j=1$  to  $N$  do
    if  $Group[k][j]=1$  then
         $position_k[j]=p$ .
         $A_k[p]=A[j]$ 
         $p=p+1$ .
    end if
end for
    Traverse the  $k^{th}$  validation tree.
for Each node traversed (except  $root_k$ ) do
    Let the index of the node be given by index.
    Replace index by  $position_k[index]$ .
end for
    
```

For each group, we also need to calculate the array containing the aggregate constraint values in the redistribution licenses in the group. Let it be A_k for the k^{th} group (corresponding to k^{th} validation tree). The size of array A_k is given by N_k , where N_k is the number of redistribution licenses in the k^{th} group. A_k is derived using the initial array A containing aggregate constraint counts for all redistribution licenses. The procedure for derivation of A_k using A is mentioned in Algorithm 10. After the modification of indexes and calculation of aggregate constraint array, we can directly use the validation algorithm (Algorithm 2) for each validation tree. So, we call $Validation(root_k, A_k, N_k)$ for each value of $k \leq g$, where g is the total number of groups of redistribution licenses.

The validation time performance gain of our proposed graph theory based approach depends on the number of groups formed and the number of redistribution licenses in each group. If N_k number of redistribution licenses are present in the k^{th} group then $2^{N_k} - 1$ validation inequalities are needed for the k^{th} group. Therefore, the total number of validation inequalities required to validate will be $\sum_{k=1}^g (2^{N_k} - 1)$. Whereas, without using graph theory based approach the total number of validation inequalities are $2^N - 1$. Thus, the approximate performance gain (G) can be given as:

$$G \approx \frac{2^N - 1}{\sum_{k=1}^g (2^{N_k} - 1)} \quad (5.11)$$

Figure 5.8: Modification of indexes

The value of G varies from 1 to $(2^N - 1)/N$ for $m=1$ to $m=N$. Thus, the performance gain always remains greater than or equal to 1.

5.6 Experimental Performance Analysis

In this section, we present the experimental analysis of our proposed approaches namely, validation tree, two-dimensional array based data structure, and graph theory based approach in terms of compactness, validation time, memory requirements and insertion time. All the experiments were performed on Intel(R) Xeon(R) 3.20 GHZ 32-bit CPU with 6 GB RAM. All the programs were written in Java. To our knowledge, any standard benchmark data related to licenses in multi-level DRM is not available. So, for the experiments we generate our own simulation data.

To perform the experiments, we first created a number of redistribution licenses and issued licenses. The set of redistribution licenses to which each issued license can be instance-based validated along with the aggregate constraint counts is saved in the log records. For experiments, each redistribution license is assumed to contain five instance-based constraints and one aggregate constraint with aggregate constraint's count between 5,000 and 15,000. Out of the five instance-based constraints

one instance-based constraint refers to the validity period of the redistribution licenses. Other four instance-based constraints refer to range type constraints with up to 200, 5, 50, and 300 values in their domains respectively. Each issued license is assumed to contain aggregate constraint counts in between 10 and 30. So, the number of log records in our experiments varies from about 600 for $n=1$ redistribution license to 22000 for $n=35$ redistribution licenses.

To our knowledge, there is no other work that discusses the efficient approaches for doing the validation using the validation inequalities of the form discussed in this chapter. So, we evaluate our proposed approaches against two alternative approaches namely direct approach and modified approach. In the direct approach, we directly scan all the log records to find the subsets for the set corresponding to each validation inequality and then aggregate the *set counts* for each subset. The direct approach can be considered equivalent to the online validation method. In modified approach, we preprocess the log records to first aggregate the *set counts* for the sets containing the same redistribution licenses to obtain the modified logs. The records in modified logs then can be scanned to find the subsets of the set corresponding to each validation inequality and then aggregate the *set counts* for each subset. Since, the number of records in the modified logs is generally much less therefore it performs better than the direct method.

Figure 5.9: Distinct sets of redistribution licenses

Figure 5.10: Average length of sets of redistribution licenses

5.6.1 Property of Datasets

There are different instance-based constraints in the redistribution licenses in different datasets. So, for different datasets, different number and length of distinct sets of redistribution licenses are generated during the online instance-based validation, which affect the validation time performance, memory requirement, and insertion time performance of all three methods. Figure 5.9 shows the variation of number of distinct sets (which is also equal to the number log records in the modified logs) of redistribution licenses with N for all three datasets. Figure 5.10 shows the variation of the average length (number of redistribution licenses in a set) of the sets of redistribution licenses with N in the modified logs.

5.6.2 Compactness Analysis

In this section, we do the compactness analysis of the validation tree. The compactness can be measured as the ratio of the number of nodes (excluding the *root* node) in the validation tree and the number of distinct sets of redistribution licenses (or number of records in the modified logs). Each set that is different from all the previous sets generates at least one node in the validation tree. As shown in Figure 5.11, the compactness ratio lies between 1 and 1.1 for all values of N for all three datasets (note that the compactness ratio for datasets 1 and 2 is almost 1 so the curves for number of nodes and number of distinct sets overlap). It implies that each record of the redistribution licenses (containing up to N redistribution licenses) can be represented approximately using only one node. Thus, the representation using the validation tree is much compact and would lead to lower validation time and storage space requirements.

The compactness ratio depends on the common paths shared by distinct sets of redistribution licenses generated using a dataset as discussed in Section 5.3.2.2. From the experiments, we observe that the compactness ratio increases slightly from dataset 1 to dataset 3. This slight increase is because, as shown in Figure 5.10,

Figure 5.11: Compactness analysis of the validation tree

the average length of sets of redistribution licenses is relatively more in dataset 3 as compared to dataset 1 and dataset 2. So, in dataset 3 there may be more sets with more redistribution licenses for which only a small path is shared with other sets with less number of redistribution licenses. The compact representation along with the efficient validation and insertion algorithms leads to efficient validation time and insertion time performance as well as lower storage space requirements.

For the two-dimensional array based data structure, the number of *bit-vector nodes* always remains equal to the number of distinct sets of redistribution licenses. This is because each distinct set of the redistribution licenses creates one and only one *bit-vector node*. But the additional space is required to store the two-dimensional $N \times N$ array. So, as shown in Section 5.6.5, it takes more storage space as compared to the validation tree. However, the validation time only depends on the traversal of *bit-vector nodes* and in the two-dimensional array based data structure we need to traverse much lesser number of *bit-vector nodes* so the validation time performance becomes better with two-dimensional array based data structure as compared to the validation tree.

5.6.3 Validation Time Performance

Figure 5.12 compares the validation time performance of the direct method, modified method and our proposed methods (namely, validation tree based method, two-dimensional array based method and graph based method) in terms of the validation time requirements for different number of redistribution licenses for dataset 2. The experiments show that all three of our proposed algorithms always give better performance as compared to the direct and the modified methods and the comparative performance of our proposed algorithms become better as the value of N increases. In particular, at $N=1$ both the modified method and our proposed methods have almost same validation time performance but the relative performance of our proposed methods increase significantly as N increases. For instance, for $N=30$, the validation tree and two-dimensional array based data structures perform about 9 and 142 times respectively (in terms of validation time) as compared to the modified approach. The gain of the validation tree and two-dimensional array based method remains more than 200 and 7,000 times respectively as compared to the direct method for all values of $N > 1$. The reason for such a large performance enhancement as compared to the direct method is the compact data representation property of the prefix tree based structure. However, in modified approach, the main reason for performance enhancement is the efficiency of our proposed subset search algorithm for the validation tree. Note that the graph based method gives almost same performance as compared to validation tree based method for the dataset 2 (but it can also improve the performance very significantly depending on the properties of a dataset). An elaborate discussion on the performance of the graph based method with the properties of the datasets is presented later in this section.

Figure 5.13 shows the comparison of performance of the validation tree based method with the modified method for all three datasets. We observe that for all three datasets, the validation time with the modified and our proposed method is the least with dataset 1 and most for dataset 3. This is because with more number

Figure 5.12: Validation time performance of all three methods for dataset 2

of distinct sets of redistribution licenses, the number of modified log records as well as the number of nodes in the validation tree also becomes more. So, during the validation, more time is required to scan both the modified logs and validation tree. We also observe that the relative gain of our method as compared to the modified method in general increases from dataset 1 to dataset 3 for all values of N . This is because, from dataset 1 to dataset 3, the relative increase in the average length of records in the modified logs (see Figure 5.10) is more than the relative increase in average number of nodes (see Figure 5.11) to store each record. For example, for $N=20$, the average length of records in modified logs is 3.6 and 5.2 for dataset 1 and dataset 3 respectively (i.e. the increment of about 44%); whereas for the validation tree based method the average number of nodes per modified log record is 1 and 1.15 for dataset 1 and dataset 3 respectively (i.e. the increment of about 15% only).

In Figure 5.14, we compare the validation time performance for two-dimensional array based data structure, and validation tree based method. The experimental results are in accordance with the theoretical analysis in Section 5.4.4 i.e. the proposed

Figure 5.13: Validation time performance of validation tree

method performs better than both the modified and validation tree based method for all values of N . The main reason for such a performance gain is that the proposed data structure divides the whole license data into $(N)*(N+1)/2$ parts. For a particular validation inequality for a set containing r redistribution licenses, we only need to traverse through r parts out of the total $(N)*(N+1)/2$ parts, where $r \leq N$.

To compare the validation time performance after applying the graph theory based algorithm with the validation tree, we first show the variation of the number of groups for all three datasets with the number of redistribution licenses (N) in Figure 5.15. The number of groups in our experiments varies from 1 to 13 for different values of N for dataset 1 but remains equal to 1 for datasets 2 and 3 for most of the values of N . As in Figure 5.15, the number of disconnected groups may remain same, increase or decrease after addition of a new redistribution license. This is because a new redistribution license can become a part of an existing group, form a separate group, or merge two or more disconnected groups.

Figure 5.14: Comparison of validation time performance of two-dimensional array based data structure and validation tree

Figure 5.15: Variation of number of groups

Figure 5.16 shows a comparative analysis of the validation time after applying the graph theory based method to the validation tree with the validation tree based

method. From the Figure 5.16, it can be observed that there is no gain for datasets 2 and 3. This is because for most of the values of N , the number of groups remain equal to 1 for datasets 2 and 3 as shown in Figure 5.15. So, in accordance with Equation 5.11, the gain remains 1. However, a significant gain is obtained for dataset 1 for $N > 11$. This is because as shown in Figure 5.15, for dataset 1, the number of groups remain 1 for all values of N up to 11 and becomes more than 1 for $N > 11$. In particular, the gain for $N = 35$ is about 10,000 times, which is a significant gain. So, the graph theory based method can be much useful if more than one group of redistribution licenses are formed.

Figure 5.16: Validation time complexity after using graph theory based approach

The validation time gain obtained using our proposed algorithms is significant in practical scenarios. In practical scenarios, in most of the cases we expect up to 20 to 30 active (i.e. not expired) redistribution licenses per content. Assuming that there are 1000 contents with a distributor and there are 30 redistribution licenses per content. The time required for validation using our proposed two-dimensional array based data structure for $N = 30$ is of the order of 10^4 milliseconds. So, the time

required for validation will be of the order of several hours for 1,000 contents. This is quite relevant practically as validations need to be done periodically (may be once everyday). This is a significant improvement over direct and modified methods, which perform the same task in time which is of the order of several years and several weeks respectively.

5.6.4 Insertion Time Performance

In Figure 5.17, we compare the average insertion time over all three datasets for the modified method, validation tree based method, two-dimensional array based method, and graph theory based method. The modified method takes the highest insertion time among all four methods because it has the complexity of $O(D)$ whereas the complexity of other methods is $O(N)$. And the value of D in general is much larger than N . As shown in Figure 5.17, the two-dimensional array based method performs better than the validation tree based method (although both the methods have $O(N)$ complexities). This is because in the validation tree based method if the index of the highest indexed redistribution license in a record is k_r ($1 \leq k_r \leq N$) then up to k_r comparisons may be needed. Whereas in the two-dimensional array based approach the number of comparisons (in the pre-processing step) depends on the number of redistribution licenses in the set to be inserted. Since the number of redistribution licenses in a set cannot be more than the index of the last redistribution licenses in the set so the two-dimensional array based method performs better than the validation tree based method.

The insertion time required for the graph theory based method is the sum of the insertion time in the validation tree and time required for division of the original validation tree. As shown in Figure 5.17, it remains almost same as that of the insertion time in the validation tree based method (assuming that division of the validation tree is performed only once during the insertion of log records). This is because the time required to divide the validation time is only about 7–8 times of the

Figure 5.17: Insertion time performance

time required to insert a single log record, but the insertion process generally requires the insertion of thousands of the log records. A comparison between the time required for the division of the validation tree and insertion of a single log record is shown in Figure 5.18.

5.6.5 Storage Space Requirements

Figure 5.19 compares the average storage space required for the validation tree based method, two-dimensional array based method, graph based method and modified method. Among all the methods, the two-dimensional array based data structure takes highest storage space, and validation tree based data structure takes the lowest storage space. The storage space for graph based method is only slightly more as compared to storage space for the validation tree (the curves for both the methods almost overlap in Figure 5.19). The additional storage space for the graph based method is due to the generation of new validation trees after the division of original validation tree. However, the only slight increase is attributed to the fact that after

Figure 5.18: Insertion time vs validation tree division time

the division of the original validation tree no new nodes except several root nodes are created.

Figure 5.19: Storage space comparison

Figure 5.20 compares the performance of validation tree based method with the modified approach in terms of storage space requirements. From the experiments it is evident that the storage space requirements for validation tree is much less than the modified approach for all three datasets. It is mainly due to the compact representation of the validation tree. The storage space requirements for the modified method, and validation tree increase from dataset 1 to dataset 3 because modified logs contain more number of log records for dataset 3, and number of nodes in the validation tree also increases with the increase in the number of distinct sets of redistribution licenses.

Moreover, the storage space requirements for all three methods for a single content is of the order of several kilobytes. This is very less as compared to storage space available in modern computers (of the order of gigabytes and terabytes). So, it remains practically feasible to use our proposed methods with modern computers.

Figure 5.20: Storage space requirements for modified and validation tree based approaches

Figure 5.21: Storage space requirements for modified and two-dimensional array based approaches

The storage space for two-dimensional array is compared with the storage space for the modified method in Figure 5.21. Although, for datasets 1 and 2 the two-dimensional array based data structure takes more storage space, but the storage space requirement remains almost constant as the number of distinct sets increases. The more storage space for datasets 1 and 2 is due to the overhead due to the storage of $N \times N$ array. With more number of distinct sets in a dataset, as for the dataset 3, the overhead due to the storage of $N \times N$ array becomes relatively small as compared to the storage of bit-vector nodes. Thus, the proposed data structure is scalable in terms of the storage space.

5.7 Conclusion

In this chapter, we presented the problem of aggregate licenses validation in multi-level DRM system. The aggregate validation task is computationally intensive due to exponential number of validation inequalities and exponential number of terms in

each validation inequality. So, efficient ways of aggregate validation are required. We proposed several data organization based solutions namely validation tree and two-dimensional array based data structure and a graph theory based algorithm for this purpose. All these methods work efficiently as compared to existing methods of validations. Among the validation tree and two-dimensional array based data structure, validation tree has relatively less storage space requirements and two-dimensional array based data structure is relatively efficient in terms of the validation time, and both perform almost similar in terms of insertion time performance. The graph theory based algorithm improves the performance significantly if the redistribution licenses can be divided in to several non-overlapping groups. So, with the use of the proposed methods, aggregate validation can be performed efficiently.

Chapter 6

DRM Violations Detection Using Consumer Logs Analysis

In this chapter, we present log files analysis based method for the detection of violations by the consumers. We start the chapter by analyzing the problem of the consumers' violation detection in DRM systems. Then, we present the concept of the global and semi-global analysis of the consumer logs, and a sequential pattern mining based algorithm for effective violation detection. We also present a bit-vectors based algorithm for doing the global and semi-global analysis efficiently. Finally, we demonstrate the effectiveness and efficiency of the proposed violation detection method using the theoretical analysis and experimental results.

6.1 Problem Definition

Content usage control in DRM systems is usually done using the digital licenses. The conditions of content usage in the digital licenses are expressed in the form of permissions and constraints [29]. The permissions and constraints in the licenses may be violated due to various vulnerabilities [125,155] in operating systems, DRM agent software, hardware associated with DRM systems, etc., and various possible attacks due to these vulnerabilities [11,71,82,92,125,135,155].

The main attacks exploiting these vulnerabilities are compromise of DRM agent and content keys [11], modification of rights object [146], replay attacks [82], system

modifications such as modification in system time, and attacks due to external softwares and malwares [11, 155]. A study on commercial DRM systems in [155] suggests that most of the commercial DRM systems such as Microsoft WMRM [10], FairPlay DRM [5], etc. have been attacked in the past exploiting these vulnerabilities. Furthermore, once a vulnerability is exposed by an attacker, the attacker can easily share the attack knowledge with others. This causes a huge loss to the owners. So, it is necessary to detect the violations, and determine and fix the vulnerabilities causing the violations. We use the log files obtained from consumers as a means for violation detection. We assume that the log files can be securely stored at consumers' devices, and any modification in the log files can be detected [44, 86].

A direct way of detecting the violation is to collect the usage logs (of the form shown in Table 6.2) from consumers [27] and compare the usage of contents with the permission and constraints granted in the respective licenses. For example, if a license allows 10 counts and the usage within a certain time period then from the logs we can easily identify if the number of times content played is more than 10 or if it is not played within the allowed time period. However, analyzing the logs directly, the owner may not be able to determine the types of violations and vulnerabilities causing the violations.

To determine the types of violations, it may require the analysis of the sequence of events in the consumer logs e.g. if a violation has resulted from the compromise of DRM agent (i.e. DRM agent is modified such that it does not respect permissions and constraints in the licenses) then every time the content is played, the remaining counts (total issued counts–total used counts) in the logs will not change and also the content can be played even after the expiry date. On the other hand, if a violation is resulted from the modification of licenses (say count constraint) then the remaining count will change every time content is played but in some initial logs the value of remaining count may be more than the issued count in the respective license. Here the sequences of activities are different in both the cases and analyzing the sequence of activities in the logs, an analyst may predict the types of violations.

However, DRM systems generally contain thousands of consumers and analysis of the logs obtained from every consumer by an analyst may not be practically feasible. Also, the system owners are generally more interested in obtaining the violation statistics (e.g. most frequently occurring violations) in the complete system rather than the analysis of violation for each consumer individually. So, it would be much beneficial if we can get the representative sequences in the logs of different consumers. For this purpose, we propose the use of sequential pattern mining [31,93]. The advantage of the sequential pattern mining algorithms is that the analyst gets only a small set of representative sequences (or patterns) of violations. The analyst can easily analyze these small set of patterns of violations.

But the format of the logs obtained from the consumers is not suitable to apply the sequential pattern mining algorithms directly. So, in this chapter, we first propose an approach of static and dynamic state models to convert the logs obtained from the consumers into the alert logs, which are in a format suitable for the analysis with the sequential pattern mining algorithms.

Additionally, there are challenges associated with making the traditional sequential pattern mining algorithms [31, 84, 93, 131, 132] more suitable for the violation analysis. The traditional sequential pattern mining algorithms can find the violations only for the complete system. However, to obtain more useful information (e.g. vulnerability causing violation) it requires obtaining the violation information in different parts of the system e.g. the types of violations occurring only in certain region, operating system, or a combination of region and operating system etc. For instance, if we find that a violation occurs most frequently in Windows XP then most likely the cause of violation will be associated with a vulnerability in Windows XP. To obtain such information, the traditional sequential pattern mining algorithms need to be applied multiple times (once for obtaining information about each specific part of the system), which may be inefficient. So, we propose an efficient bit-vectors (a string of bits 0 and 1) based algorithm called global and semi-global violation pattern mining,

where global refers to the complete system and semi-global refers to a part of the system. The method is efficient as it eliminates the need of applying the sequential pattern mining algorithm repeatedly on data for each case of global and semi-global violation analysis. The efficiency of the proposed method is shown in Section 6.7. To the best of our knowledge, there is no other work in DRM that focuses on the violation detection using the consumers' logs analysis.

6.2 Preliminaries

In this section, we discuss preliminaries and related works using sequential pattern mining algorithms those resemble the problem in our case.

6.2.1 Sequential Pattern Mining

We discuss preliminaries on sequential pattern mining (SPM) by first defining the terms *item*, *itemset*, *sequence*, *subsequence*, *support*, and *frequent pattern*.

An *item* is the basic unit in SPM. Assuming that there are total n *items* given by $I = \{i_1, i_2, \dots, i_n\}$, an *itemset* is a subset of I . A *sequence* is the set of *itemsets* in the time order. A *sequence* containing m *itemsets* is denoted as $a_1 a_2 \dots a_m$, where a_j is the j^{th} *itemset*. A *sequence database* S is a collection of tuples of the form (tid, s) , where tid is the ID of the tuple and s is a *sequence*. Let $I = \{a, b, c, d, e\}$ then $\alpha = \langle a, c, d \rangle \langle b, e \rangle \langle a \rangle \langle b, c, d, e \rangle$ is a possible *sequence* with $\langle a, c, d \rangle$, $\langle b, e \rangle$, $\langle a \rangle$ and $\langle b, c, d, e \rangle$ as *itemsets*.

A *sequence* $s' = a_1 a_2 \dots a_m$ is called a *subsequence* of another *sequence* $s = b_1 b_2 \dots b_l$ if there exist integers $j_1, j_2, \dots, j_m$ such that $1 \leq j_1 < j_2 < \dots < j_m \leq l$ and $a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, \dots, a_m \subseteq b_{j_m}$. For example, $\beta = \langle a, d \rangle \langle b, e \rangle \langle b \rangle$ is a *subsequence* of α (in the previous paragraph) as $\langle a, d \rangle \subseteq \langle a, c, d \rangle$, $\langle b, e \rangle \subseteq \langle b, e \rangle$, and $\langle b \rangle \subseteq \langle b, c, d, e \rangle$. The *support* of a *sequence* s in a *sequence database* S is equal to the number of tuples such that s is a *subsequence* of the *sequence* in the tuples. A *sequence* is called (frequent) *sequential pattern* with respect to a database S if the *support* of a *sequence* in the database S

Table 6.1: Table of *sequences*

<i>tid</i>	Sequence	$\langle a \rangle \langle b \rangle$ projected
1	$\langle a \rangle \langle a, b \rangle \langle b, d \rangle$	$\langle b, d \rangle$
2	$\langle a, c, d \rangle$	
3	$\langle a, b \rangle \langle b, d \rangle$	$\langle _d \rangle$
4	$\langle c, d \rangle \langle c, d, e \rangle$	

is greater than or equal to the given *support threshold* T . The task of the sequential pattern mining is to find all such sequential patterns.

Example 6.4 Consider the sequence database (columns 1 and 2) in Table 6.1. Let the support threshold be $T=2$. The sequence $\langle b \rangle \langle b \rangle$ is a sequential pattern as it is a subsequence of sequences corresponding to 2 tuples with tids 1 and 3.

There exist various algorithms for efficient sequential pattern mining. The algorithms can be mainly divided into two categories viz. apriori property based and projection based [31, 84, 93, 131, 132]. The apriori property based algorithms such as AprioriAll [31], GSP [132], SPADE [117], SPAM [84], etc. recursively generate $l+1$ length patterns from l patterns using the *anti-monotone* property. The *anti-monotone* property states that for a *sequence* to be frequent, all its *subsequences* must be frequent. However, algorithms based on the apriori property suffer the drawbacks of generation of large number of candidate patterns and scan of the complete database during each pass. To overcome these drawbacks, Pei et al. [93] proposed a projection based algorithm called PrefixSpan. In PrefixSpan algorithm, during each pass the algorithm generates a projection, which contains only a part of the complete data. The algorithm then needs to scan only a part of database instead of the complete database. We use PrefixSpan algorithm as the basis of our work. Next, we define the terms prefix, suffix, and projected database used in the PrefixSpan algorithm. Then we describe the PrefixSpan algorithm.

Prefix: A *sequence* $s' = a_1 a_2 \dots a_m$ is a prefix of a *sequence* $s = b_1 b_2 \dots b_n$ ($m \leq n$) if $a_i = b_i$ for $i \leq m - 1$, $a_m \subseteq b_m$ and all the *items* in $(b_m - a_m)$ are in the lexicographic order after the *items* in a_m .

Example 6.5 Consider the sequences $s = \langle a \rangle \langle a, b \rangle \langle b, d \rangle$ and $s' = \langle a \rangle \langle a \rangle$. s' is a prefix of s as it satisfies all the conditions required for a prefix. But $\langle a \rangle \langle b \rangle$ is not a prefix of s as $\langle a, b \rangle - \langle b \rangle = \langle a \rangle$, which is not lexicographically after $\langle b \rangle$.

Suffix: Let a sequence $s' = a_1 a_2 \dots a_m$ be a subsequence of another sequence $s = b_1 b_2 \dots b_n$ ($m \leq n$). Sequence $s'' = b'_l b_{l+1} \dots b_n$ is the suffix of s with respect to prefix s' if **(a)**. there exist $i_1, i_2, \dots, i_m$ such that $1 \leq i_1 < i_2 < \dots < i_m \leq n$, $l = i_m$, $a_j \subseteq b_{i_j}$ and i_m has the least possible value. **(b)**. let b'_l be $b_l - a_m$ then all items in b'_l must be lexicographically after the items in a_m .

Example 6.6 Consider the sequence with tid=1 in Table 6.1. The sequence $s'' = \langle b, d \rangle$ will be the suffix of the sequence $s = \langle a \rangle \langle a, b \rangle \langle b, d \rangle$ with respect to the sequence $s' = \langle a \rangle \langle b \rangle$.

Projected database: If α is a frequent sequence in a database S then the α -projected database is the collection of all the suffixes with respect to α in S .

Example 6.7 The projected database for $\alpha = \langle a \rangle \langle b \rangle$ for the sequences in column 2 in Table 6.1 is as shown in column 3 in Table 6.1. Blank entry for a tid indicates that the sequence for the tid does not contain the suffix with respect to α . The sequence $\langle _ d \rangle$ for tid=3 indicates that $\langle _ d \rangle$ was in the same itemset as of b in α . The α -projected database contains only 2 entries.

PrefixSpan Algorithm: In the PrefixSpan algorithm initially all 1 length sequential patterns are generated. Then the projected databases corresponding to 1 length sequences are constructed. In each projected database (say α -projected) frequent items are found and a new frequent sequence (say α') is constructed by appending each frequent item to the sequence α . Then α' -projected database is constructed using α -projected database. The process is recursively applied until all the sequential patterns are found. The advantage of PrefixSpan algorithm is that we need not scan the entire database for finding the $l + 1$ length frequent sequences from an l length sequence, only the partial projected database corresponding to the l length sequence needs to be scanned.

6.2.2 Related Works

The originally proposed sequential pattern mining problem has been extended to accommodate various real life scenarios e.g. mining by applying constraints (such as length, gap, duration, etc.) [67], multidimensional sequential pattern mining [55, 67, 78], etc. The closest problem to the problem in this chapter is the multidimensional sequential pattern mining.

The multidimensional sequential pattern mining problem is the problem of sequential pattern mining by integrating the multidimensional analysis into the sequential pattern mining problem. If there are total d dimensions for the multidimensional analysis then a d dimensional tuple is associated with every *sequence*. The task is to find the frequent *sequences* for all the possible combinations of d dimensional tuples. But like the original sequential pattern mining the same threshold is considered for all cases of multidimensional analysis. This makes the multidimensional sequential pattern mining problem as a simple extension of the original sequential pattern mining problem by considering multidimensional tuple as another *itemset* [67, 78]. However, in our case, there may be different *support thresholds* for different cases of global and semi-global analysis. So, original sequential pattern mining algorithms may not be directly applicable. We need to apply the sequential pattern mining algorithm to the data related to each case of global and semi-global analysis separately, which may be inefficient. However, our proposed bit-vectors based algorithm can mine the sequential patterns for all the cases efficiently without the need of applying the sequential pattern mining algorithm separately for each case.

6.3 Log Transform for Sequential Pattern Mining

As discussed in Section 6.1, the logs originally obtained from the consumers are not suitable for the SPM algorithms. In this section, we present a method of converting the logs originally obtained from the consumers into the alert logs, which are in a

Table 6.2: Consumer log

Consumer Attr: region-UK, operating system-Windows XP, DRM agent version-5				
Log ID	Start time	End time	counts	Region used
1	4 March 2010, 14:10:07	4 March 2010, 15:08:00	9	UK
2	6 March 2010, 08:10:05	6 March 2010, 09:10:04	8	America
3	6 March 2010, 09:20:05	6 March 2010, 09:50:55	8	Germany
4	8 March 2010, 10:05:00	8 March 2010, 12:00:04	8	UK

form suitable for SPM algorithms. Before discussing the log conversion process, we first discuss the format of the logs obtained from the consumers.

6.3.1 Format of Logs

The logs on the consumers' devices are generated when they consume the contents using their licenses. The format of logs is as shown in Table 6.2. Consumer attributes (in the first row in Table 6.2) are the parameters associated with the consumers. The entries in the logs (row 3 onwards in Table 6.2) correspond to the usage statistics of the consumer. For example, the entry with Log ID=1 states that the content is consumed on 4th march 2010 between 14:10:07 and 15:08:00, the remaining counts after the usage of content are 9, and the content is consumed in the UK (here we assume that the license associated with the content allows consumption of the content between 1st March 2010 to 31st march 2010, it is valid for the content consumption in Europe, and it allows total 10 number of counts).

6.3.2 Alert Logs Generation

The alert logs consist of predefined set of alert symbols. The alert symbols represent the anomalous states in logs as determined using static and dynamic state models. These models use the licenses obtained by the consumers and the respective logs collected from the consumers.

6.3.2.1 Static State Model

In the static state model, we verify whether the state of a constraint in a log record is within the states allowed in the respective license. Let U be the set of valid states

for a constraint in a license and u be a state in a log record. For example, for a license allowing total 10 number of counts, the set U for the counts constraint will be $\{0, 1, 2, \dots, 10\}$. Based on the static state model, the log is normal if $u \in U$. As we capture only the anomalous states in the alert logs therefore no symbol will be generated when $u \in U$. Else a symbol will be generated. An analyst may choose to generate different symbols depending upon the ways of violation. For example, the constraint dates allowed for playing a content may be violated in two ways, first if the date in a log record is prior to the start date of the license and second if the date is after the expiry date of license. So, the two types of symbols will be generated for this constraint.

6.3.2.2 Dynamic State Model

The static state model is not effective if after a violation activity, the state of a constraint remains in a valid state. For example, let somebody modifies the DRM agent such that the DRM agent does not change the remaining counts in the licenses even after consuming the content. In this case, the remaining counts will always represent a normal state although an illegal state change is involved.

The dynamic state model verifies that the transition from one state to other state in the successive log records is normal. Alert symbols are generated for the transitions that are not normal. Let the state of the k^{th} constraint in the i^{th} log record be u_i^k and in the $(i-1)^{th}$ log record be $u_{(i-1)}^k$. An alert symbol is generated if $u_{(i-1)}^k \rightarrow u_i^k$ is not a normal transition. For this purpose, we define a variable $t_i^k = u_i^k - u_{(i-1)}^k$ (note that u_0^k is equal to value of k^{th} constraint in the license). The alert symbols are generated based on the value of t_i^k . An analyst may generate different symbols for different values of t_i^k , e.g. for the constraint remaining counts different symbols may be generated for $t_i^k=0$, $t_i^k<0$, and $t_i^k>1$ (note that $t_i^k=1$ is the normal transition).

The static and dynamic models are applied on the logs obtained from every consumer. As illustrated in Example 6.8, if the logs obtained from a consumer contain

anomalous entries then an alert *sequence* is generated. The alert symbols generated for the same entry in the original logs either by the static or dynamic model are placed in the same *itemset* (this shows that these alerts occur simultaneously). The complete alert *sequence seq* is the collection of all such *itemsets*. As shown in Table 6.3, the alert log of a consumer can be represented as a tuple $(tid; v^1, v^2, \dots, v^d; seq)$, where tid is the tuple ID in the alert logs, and $v^1, v^2, \dots$, and v^d are the analysis dimension values. The analysis dimension values are the attributes associated with a consumer such as, region, operating system, etc.

Example 6.8 Consider a license issued to a consumer with play permission and the constraints allowed time period, region allowed, allowed counts, and duration as 1 March 2010 to 30 March 2010, Europe, 10, and 1 hour respectively. Let the logs obtained from the consumer are as shown in Table 6.2. The consumer attributes are as in the first row in Table 6.2.

In the alert logs there will be three analysis dimensions viz. the region, operating system, and DRM agent version with the values UK, windows XP and 5 respectively. For the static state model for *allowed time period* constraint, let there be two alert symbols, a_1 and a_2 for the cases when the time of play is before the start date and after the expiry date in the license respectively. For the dynamic model, let there be one alert symbol a_3 to indicate if the time gap between the start time in current log and the end time in previous log is less than a certain threshold (say 1 hour). This is to detect if the system date is not modified to a previous date. For the *duration* constraint, let there be only one static alert symbol a_4 for the condition if the duration of play exceeds the certain threshold. For the *region* constraint, let there be one static and one dynamic alert symbols, a_5 and a_6 respectively. For the *allowed counts* constraint, let there be one static alert symbol a_7 for the case the remaining counts exceed the counts in the license and three dynamic alert symbols, a_8, a_9 , and a_{10} for no change in the remaining counts, negative change by more than

1, and positive change respectively. Using the static and dynamic models, no alert symbol will be generated for the first entry in Table 6.2. For the second entry, two alert symbols, a_5 and a_6 will be generated. For the third and fourth entries, symbols (a_3 , a_6 , and a_8), and (a_4 , a_6 , and a_8) will be generated respectively. So, the tuple corresponding to the consumer in the alert logs (let tuple be at the k^{th} position) will be ($tid=k$; $v^1=UK$, $v^2=windows$ XP, $v^3=5$; $seq=<a_5,a_6><a_3,a_6,a_8><a_4, a_6, a_8>$).

6.3.2.3 Alert Logs Database

The alert logs database S can be expressed as a collection of the tuples of the form $<tid; v^1, v^2, \dots, v^d; seq>$ obtained after the log transformation of the original consumers log data. The form of the alert database with seven log entries and three analysis dimensions (viz. region, operating system, DRM agent version) is as shown in Table 6.3. If the domain of the j^{th} analysis dimension is given by D_j then $v_i^j \in D_j$ for all the values of i . For the alert log database, for ease of understanding we use the terms alert symbol, alert set, alert *sequence*, and *violation pattern* in the place of the terms *item*, *itemset*, *sequence*, and sequential pattern respectively (as in Section 6.2) further in this chapter.

Table 6.3: Alert logs database (S)

tid	Analysis dimensions	Alert <i>sequence</i>
1	(Italy, Linux, 5)	$<a_1><a_3,a_5><a_3,a_5,a_7><a_3,a_5,a_7, a_9><a_5><a_3,a_{10}>$
2	(India, Windows, 4)	$<a_1,a_2,a_4><a_3,a_6><a_3,a_{10}><a_2,a_3,a_5,a_7, a_9,a_{10}>$
3	(France, windows, 4)	$<a_1,a_2><a_3,a_5,a_7><a_3,a_5,a_7, a_{10}>$
4	(USA, MAC, 3)	$<a_3,a_4><a_3><a_2><a_2,a_3,a_7, a_{10}>$
5	(India, MAC, 5)	$<a_1,a_3,a_{10}><a_3,a_5,a_8><a_4,a_5,a_6><a_1,a_2,a_3, a_5,a_8,a_{10}>$
6	(UK, Windows, 2)	$<a_3,a_9><a_2><a_3,a_9><a_2><a_3,a_9><a_3,a_9><a_3>$
7	(China, Linux, 2)	$<a_3,a_6><a_3,a_6><a_2, a_3,a_6><a_1,a_3,a_4, a_9,a_{10}>$

6.4 Global and Semi-global Analysis

To make the SPM algorithms more suitable for the violation detection, we propose the concept of global and semi-global violation analysis. The global violation analysis

refers to the analysis of the violations in the complete system and semi-global violation analysis refers to the analysis of violation for a part of the system. For example, a semi-global case may be to analyze the violations by the consumers in Asia who are using Linux operating system. The cases of global and semi-global analysis can be expressed using d dimensional *rules* in the form of the equations 6.1 and 6.2. Let the domain of j^{th} analysis dimension be defined as D_j . Then the global *rule* (R_1) is expressed as:

$$R_1 = \{D_1, D_2, \dots, D_d : T_1\} \quad (6.1)$$

where T_1 is the *support threshold* (see Section 6.2) for the *rule* R_1 . If there are $N - 1$ semi-global analysis cases then these cases can be represented using the total $N - 1$ *rules*. The i^{th} ($i=2$ to N) *rule* can be expressed as:

$$R_i = \{s_i^1, s_i^2, \dots, s_i^d : T_i\} \quad (6.2)$$

where s_i^j is the set of values associated with the j^{th} analysis dimension in the i^{th} *rule* such that $s_i^j \subseteq D_j$ and T_i is the *support threshold* for the i^{th} *rule*.

Interpretation of the Rules: The *rule* R_i ($1 \leq i \leq N$) can be interpreted as to find all the *violation patterns* with the *support* greater than T_i using only the alert log entries satisfying the i^{th} *rule*. An alert log entry with the d analysis dimensions' values as $v^1, v^2, \dots, v^d$ satisfies the i^{th} *rule* if $v^1 \in s_i^1, v^2 \in s_i^2, \dots$, and $v^d \in s_i^d$. For example, let there be a *rule* {Asia, Linux: 1000} then the task is to mine all the violation patterns with the *support* greater than 1000 using only the log entries in which consumers belong to Asia and use Linux operating system. Since the number of entries satisfying each *rule*, in general, are different so a separate threshold for each *rule* is required.

To use the existing SPM algorithms, we need to create the database of the alert log entries satisfying each *rule* and then need to do the *violation pattern* mining separately on these databases. But there may be a significant amount of overlapping

data between these *rules* so it may become inefficient to use the existing SPM algorithms. So, next we propose an efficient global and semi-global *violation pattern* mining algorithm.

6.5 Proposed Efficient Algorithm

The proposed algorithm uses a bit-vector (it is a string of bits 0 or 1) based technique to efficiently mine the *violation patterns* for all the *rules*. The algorithm consists of the two steps namely, log data classification, and *violation pattern* mining.

6.5.1 Log Data Classification

In this step, entries in the alert log database are classified using the dynamically generated N length bit-vectors as classifiers, where N is the number of total global and semi-global *rules*. Each class obtained after the classification acts as a separate sub-database as shown in Table 6.4. The bit-vector calculated for the sub-databases are used for the classification. The i^{th} bit (from LSB) in a bit-vector corresponds to the i^{th} rule.

Table 6.4: Table of sub-databases

<i>subDB ID</i>	Bit-vector	Alert sequences
1	$\mathbf{b}_1=00101$	1. $\langle a_1 \rangle \langle a_3, a_5 \rangle \langle a_3, a_5, a_7 \rangle \langle a_3, a_5, a_7, a_9 \rangle \langle a_5 \rangle \langle a_3, a_{10} \rangle$
		2. $\langle a_1, a_2 \rangle \langle a_3, a_5, a_7 \rangle \langle a_3, a_5, a_7, a_{10} \rangle$
2	$\mathbf{b}_2=01011$	1. $\langle a_1, a_2, a_4 \rangle \langle a_3, a_6 \rangle \langle a_3, a_{10} \rangle \langle a_2, a_3, a_5, a_7, a_9, a_{10} \rangle$
3	$\mathbf{b}_3=00001$	1. $\langle a_3, a_4 \rangle \langle a_3 \rangle \langle a_2 \rangle \langle a_2, a_3, a_7, a_{10} \rangle$
		2. $\langle a_3, a_9 \rangle \langle a_2 \rangle \langle a_3, a_9 \rangle \langle a_2 \rangle \langle a_3, a_9 \rangle \langle a_3, a_9 \rangle \langle a_3 \rangle$
4	$\mathbf{b}_4=01001$	1. $\langle a_1, a_3, a_{10} \rangle \langle a_3, a_5, a_8 \rangle \langle a_4, a_5, a_6 \rangle \langle a_1, a_2, a_3, a_5, a_8, a_{10} \rangle$
		2. $\langle a_3, a_6 \rangle \langle a_3, a_6 \rangle \langle a_2, a_3, a_6 \rangle \langle a_1, a_3, a_4, a_9, a_{10} \rangle$

Algorithm 11 is used to classify the alert log entries (in Table 6.3) into different classes. We assume that no bit-vector classifier is initially present. The first bit-vector classifier is derived using the first log entry in the alert logs database. If the log entry satisfies the i^{th} ($1 \leq i \leq N$) rule then the i^{th} bit (from LSB) in the bit-vector classifier

Algorithm 11: Log_ Classification

Initialize an empty table of sub-databases (see Table 6.4).

foreach LogEntry **do**

variable: *bitVec*=0.

$V = \{v^1, v^2, \dots, v^d\}$ analysis dimensions in the LogEntry.

for $i=1$ to N **do**

if satisfies(R_i, V) **then**

$\text{bitVec} = \text{bitVec} + 1 \ll (i-1)$.

 int $p = \text{sizeOf}(\text{subDB})$.

for $j=1$ to p **do**

if $\mathbf{b}_j = \text{bitVec}$ **then**

 /* $\mathbf{b}_j$: bit-vector for j^{th} subDB. */

 Add the record to the j^{th} subDB.

 Exit and process the next LogEntry.

 Create a new subDB with $ID=p+1$, $\mathbf{b}_{p+1} = \text{bitVec}$.

 Add the record to the $(p+1)^{\text{th}}$ subDB.

Subroutine: satisfies(R_i, V) {

m : number of classification dimensions.

$s_i^1, s_i^2, \dots, s_i^d$: set of analysis dimension values in R_i .

 Let $V = \{v^1, v^2, \dots, v^d\}$.

if $(v^1 \in s_i^1) \wedge (v^2 \in s_i^2) \wedge \dots \wedge (v^d \in s_i^d)$ **then** Return True.

else Return False.}

is assigned a bit value 1 else it is assigned a value 0. It creates the first sub-database with the calculated bit-vector and the first alert log entry.

For all other alert log entries, we first calculate the bit-vector as calculated for the first log entry. Then compare it with the bit-vectors associated with the existing sub-databases. If we find an existing sub-database having the same bit-vector then the alert log entry is added to that sub-database. If no such sub-database is found then a new sub-database with the bit-vector equal to the bit-vector calculated for the current log entry is created and the alert log entry is placed into the newly generated sub-database. We assume that after the classification of all the alert log entries total p sub-databases are formed.

Example 6.9 Consider the alert logs in Table 6.3 and following one global rule (R_1) and four semi-global rules (R_2 - R_5).

$$R_1 = \{s_1^1 = [\text{Europe, Asia, USA}], s_1^2 = [\text{Windows, MAC, Linux}], s_1^3 = [1, 2, 3, 4, 5]: 4\}.$$

$$R_2 = \{s_2^1 = [\text{Asia}], s_2^2 = [\text{Windows}], s_2^3 = [1, 2, 3, 4, 5]: 2\}.$$

$$R_3 = \{s_3^1 = [\text{Europe}], s_3^2 = [\text{Windows, MAC, Linux}], s_3^3 = [3, 4, 5]: 3\}.$$

$$R_4 = \{s_4^1 = [\text{Asia}], s_4^2 = [\text{Windows, MAC, Linux}], s_4^3 = [1, 2, 3, 4, 5]: 2\}.$$

$$R_5 = \{s_5^1 = [\text{USA}], s_5^2 = [\text{Windows}], s_5^3 = [1, 2, 3, 4, 5]: 2\}.$$

Now according to Algorithm 11, each alert log record in Table 6.3 is scanned to derive the bit-vector based classifier. The alert log record with $tid=1$ in Table 6.3 satisfies the rules R_1 and R_3 as the analysis dimension values (Italy, Linux, and 5) are elements of the respective sets in R_1 and R_3 . So, according to Algorithm 11, the bit-vector corresponding to the alert log record will be $b_1 = 1 \ll (1-1) + 1 \ll (3-1) = 5$ (00101 in the binary bit-vector format). The first sub-database is created with the bit-vector $b_1 = 00101$ and the alert *sequence* is placed into it. The second alert log record in Table 6.3 satisfies the rules R_1 , R_2 and R_4 so the bit-vector corresponding to it is 01011. On comparing the bit-vector with the bit-vectors of existing sub-database, 01011 is

not same as b_1 . So, a new sub-database with $b_2=01011$ is created and the *sequence* in the second alert log record is added to the sub-database. The third alert log record satisfies the rules R_1 , and R_3 . So, the bit-vector corresponding to the third alert log entry is 00101 . On comparing it with existing bit-vectors, it is found that 00101 is equal to the bit-vector corresponding to the existing sub-database with SubDB ID=1, so the alert log record is added to the sub-database with subDB ID=1. Similarly, all other alert log records in Table 6.3 are added to Table 6.4. After addition of all the alert log records in Table 6.3, we get four sub-databases ($p=4$) as in Table 6.4.

The classification algorithm classifies the alert log data such that each alert *sequence* in the original alert log database is classified to only one sub-database. The classification is such that if the i^{th} bit in a bit-vector corresponding to a sub-database is 1 then every entry in that sub-database satisfies the i^{th} rule, else if it is 0 then no entry in that sub-database will satisfy the i^{th} rule.

6.5.2 Global and Semi-global Violation Pattern Mining

In this section, we first present the basis of efficient violation pattern mining using the bit-vectors based sub-database division as in Table 6.4. Then we present our proposed bit-vectors based efficient *violation pattern* mining algorithm.

6.5.2.1 Basis of Efficient Mining Using Bit-vectors

Using the sub-databases and bit-vectors associated with the sub-databases, the proposed algorithm gains efficiency in two ways viz. efficient *support* calculation for each global and semi-global rule, and identification of the redundant sub-databases.

6.5.2.1.1 Efficient *Support* Calculation for Rules: From discussions in Section 6.2, a *violation pattern* α is frequent if its *support* in the given alert database S is greater than the minimum absolute threshold required. The *support* ($Supp_S(\alpha)$) of α is defined as $Supp_S(\alpha)$ =Number of tuples of the form $\langle tid; V = (v_1, v_2, \dots, v_d); seq \rangle$ in S such that α is a *subsequence* of seq i.e. $\alpha \sqsubseteq seq$.

We require to calculate the *violation patterns* separately for each global and semi-global *rule*. So, we need to separately calculate the *support* for each *rule*. For an alert *sequence* to be frequent for a *rule* the *support* calculated for that *rule* must be greater than the minimum *support* required for that *rule*. Let the symbol $Supp_S(R_i, \alpha)$ defines the *support* of an alert *sequence* α for *rule* R_i in alert database S then $Supp_S(R_i, \alpha)$ = Number of tuples of the form $\langle tid; V = (v_1, v_2, \dots, v_d); seq \rangle$ in S such that $satisfies(R_i, V)$ (see the subroutine satisfies in Algorithm 11) is true and $\alpha \sqsubseteq seq$.

Since each alert log record is classified to one and only one sub-database so if there are p number of sub-databases (let j^{th} sub-database be defined as $subDB_j$) in S then $Supp_S(R_i, \alpha)$ can also be defined as:

$$Supp_S(R_i, \alpha) = \sum_{j=1}^p Supp_{subDB_j}(R_i, \alpha). \quad (6.3)$$

where $Supp_{subDB_j}(R_i, \alpha)$ is the *support* of α for the *rule* R_i in the j^{th} sub-database. But as discussed in Section 6.5.1, either all the log entries in the j^{th} sub-database satisfy the *rule* R_i or no entry satisfies the rule R_i depending upon the i^{th} bit (say $b_{j,i}$) of the bit-vector $\mathbf{b}_j$ associated with the j^{th} sub-database. So, $Supp_S(R_i, \alpha)$ can also be expressed as:

$$Supp_S(R_i, \alpha) = \sum_{j=1}^p b_{j,i} * Supp_{subDB_j}(R_i, \alpha). \quad (6.4)$$

Now since $b_{j,i}$ gives information on whether to calculate the *support* of the i^{th} rule in the j^{th} sub-database. So, if $b_{j,i}$ is present in Equation 6.4, R_i becomes redundant and it can be removed. Let $Supp_{subDB_j}(\alpha)$ be the *support* of α in j^{th} sub-database. Then $Supp_S(R_i, \alpha)$ can also be written as:

$$Supp_S(R_i, \alpha) = \sum_{j=1}^p b_{j,i} * Supp_{subDB_j}(\alpha). \quad (6.5)$$

From Equation 6.5 for a general *rule* R_i , we can conclude that if we first calculate the *support* of α in each sub-database independently then the *support* counts obtained for all the sub-databases can be used to compute the *support* of the *rules*. *Rules support* calculation in this way is efficient as we need not separately calculate the *support* of a *sequence* for each *rule*. Only p additional summations are required to compute the *support* of each *rule* after a single database scan. The value of p , in general, is very small as compared to the number of entries in the database so the overhead due to the computation of the *support* of the *rules* is very small.

6.5.2.1.2 Identification of Redundant Sub-database: Let α be an l length *violation pattern* and α' be an $l + 1$ length *violation pattern* constructed by adding an alert symbol to α . Using the *anti-monotone* property (refer to Section 6.2), if α is not frequent for a *rule* then α' also cannot be frequent for that *rule*.

Let X_α and X'_α be the set of all those *rules* for which α and α' are frequent respectively. Using the fact $\alpha \sqsubseteq \alpha'$ and the *anti-monotone* property.

$$X'_\alpha \subseteq X_\alpha \quad (6.6)$$

Let Y_j be the set of all such *rules* those are satisfied by the alert log entries in the j^{th} sub-database. All the alert log entries in the j^{th} sub-database satisfy *rule* R_i if the i^{th} bit $b_{j,i}$ in the bit-vector $\mathbf{b}_j$ is 1. So, Y_j can be expressed as:

$$Y_j = \{R_i : b_{j,i} = 1\} \quad (6.7)$$

Using Equations 6.6 and 6.7, we can also write

$$(X'_\alpha \cap Y_j) \subseteq (X_\alpha \cap Y_j) \quad (6.8)$$

The term $X_\alpha \cap Y_j$ in Equation 6.8 will result in the null set if α is not a frequent *violation pattern* for any of the *rules* satisfied by the alert *sequences* in the j^{th} sub-database. Using Equation 6.8, if the term $X_\alpha \cap Y_j$ is null then the term $X'_\alpha \cap Y_j$ will

also be null. Hence, if α is not frequent for any of the *rules* satisfied by the alert *sequences* in the j^{th} sub-database then α' also cannot be frequent for any of the *rules* satisfied by the alert *sequences* in the j^{th} sub-database.

To use this information, we calculate a bit-vector for each *violation pattern* to give the information about the *rules* for which it is frequent (the i^{th} bit in the bit-vector is 1 if α is frequent for the i^{th} rule). The bit-vector calculated for an l length *violation pattern* can be used to decide whether we need to calculate the *support* in a particular sub-database during the *support* calculation for the $l + 1$ length alert *sequences* constructed by adding an *item* to α . For this purpose, we perform the AND operation between the bit-vector corresponding to a sub-database and the bit-vector for α ; if it is 0 then we don't calculate the *support* in the sub-database for the $l + 1$ length alert *sequences* constructed by adding an *item* to α .

6.5.2.2 Violation Pattern Mining Algorithm

In this section, we propose a PrefixSpan based algorithm for efficiently mining the *violation patterns* for each global and semi-global *rule* using the basis discussed in Section 6.5.2.1. Similar to PrefixSpan algorithm, the frequent *violation patterns* for all the rules are recursively computed by constructing the projected databases.

The process of *violation pattern* mining is as discussed in Algorithm 12. The algorithm recursively finds the violation patterns that can be formed by adding an alert symbol to a known violation pattern α . To find such violation patterns, the algorithm uses the α -projected database ($S|_{\alpha}$), and the bit-vector ($\mathbf{bv}_{\alpha}$) associated with α . The α -projected database is the set of the suffixes with respect to α . It is of the same form as the original database in Table 6.4. Table 6.5 illustrates the α -projected database for $\alpha=a_1$ (refer to Section 6.2 for more details on the construction of α -projected database). The bit-vector $\mathbf{bv}_{\alpha}$ is an N bit length bit-vector associated with the *violation pattern* α . It determines the rules for which α is frequent. The i^{th} bit in $\mathbf{bv}_{\alpha}$ from the LSB corresponds to the i^{th} rule and its value is 1 if α is frequent

(or a *violation pattern*) for the i^{th} rule. Else the value of i^{th} bit is 0. For example, consider the 5 rules in Example 6.9 and suppose a *violation pattern* α is frequent for the rules 1, 2, and 5. Since there are total 5 rules so there will be total 5 bits in $\mathbf{bv}_\alpha$. And α is frequent for the rules 1, 2, and 5 so the 1st, 2nd and 5th bits in $\mathbf{bv}_\alpha$ from the LSB will be 1 and the bit-vector will be 10011.

Algorithm 12: Mining_Algo($\alpha, S|_\alpha, \mathbf{bv}_\alpha$)

Inputs:

α : A *violation pattern* (except the initial null value).

$S|_\alpha$: α projected database of the form as in Table 6.5.

$\mathbf{bv}_\alpha$: Bit-vector of the size N bits associated with α to give information about the rules for which α is frequent.

Algorithm:

1. Obtain the array of frequent alert symbols *frequent_symbols*, corresponding to the bit-vectors array *arr_BV*, and array of projected databases *arr_projected_DB* by calling *FrequentSymbols*($S|_\alpha, \mathbf{bv}_\alpha$).
 2. For each alert symbol in *frequent_symbols*
 - a). Form a new *violation pattern* by appending the alert symbol to α , let it be α' . The projected database with respect to α' will be same as obtained using Algorithm 13 for the current alert symbol.
 - b). Let the bit-vector and projected database (in *arr_BV* and *arr_projected_DB* respectively) corresponding to the current alert symbol be $\mathbf{bv}_{\alpha'}$ and $S|_{\alpha'}$ respectively. Call Mining_Algo($\alpha', S|_{\alpha'}, \mathbf{bv}_{\alpha'}$).
-

 Table 6.5: a_1 projected database

subDB ID	Bit-vector	Alert sequences in a_1 projected database
1	$\mathbf{b}_1=00101$	1. $\langle a_3, a_5 \rangle \langle a_3, a_5, a_7 \rangle \langle a_3, a_5, a_7, a_9 \rangle \langle a_5 \rangle \langle a_3, a_{10} \rangle \langle a_1, a_4 \rangle$
		2. $\langle _a_2 \rangle \langle a_3, a_5, a_7 \rangle \langle a_3, a_5, a_7, a_{10} \rangle$
2	$\mathbf{b}_2=01011$	1. $\langle _a_2, a_4 \rangle \langle a_3, a_6 \rangle \langle a_3, a_{10} \rangle \langle a_2, a_3, a_5, a_7, a_9, a_{10} \rangle$
4	$\mathbf{b}_4=01001$	1. $\langle _a_3, a_{10} \rangle \langle a_3, a_5, a_8 \rangle \langle a_4, a_5, a_6, a_{10} \rangle \langle a_1, a_2, a_3, a_5, a_8, a_{10} \rangle$
		2. $\langle _a_3, a_4, a_9, a_{10} \rangle$

Algorithm 12 is initialized with the following values $\alpha=\text{null}$, $S|_\alpha=S$, $\mathbf{bv}_\alpha=\text{Bit-vector with all } N \text{ bits}=1$, $W_\alpha=0$. To recursively generate the violation patterns, the algorithm uses the fact that the alert symbols that are frequent according to any of the rules (which is obtained by calling Algorithm 13) in the α -projected database can be added to α to form a new violation pattern. This is because the α -projected

database is the set of the suffixes with respect to α . So, a frequent alert symbol in the α -projected database corresponds to the violation pattern formed by adding the alert symbol to α in the original database (S). For example, if the alert symbol a_2 is frequent according to any of the five rules in Example 6.9 for the a_1 -projected database then the violation pattern $\langle a_1 \rangle \langle a_2 \rangle$ will be frequent in S . The set of frequent alert symbols is found by calling Algorithm 13 with the parameters $S|_\alpha$, $\mathbf{bv}_\alpha$, and W_α .

Algorithm 13 uses $S|_\alpha$, $\mathbf{bv}_\alpha$, and W_α as input to obtain the set of the frequent alert symbols that can be added to α to generate the new *violation patterns*. The algorithm consists of three steps. In the first step, we calculate the *support* of all the alert symbols in $S|_\alpha$. The *support* is calculated over all the sub-databases and stored in two dimension array *supp_alerts*. To efficiently calculate the *support*, we use the second basis for efficient *violation pattern* mining in Section 6.5.2.1. For this purpose, we take the AND operation between $\mathbf{bv}_\alpha$ and bit-vector corresponding each sub-database. We calculate the *support* from a sub-database only if the result of the AND operation between $\mathbf{bv}_\alpha$ and the bit-vector corresponding to that sub-database is not 0. In addition, during the *support* calculation, the projected databases with respect to all the alert symbols are also formed. Later in step 3, projected databases corresponding to the frequent alert symbols are passed to Algorithm 12.

In the third step in Algorithm 13, using the *support* of the alert symbols calculated for different sub-databases in array *supp_alerts*, we determine the alert symbols among all the alert symbols in I that are frequent according to any of the N rules. Initially, in Algorithm 13, for each alert symbol in I , a bit-vector $\mathbf{bv}'$ is initialized with value equal to 0. $\mathbf{bv}'$ stores the information about the rules for which the current alert symbol is frequent, which is calculated using the *support* counts. Next in Algorithm 13, we calculate the *support* of all N rules using Equation 6.5 discussed in the first basis in Section 6.5.2.1. An alert symbol is frequent for a rule (say i^{th}) if the *support* calculated for the i^{th} rule become greater than the *support* threshold $T[i]$ for

Algorithm 13: *FrequentSymbols*($S|_{\alpha}$, $\mathbf{bv}_{\alpha}$)

Output: Arrays containing frequent alert symbols (*frequent_symbols*), corresponding Bit-vectors (*arr_BV*), and projected databases with respect to alert symbols (*arr_projected_DB*).

Constants: T – Array of size N containing *support threshold* for N rules.

I –Set of all alert symbols, let there be total l alert symbols.

Variables: p – Number of sub-databases in $S|_{\alpha}$.

supp_alerts: Two dimensional array of size $l \times p$ to store the *support* of alert symbol in each sub-database, each element initialized to 0.

Step 1: Support calculation for alert symbols

for $j=1$ to p (*i.e. for all sub-databases in $S|_{\alpha}$*) **do**

 Calculate the value $\mathbf{b}=\mathbf{bv}_{\alpha}$ AND $\mathbf{b}_j$.

if $\mathbf{b} \neq 0$ **then**

 1. Calculate the *support* of the each alert symbol in the j^{th} sub-database.

 2. Update *supp_alerts*[k][j] with the *support* calculated for k^{th} alert symbol.

Step 2: A projected database with respect to each alert symbol is also generated during the *support* calculation in step 1.

Step 3: Support calculation for rules and finding frequent alert symbols

for $k=1$ to l (*i.e. for all alert symbols in I*) **do**

 Initialize a new bit-vector $\mathbf{bv}'=0$.

for $i=1$ to N (*i.e. for all N rules*) **do**

support=0.

for $j=1$ to p **do**

if i^{th} bit in $\mathbf{b}_j$ is 1 **then**

support=*support*+*supp_alerts*[k][j].

if *support* $\geq T[i]$ **then**

$\mathbf{bv}'=\mathbf{bv}'+1 \ll (i-1)$.

if $\mathbf{bv}' \neq 0$ **then**

 Add the current alert symbol to *frequent_symbols*, $\mathbf{bv}'$ to *arr_BV* and the projected database corresponding to the current alert symbol to

arr_projected_DB.

Return(*frequent_symbols*, *arr_BV*, and *arr_projected_DB*).

the i^{th} rule. Then the bit-vector $\mathbf{bv}'$ for an alert symbol is modified by assigning a bit equal to 1 at the positions corresponding to the *rules* for which the alert symbols are frequent. Finally, all frequent alert symbols, corresponding bit-vectors, and projected databases are returned to Algorithm 12.

Finally, in the second step in Algorithm 12, a new *violation pattern* is created for each frequent alert symbol by appending the frequent alert symbol to α . Let the *sequence* be α' for the current alert symbol. The projected database with respect to α' will be same as that of the projected database for the current alert symbol calculated in Algorithm 13. Next, Algorithm 12 is recursively called with the parameters α' , $S|_{\alpha'}$, and $\mathbf{bv}_{\alpha'}$, where $\mathbf{bv}_{\alpha'}$ is the bit-vector corresponding to the *violation pattern* α' (same as the bit-vector for the current alert symbol).

Both the basis used in the proposed algorithm ensure that the algorithm can find the *violation patterns* efficiently. As proved mathematically and experimentally in Section 6.7, the proposed algorithm performs significantly better as compared to the method by using sequential pattern mining algorithm separately on each *rule* database.

6.6 Discussion

In this section, we discuss about analysis of the violation patterns and practical issues with the logs based solution.

6.6.1 Analysis of the Violation Patterns

The objective of the analysis of the violation patterns is to identify and analyze the vulnerabilities and violations based on the violation patterns and their support counts obtained for global and semi-global *rules*. An analyst can analyze the violations in various possible ways as we discuss in the next paragraph. The discussions in the next paragraph are with respect to the several types of violations and their characteristics

(with respect to the alert symbols in Example 6.8 in Section 6.3) as shown in Table 6.6.

Table 6.6: Different types of violations in DRM

Type of violation	Effect of violation	Main characteristics of alerts
Compromise of DRM agent	DRM agent may not respect some or all the constraints.	<ul style="list-style-type: none"> – Dynamic count alert (a_8) will be generated every time content is used. – Static alert a_2 will be generated after the expiry date. – Static alert a_5 will be generated if the content is played in different region than in license. – Dynamic alert a_6 can also be generated if the region of content play is changed.
Modification of licenses	Parameters in the licenses may be modified to use the content more number of times, longer duration, or in any other region.	<ul style="list-style-type: none"> – If count parameter is modified then initially a dynamic count alert a_{10} will be generated followed by several static count alert a_7. – If time parameter is modified then several alerts a_1 or a_2 may be generated.
Replay attack	A copy of license may be stored in cache and may be restored later to use a content again.	<ul style="list-style-type: none"> – Only dynamic count alert a_{10} will be generated every time replay attack is performed.
Operating system vulnerabilities	Usage counters in the system may be modified using system registry, and system clocks may be drifted to a previous state.	<ul style="list-style-type: none"> – Dynamic alert a_{10} will be generated if usage counters are modified. – Static alert symbols a_7 may also be generated. – Dynamic alert a_3 will be generated if clocks are drifted.
Cloning of DRM agent	DRM agent may be cloned to use the same content multiple times.	<ul style="list-style-type: none"> – Dynamic alerts a_8, a_9, and a_{10} may be generated depending upon the content usage.

Following are some of the possible ways of the analysis of the violation patterns.

- (1) **Analysis of alert symbols in violation patterns:** The alert symbols in a frequent violation pattern can be used to decide the likelihood of occurrence of different types of violations. If an alert symbol that is present in a violation pattern is not likely to be generated from a particular type of violation/vulnerability then it is less likely that the violation pattern to be associated with that violation/vulnerability. For example, in Table 6.6, a violation pattern containing the alert symbol a_8 is more likely to be associated with the violations ‘compromise of DRM agent’ or ‘cloning of DRM agent’ as compared to other types of violations. This is because as shown in Table 6.6, the symbol a_8 is present only in the characteristics of the violations ‘compromise of DRM agent’

and ‘cloning of DRM agent’, but not in the characteristics of the other violations.

- (2) **Analysis based on likelihood of co-occurrence of different alert symbols:** Likelihood of the co-occurrence of different alert symbols may further help an analyst to determine the types of violations. It is based on the fact that some alert symbols have high probability of occurring together only for certain types of violations.

For example, as mentioned in column 3 of Table 6.6, the alert symbol a_{10} in a violation pattern may refer to four different types of violations namely, ‘modification of licenses’, ‘replay attack’, ‘operating system vulnerability’ and ‘cloning of DRM agent’. So, this information alone may not be sufficient for determination of the type of violation. However, as presented next the co-occurrence of other symbols together with a_{10} in the violation patterns may give additional information to determine the type of violation.

- In case of modification of licenses, as discussed in Table 6.6, the alert symbol a_7 will also be generated in the logs after the generation of the alert symbol a_{10} so the pattern $\langle a_{10} \rangle$ as well as the patterns $\langle a_{10} \rangle \langle a_7 \rangle$, $\langle a_{10} \rangle \langle a_7 \rangle \langle a_7 \rangle$, etc. will also be frequent. However, for other types of violations although the pattern $\langle a_{10} \rangle$ may be frequent, but the patterns $\langle a_{10} \rangle \langle a_7 \rangle$, $\langle a_{10} \rangle \langle a_7 \rangle \langle a_7 \rangle$, etc. are less likely to be frequent.
- In case of replay attack, only a_{10} can be generated every time replay attack is performed. So, the support count of the patterns made with only a_{10} (e.g. $\langle a_{10} \rangle$, $\langle a_{10} \rangle \langle a_{10} \rangle$, etc.) will be much more than the support count for the patterns (e.g. $\langle a_{10} \rangle \langle a_6 \rangle$, $\langle a_{10} \rangle \langle a_7 \rangle$) made by combination of a_{10} and any other symbol other than a_{10} .

The above analysis may help to distinguish between several types of violations, but some times similar patterns may be generated for two or more types of

violations. For example, similar patterns as discussed for ‘modification of licenses’ and ‘replay attack’ may also occur for ‘operating system vulnerability’ and ‘cloning of DRM agent’ respectively. The global and semi-global analysis as discussed in the next point may help in distinguishing between violations in such scenarios.

- (3) **Global and semi-global analysis:** Global and semi-global analysis of the violation pattern may help to further identify the type of violations/vulnerabilities. It may be particularly useful when two or more types of violations result in similar violation patterns. Then based on the support of the violation pattern for different global and semi-global *rules*, the analyst can determine the type of violation. Next, we present the use of global and semi-global analysis in distinguishing types of violations for which the violation patterns may look similar.

- In case of operating system vulnerability, the patterns generated may be similar to that of violations using modification of licenses (as discussed in point 2). However, the patterns related to this type of violation may only be frequent for the semi-global *rules* corresponding to the operating systems which are vulnerable. For example, if a certain version of Windows operating system is vulnerable then the patterns $\langle a_{10} \rangle$, $\langle a_{10} \rangle \langle a_7 \rangle$, $\langle a_{10} \rangle \langle a_7 \rangle \langle a_7 \rangle$, etc. may not be frequent (or may have relatively very less support counts) for the semi-global *rules* corresponding to other operating systems.
- In case of cloning of DRM agent, patterns similar to the replay attack ($\langle a_{10} \rangle$, $\langle a_{10} \rangle \langle a_{10} \rangle$, etc.) can be generated. However, these patterns are likely to be more frequent for the *rules* corresponding to some particular versions of DRM agent, which are more likely to be vulnerable. Whereas, replay attack may not be only due to vulnerability in DRM agent, but

may also be due to a operating system vulnerability or hardware based vulnerability. Based on the support of patterns for different rules, an analyst may decide the type of violation.

6.6.2 Practical Issues with Violation Detection Using Logs

The proposed method uses the usage data collected from consumers to do the violation detection. This presents following two concerns. First, whether it is feasible to collect the usage data in the form of log files (as in Table 6.2) from the consumers. Second, the privacy concerns associated with the data collection.

To retrieve the log files from a consumers' device, the device must be connected to consumer. However, it is not practically feasible that all the devices associated with a consumer remain connected to Internet. So, we follow the protocol of either collecting the logs from a device when the device becomes online, or collecting the logs from a device whenever consumer purchases the next content for the device. This may be particularly useful in the scenario where a consumer purchases contents for offline devices (e.g. DVD players not connected to Internet) through an online device (e.g. laptop). In such cases, a consumer may be asked to first transfer the logs from offline device to the online device for transfer of the logs to log collection center (LCC).

The privacy concerns associated with the data collection are that the collected log data contains viewing pattern of the consumer. It may reveal the sensitive information such as type of content, frequency of viewing the content, etc. However, in our proposed method, the consumer data may be collected in privacy preserving manner because of the fact that the aim of our proposed method is not to detect the individual violations by consumers. But, the aim of our proposed method is to analyze the log data for frequently occurring violations and vulnerabilities in the system. Thus, as we can analyze from Table 6.2, we do not need to collect any personal information (e.g. ID of the consumer, DRM agent ID or any other information that can directly reveal the consumer associated with logs) about consumers, but we only need to

collect the information about consumer attributes such as region, operating system or version of DRM agent. Furthermore, to achieve even more privacy during the data collection, one may use existing privacy preserving data collection and data analysis techniques [47] such as k-anonymity, l-diversity, etc. However, the main focus of our current work is violation detection and we would like to address these issues in detail in our future works.

6.7 Performance Analysis

In this section, we first analyze the theoretical performance to theoretically determine the efficiency of the proposed approach. Then we experimentally analyze the performance in terms of the violation pattern mining time, alert log classification time and storage space requirements.

6.7.1 Mathematical Complexity Analysis

We discuss the complexity of generation of new *violation patterns* using a general l length *violation pattern* α . We calculate and compare the complexity involved in generating the violation patterns by using the separate rules databases and our proposed approach in Algorithms 12 and 13. For this purpose, we first briefly discuss the approach of using separate rules databases. Then we present the complexity of calculation of the *support* of a general alert symbol β in α -projected database and determining whether β is frequent in α -projected database. Since α and β are general *violation pattern* and alert symbol respectively, so the analysis is general and applicable to all such pairs of α and β .

6.7.1.1 Separate Rules Databases Based Method

In the separate rule database generation based method, a separate rule database can be generated for each rule. An alert log entry is placed in all the rule databases that the entry satisfies. For instance, alert log entry with $tid=1$ in Table 6.3 satisfies

the rules R_1 and R_3 in Example 6.9 so in separate rule database based approach it will be added to both the rules databases corresponding to the rules R_1 and R_3 . The same process is used for all the alert log entries. Finally, each rule database is mined separately using the sequential pattern mining algorithms to obtain the frequent violation patterns for the rules according to the *support thresholds* associated with each rule.

6.7.1.2 Notations

Let us first define the notations required. Let S be the initial alert log database, S^i be the alert log database corresponding to the i^{th} rule, and $SubDB_j$ be the alert database corresponding to the j^{th} sub-database in our approach; $S|_\alpha$, $S^i|_\alpha$ and $SubDB_j|_\alpha$ be the α -projected databases corresponding to S , S^i , and $SubDB_j$ respectively.

6.7.1.3 Time Complexity Using Separate Rule Databases

Let the rules for which α is a frequent *violation pattern* are given by position of bits=1 in a bit-vector $\mathbf{bv}$ i.e. if α is a frequent *violation pattern* for the rule i then the i^{th} bit in $\mathbf{bv}$, bv_i , be 1. The number of alert sequences in α -projected alert database for i^{th} rule, $S^i|_\alpha$, will be equal to the number of alert sequences satisfying the i^{th} rule in the complete database $S|_\alpha$. It can also be expressed in the terms of bit-vectors associated with sub-databases in $S|_\alpha$ (there is no sub-database based division in rules database based approach but to correlate the complexity with our algorithm, we derive the complexity in terms of sub-databases in our algorithm). Let the bit-vector for the j^{th} sub-database ($SubDB_j$) be $\mathbf{b}_j$ and i^{th} bit (corresponding to the i^{th} rule) in it be given by $b_{j,i}$. All the alert log records in the j^{th} sub-database will also be present in the i^{th} rule database if the i^{th} bit, $b_{j,i}$, in $\mathbf{b}_j$ is 1. So, the total number of alert log entries in the i^{th} rules database are given by the following equation.

$$|S^i|_\alpha = bv_i * \sum_{j=1}^p b_{j,i} * |SubDB_j|_\alpha \quad (6.9)$$

where $|S^i|_\alpha|$, and $|SubDB_j|_\alpha|$ be the number of entries in $S^i|_\alpha$, and $SubDB_j|_\alpha$ respectively. The factor bv_i is to determine whether α is a *violation pattern* for the i^{th} rule. The value of $|S^i|_\alpha|$ will be 0 if $bv_i=0$ i.e. α is not a frequent *violation pattern* for the i^{th} rule. Else if $bv_i=1$, the rest of the terms will not be affected.

$|S^i|_\alpha|$ is also equal to the number of entries need to be traversed while calculating the *support* of β in $S^i|_\alpha$. So, if we perform the *violation pattern* analysis separately on each *rule* database then the total number of alert *sequences* (let $count_1$) need to be traversed for all the *rules* can be given by performing the summation over the values of i from 1 to N on $|S^i|_\alpha|$.

$$\begin{aligned} count_1 &= \sum_{i=1}^N |S^i|_\alpha| = \sum_{i=1}^N bv_i * \sum_{j=1}^p b_{j,i} * |SubDB_j|_\alpha| \\ &= \sum_{j=1}^p \sum_{i=1}^N (bv_i * b_{j,i}) * |SubDB_j|_\alpha| \end{aligned} \quad (6.10)$$

The term $bv_i * b_{j,i}$ in Equation 6.10 will be 1 for all the values of i such that the i^{th} bit is 1 in both the bit-vectors $\mathbf{bv}$ and $\mathbf{b}_j$. So, the value of $\sum_{i=1}^N bv_i * b_{j,i}$ will be equal to the number of bits=1 in $(\mathbf{b}_j \text{ AND } \mathbf{bv})$. It can also be expressed as $H(\mathbf{b}_j \text{ AND } \mathbf{bv})$, where $H(\mathbf{b}_j \text{ AND } \mathbf{bv})$ is the hamming weight (i.e. number of bits=1) of $(\mathbf{b}_j \text{ AND } \mathbf{bv})$.

$$count_1 = \sum_{j=1}^p H(\mathbf{b}_j \text{ AND } \mathbf{bv}) * |SubDB_j|_\alpha| \quad (6.11)$$

Now let the average time required to mine the alert symbol β in the j^{th} sub-database be avg_j . Then $time_1$ will be the total time required by mining the database corresponding to each *rule* individually.

$$time_1 = \sum_{j=1}^p avg_j * H(\mathbf{b}_j \text{ AND } \mathbf{bv}) * |SubDB_j|_\alpha| \quad (6.12)$$

6.7.1.4 Time Complexity Using Our Proposed Approach

Using our method, we only need to mine a sub-database only once and the j^{th} sub-database needs to be mined only when $\mathbf{b}_j \text{ AND } \mathbf{bv} \neq 0$ (see step 1 in Algorithm 13). But to get this information, we need to perform additional $p \text{ AND}$ operations. And finally in step 3 in Algorithm 13 we need to perform $N * p$ number of addition operation. If the logical operations can be done for k bits simultaneously, each logical AND operation takes t_1 time, and each addition operation takes t_2 time. Then the total time ($time_2$) can be denoted by the following equation.

$$time_2 = t_1 * p * \lceil N/k \rceil + t_2 * N * p + \sum_{j=1}^p avg_j * sgn(H(\mathbf{b}_j \text{ AND } \mathbf{bv})) * |SubDB_j|_\alpha \quad (6.13)$$

where sgn is the signum function. It will be 1 if the hamming weight $H(\mathbf{b}_j \text{ AND } \mathbf{bv}) \neq 0$ and 0 if $H(\mathbf{b}_j \text{ AND } \mathbf{bv}) = 0$.

But overhead due to AND operations and addition operations, in general, is very small due to small value of p as compared to the total number of *sequences*. So, the time required to perform the AND operations and summations can be removed from the expression. The $gain_\beta (=time1/time2)$ while calculating *support* for β in our case can be given by the following expression.

$$gain_\beta \approx \frac{\sum_{j=1}^p avg_j * H(\mathbf{b}_j \text{ AND } \mathbf{bv}) * |SubDB_j|_\alpha}{\sum_{j=1}^p avg_j * sgn(H(\mathbf{b}_j \text{ AND } \mathbf{bv})) * |SubDB_j|_\alpha} \quad (6.14)$$

Since $H(\mathbf{b}_j \text{ AND } \mathbf{bv}) \geq sgn(H(\mathbf{b}_j \text{ AND } \mathbf{bv}))$ for every value of j . So, the gain for our algorithm, in general, will be greater than 1 for all α and β values.

6.7.2 Experimental Performance Analysis

In this section, we analyze the performance of our proposed approach in terms of time required to find the frequent *violation patterns*, alert logs classification time, and storage space required. All the experiments were performed on Intel(R) core(2)

2.40 GHZ CPU with 2 GB RAM. All the programs were written in Java. We are not aware of any standard consumers log data or work in DRM that does violation detection using the logs collected from the consumers. The privacy concerns of the consumers and companies are reasons for non-availability of standard data. So, we generated our own simulation data for the experiments.

For the experiment purpose, we initially generated licenses and logs for 10,000 consumers for a single content. These were processed for generating the alert logs. The log entries of about 4,500 consumers were containing one or more than one anomalous activities. So, the alert logs consist of the *sequences* corresponding to 4,500 consumers. The alert log data was classified using four classification dimensions and total 11 alert symbols were used in the alert logs. The domain of these four classification dimensions were containing 10, 9, 5, and 3 values respectively.

6.7.2.1 Violation Pattern Mining Time

We perform the experimental analysis in terms of variation of *violation pattern* mining time with the number of *rules* for the number of *rules* between 1 and 50 by randomly generating the *rules*. As shown in Figure 6.1, our proposed method always performs better than the method by applying PrefixSpan algorithm on separate *rule* databases. Also, the relative performance becomes better as the value of N is increased. The performance gain as compared to method using separate *rule* databases is attributed to our bit-vectors based classification scheme and the basis mentioned in Section 6.5.2.1.

Comparing the gain with the gain obtained mathematically in Equation 6.14, the maximum value of $H(\mathbf{b}_j \text{ AND } \mathbf{b}_v)$ can be N bits so the maximum possible gain can also be about N times (50 for $N=50$ in our experiments). But it is only possible when the number of entries in each semi-global rule database becomes equal to the number of entries in the global rule database. However, the semi-global rule databases contain the entries corresponding to only a part of the complete system e.g. entries

Figure 6.1: Violation pattern mining time

corresponding to Asia and for Windows XP only, etc. Thus, the number of entries in the rules corresponding to sub-databases will always be smaller than the total entries in the global rule database. So, we observe the variation of $H(\mathbf{b}_j)$ between 1 and 14, with average about 8, for different sub-databases for $N=50$. Since $H(\mathbf{b}_j \text{ AND } \mathbf{bv}) \leq H(\mathbf{b}_j)$ so the observed gain about 3.5 times is in accordance with the average value of $H(\mathbf{b}_j)$. Note that $\mathbf{bv}$ is not a constant, it is different for different *violation patterns*.

In real world scenario, let us assume there are 1,000 contents, 40 rules for each content and violation analysis is done periodically once the new logs from 10,000 consumers using the content are collected. The violation pattern mining time will be about 4.6 days for the RDB approach and 1.1 days for our proposed approach. In practical scenario, it is desirable to have periodic analysis daily or over a few days period. So, the pattern mining time for both the methods is within the practically feasible limits. However, it is desirable to have shorter pattern mining time as it helps in doing the same task with less computing resources. Several possible ways of reducing the pattern mining time are discussed further in Chapter 8.

Figure 6.2: Alert logs classification time

6.7.2.2 Alert Logs Classification Time

The alert logs classification time is equal to the time required to classify all the log entries in the alert log database S into one of the sub-databases. Figure 6.2 shows the variation of alert log classification time with the number of *rules* N . From Figure 6.2, we can conclude that the alert log classification time increases almost linearly with N (the value is 0 for $N=1$ because we need not to do the classification if there is only 1 *rule*). But for all the values it is very small as compared to the *violation pattern* mining time (all the *violation pattern* mining times in Figure 6.1 are of the order 10^5 milliseconds as compared to the classification time only up to 3 milliseconds). In real world setting, the classification time for 1,000 contents assuming 40 rules for each content will be about 3 seconds, which is very small as compared to violation pattern mining time. So, the overhead due to the alert log classification remains very small.

6.7.2.3 Storage Space Required

Figure 6.3 compares the storage space required to store the alert logs in the proposed algorithm and to store the *rule* databases in the method using the separate *rule* databases. As shown in Figure 6.3, the storage space in our case is always less as

Figure 6.3: Storage space required

compared to the method using separate *rule* databases and also it remains almost constant with the increase in the number of *rules*. This is because only overhead (over the original storage space) on the storage in our case is due to the storage of indexes and bit-vectors associated with sub-databases, which in general is very small as the number of sub-databases are generally small as compared to the total log entries in database. Whereas in the *rule* database based approach, separate storage space is required for each *rule* database.

In real world setting, the storage space for 1,000 contents assuming 40 rules for each content will be about 1 gigabytes, which is easily feasible with storage space of several gigabytes and terabytes available in modern computers.

6.8 Conclusion

In this chapter, we proposed an approach of DRM violations detection by doing the analysis of consumer logs. We also presented the concept of global and semi-global violation analysis for effective analysis of consumer logs. However, doing the global and semi-global violation analysis using the traditional sequential pattern mining is inefficient. So, we proposed a bit-vectors based algorithm to efficiently mine the

global and semi-global violation patterns. As shown experimentally, our proposed *violation pattern* mining algorithm performs significantly better than the existing pattern mining algorithms and storage space required for our method is also small.

Chapter 7

Application of the Bit-vector Transform in Multimedia Social Networks

7.1 Introduction

Over the past few years with the advent of web 2.0 technologies, social media such as blogs, images, videos, etc. have become much popular due to their wide reach and easy accessibility. Social media depending on the type of media is generally shared over various types of social networks. In particular, the multimedia social networks (MMSNs) such as YouTube [14], Facebook [2] and, Flickr [4], etc. that allow the sharing of multimedia contents, such as images, videos, audios, etc. have become much popular. MMSNs allow easy sharing and spread of multimedia contents. People can easily share any multimedia contents from anywhere in the world and the shared contents can be accessed conveniently anywhere.

Although people can easily share their multimedia contents over MMSNs, there are often various privacy, security, and legal concerns [23, 72, 99, 160] due to online sharing of their personal multimedia contents. Privacy and security concerns are about unwillingness to disclose contents to a particular group of people, unintentional disclosure of identity and sensitive information, and threat of content modification for unethical purposes. For example, a cancer patient may wish to share the pictures related to his disease only with his doctors and family members [99]. Or due to the

threat of content modification, a person may wish to share his pictures only with trustworthy friends. Legal concerns are about unsuitability of content to a group of persons and restrictions associated with the copyrighted contents. For example, due to legal restrictions a person may wish to share a video containing violent scenes with the persons above some specific age only. Or due to the copyright issues, some contents may not be shared to the users in some countries. All privacy, security and legal issues may socially and economically harm a person. Therefore, people usually want to selectively share their multimedia contents.

Access control mechanisms [34, 35, 149] provide one way to selectively share the contents in MMSNs. By access control mechanisms a person can define the credentials [57] required by anybody else to access the contents shared by the person. Credentials are in the form of social relationship parameters such as friendship level, trust level, age, allowed countries, etc. [35, 69, 137]. Similarly, each individual has the credentials associated with his profile. The access control mechanisms work by verifying the credentials of an individual accessing a content with the credentials required to access the content. If an individual has sufficient credentials to access a content, then only he is allowed to access the content.

Access control mechanisms are simple and only require comparison of the credentials. But large number of content uploads and accesses make the access control task difficult. For instance, according to the statistics provided by Youtube [17] currently every minute about 48 hours of video is uploaded on Youtube and 3 billion videos are watched everyday on Youtube. Furthermore, large number of users (according to Facebook [4] there are more than 800 million active users are on Facebook), and rapid growth rate of users over MMSNs (according to Nielsen online [12], facebook recorded a growth rate of 228% over the period of February, 2008 to February, 2009) make the task of access control more difficult.

Also, as suggested in [30, 49, 113], users may require fine grained access control for their shared contents. For example, a user may wish to protect some words in

his blog, or specific parts in a picture from the specific persons. Another specific requirement for multimedia contents may be to allow the users to specify several discrete access control parameters instead of only the binary access parameters (in which someone either can or cannot access a content). For example, an access value of 0.5 may refer to the access to a low resolution version of the original picture. To satisfy these multimedia specific requirements, we need to treat each specific part of a content (or contents with different access parameters) as an independent content.

Thus, due to large growth rate of MMSNs, large number of content uploads and accesses, fine grained access control requirements, and multimedia specific access control requirements, the access control time may grow several times in MMSNs. This necessitates an efficient access control mechanism suitable for MMSNs. However, most of the existing works on access control in social networks focus only on new social network models [149], policy description [57, 137], and privacy and security [72, 113] issues. Authors in reference [164] discussed the issue of complexity of access control in social networks and requirements of an efficient access control solution, and proposed an efficient access control scheme. However, as we discuss in Section 7.2, the proposed efficient control solutions in [164] is user based (no access rights are associated with the contents so if a person is allowed to access then the person can access all the contents) not the content based, which may not be suitable for access control in present day MMSNs. This is because the content-based access control in MMSNs provides user more control over his/her shared contents. As an example, a user may wish to share family pictures only with his/her family members or may wish to share some videos only with some friends, but not with family members. To the best of our knowledge there is no existing work that discusses efficient content based efficient access control mechanism.

In this chapter, we focus on designing an efficient access control mechanism suitable for access control in present day MMSNs. For this purpose, we propose a bit-vector transform [29] based access control method. In the proposed approach we

model the contents, content access rights and users in MMSNs for efficient access control. The proposed approach is also able to provide the functionalities required in MMSNs such as multimedia content deletion and access rights modification using data in the bit-vector transform domain. The proposed access control mechanism is efficient due to efficient representation of data in the bit-vector transform domain and efficient operations on the data obtained after the bit-vector transform. Mathematical performance analysis and experimental results show the effectiveness and efficiency of our proposed mechanism.

The rest of this chapter is organized as follows. Section 7.2 discusses the related works. In Section 7.3, we present the access control model. In Section 7.4 our proposed access rights organization based access control mechanism is discussed. Performance analysis is in Section 7.5. Finally, the chapter is concluded in Section 7.6.

7.2 Related Works

In this section, we discuss about various access control mechanisms in social networks, and the requirement of efficient access control in social networks.

7.2.1 Access Control in Social Networks

In literature, a variety of access control mechanisms have been proposed for the social networks to address various aspects, such as new models of access control [34, 35, 57, 149], security and privacy [72, 113, 116], and policy description [137], etc. have been proposed in literature.

Carminati et al. in [34, 35] described the access control policies using friend of a friend (FOAF) [57] scripting language. FOAF provides the vocabulary to manage the access control based on the relationship between the individuals. Each relationship can be defined on a scale from 0 to a maximum value called the domain of the relationship (D) [149]. For example, the level of friendship can be defined using the values between 0 and 100. The authors in [35] defined access rule for an object

with ID oid using the tuple $(oid, cset)$, where $cset$ is the set of conditions $\{cond_1, cond_2, \dots, cond_m\}$ that must be fulfilled by an individual to access the object oid . All the conditions must be fulfilled by an individual who wants to access oid . There can be more than one rule defined for an object. In this case all the rules need to be verified one by one until a valid proof, if any, is obtained. The conditions in the access rules can be trust level, friendship level, country, name of persons, etc. [35, 69, 137].

In [149] authors proposed a digital rights management (DRM) [28, 64, 138] based solution for doing the access control in social networks. In the proposed mechanism digital contents are considered as physical good. The authors proposed solutions similar to that exist in physical world. For example, in physical world a book can be rendered to only one person at a time so the authors similarly allow the access of an e-book only to one person at a time. The proposed method is suitable for the copyrighted contents. However, it may limit the distribution capability of Internet.

Ianella in [137] emphasized the need of inter-operability between the contents shared between different social networks so that users need not share their contents multiple times. But it requires the same access control parameters for different social networks so that the same policy can be applied. It may be difficult as presently different existing social networks have different access policies. Furthermore, due to privacy and business related reasons social network providers may not agree for an inter-operable framework.

Various authors have discussed about different types of privacy concerns in the social networks and access control mechanisms to deal with the privacy concerns. Beato et al. in [113] discussed about the privacy threats from the social network providers. The authors proposed an access control model that is not regulated by the social network provider but it is under the control of users. But in order to use the proposed system users must know the technical details of the access control mechanism and cryptographic protocols, which may be very difficult practically. Loukides and Gkoulalas-Divanis in [72] discussed privacy concerns of a user resulting from other

users in the social network and social network provider. The main privacy concerns due to other users in social networks according to authors are their identity and content disclosure. Access control mechanisms mainly deal with the privacy concerns due to the content disclosure.

Authors in [36, 46] argued the use of a decentralized approach to online social networking to handle the privacy and inter-operability issues in present day social networks. Using the decentralized system, the users can privately share their own data by maintaining their data on their own trusted servers. The data of other users can be accessed using FOAF vocabulary [57]. But enforcement of decentralized access control policies may be difficult and inefficient in present day social networks due to relationship based access control in present day social networks [36]. Furthermore, although a decentralized approach may be more effective in terms of privacy but it lacks a good user interface and interaction (e.g. in the form of status updates) between users as provided by the present day social networks. A good user interface and interaction are some of the main reasons of the popularity of the present day social networks over other mediums of sharing the contents such as email. To tackle these problems, Baden et al. in [133] presented a social network model called Persona using public key cryptography and attribute based encryption. Persona uses a decentralized approach yet it is able to provide the interface and interaction as provided by the present day social networks. For this purpose, it uses a group based content sharing, where users in a group can access any content shared by a user in the group for the group. However, the proposed mechanism may involve a large overhead due to the cryptographic operations during formation of group, or removal of any member from the group. Furthermore, using decentralized approach users have to pay for the storage and computations involved as opposed to free storage and computation power provided in the present day social networks, which make them much popular.

Shehab et al. in [116] discussed the privacy infringement due to third party applications on the social networks. These applications require user data but users

often don't know how much data is being disclosed to the third party applications. So, authors proposed a finite state machine based approach for providing data access to the third party applications.

Although various access control mechanisms exist for different privacy requirements, in this work we mainly focus on access control mechanisms for content disclosure to other users in the social networks.

From the access control models studied in this section, we found that the access control model proposed by Carminati et al. in [34,35] is more suitable for the current social networks. So, we use an extension of the social network model proposed by Carminati et al. as the basis, which we discuss in Section 7.3.

7.2.2 Efficient Access Control

Various authors recently have discussed the need of finer and efficient access control requirements [30,36,49,164] in various scenarios. In [30], Tootoonchian et al. proposed an inter-operability framework for different social networks. The motivation behind that is that users are generally on different social networks and moving a content from one social network to another social network is difficult. In such cases, due to the different policies in different social networks, finer and more number of access control parameters may be needed. Gates in [49] have stated the four key requirements while designing an access control system for social networks. One of the requirements according to Gates is the fine grained access control i.e. access control mechanisms should also be able to manage the access control for fine grained details inside the object. For example, a user may wish to protect some words in his blog, or specific parts in a picture from specific persons.

Authors in [164] discussed the effect of fine grained access control on rising access control time complexity. In [164] authors proposed a trusted distance classifier based scheme to do the efficient access control. In the scheme, for each user's profile, other users are classified into three categories viz. acceptance, attestation, and rejection.

Users in the acceptance zone are accepted immediately and the users in the rejection zone are rejected immediately. Requests in the attestation zone require additional authorization, which is done by the attesters designated by the user. The proposed scheme in [164] can reduce the access control time as it eliminates the need of comparison of the attributes associated with users accessing the contents with the attributes associated with the contents. But the scheme may not be suitable for access control in the present day social networks because of the following two reasons. First, the proposed access control mechanism does the access control on the basis of users not the contents but practically users may have different access control requirements for different contents. Second, for the users in the attestation zone manual attestation by the designated attesters is required, which may cause significant delay and inconvenience to users. The bit-vector transform based method presented in this chapter does efficient access control and it is also suitable for present day social networks.

7.3 Model of Access Control

In this work, we model the system by modifying the access control model proposed by Carminati et al. in [34, 35]. In the access control model in [34, 35], an access rights vector is associated with each content. Let the access rights vector (R^c for content c) be given by: $R^c = \{r_1, r_2, \dots, r_M\}$, where $r_1, r_2, \dots$, and r_M are M access rights. Each access right is in the form of an interval and has a domain of all possible allowed values. If the domain of the j^{th} access right is given by D_j then $r_j \in D_j$. Let the attribute vector of an individual accessing the content be defined as $A = \{A_1, A_2, \dots, A_M\}$, where $A_1, A_2, \dots$, and A_M are M attribute values. Each attribute in the attribute vector corresponds to the respective access right in the access right vector (R^c). Note that some attributes such as trust level, friendship level, etc. in the attribute vector of an individual are not fixed and dependent upon the profile individual is visiting. Such attributes need to be obtained first during the access control process.

The individual can access the content c if $A_j \in r_j, \forall j \leq M$. When the profile of an individual I_1 is visited by another individual I_2 then all the contents to which I_2 is authorized to view are identified and displayed, as shown in Example 7.10.

Example 7.10 *Let an individual I_1 has 3 contents, c_1 , c_2 and c_3 in his profile with access rights vector R^{c_1} , R^{c_2} and R^{c_3} given as (with access rights as friendship level (F), trust level (T) and age (AG), each having domain 0 to 100):*

$$R^{c_1} = \{F=(40, 100], T=(30, 80], AG=(0, 100]\}$$

$$R^{c_2} = \{F=(24, 60], T=(63, 100], AG=(18, 100]\}$$

$$R^{c_3} = \{F=(40, 80], T=(70, 100], AG=(16, 100]\}$$

Access right vector R^{c_1} states that the content c_1 can be accessed by a person having the friendship level between 40 and 100, trust level between 30 and 80, and age between 0 and 100. Similarly, R^{c_2} and R^{c_3} can be interpreted. Let an individual I_2 with the attribute vector $A=\{50, 40, 25\}$ visits the profile of I_1 . Now, the attributes 50, 40, and 25 are within the ranges $(40, 100]$, $(30, 80]$, and $(0, 100]$ respectively. Therefore I_2 can access c_1 . But for c_2 and c_3 , trust level=40 of the individual I_2 is not within the access rights ranges $(63, 100]$ and $(70, 100]$ respectively. So, I_2 cannot access the contents c_2 and c_3 , and only the content c_1 will be displayed.

The model presented above uses only a single range for each access right, but the access rights in present day social networks may be discrete and may consist of multiple discrete ranges. For example, a user may allow the access of a content in few different regions, which may not be represented with a single range. So, to reflect the existing social networks, instead of directly using the same model in [34, 35], we use a social network model in which access rights can have multiple discrete ranges. However, in the rest of this chapter, for the sake of simplicity, we first present the case of all access rights with a single range and then provide the extensions required to accommodate the multiple access rights ranges.

7.4 Proposed Efficient Access Control Mechanism

In this section, we present our proposed bit-vector transform based access rights organization and an access control mechanism using the data after the organization. We also propose the mechanisms (in the bit-vector transform domain) for the deletion of contents from the user's profile and modification of access rights associated with the contents.

7.4.1 Access Rights Organization

Let each content has M access rights associated with it. We organize the access rights associated with the contents into an M dimensional space. As shown in Figure 7.1, each dimension corresponds to a particular access right and divided into several elementary ranges. Each elementary range has a bit-vector associated with it. A bit-vector is a string of bits '0' and '1'. The number of bits in a bit-vector is equal to the number of contents in the profile of the user. Each bit in a bit-vector corresponds to a particular content. Thus, if a user has N contents in his profile then the length of bit-vectors will be N bits; and the n^{th} bit in the bit-vector will correspond to the n^{th} content. For example, Figure 7.3(e) represents the dimension for friendship level (F) in Example 7.10. There are five elementary ranges viz. 0 to 24, 24 to 40, 40 to 60, 60 to 80, and 80 to 100. Since there are three contents in Example 7.10 so in Figure 7.3(e) each elementary range has a 3 bits length bit-vector (shown in rectangular box) associated with it.

Initially, each access right's dimension is assumed to contain only one elementary range in the entire domain $D_j = (D_j^{min}, D_j^{max}]$ of the j^{th} access right. D_j^{min} and D_j^{max} are the minimum and maximum values respectively in the j^{th} access right's domain. No bit-vector is assumed to be present for the initial elementary range (the first bit in a bit-vector is inserted when the first content is inserted). Two steps viz. access right's range insertion and modification of bit-vectors are required for the insertion of a new content. In the first step, the range of access rights in the contents is inserted

Figure 7.1: Illustration of elementary ranges and bit-vectors

along the respective access right's dimension. In the second step, existing bit-vectors are modified by appending a new bit to them.

1) Access Right's Range Insertion: Let the range of the j^{th} access right for the n^{th} content be $[a_j^n, b_j^n]$. We call $AR_Insert(a_j^n)$ and $AR_Insert(b_j^n)$ for the insertion of $(a_j^n, b_j^n]$. The process $AR_Insert(x_j)$ is as defined in Algorithm 1. During the insertion process the point x_j is inserted in the numerical order if it was not already present along the j^{th} access right's dimension. If x_j is inserted then the same bit-vector as the elementary range in which it is inserted is assigned to both the elementary ranges formed after the insertion. The same process is used for all M dimensions.

Figure 7.2 (a) shows the initial range of friendship dimension in Example 7.10. To insert the access right's range (which is $(40, 100]$) in the first content initially Algorithm 1 is called with the parameter $x_j=40$. Since $x_j=40$ was not already present along the access right's range in Figure 7.2 (a) so $x_j=40$ inserted in the numerical order between 0 and 100 as shown in Figure 7.2 (b). Since there was no bit-vector present for initial elementary range $(0, 100]$ so no bit-vector is assigned to newly formed elementary ranges viz. $(0, 40]$ and $(40, 100]$. Next, Algorithm 1 is called with the parameter $x_j=100$. Since 100 is already present along the access right's dimension in Figure 7.2 (b) so we need not insert it again. The access right's dimension remains same as in Figures 7.2 (b) and (c) after calling the algorithm with $x_j=100$.

Figure 7.2: (a). Initial friendship(F) access right dimension. (b). Access right dimension after insertion of the end point 40 in the first content.(c). Access right dimension after insertion of the end point 100 in the first content.

Algorithm 1 $AR_Insert(j, x_j)$

Input: j : A numerical value between 1 and M to determine the access right dimension.

x_j : A numerical value to be inserted along the j^{th} access right's dimension.

Output: Modified j^{th} access right's dimension.

Procedure:

Search x_j along the j^{th} dimension.

if x_j is already present along the j^{th} access right's dimension **then**

 Exit.

else

1. Insert x_j in numerical order along the j^{th} access right's dimension.
2. Assign the same bit-vector as previous elementary range to both the newly generated elementary ranges formed due to the insertion of x_j (as in Figures 7.3(b) after insertion of points $x_j=24$ and $x_j=60$ in Figure 7.3(a)).
3. Return the modified j^{th} access right's dimension.

end

2) Modification of Bit-vectors: Each bit in a bit-vector corresponds to a particular content. Thus, if $n - 1$ contents are present then the bit-vectors will be of the length $n - 1$ bits, which needs to be changed to n bits while inserting a new content. Bit-vectors of all the elementary ranges are modified by appending a bit equal to 0 or 1 to them. We call the Algorithm 2, $BV_Modify(j, a_j^n, b_j^n)$, to modify the bit-vectors. The algorithm appends a bit equal to 1 in the bit-vectors corresponding to all the elementary ranges between a_j^n and b_j^n and 0 to rest of the bit-vectors. Thus, in the bit-vector transform domain if the n^{th} bit for a content is 1 in an elementary range then the elementary range will be within the respective access right's range in

Figure 7.3: Illustration of the content insertion process

the n^{th} content. This is used as a basis in access control mechanism in Section 7.4.2.

Algorithm 2 $BV_Modify(j, a_j^n, b_j^n)$

Input: a_j^n, b_j^n : End points of the j^{th} access right's range in the n^{th} content.

Output: Modified j^{th} access rights dimension.

Procedure:

- (1) Append a bit equal to 0 to the LHS of all the bit-vectors present between D_j^{min} and a_j^n along the j^{th} access right's dimension.
 - (2) Append a bit equal to 1 to the LHS of all the bit-vectors present between a_j^n and b_j^n along the j^{th} access right's dimension.
 - (3) Append a bit equal to 0 to the LHS of all the bit-vectors present between b_j^n and D_j^{max} along the j^{th} access right's dimension.
 - (4) Return the modified j^{th} access right's dimension.
-

Figure 7.3(a) shows the friendship access right's dimension after the insertion of the initial bit-vectors in Figure 7.2(b). Figures 7.3(b) and 7.3(d) show access right's range insertion steps for the second and third content respectively. 7.3(c) and 7.3(e) show bit-vector modification step after the insertion of second and third content respectively.

If multiple discrete ranges are present for an access right associated with a content then insertion process is modified slightly. Let d number of discrete ranges viz. $(a_{j_1}^n, b_{j_1}^n]$, $(a_{j_2}^n, b_{j_2}^n]$, ..., $(a_{j_d}^n, b_{j_d}^n]$ are present for the j^{th} access right. In the first step, all the end points of all the ranges are inserted along the j^{th} dimension using Algorithm 1. For the bit-vector modification step, initially Algorithm 2, $BV_Modify(a_{j_1}^n, b_{j_1}^n)$

is called (for the first range). For doing the bit-vector modification using all other ranges, the bit-vector modification step is slightly modified. In the modified algorithm for the i^{th} ($1 < i \leq d$) range, we don't insert a new bit but change the n^{th} bit to 1 in all the bit-vectors present between a_{ji}^n and b_{ji}^n . This ensures that the size of the bit-vectors remain n bits if n contents are inserted, irrespective of the number of ranges.

7.4.2 Access Control Mechanism

Let the attribute vector of an individual accessing the contents be given by $A = \{A_1, A_2, \dots, A_M\}$. Initially, for each value of j , a search is made for A_j along the j^{th} dimension. Let the elementary range that contains A_j be given by E_j and the respective bit-vector be given by B_j . The contents for which the j^{th} access right's range contains E_j will have their corresponding bit will be equal to 1 in B_j (see Section 7.4.1). Thus, the j^{th} access right's range will also contain A_j as $A_j \in E_j$.

A content (say n^{th}) can be accessed by an individual if he satisfies all M access rights conditions to access the content. Thus, the n^{th} bit must be equal to 1 in all M bit-vectors searched over all access right's dimensions. This information can be obtained by taking an AND operations among the bit-vectors obtained for all the access right dimensions (if a particular bit is 1 in all the bit-vectors then that bit will also be equal to 1 in the bit-vector obtained after the AND operations). As shown in Equation 7.1, final bit-vector B is obtained by performing the AND operations among all the bit-vectors.

$$B = B_1 \wedge B_2 \wedge \dots \wedge B_M \quad (7.1)$$

Finally, we scan the final bit-vector B to identify the contents bits corresponding to which are equal to 1. To identify the n^{th} ($1 \leq n \leq N$) bit, we perform an AND operation of B with a bit-vector containing n^{th} bit 1 and other bits 0. If the result is 1 then the n^{th} content can be accessed.

Figure 7.4: (a). Friendship dimension (F), (b). Trust dimension (T), and (c). Age dimension (AG) after insertion of contents in example 7.10.

Example 7.11 Consider the individual I_2 in Example 7.10 accesses the contents shared by I_1 . The attribute vector of I_2 is $A=\{50, 40, 25\}$. Figure 7.4 shows friendship (F), trust (T) and age (AG) dimensions after the insertion of all three contents in Example 7.10. The first attribute (having value 50), belongs to the elementary range 40 to 60 in Figure 7.4(a). Hence, $B_1=111$ in this case. Similarly, for other two access rights bit-vectors can be calculated (using Figures 7.4(b) and 7.4(c)) and are given as: $B_2=001$ and $B_3=111$. In this case, $B=111\wedge 001\wedge 111=001$. Since, only the bit corresponding to c_1 is 1 therefore I_2 can only access the content c_1 , which is in accordance with Example 7.10.

The access control mechanism is also valid if multiple ranges are possible for an access right as the bit corresponding to a content is 1 in the bit-vectors corresponding to all the ranges present for the content.

7.4.3 Modification of Access Rights

The access rights associated with the contents can be modified by the content owners. These changes must be reflected in the bit-vector transformed domain. Let the user changes a range in the j^{th} access right associated with the n^{th} content from $[a_j, b_j]$ to $[a'_j, b'_j]$. Then Algorithm 3, $AR_Modification(n, j, a_j, b_j, a'_j, b'_j)$, is used to reflect the change in bit-vector transform domain. In the algorithm, we first change the n^{th} bit in every bit-vector between a_j and b_j to 0. After that the algorithm calls

$AR_Insert(a'_j)$ and $AR_Insert(b'_j)$ (Algorithm 1) to insert the points a'_j , and b'_j if they are not already present. Then it sets n^{th} bit in all the bit-vectors between a'_j , and b'_j to 1. Finally, the algorithm removes a_j and/or b_j if no other content shares the end points a_j and/or b_j using Algorithm 4. This is to reduce the search time and storage space by removing any redundant point along the access right's dimension.

Algorithm 3 $AR_Modification(n, j, a_j, b_j, a'_j, b'_j)$

Input: a_j and b_j : Original end points of the j^{th} access right's range for the n^{th} content.

a'_j and b'_j : New end points of the j^{th} access right's range for the n^{th} content.

Output: Modified j^{th} access right's dimension.

Procedure:

- (1) Make the n^{th} bit to 0 in all the bit-vectors between a_j and b_j .
 - (2) Call $AR_Insert(a'_j)$ and $AR_Insert(b'_j)$.
 - (3) Set the n^{th} bit to 1 in all the bit-vectors between a'_j and b'_j .
 - (4) Call $delete_point(j, a_j)$.
 - (5) Call $delete_point(j, b_j)$.
 - (6) Return j^{th} access right's dimension.
-

Algorithm $delete_point(j, x_j)$ first obtains the bit-vectors for the elementary ranges just before and after x_j along the j^{th} access right's dimension. If both are equal this implies that no content has x_j as an end point for the j^{th} access right. Thus, the point x_j can be deleted.

7.4.4 Deletion of a Content

In social networks, a user can delete a content from his profile. The Algorithm 5, $delete_content(n)$, is used to delete the n^{th} content from a user's profile. Two steps are required for the deletion of a content. In the first step, the n^{th} bit in all the bit-vectors is made 0. Next, if no other content shares the end points a_j and/or b_j then remove a_j and/or b_j from the j^{th} dimension. Algorithm 4 is used to remove a_j and/or b_j .

Algorithm 4 *delete_point(j, x_j)*

Input: x_j : The point to be deleted from the j^{th} access right's range.

Output: Modified j^{th} access right's dimension.

Procedure:

if $x_j = D_j^{min}$ **or** $x_j = D_j^{max}$ **then**
 | Exit.

else

 | Let the bit-vectors for the elementary ranges just before and after x_j be given by
 | B_{low} and B_{high} respectively.

if $B_{low} = B_{high}$ **then**

 | Delete x_j from the j^{th} access right's range.

Return j^{th} access right's dimension.

Algorithm 5 *delete_content(n)*

Input: n : ID of the content to be deleted.

Output: Modified version of all M access rights' dimensions.

Procedure:

for $j=1$ **to** M **do**

 (1) Make the n^{th} bit in each bit-vector equal to 0.

 (2) Let the range of j^{th} access right in the n^{th} content be given by $[a_j, b_j]$.

 (3) Call *delete_point(j, a_j)*.

 (4) Call *delete_point(j, b_j)*.

end

7.4.5 Efficient Data Representation

In the bit-vector transform domain, end points of elementary ranges along each access right dimension are present in the increasing order of magnitude. This suggests use of efficient data structures such as binary search trees (BST), AVL trees, red black trees [20] etc. to represent each access right dimension. As AVL trees are more search efficient [20, 41] so we use an AVL tree to represent each access right's dimension. A separate AVL tree is designed for each access right's dimension. In this section, we briefly describe the procedure for insertion and searching of data in AVL trees.

There are two types of nodes in each AVL tree: internal nodes and leaf nodes. As shown in Figure 7.5, the internal nodes (in circles) in each AVL tree stores the points along the access right's dimension. Whereas, leaf nodes (in rectangles) store the bit-vector corresponding to an elementary range determined by the values stored in internal nodes. For example, in Figure 7.5(b) the bit-vectors 00, 10, 11 and 01 (from left to right) corresponds to elementary ranges 0 to 24, 24 to 40, 40 to 60, and 60 to 100 respectively.

Two steps (corresponding to access rights insertion and bit-vector modification steps in Section 7.4.1) are required for the insertion of access rights in an AVL tree. The first step is access right's range insertion and the second step is bit-vector modification. During the access right's range insertion process, the end points of the access right's ranges are inserted one by one using the normal insertion process in AVL trees [20]. During the bit-vector modification step, an in-order traversal (recursive processing of the tree in the order: the left sub-tree, root, and right sub-tree) in the AVL tree is performed. If the range of the j^{th} access right is given by $[a_j, b_j]$ then all the bit-vectors in between a_j and b_j are appended with a bit equal to 1 and rest of the bit-vectors are appended with a bit equal to 0. Figure 7.5(a), 7.5(b), and 7.5(c) illustrates an AVL tree designed for the access right dimension shown in Figure 7.3(a) 7.3(c), and 7.3(e) respectively.

Figure 7.5: Insertion process in AVL tree

For the access control mechanism discussed in Section 7.4.2, a search along each access right's dimension is required. Following algorithm (Algorithm 6) is used to search along the bit-vector along the j^{th} dimension (let the j^{th} attribute in the attribute vector of a person accessing the content be given by A_j).

Algorithm 6 *Search(j, A_j)*

Input: j : Dimension to search.

A_j : A numerical value to be searched along j^{th} dimension.

Output: Bit-vector corresponding to A_j .

Procedure:

- (1) Set the root node as current node.
 - (2) Compare the value A_j with the value stored in the current node. If A_j is greater than the value stored in the current node then assign the right child of the current node as current node. Else assign the left child of the current node as current node.
 - (3) If current node is a leaf node then bit-vector stored in the node is the bit-vector required. Else go to step 2.
-

7.5 Performance Analysis

In this section, we analyze both mathematical as well as experimental performance of our proposed method in terms of the time required to access the contents, storage space, and insertion time. For experimental performance analysis, we use the relational database (RDB) model as reference for storing the access rights. This is

because most of the literature [34,35] and existing social networks [165] use relational database for storing the access rights. In RDB model, access rights are stored in the tabular format in the relational database. Each row in the table corresponds to a particular content and each column corresponds to a particular access right. In this section, we first present the reason for efficient access control and then we present the mathematical and experimental analysis.

The proposed method of access rights organization using bit-vector transform outperforms the RDB model of access control because of following reasons. First, the maximum possible elementary ranges along each access right's dimension cannot be more than maximum possible values in the domain of that access right. Thus, in our proposed arrangement the search space is limited to the number of values in the domain of the access right. Second, the search time along each access rights dimension is further reduced as each dimension can be represented using AVL tree structure. For example, if there are 1000 contents with a person. To verify the credential of an individual accessing the content (assuming a single access right), 2000 comparisons are needed (assuming only single range with each access right) if we use RDB based access control model. If the domain of that particular access contains 100 values then our method requires 100 comparisons if we do not represent access rights with an AVL tree. Only 8 to 10 comparisons are needed for the same purpose when we represent the dimension using an AVL tree. In case of multiple access rights associated with each content, our method requires $M-1$ more AND operation between the bit-vectors obtained for each access right's dimension, where M is the number of access rights associated with each content. But still our method remains much more efficient. As discussed in mathematical and experimental analysis, the relative gain becomes even more for the case of multiple ranges per access right.

7.5.1 Mathematical Performance Analysis

To analyze the mathematical performance, we assume that there are N number of content present and each content is defined with M access rights. For ease of un-

derstanding, we derive the expressions for access rights with single range only. We discuss the effect of multiple discrete ranges on performance in Section 7.5.2.

1) Access Time Required: Let after insertion of N contents, j^{th} access right's dimension is divided into n_j number of elementary ranges. The complexity for the searching step will be $O(\log_2(n_j))$ for the j^{th} access right's dimension. If there are N contents then each bit-vector will be N bits long. It can be represented using $\lceil N/k \rceil$ integers in computer, where k is the number of bits in the data format (e.g. integer or long) using which bit-vectors represented in computers (N bit long bit-vectors are implemented using a linked list containing $\lceil N/k \rceil$ elements). Further, $M - 1$ AND operations will be required to get the final bit-vector. Thus, the time required for the second step will be $O((M - 1) * \lceil N/k \rceil)$. Finally, the bit-vector obtained is scanned to find the bits that are equal to 1 for finding the contents which a person can access. This step requires N number of AND operations. Thus, the total time required can be given as $k_1 * \sum_{j=1}^M (\log_2(n_j)) + k_2 * ((M-1) * \lceil N/k \rceil + N)$. Here, k_1 and k_2 are arbitrary system dependent constants.

2) Space Complexity: The space complexity is given by the space required to store the bit-vectors and end points of elementary ranges. Each bit-vector consists of N bits and there n_j number of bit-vectors along the j^{th} access rights dimension. Thus, the space required to store the bit-vectors will be $\sum_{j=1}^M n_j * N$ bits or if bits are represented in integer format then the space required will be $\sum_{j=1}^M n_j * \lceil N/k \rceil * k$. To store the end points of elementary ranges, we require to store n_j integers along the j^{th} access right's dimension. Thus, the space required to store the elementary ranges will be $\sum_{j=1}^M n_j * k$. In addition, $2 * N * M * k$ bits of space is required to provide an interface for the access rights. Thus, the total space required will be $\sum_{j=1}^M n_j * (\lceil N/k \rceil * k + k) + 2 * N * M * k$ bits.

3) Insertion Time complexity: The first step of the insertion process is to insert the both end points of all access rights. The time required for this step is $k_1 * 2 * \log(n_j)$ for the j^{th} access right's dimension. The second step is the bit-vector

Figure 7.6: Access control time

modification. Let the range of j^{th} access right be given by $[a_j, b_j]$. Let the number of elementary ranges between a_j and b_j be given by n'_j . Then the time required for the second step would be $k_2 * \sum_{j=1}^M n'_j$. Thus, the overall time complexity would be $2 * k_1 * \sum_{j=1}^M \log(n_j) + k_2 * \sum_{j=1}^M n'_j$.

7.5.2 Experimental Performance Analysis

All the experiments were performed on Intel(R) core(2) 2.40 GHZ 32-bit CPU with 2 GB RAM. We perform the experiments for access time required for the number of concurrent profile accesses in the system, assuming that on an average 1000 contents are present in each user's profile. For storage space and insertion time complexity, we perform experiments for up to 10,000 contents (N) in the user's profile. For the experiments purpose, we assume that the number of access rights associated with the contents is between 7 and 15. The number of values in the domain of access rights is randomly chosen in between 10 and 100. The value of k is 32 in our case as we represent bit-vectors using integer format. We compare the performance after considering both single ranged and multiple ranged access rights. For multi-ranged access rights we assume that 3 access rights out of all access rights can have multiple discrete ranges. To show the effect of multi-ranged access rights, we do the experiments by assuming up to 4 ranges for these access rights.

Figure 7.7: Storage Space Required

1) Access Time Required: Figure 7.6 shows the comparison of access time performance of our approach with RDB based approach for the case of single range and multi-ranged access rights. As shown in Figure 7.6, our approach outperforms the RDB based approach for all the values of concurrent content accesses. It can also be observed that, in contrast to RDB approach, our approach takes almost same time in both single-ranged and multi-ranged access rights cases. This is because in our method number of *AND* operations required are same for both the cases. The only difference is searching along the access right dimensions. The search time along each access right's dimension also becomes almost same as the number of elementary ranges (n_j) reaches to maximum possible elementary ranges in access rights' domain. Thus, the proposed mechanism is more suitable for present day social networks, which may have access rights in the form of multiple discrete ranges.

2) Space Complexity: The storage space is given by the space required to store the bit-vectors, end points of elementary ranges and access rights in the original RDB form (required for providing an interface to users). Figure 7.7 compares the performance in terms of storage space required to store N contents in a profile. Although, the storage of access rights in bit-vector transform takes lesser time but additional space is required to store the access rights in the original RDB form along with the contents. Overall overhead for the case of single-ranged access rights is about 60-70%.

Figure 7.8: Insertion time complexity

The overhead is relatively smaller for the case of multi-ranged access rights due to finite possible access rights' ranges.

3) Insertion Time Complexity: Figure 7.8 shows the insertion time complexity for insertion of N^{th} content when $N - 1$ contents are already present. As derived in Section 7.5.1, the insertion time depends on n_j and n'_j , it does not depend on N . So, the curve initially rises until maximum elementary ranges are reached then it becomes almost constant. The overhead due to the insertion is very less. For most of the values of N , it is much less than the time required to do the access control with RDB approach. In particular, we observe that it is about 3 times to 30 times less for $N=1000$ to $N=10000$ (see Figures 7.6 and 7.8). In case of multi-ranged access rights, the insertion time is proportional to the number of ranges. However, we need to insert the contents only once. And in general, every content in MMSNs is accessed several times so the overall effect of insertion complexity on the system performance would be small.

7.5.3 Practical Significance of the Results

To demonstrate the practical significance of the results, we use the YouTube statistics provided by YouTube in reference [17]. According to statistics provided by YouTube about 48 hours of video is uploaded to YouTube every minute and about 3 billion

videos are accessed everyday. Assuming the average length of each video uploaded over YouTube be 5 minutes, about 10 videos are being uploaded on YouTube every second, and 35,000 videos are being accessed every second. Our proposed approach requires about 200 ms to access 10,000 contents so the access of 35,000 contents per second can be easily managed. Insertion of access rights with 10 videos require about 3 seconds (300 milliseconds per video) using the insertion mechanism, which is very close to desirable performance.

In terms of storage space, our approach requires about 1.5 MB space for storing the access rights associated with 10,000 contents. Assuming each uploaded video to be of duration 5 minutes, daily about 900,000 videos are being uploaded. This require 135 MB of storage space, which can be easily managed.

7.6 Conclusion

Efficient access control in MMSNs is an important requirement with the fast rate of proliferation of MMSNs and new innovative access control requirements. In this chapter, we presented an efficient access control mechanism for MMSNs. The proposed mechanism is suitable and scalable for present social networks as it can easily handle fine grained access control and other functionalities required in existing MMSNs. Experimental results show that our technique is about 30 times more efficient than the existing access control mechanisms. The overhead due to insertion of access rights and storage space is also small. Thus, our proposed bit-vector transform based access control mechanism turns out to be a good choice for storing the access rights associated with the contents in MMSNs.

Chapter 8

Conclusion and Recommendation for Future Work

8.1 Conclusion

In this thesis, our motivation was to design a multi-level DRM system suitable for the real world scenario consisting of untrusted distributors and consumers. In order to fulfil this requirement, we proposed a multi-level DRM architecture, two types of license validation mechanisms namely, instance-based validation and aggregate validation, and a sequential pattern mining based method for consumers' violation detection. In addition we also proposed an application of the bit-vector transform technique in multimedia social networks. In the following section, we briefly conclude these contributions. We will also present several recommendations for future works later in this chapter. Finally, we present a short discussion on legal mechanisms related to DRM and role of our proposed framework in stronger legal-technical interaction.

8.1.1 Our Contributions

a. Design of Multi-level DRM System

In Chapter 3, we proposed a multi-level DRM architecture useful for scalable business models consisting of owner, multiple levels of distributors, and consumers. We designed cryptographic protocols for the content packaging and distribution in the

proposed architecture. We also proposed the rights distribution mechanism suitable for the architecture. The rights distribution in the proposed DRM architecture is handled using two types of licenses namely, redistribution and usage licenses. In addition, we also provided the mechanism for preserving the privacy of the consumers. We compared our proposed architecture with the existing DRM systems and architectures. Using the comparison with other DRM systems and architectures, we demonstrated that the proposed architecture is more suitable as compared to the existing DRM systems and architecture for content distribution in the scenarios in which distributors and consumers may not be trusted to the owner.

b. Redistribution Rights Violation Detection Using Instance-based Validation

In Chapter 4, we proposed the instance-based license validation mechanism as a means for detecting the rights violations by the distributors. There can be large number of validations required in a distributed multi-level DRM system, so the licence validation algorithm needs to be efficient. We proposed a bit-vector transform based data structure for efficient instance-based license validation. We compared the proposed data structure with the various other data structures that can be used for instance-based validation. The proposed data structure is found to be more suitable for efficient instance-based validation as well as the DRM related tasks such as license revocation.

c. Redistribution Rights Violation Detection Using Aggregate Validation

In Chapter 5, we proposed the aggregate validation mechanism as a means of detecting the rights violation by distributors when an aggregate constraint is present in the redistribution licenses. The aggregate validation requires the validation using an exponential number of validation inequalities and each validation inequality may also contain exponential number of summation terms. This makes the aggregate validation process much time consuming and necessitates to do the aggregate validation

efficiently. We proposed two data structures and a graph theory based algorithm to do the aggregate validation efficiently. All the proposed approaches help to do the aggregate validation efficiently as compared to the existing approaches.

d. Sequential Pattern Mining based Algorithm for Usage Rights Violation Detection

In Chapter 6, we presented the problem of consumers' rights violation detection and vulnerability analysis in DRM systems. We proposed a sequential pattern mining based algorithm called global and semi-global violation detection for this purpose. The proposed algorithm uses the logs obtained from consumers for the violation detection. We also analyzed the effectiveness and efficiency of the proposed algorithm. The proposed algorithm is found to be more effective and efficient for the violation and vulnerability detection as compared to the other existing algorithms.

e. Application of Bit-vector Transform Technique in Multimedia Social Networks

In Chapter 7, we presented the application of the proposed bit-vector transform technique for efficient access control purpose in the multimedia social networks. We used the rule based access control model and we modelled the users and contents in the multimedia social networks in the bit-vector transform domain. The proposed algorithm works efficiently as compared to the existing rule based access control mechanisms. In addition, the tasks required in social network such as modification of access rights and deletion of contents can also be performed with the proposed algorithm.

8.2 Recommendation for Future Work

In this section, we present few suggestions for further developing the ideas proposed in this thesis.

a. Privacy Preserving License Validation

In the bit-vector transform based validation method proposed in Chapter 4, each

issued license is sent to the validation authority for the validation purpose. The validation authority compares the constraint values in the issued license with the elementary constraint ranges formed along each constraint dimension. The validation process may lead to breach in the privacy of the distributors. It may raise their concerns over disclosure of their business statistics and strategies to the validation authority. Thus, the distributors may need more privacy during the validation process.

A possible way of providing more privacy to the distributors is to allow them to specify the validation attributes in a privacy preserving way. For example, if a distributor issues a license for a consumer in Singapore, but the distributor may wish that the validation authority can only recognize that the license is issued for Asia. The issued license can be validated with the licenses with the distributor that are valid in Asia. However, the task of validation in privacy preserving manner is challenging because the licenses issued to consumers must reflect the actual permissions and constraints granted to the consumers. Thus, a mechanism is required such that the consumers can see actual permissions and constraints but the validation authority can only determine the required information in a privacy preserving manner.

b. Design of a Better Violation Pattern Mining Algorithm

We use sequential pattern mining algorithms as basis for the global and semi-global violation pattern mining from the alert logs. The future research can be in the direction of improving the performance of the proposed algorithm by finding more useful violation patterns. Some of the ways in which the patterns obtained can be improved are discussed below.

- Some alert symbols are less important than other alert symbols. Several less important alert symbols may occur more frequently than more important alert symbols. So, if we keep higher global and semi-global support thresholds during the *violation pattern* mining, then we may miss frequent *violation patterns*

consisting of more important alert symbols and if we keep lower thresholds then we may obtain a large number patterns related to less important alerts. So, a pattern mining in which importance of the alert symbols can be considered may be designed to obtain better violation patterns.

- Another improvement could be to take into account the relative closeness of different itemsets in a sequence while calculating the support of a sequence. It might be important to keep the closeness between the items because if two items are not close enough in a sequence then they might not be correlated to each other. For example, the sequence $\langle a \rangle \langle c, d \rangle$ is subsequence of two sequences $s_1 = \langle a \rangle \langle c, d, e \rangle$ and $s_2 = \langle a \rangle \langle d \rangle \langle c \rangle \langle e \rangle \langle c, d \rangle$ so if both these sequences are in the sequence database, both these contribute support count=1 to the total support. However, the distance between the itemsets $\langle c, d \rangle$ and $\langle a \rangle$ is different in s_1 and s_2 so $\langle c, d \rangle$ might not be related to $\langle a \rangle$ in s_2 in a similar extent as it is related to $\langle a \rangle$ in s_1 . Better violation patterns can be obtained after considering the closeness factor.

c. Improved Access Control Modeling in Social Networks

Existing access control models for social networks are mainly based on friend-of-a-friend (FOAF) concept, role based access control (RBAC) model, DRM based access control model, rule based access control model. The FOAF based and RBAC based access control models suffer the drawback that each user at the same depth is assumed to be at the same level of trust. This might not be applicable in real world scenarios, e.g. the second level users connected to a more trusted friend might be more trustable. Furthermore, the friends who are more trusted by other friends may be even more trusted. Rule based access control model tries to eliminate this drawback by assigning a trust value (between 0 and 1) to each user, access control is maintained based on this trust value. However, in rule based access control model, it is difficult to define the trust value to the users who are not known. So, automatic trust modeling may be required for the users who are not known. However, automatic trust

assignment method may cause privacy infringement to a user if two or more users over the social network collude. So, it is also necessary to do the assignment of access control parameters in a privacy preserving manner.

8.3 Legal Mechanisms in DRM

There exist various laws internationally to protect the interest of copyright holders. Digital Millennium Copyright Act (DMCA) [3] in USA, Australian Copyright Amendment (Digital Agenda) Act (DACA) [120] in Australia and European Union Copyright Directive (EUCD) [2] in European Union are among several well known laws related to copy protection. These laws make it compulsory to outlaw any attempt to circumvent the DRM. Also, these laws outlaw creation and distribution of circumvention tools. Intellectual property law in Singapore protects the original works that are created in tangible form such as in writing or recording. It protects anyone making copy of the work.

To supplement these legal frameworks, technical frameworks to detect circumvention or violations in DRM is necessary. However, existing DRM technology mainly focus on violation prevention, but not on violation detection. But as discussed in Chapters 3 to 6, rights given to distributors and consumers may be violated by malicious distributors or consumers. The proposed work in this thesis provides the ways of detecting the violation of rights by distributors and consumers and hence provides a technical framework to strongly apply the existing copyright laws. Thus, the proposed work would help towards establishing stronger legal-technical interaction towards strong DRM.

Author's Publications

Book Chapters

- A. Sachan and S. Emmanuel, “Efficient access control in multimedia social networks,” *Social Media Modeling and Computing*, Springer, 2011.

Journals

- A. Sachan, S. Emmanuel and M. S. Kankanhalli, “Aggregate licenses validation for digital rights violation detection,” *ACM Transactions on Multimedia Computing Communications and Application (TOMCCAP)*, to appear.
- A. Sachan and S. Emmanuel, “Violation detection in multi-level digital rights management system,” *Elsevier Computers and Security (COSE)*, vol. 30, no. 6–7, pp. 498–513, 2011.
- A. Sachan and S. Emmanuel, “Computation time efficient approach for DRM licenses validation,” *Springer Multimedia Tools and Applications*, under review.

Conferences

- A. Sachan, S. Emmanuel, A. Das and M. S. Kankanhalli, “Privacy preserving multiparty multilevel DRM architecture,” *IEEE Consumer Communications & Networking Conference (CCNC)*, Las Vegas, 2009, pp. 1–5.
- A. Sachan, S. Emmanuel and M. S. Kankanhalli, “Efficient license validation in MPML DRM architecture,” *9th ACM Workshop on Digital Rights Management (DRM)*, Chicago, 2009, pp. 73–82.

- A. Sachan, S. Emmanuel and M. S. Kankanhalli, “Efficient aggregate licenses validation in DRM,” 15th International Conference on Database Systems for Advanced Application(DASFAA), Tsukuba, 2010, pp. 313–319.
- A. Sachan, S. Emmanuel and M. S. Kankanhalli, “A geometric approach for efficient licenses validation in DRM,” 7th VLDB Workshop on Secure Data management(SDM), Singapore, 2010, pp. 132–149.
- A. Sachan, S. Emmanuel and M. S. Kankanhalli, An efficient access control method for multimedia social networks, 2nd ACM Workshop on Social Media(WSM), in conjunction with ACM Multimedia, Firenze, 2010, pp. 33–38.
- A. Sachan and S. Emmanuel, DRM violation detection using consumers logs analysis, Workshop on Content Protection and Forensics, IEEE International Conference on Multimedia and Expo (ICME), 2011.

Bibliography

- [1] “Cisco end-to-end solutions for IPTV,” Tech. Rep., 2007, Available at: <http://www.cisco.com/> [Last accessed: December 27, 2011].
- [2] “Copyright laws in digital Europe,” Available at: <http://www.euro-copyrights.org/index.php/13> [Last accessed: January 7, 2012].
- [3] “The digital millenium copyright act of 1998,” 1998, Available at: <http://www.cisco.com/> [Last accessed: January 7, 2012].
- [4] Facebook statistics, Available at: <http://www.facebook.com/press/info.php?statistics> [Last accessed: December 27, 2011].
- [5] FairPlay DRM, Available at: <http://www.apple.com/support/itunes/#authorization> [Last accessed: December 27, 2011].
- [6] Flickr, Available at: www.flickr.com/ [Last accessed: December 27, 2011].
- [7] Intelsat IPTV, Available at: <http://www.intelsat.com/services/media/iptv.asp> [Last accessed: December 27, 2011].
- [8] “Intertrust rights management system,” Available at: <http://www.intertrust.com/> [Last accessed: December 27, 2011].
- [9] “Introduction to basic concepts in OMA DRM v 1.0,” Tech. Rep., Motorola Inc., 2007.
- [10] Microsoft windows media DRM, Available at: <http://www.microsoft.com/windows/windowsmedia/forpros/drm/default.mspx> [Last accessed: December 27, 2011].
- [11] “Mobile phone work group- selected user case analyses- v 1.0,” Trusted Computing group, Tech. Rep., 2009.
- [12] “Nielsen online technical report 2009,” Available at: <http://en-us.nielsen.com/> [Last accessed: December 27, 2011].

- [13] “Open digital rights language (ODRL) version 1.1,” Tech. Rep., 2002, Available at: <http://www.w3.org/TR/odrl/> [Last accessed: December 27, 2011].
- [14] Open mobile alliance, Available at: <http://www.openmobilealliance.org> [Last accessed: December 27, 2011].
- [15] Singapore intellectual property law, Available at: <http://www.ipos.gov.sg/> [Last accessed: January 8, 2012].
- [16] “Xrml 2.0 technical overview,” Tech. Rep., 2002, Available at: <http://www.xrml.org/Reference/XrMLTechnicalOverviewV1.pdf> [Last accessed: December 27, 2011].
- [17] YouTube statistics, Available at: http://www.youtube.com/t/press_statistics [Last accessed: December 27, 2011].
- [18] A. Adelsbach, “Towards multilateral secure digital rights distribution infrastructures,” in 5th ACM Workshop on Digital Rights Management (DRM), 2005, pp. 45–54.
- [19] A. Alessi, “Internet protocol television (IPTV),” M.S. thesis, Bowie State University, Maryland in Europe, 2007.
- [20] A. Andersson, “General balanced trees,” *Journal of Algorithms*, vol. 30, no. 1, pp. 1–18, 1999.
- [21] A. Arnab and A. Hutchison, “Fairer usage contracts for DRM,” in 5th ACM Workshop on Digital Rights Management (DRM), 2005, pp. 1–7.
- [22] A. Arnab and A. Hutchison, “Ticket based identity system for DRM,” in Information Security for South Africa (ISSA), 2006, pp. 1–10.
- [23] A. C. Squicciarini, M. Shehab and F. Paci, “Collective privacy management in social networks,” in 18th International Conference on World Wide Web (WWW), 2009, pp. 521–530.
- [24] A. Feldmann, S. Muthukrishnan, “Tradeoffs for packet classification,” in IEEE International Conference on Computer Communications (INFOCOM), 2000, pp. 1193–1202.
- [25] A. M. Tenenbaum and M. J. Augenstein, *Data structures using Pascal*, Prentice-Hall Inc., 1986.

- [26] A.-R. Sadeghi, “Trusted computing $\hat{U}$ special aspects and challenges,” in 34th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM), 2008, pp. 98–117.
- [27] A. Sachan, S. Emmanuel, A. Das and M. S. Kankanhalli, “Privacy preserving multiparty multilevel DRM architecture,” in 6th IEEE Consumer Communications and Networking Conference (CCNC), 2009, pp. 1–5.
- [28] A. Sachan, S. Emmanuel and M. Kankanhalli, “Efficient aggregate licenses validation in DRM,” in proc. Database Systems for Advanced Application(DASFAA), 2010, pp. 313–319.
- [29] A. Sachan and S. Emmanuel, “Violation detection in multi-level digital rights management system,” *Elsevier Computers and Security (COSE)*, vol. 30, no. 6–7, pp. 498–513, 2011.
- [30] A. Tootoonchian, K. K. Gollu, S. Saroiu, Y. Ganjali and A. Wolman, “Lockr: social access control for web 2.0,” in First ACM Workshop on Online Social Networks WOSN), 2008, pp. 43–48.
- [31] A. Yukhuu, S. Garamragchaa and H. Y. Sup, “Mining sequential patterns by prefixspan algorithm with approximation,” in 8th International Conference on Applied Computer Science (ICACSE), 2008, pp. 159–164.
- [32] B. B. Zhu, C. Yuan, Y. Wang and S. Li, “Scalable protection for mpeg-4 fine granularity scalability,” *IEEE Transactions on Multimedia (TMM)*, vol. 7, no. 2, pp. 222–233, 2005.
- [33] B. C. Popescu, B. Crispo, A. S. Tanenbaum and F. Kamperman, “A DRM security architecture for home networks,” in 4th ACM workshop on Digital Rights Management (DRM), 2004, pp. 1–10.
- [34] B. Carminati, E. Ferrari and A. Perego, “Enforcing access control in web-based social networks,” *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 1, pp. 191–223, 2009.
- [35] B. Carminati, E. Ferrari, A. Perego, “Rule-based access control for social networks,” in On the Move to Meaningful Internet Systems: OTM Workshops, 2006, pp. 1734–1744.
- [36] B. Carminati, E. Ferrari and A. Perego, “Privacy-Aware access control in social networks: issues and solutions,” in Privacy and Anonymity in Information Management Systems, 2010, pp. 181–195.

- [37] B. Furht and D. Kirovski, *Multimedia security handbook*, CRC Press, 2005.
- [38] B. Jon Louis and H. F. Jerome, “Data structures for range searching,” *ACM Computing Surveys*, vol. 11, no. 4, pp. 397–409, 1979.
- [39] B. Palmer, K. Bubendorfer and I. Welch, “A Protocol for anonymously establishing digital provenance in reseller chains,” in *Financial Cryptography (FC)*, 2011.
- [40] B. Palmer, “Anonymously establishing digital provenance in reseller chains,” *Australasian Computing Doctoral Consortium*, 2011.
- [41] B. Pfaff, “Performance analysis of BSTs in system software,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 32, no. 1, pp. 410–411, 2004.
- [42] B. Rosenblatt, W. Trippe and S. Mooney, “Digital rights management: business and technology,” New York, NY: John Wiley & Sons, Inc., 2001.
- [43] B. Schneier and P. Sutherland, *Applied Cryptography: Protocols, Algorithms, and Source Code in c*, New York, NY: John Wiley & Sons, Inc., 1995.
- [44] B. Schneier and J. Kelsey, “Secure audit logs to support computer forensics,” *ACM Transactions on Information and System Security (TISSEC)*, vol. 2, no. 2, pp. 159–176, 1999.
- [45] B. Xu, D. Jiang and J. Li, “HSM: A fast packet classification algorithm,” in *19th International Conference on Advanced Information Networking and Applications (AINA)*, 2005, pp. 987–992.
- [46] C.A. Yeung, I. Liccardi, K. Lu, K. Seneviratne and T. Berners-Lee, “Decentralization: The future of online social networking,” in *W3C Workshop on the Future of Social Networking*, 2009.
- [47] C. C. Aggarwal and P. S. Yu, *Privacy-preserving Data Mining: Models and Algorithms*, Springer Publishing Company, Inc., 2008.
- [48] C. C. Foster, “Information retrieval: information storage and retrieval using AVL trees,” in *20th National Conference*, 1965, pp. 192–205.
- [49] C. Gates, “Access control requirements for web 2.0 security and privacy,” in *IEEE Web 2.0 Privacy and Security Workshop*, 2007.

- [50] C. K. Chui, B. Kao and E. Hung, “Mining frequent itemsets from uncertain data,” in 11th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD), 2007, pp. 47–58.
- [51] C. Nguyen, D. Hoang and A. Symvonis, “Multi-level caching with delayed-multicast for video-on-demand,” in Internet and Multimedia Systems and Applications Conference (IMSA), 2003, pp. 699–705.
- [52] C. Rojas and O. Nasraoui, “Summarizing evolving data streams using dynamic prefix trees,” in IEEE/WIC/ACM International Conference on Web Intelligence (WI), 2007, pp. 221–227.
- [53] C. W. Thompson and R. Jena, “Digital licensing,” *IEEE Internet Computing*, vol. 9, no. 4, pp. 85–88, 2005.
- [54] C. Yuan, B. B. Zhu, M. Su, X. Wang, S. Li and Y. Zhong, “Layered access control for MPEG-4 FGS video,” in IEEE International Conference on Image Processing (ICIP), 2003, pp. 517–520.
- [55] C. Yu and Y. Chen, “Mining sequential patterns from multidimensional sequence data,” *IEEE Transactions on knowledge and data engineering*, vol. 17, no. 1, pp. 136–140, 2005.
- [56] D. Austerberry, *The technology of video and audio streaming*, Focal Press, 2004.
- [57] D. Brickley and L. Miller, FOAF vocabulary specification, 2005.
- [58] D. Burdick, M. Calimlim, J. Flanick, J. Gehrke and T. Yiu, “MAFIA: A maximal frequent itemset algorithm,” *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 17, no. 11, pp. 1490–1504, 2005.
- [59] D. Kundur and K. Karthik, “Video fingerprinting and encryption principles for digital rights management,” *Proceedings of the IEEE*, vol. 92, no. 6, pp. 918–932, 2004.
- [60] D. Le Métayer and S. Monteleone, “Computer assisted consent for personal data processing,” in 3rd LSPI Conference on Legal Security and Privacy Issues in IT, 2008.
- [61] D. Naor, M. Naor and J. Lotspiech, “Revocation and tracing schemes for stateless receivers,” in Advances in Cryptography (CRYPTO), 2001, pp. 41–62.
- [62] D. West, *Introduction to Graph Theory*, Prentice Hall Upper Saddle River, 2001.

- [63] E. Becker, *Digital Rights Management: Technological, Economic, Legal and Political Aspects*, Springer, 2003.
- [64] E. Diehl, “A four-layer model for security of digital rights management,” in 8th ACM Workshop on Digital Rights Management (DRM), 2008, pp. 19–28.
- [65] G. Boccon-Gibod, J. Boeuf and J. Lacy, “Octopus: An application independent DRM toolkit,” in 6th IEEE Consumer Communications and Networking Conference (CCNC), 2009, pp. 1–7.
- [66] G. Doerr and T. Kalker, “Design rules for interoperable domains: controlling content dilution and content sharing,” in 8th ACM Workshop on Digital Rights Management (DRM), 2008, pp. 39–50.
- [67] G. Dong and J. Pei, “Frequent and Closed Sequence Patterns,” *Sequence data mining*, Springer, 2007.
- [68] G. Durfee and M. Franklin, “Distribution chain security,” 7th ACM Conference on Computer and Communications Security (CCS), 2000, pp. 63–70.
- [69] G. Governatori and R. Iannella, “Modelling and reasoning languages for social networks policies,” in 13th IEEE International Conference on Enterprise Distributed Object Computing (EDOC), 2009, pp. 193–200.
- [70] G. Grimen, C. Monch and R. Midtstraum, “Software-based copy protection for temporal media during dissemination and playback,” in Conference on Information Security and Cryptology (ISCISC), 2005, pp. 362–377.
- [71] G. Grimen, C. Monch and R. Midtstraum, “Building secure software-based DRM systems,” Norsk informatik konferanse (NIK), 2006.
- [72] G. Loukides and A. Gkoulalas-Divanis, “Privacy challenges and solutions in the social web,” *ACM Crossroads*, vol. 16, no. 2, pp. 14–18, 2009.
- [73] G. O’Driscoll, *Next generation IPTV services and technologies*, Wiley-Interscience, 2008.
- [74] G. Taban, A. A. Cardenas and V. D. Gligor, “Towards a secure and interoperable drm architecture,” in 6th ACM Workshop on Digital Rights Management (DRM), 2006, pp. 69–78.

- [75] H-C. Huang and Y. H. Chen, “Genetic fingerprinting for copyright protection of multicast media,” *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 13, no. 4, pp. 383–391, 2009.
- [76] H. Li, L. Zhao, C. Wang and F. Ma, “DRM system for multiple cascaded business operators,” in *IEEE International Conference on Multimedia and Expo (ICME)*, 2010.
- [77] H. Lu and S. Sahni, “Enhanced interval trees for dynamic IP router-tables,” *IEEE Transactions on Computers (TC)*, vol. 53, no. 12, pp. 1615–1628, 2004.
- [78] H. Pinto, J. Han, J. Pei, K. Wang, Q. Chen and U. Dayal, “Multi-dimensional sequential pattern mining,” *Tenth ACM international conference on Information and knowledge management (CIKM)*, 2001, pp. 81–88.
- [79] H. Yin, C. Lin, Q. Zhang, Z. Chen and D. Wu, “Truststream: A secure and scalable architecture for large-scale internet media streaming,” *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, vol. 18, no. 12, pp. 1692–1702, 2008.
- [80] J. Delgado, V. Torres, S. Llorente and E. Rodríguez, “Rights and trust in multimedia information management,” in *Communications and Multimedia Security (CMS)*, 2005, pp. 55–64.
- [81] I. J. Cox, J. Kilian, F. T. Leighton and T. Shamoon, “Secure spread spectrum watermarking for multimedia,” *IEEE Transactions on Image Processing (TIP)*, vol. 6, no. 12, pp. 1673–1687, 1997.
- [82] I. M. Abbadi and M. Alawneh, “Replay attack of dynamic rights within an authorised domain,” in *Emerging Security Information, Systems and Technologies (SECUREWARE)*, 2009, pp. 148–154.
- [83] J. A. Bloom, “Security and rights management in digital cinemas,” in *proc. IEEE Conference on Multimedia and Expo (ICME)*, 2003, pp. 621–624.
- [84] J. Ayres, J. Flannick, J. Gehrke and T. Yiu, “Sequential pattern mining using a bitmap representation,” in *proc. Knowledge Discovery and Data Mining (KDD)*, 2002, pp. 429–435.

- [85] J. C. N. Chong, R. Van Buuren, P. H. Hartel and G. Kleinhuis, "Security attributes based digital rights management," in *proc. Joint International Workshops on Interactive Distributed Multimedia Systems and Protocols for Multimedia Systems: Protocols and Systems for Interactive Distributed Multimedia*, 2002, pp. 339–352.
- [86] J. E. Holt, "Logcrypt: forward security and public verification for secure audit logs," *Australasian workshops on Grid computing and e-research*, 2006, pp. 203–211.
- [87] J. Gross and J. Yellen, *Handbook of Graph Theory and Applications*, CRC press, 2003.
- [88] J. Han, J. Pei, Y. Yin and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," *Data Mining and Knowledge Discovery*, vol. 8, no. 1, pp. 53–87, 2004.
- [89] J. Katz, "Public-key cryptography," *Handbook of Information and Communication Security*, pp. 21–34, 2010.
- [90] J. Lee, S. O. Hwang, S. W. Jeong, K. S. Yoon, C. S. Park and J. C. Ryou, "A DRM framework for distributing digital contents through the internet," *ETRI Journal*, vol. 25, no. 6, pp. 423–436, 2003.
- [91] J. Mirkovic and P. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004.
- [92] J. Nützel and A. Beyer, "How to increase the security of digital rights management systems without affecting consumer's security," in *Emerging Trends in Information and Communication Security (ETRICS)*, 2006, pp. 368–380.
- [93] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal and M. C. Hsu, "Mining sequential patterns by pattern-growth: The prefixspan approach," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 16, no. 10, pp. 1424–1440, 2004.
- [94] J. Shah and D. Saxena, "Video encryption: a survey," *IJCSI International Journal of Computer Science Issues*, vol. 8, no. 2, pp. 525–534, 2011.
- [95] J. Van der Veen and V. Venugopal, "Using revenue sharing to create win-win in the video rental supply chain," *Journal of the Operational Research Society*, vol. 56, no. 7, pp. 757–762, 2005.

- [96] J. Wang, S. Wu, H. Gao, J. Lee and B. C. Ooi, "Indexing multi-dimensional data in a cloud system," in International Conference on Management of Data (SIGMOD), 2010, pp. 591–602.
- [97] J. Weber and T. Newberry, *IPTV Crash Course*, McGraw-Hill, 2007.
- [98] J. Weise, "Public key infrastructure overview," Tech. Rep., 2001.
- [99] J. Williams, "Social networking applications in health care: threats to the privacy and security of health information," in ICSE Workshop on Software Engineering in Health Care (SEHC), 2010, pp. 39–49.
- [100] J. Y. Halpern and V. Weissman, "A formal foundation for xml," *Journal of the ACM (JACM)*, vol. 55, no. 1, pp. 1–42, 2008.
- [101] J. Zhang, N. Wu, J. Luo and S. Yang, "A scalable digital rights management framework for large-scale content distribution," International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS), 2005, pp. 761–764.
- [102] K. Coyle, "The technology of rights: Digital rights management," *Berkeley Technology Law Journal*, vol. 12, no. 1, pp. 1–20, 1997.
- [103] K. T. Chaung, J. L. Huang and M. S. Chen, "Mining top-k frequent patterns in the presence of the memory constraint," *The VLDB Journal*, vol. 17, no. 5, pp. 1321–1344, 2008.
- [104] L. Chiariglione, "A walkthrough in the interoperable DRM platform v. 3.0 specification," Tech. Rep., Available at: http://www.dmpf.org/documents/walkthrough_in_idp-3.htm [Last accessed: December 27, 2011].
- [105] L. L. Win, T. Thomas, S. Emmanuel and M. S. Kankanhalli, "Secure domain architecture for interoperable content distribution," Pacific-Rim Conference on Multimedia (PCM), 2009, pp. 1313–1318.
- [106] L. Qiao and K. Nahrstedt, "Comparison of MPEG encryption algorithms," *Computers and Graphics*, vol. 22, no. 4, pp. 437–448, 1998.
- [107] M. Abomhara, O. Zakaria and O. O. Khalifa, "An overview of video encryption techniques," *International Journal of Computer Theory and Engineering*, vol. 2, no. 1, pp. 103–110, 2010.

- [108] M. De Berg, O. Cheong, M. V. Kreveld and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd Edition, Springer-Verlag New York Inc., 2008.
- [109] M. Blanton and M. Atallah, "Succinct representation of flexible and privacy-preserving access rights," *The VLDB Journal*, vol. 15, no. 4, pp. 334–354, 2006.
- [110] M. FENG and B. ZHU, "A drm system protecting consumer privacy," in 5th IEEE Consumer Communications and Networking Conference (CCNC), 2008.
- [111] M. Hilty, A. Pretschner, D. Basin, C. Schaefer and T. Walter, "A policy language for distributed usage control," in 12th European Symposium on Research in Computer Security (ESORICS), 2007, pp. 531–546.
- [112] M. K. Aguilera, W. Golab and M. A. Shah, "A practical scalable distributed B-tree," in 34th Very Large Databases Conference Conference (VLDB), 2008, pp. 598–609.
- [113] M. K. F. Beato and K. Wouters, "Enforcing access control in social networks," in Hot Topics in Privacy Enhancing Technologies (HotPets), 2009, pp. 1–10.
- [114] M. Lim and D. Lee, "A task-based load distribution scheme for multi-server-based distributed virtual environment systems," *Presence: Teleoperators and Virtual Environments*, vol. 18, no.1, pp. 16–38, 2009.
- [115] M. Rafi and Z. Guennoun, "DRMAA : DRM adaptable architecture," in 3rd International Conference on Information and Communication Technologies: From Theory to Applications, 2008, pp. 1–4.
- [116] M. Shehab, A. Squicciarini and G.J. Ahn, "Beyond user-to-user access control for online social networks," Information and Communications Security (ICICS), 2008, pp. 174–189.
- [117] M. Zaki, "Spade: An efficient algorithm for mining frequent sequences," *Machine Learning*, vol. 42, no. 1, pp. 31–60, 2001.
- [118] N. F. Sharpe and O. B. Orewa, "Is Apple Playing Fair? Navigating the iPod FairPlay DRM Controversy," *Northwestern Journal of Technology and Intellectual Property*, Vol. 5, no. 2, pp. 331–349, 2007.
- [119] N. P. Sheppard and R. Safavi-Naini, "Sharing digital rights with domain licensing," in 4th ACM International Workshop on Contents Protection and Security, 2006, pp. 3–12.

- [120] N. Smith, “Digital Copyright: The Copyright Amendment (Digital Agenda) Act 2000 for Libraries,” Available at: www.digital.org.au/alcc/dasem.ppt [Last accessed: January 7, 2012].
- [121] P. A. Jamkhedkar and G. L. Heileman, “The role of architecture in DRM vendor economics,” in International Conference on e-Commerce (ICEC), 2008.
- [122] P. A. Jamkhedkar and G. L. Heileman, “Digital rights management architectures,” *Computers and Electrical Engineering*, vol. 35, no. 2, pp. 376–394, 2009.
- [123] P. Gupta and N. McKeown, “Algorithms for packet classification,” *IEEE Network*, vol. 15, no. 2, pp. 24–32, 2001.
- [124] P. Gupta and N. McKeown, “Packet classification using hierarchical intelligent cuttings,” in IEEE Hot Interconnects, 1999, pp. 34–41.
- [125] P. Loscocco, S. Smalley, P. Muckelbauer, R. Taylor, S. Turner and J. Farrell, “The inevitability of failure: The flawed assumption of security in modern computing environments,” in 21st Information Systems Security Conference (ISC), 1998, pp. 303–314.
- [126] P. W. Wong, “A public key watermark for image verification and authentication,” in proc. IEEE International Conference on Image Processing (ICIP), 1998, pp. 455–459.
- [127] Q. Dong, A. Shukla, V. Shrivastava, D. Agrawal, S. Banerjee and K. Kar, “Load balancing in large-scale RFID systems,” *Computer Networks*, vol. 52, no. 9, pp. 1782–1796, 2008.
- [128] Q. Liu, R. Safavi-Naini, and N. P. Sheppard, “Digital rights management for content distribution,” in Australasian Information Security Workshop (AISW), 2003, pp. 49–58.
- [129] R. Accorsi, “BBox: A distributed secure log architecture,” in 7th European Workshop on Public Key Services (EuroPKI), 2010, pp. 349–353.
- [130] R. Agrawal and R. Srikant, “Fast algorithms for mining association rules,” in 20th International Conference on Very Large Data Bases (VLDB), 1994, pp. 487–499.
- [131] R. Agrawal and R. Srikant, “Mining sequential patterns,” in 11th International Conference on Data Engineering (ICDE), 1995, pp. 3–14.

- [132] R. Srikant and R. Agrawal, “Mining sequential patterns: Generalizations and performance improvements,” in *Advances in Database Technology (EDBT)*, 1996, pp. 1–17.
- [133] R. Baden, A. Bender, N. Spring, B. Bhattacharjee and D. Starin, “Persona: an online social network with user-defined privacy,” in *ACM SIGCOMM*, 2009, pp. 135–146.
- [134] R. Drechsler and B. Becker, *Binary Decision Diagrams: Theory and Implementation*, Springer, 1998.
- [135] R. Garcia and R. Gill, “Content value chains modelling using a copyright ontology,” *Information Systems*, vol. 35, no. 4, pp. 483–495, 2010.
- [136] R. H. Koenen, J. Lacy, M. MacKay and S. Mitchell, “The long march to interoperable digital rights management,” *Proceedings of the IEEE*, vol. 92, no. 6, pp. 883–897, 2004.
- [137] R. Iannella, “A framework for the policy-oriented web in social networks,” in *7th International Workshop for Technical, Economic and Legal Aspects of Business Models for Virtual Goods*, 2009.
- [138] R. Iannella, “Digital rights management (DRM) architectures,” *D-Lib Magazine*, vol. 7, no. 6, 2001.
- [139] R. Iannella, “Digital rights management,” *Handbook of Information Security*, Vol. 3, Wiley, 2006.
- [140] R. Iannella, “Rights management and licensing multimedia services,” *Studies in Computational Intelligence (SCI)*, 2008, pp. 327–349.
- [141] R. K. C. Chang, “Defending against flooding-based distributed denial-of-service attacks: A tutorial,” *IEEE Communications Magazine*, vol. 40, no. 10, pp. 42–51, 2002.
- [142] R. Mori and M. Kawahara, “Superdistribution: the concept and the architecture,” *Transactions of the IEICE*, vol. E73, no. 7, pp. 1133–1146, 1990.
- [143] R. Safavi-Naini, N. P. Sheppard and T. Uehara, “Import/export in digital rights management,” in *4th ACM Workshop on Digital Rights Management (DRM)*, 2004, pp. 99–110.

- [144] R. T. Snodgrass, S. S. Yao and C. Collberg, “Tamper detection in audit logs,” in 13th International Conference on Very Large Databases (VLDB), 2004, pp. 504–515.
- [145] S. B. Akers, “Binary decision diagrams,” *IEEE Transactions on computers (TC)*, vol. 27 no. 6, pp. 509–516, 1978.
- [146] S. Choi, J. Han, and S. Jun, “Improvement on TCG attestation and its implication for DRM,” in International Conference on Computational Science and its Applications (ICCSA), 2007, pp. 912–925.
- [147] S. Emmanuel and M. S. Kankanhalli, “A digital rights management scheme for broadcast video,” *Multimedia Systems*, vol. 8, no. 6, pp. 444–458, 2003.
- [148] S. Emmanuel and M. S. Kankanhalli, “Digital rights management issues for video,” *Multimedia Security Handbook*, Chapter 26, pp. 759–787, CRC press, 2005.
- [149] S. Grzonkowski, B. Ensor, S. R. Kruk, A. Gzella, S. Decker and B. McDaniel, “A DRM solution based on social networks and enabling the idea of fair use,” in proc. Media in Transition: Creativity, Ownership and Collaboration in the digital age, 2007.
- [150] S. K. Nair, A. S. Tanenbaum, G. Gheorghe and B. Crispo, “Enforcing DRM policies across applications,” in 8th ACM workshop on Digital Rights Management (DRM), 2008, pp. 87–94.
- [151] S. L. Keoh and K. Vrieling, “An implementation experience of domain management in marlin,” in 9th ACM workshop on Digital Rights Management (DRM), 2009, pp. 29–38.
- [152] S. O. Hwang, K. S. Yoon, K. P. Jun and K. H. Lee, “Modeling and implementation of digital rights,” *The Journal of Systems and Software (JSS)*, vol. 73, no. 3, pp. 533–549, 2004.
- [153] S. I. M. SangGyoo, O. H. YunSang and L. E. E. SukBong, “Extending DRM features to distributed environments,” *Journal of Multimedia (JMM)*, vol. 1, no. 5, pp. 36–41, 2006.
- [154] T. Eavis and X. Zheng, “Multi-level frequent pattern mining,” Database Systems for Advanced Applications (DASFAA), 2009, pp. 369–383.

- [155] T. Hauser and C. Wenz, “DRM under attack: weaknesses in existing systems,” Digital Rights Management, 2003, pp. 206–223.
- [156] T. H. Gormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms*, MIT Press/McGraw-Hill, 1990.
- [157] T. Sans, F. Cuppens and N. Cuppens-Boulahia, “OPA: Onion policy administration model-another approach to manage rights in DRM,” in *New Approaches for Security, Privacy and Trust in Complex Environments*, 2007, pp. 349–360.
- [158] T. Thomas, S. Emmanuel, A. V. Subramanyam and M. S. Kankanhalli, “Joint watermarking scheme for multiparty multilevel DRM architecture”, *IEEE Transactions on Information Forensics and Security (TIFS)*, vol. 4, no. 4, pp. 758–767, 2009.
- [159] T-Y. Lee and S. D. Lin, “Dual watermark for image tamper detection and recovery”, *Pattern Recognition*, vol. 41, no. 11, pp. 3497–3506, 2008.
- [160] V. Kolovski, Y. Katz, J. Hendler, D. Weitzner and T. Berners-Lee, “Towards a policy-aware web,” in *Semantic Web and Policy Workshop (SWPW)*, 2005.
- [161] V. M. Potdar, S. Han and E. Chan, “A survey of digital image watermarking techniques,” in *3rd IEEE International Conference on Industrial Informatics (INDIN)*, 2005, pp. 709–716.
- [162] V. Rosset, C. V. Filippin and C. M. Westphall, “A DRM architecture to distribute and protect digital contents using digital licenses,” in *Advanced Industrial Conference on Telecommunications (AICT)*, 2005, pp. 422–427.
- [163] V. S. Mahanta, “Multi-level supply chain disruption and coordinated response,” Ph.D. dissertation, Department of decision sciences, Drexel University, 2011.
- [164] V. Wilfred, A. Bader and M. Muthucumaru, “An access control scheme for protecting personal data,” in *proc. Sixth Annual Conference on Privacy, Security and Trust (PST)*, 2008, pp. 24–35.
- [165] W. Graham, “Reaching users through facebook: A guide to implementing facebook athenaeum,” *Code4Lib Journal*, vol. 5, 2008.
- [166] W. Luh and D. Kundur, “Digital media fingerprinting: Techniques and trends”, *Multimedia Security Handbook*, B. Furht and D. Kirovski, eds. CRC Press, pp. 577–603, 2004.

- [167] X. Liu and A. M. Eskicioglu, "Selective encryption of multimedia content in distribution networks: Challenges and new directions," in *IASTED Communications, Internet and Information Technology (CIIT)*, 2003.
- [168] X. Liu, T. Huang, L. Huo and L. Mou, "A DRM Architecture for Manageable P2P Based IPTV System," in *proc. IEEE International Conference on Multimedia and Expo (ICME)*, 2007, pp. 899–902.
- [169] X. Zhang, M. Nakae, M. J. Covington and R. Sandhu, "Toward a usage-based security framework for collaborative computing systems," *ACM Transactions on Information and System Security (TISSEC)*, vol. 11, no. 1, pp. 1–36, 2008.
- [170] Y. Danfeng, B. F. Keith, J. A. Mikhail and T. Roberto, "Private information: To reveal or not to reveal," *ACM Transaction on Information and System Security (TISSEC)*, vol. 12, no. 1, pp. 1–27, 2008.
- [171] Y. D. Wu and F. Bao, "Enriched trusted platform and its application on drm," in *3rd Asia-Pacific Trusted Infrastructure Technologies Conference (APTIC)*, 2008, pp. 91–97.
- [172] Y. Jeong, J. Park, J. Kin and K. Yoon, "DRM content adaptation scheme between different DRM systems for seamless content service," in *IEEE International Conference on Multimedia and Expo (ICME)*, 2006, pp. 867–870.
- [173] Y. Qi, L. Xu, B. Yang, Y. Xue and J. Li, "Packet classification algorithms: From theory to practice," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2009, pp. 648–656.
- [174] Y. Wang, L. Tao, Q. Liu, Y. Zhao and G. Xu, "A flexible multi-server platform for distributed video information processing," in *5th International Conference on Computer Vision Systems (ICVS)*, 2007.