Efficient License Validation in MPML DRM Architecture

Amit Sachan, Sabu Emmanuel
School of Computer Engineering
Nanyang Technological University
Singapore
{amit0009, asemmanuel}@ntu.edu.sg

Mohan S. Kankanhalli
School of Computing
National University of Singapore
Singapore
mohan@comp.nus.edu.sg

ABSTRACT

Multiparty multilevel DRM architecture (MPML-DRM-A) involves multiple parties such as owner, multiple levels of distributors and consumers. The owner issues redistribution licenses to its distributors, who in turn generate and issue variations of these redistribution licenses to their sub-distributors. Also the distributors generate and issue usage licenses to the consumers to consume the contents. But, these variations of the redistribution licenses and usage licenses generated and issued by each distributor must be validated by a validation authority against the redistribution licenses that it has received. In MPML-DRM-A, there may exist multiple, different types of redistribution licenses for a content. Validation using multiple redistribution licenses may become difficult in real time. Further, storage of multiple redistribution licenses for validation presents a challenge of reducing storage space requirements. Hence, in this paper we propose a bit-vector transform based license organizing structure, and present a method to do the validation of issued licenses in the bit-vector transform domain efficiently. Experimental results show that our license organization structure helps to achieve low validation time and storage space complexity.

Categories and Subject Descriptors

H.3.m [**Information Storage and Retrieval**]: Miscellaneous; K.5.1 [**Hardware/Software Protection**]: Metrics—*Licensing*

General Terms

Algorithms, Security

Keywords

Digital Rights Management (DRM), License Organization, License Validation

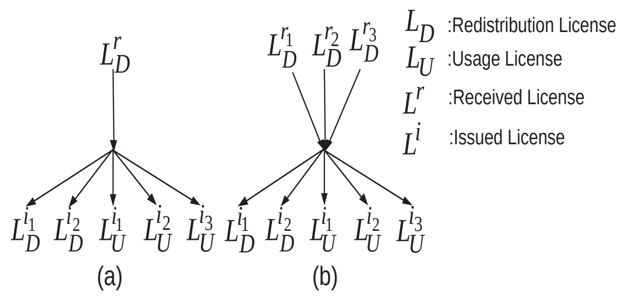

L_D^r
$L_D^{r_1}$ $L_D^{r_2}$ $L_D^{r_3}$
L_D :Redistribution License
L_U :Usage License
L^r :Received License
L^i :Issued License
$L_D^{i_1}$ $L_D^{i_2}$ $L_U^{i_1}$ $L_U^{i_2}$ $L_U^{i_3}$
(a)
(b)

Figure 1: (a). Single, and (b). Multiple received redistribution license cases

1. INTRODUCTION

Traditional two party digital rights management (DRM) architectures [12][13][20] involving the seller and buyer may not be sufficient to provide the business scalability for all regions and cultures. So, as an alternative to the two party DRM architectures, multiparty multilevel DRM architecture (MPML-DRM-A) is proposed in [16][8][10][4]. The term 'multiparty' refers to the involvement of multiple parties such as owner, distributors, sub-distributors and consumers. The term 'multilevel' refers to multiple levels of distributors/ sub-distributors. In MPML-DRM-A the owner issues rights for redistribution of contents to distributors by issuing redistribution license. The distributors in turn can use the received redistribution license to generate and issue new different types of redistribution licenses to their sub-distributors, and also generate and issue new usage licenses to consumers, as shown in figure 1(a). A redistribution license allows a distributor to redistribute the content to sub-distributors and consumers, according to the permissions and constraints specified in the redistribution license that it has received. Thus, as part of the rights violation detection[5][6], it is then necessary to validate these newly generated licenses against the redistribution license that the distributor has received.

Both redistribution and usage licenses for a content K can represented with the following structure: $L = \{K; P_1, P_2, ..., P_M; C_1, C_2, ..., C_M\}$. Where P, and C represent permissions, and set of constraints respectively, and M is the number of permissions in the license. The set C_q for a permission P_q with n_q constraints can be expanded as: $C_q = \{C_{q,1}, C_{q,2}, ..., C_{q,n_q}\}$. Permissions ($P$) in the usage licenses $'L'_U$ are the actions such as copy, play, rip, etc. those can be performed on the content. Redistribution licenses $'L_D'$ also have similar permissions as those in the usage licenses but the purpose is to distribute them in further redistribution and usage licenses. Constraints [5] [17]in usage licenses, such as number of plays, expiry date of license, region allowed to play, etc. are restrictions on the content usage and are in form of single values. Constraints in the redistribution licenses such as the region allowed for distribution, minimum and maximum number of play permission allowed to issue in each usage licenses, etc. are the restrictions associated with content redistribution and these are in form of ranges. Ranges of constraints in the further generated redistribution licenses using a redistribution license must be subsets of the respective constraint ranges in the redistribution license. Thus, in figure 1(a), the constraint ranges the issued redistribution licenses, $L_D^{i_1}$, and $L_D^{i_2}$ must be subset of respective constraint ranges in the received redistribution license L_D^r. Values of constraints in usage licenses generated using a redistribution license must be inside the respective constraint ranges in the redistribution license. Thus, the constraint values in $L_U^{i_1}$, $L_U^{i_2}$, and $L_U^{i_3}$ must be within the respective constraint ranges in the received redistribution license L_D^r.

A validation authority validates all the newly generated licenses before the licenses are finally issued to the requesters. Newly generated licenses are validated for the constraint values/ranges with the redistribution license used to generate them . The purpose of validation is to determine whether a new license can be generated using the respective received redistribution license after satisfying the constraints in the received redistribution license.

For business flexibility, distributors may need to obtain multiple redistribution licenses for the same content as shown in figure 1(b). Different pricing structure for different range of constraints may not be handled with a single redistribution license. For example, for promotional purpose per unit play permission cost may be lower if a consumer purchases play permission counts in the range $[11, 20]$ rather than $[1, 10]$, or price of a media may be different in Asia and America. In both the cases discussed above distributors are required to have different redistribution licenses for different constraint ranges. Or, it may happen that for different types of permissions and constraints the distributor might get better deal from different distributors and hence the distributor might buy them from different distributors at different time. In case of multiple received redistribution licenses the validation time and storage requirements grow proportionally with the number of licenses present. It presents challenges of reducing the validation time and storage space requirements. This is firstly because all the validation requests need to be handled in real time. Secondly, in a large and distributed DRM architecture like MPML-DRM-A there can be a large number distributors and subdistributors and each distributor might be generating a large number licenses and validation requests per unit time. And thirdly, validation of each newly generated license needs to be done with multiple constraints in multiple received redistribution licenses. Thus, for this purpose, we propose bit-vector transform based license organization technique and present a method to do the validation the newly generated licenses in the bit-vector transform domain. The license organization technique, firstly, removes the redundancy by utilizing the fact that in practical scenarios, number of ranges for most of the constraints are limited irrespective of number of licenses present. For example, number of countries in the redistribution licenses of a South Asia distributor cannot exceed a maximum fixed number of countries. Secondly, as we explain in section 4, the data after the transform can be represented using efficient data structures.

Thus, in this paper, we present a bit-vector transform based license organization technique and secondly we propose a validation method that uses the data in the bit-vector transform domain to do the validation efficiently. To the best of our knowledge, the work presented in this paper is the first for the license validation and license organization for DRM systems.

Rest of this paper is organized as follows. In section 2, we discuss related work. In section 3, we discuss about the problem of license validation. Section 4 describes our proposed license organization method. In section 5, we present our method of license validation in transformed domain. In section 6, we discuss the performance of our method. Finally, section 7 concludes the paper.

2. RELATED WORK

In this section we discuss the work and important issues in the area of digital rights management (DRM) which are related to our work.

In [9] Jamkhedkar and Heileman analyzed the importance of shift of DRM industry more towards user friendly rights expression. The MPML-DRM-A can be much useful for fulfilling user needs as distributor can deal with regionally and culturally sensitive manner. In [2] and [3] Arnab and Hutchison discussed different contract and negotiation scenarios between DRM vendors and clients. In [2] the authors proposed a technique for negotiation of rights with the license server in a two level DRM system. In the technique clients negotiate with the license server for the requested rights. The license server (LS) presents the types of licenses closely resembling with the requested license. Then the client selects one out of those licenses to purchase. However, in MPML-DRM-A an LS cannot do such negotiations as the LS is a party different than distributors. This is in contrast to the two level architectures in which only the owner distributes the contents by maintaining his own LS. Thus, we are using a model of distribution that enables the distributors to directly negotiate with the consumers.

In [18] describes a way of rights assignment to the content distributors by the owner. In the model anybody can act as a potential distributor, and distribute the rights to others without any constraint. The rights issued by each level of distributors are verified at the client device. The DRM agent verifies that each right issued is compliant to the upper level rights. The scheme presented is technically sound but in case if some distributor has distributed rights wrongly finally the consumer will be penalized which is unacceptable in most of the practical scenarios. So, if the rights can be checked at each level then it would be better for end users. In this paper we consider the model that verifies rights in form of licenses at every level.

Nair et al.[14] discussed and modeled using different types of DRM system modeling such as 'pay per use', 'use N times', and 'metered payment' in their architecture. However, the most popular is the 'use N times' model. In this model constraints modeled have some maximum limit associated with them. After expiry of the constraints associated with a permission, such as play, copy, etc, the permission cannot be used. This model is also most suitable in case of MPML-DRM-A. In MPML-DRM-A the permissions and constraints to each distributor and consumers can be given with the help of redistribution and usage licenses respectively. The redistribution licenses can be used to further distribute the redistribution and usage licenses.

In this work, we consider all the issues discussed above while defining the problem of license validation and carrying out the license validation. To the best of our knowledge, there is no existing work tackling the DRM license validation.

3. PRELIMINARIES

In this section, first we present a list of notations we use in this paper. Then we discuss in detail about structure and generation redistribution and usage licenses using the redistribution licenses. In the end of this section we discuss about the process of validation when multiple redistribution licenses are present.

3.1 Notations

We use the following notations in this paper.

- Structure of a license(details about redistribution and usage licenses are given in the next sub-section): $L = \{K; P_1, P_2, ..., P_M; C_1, C_2, ..., C_M\}$
- P_q: q^{th} permission in the license.
- C_q: Set of constraints for the permission P_q.
- n_q: Total number of constraints in the set of constraints C_q.
- L^r_D : A redistribution license received by a distributor D.
- L^i_D: A redistribution license issued by a distributor(D) to another distributor.
- L^i_U: A usage license issued by a distributor(D) to a consumer.
- N: Number of received redistribution licenses for a media.
- $B^{je}_{P_q}$: Bit-vector for the e^{th} elementary range along the j^{th} constraint's dimension of a permission P_q.
- B_{P_q}: Resultant bit-vector obtained after validation.
- r_j: Number of maximum possible elementary ranges along j^{th} constraint dimension.

3.2 Generation of New Licenses

Received redistribution licenses are used by distributors to redistribute the contents to their sub-distributors and consumers. Distributors do it by generating further redistribution licenses and usage licenses for the sub-distributors and consumers respectively. Redistribution licenses contain permissions and constraints related to further distribution of contents. Constraints in redistribution licenses generally allow a distributor to distribute over a range of allowed values for the constraints. For example, if a distributor is allowed to distribute in Asia at country level, then range of allowed values will be countries in Asia. Or, the owner may limit the distribution such that a distributor cannot issue more than 20 play permission counts in further usage licenses, then range of allowed values for this constraint is (0, 20]. Or, maximum number of days a usage or redistribution license generated using a redistribution license can be used. Thus, we assume the constraints in redistribution licenses to be in form of ranges (there can be multiple ranges for each constraint but for the sake of simplicity we discuss only the case for single range). Below, we discuss procedures for generation of redistribution licenses and usage licenses using a redistribution license in detail.

3.2.1 Redistribution License Generation

In an issued redistribution licenses, each constraint must be a subset of the same constraint in the received redistribution license, using which the license is issued. For instance, a received redistribution license that is allowed to issue further redistribution and usage licenses in Asia cannot be used to issue a redistribution license that is allowed to distribute in a country in Europe. Thus, if in a newly generated redistribution license $L^{i'}_D$, for a permission P_q, the j^{th} constraint has a range $[x'_j, y'_j]$; and it is generated using a received redistribution license L^r_D with the j^{th} constraint for a permission P_q with the range as $[x_j, y_j]$ then $[x'_j, y'_j] \subseteq [x_j, y_j]$, and this condition must be valid for all $j \leq n_q$, where n_q is the number of constraints for the permission P_q. The above condition must also hold for all the permissions.

3.2.2 Usage License Generation

A usage license allows a consumer to consume a media according to the permissions and constraints in it. In usage licenses, constraints can be in form single values or in form of ranges. For example, if a usage license allows to play a media for 20 days in Asia, then 20 can be considered in form of single value and Asia can be considered as range of allowed countries. However, the constraints which are in form of ranges, during validation, can be dealt in similar manner as that of constraints in redistribution licenses(which we discuss separately). Thus, for ease of explanation we assume each constraint in a usage license in the form of single value. For generation of a new usage license the value of each constraint must be within the range of respective constraint in the redistribution license that is used to generate it. Thus, for a permission P_q, if the j^{th} constraint value in a usage license is C_j then C_j must be within the respective range $[x_j, y_j]$ in the redistribution license used to generate it.

Example 1: Consider the received redistribution license $L^{r_1}_D$ to distribute the play permissions according to three constraints(validity period T(in days) of usage licenses generated using redistribution licenses, region allowed for distribution R, and range of number of maximum play permission counts A in the usage licenses generated using the redistribution license). Redistribution license $L^{i_1}_D$ and usage license $L^{i_1}_U$ shown below can be generated using it.

$L^{r_1}_D = \{K; \mathit{Play}; C^{r_1}_{\mathit{Play}} : T = [10, 30], R = [\mathit{Asia}, \mathit{Africa}], A = [10, 50]\}$

$L^{i_1}_D = \{K; \mathit{Play}; C^{i_1}_{\mathit{Play}} : T = [12, 30], R = [\mathit{SouthAsia}], A = [10, 45]\}$

$L^{i_1}_U = \{K; \mathit{Play}; C^{i_1}_{\mathit{Play}} : T = [15], R = [\mathit{Country}\ 'X'], A = [35]\}$

Where, $L^{i_1}_U$ can be interpreted as license allowed to play the content for at most 12 days, in the country $'X'$ (in Asia) and maximum allowed play permission counts are 35.

3.3 Validation of Newly Generated Licenses

A distributor in MPML DRM architecture is a different party than the owner. The licenses generated by these distributors may not be according to the redistribution licenses they acquired. Thus, as a part of rights violation detection[5], it is required to do the validation of all the licenses generated by all the distributors. And the licenses can only be redirected to the sub-distributors and consumers after the validation. A validation authority does the validation of all the licenses generated by all the distributors.

As discussed in section 1, due to business flexibility reasons multiple received redistribution license for the contents may be present with the distributors. Validation of each license generated by a distributor for a content is done using all the received redistribution licenses for the content with the distributor. The validation authority stores a copy of all received redistribution licenses by all distributors and validates a newly generated license using stored received redistribution licenses. A newly generated redistribution license is valid if all the constraints for each permission are subset of constraints in at least one of the received redistribution licenses. Let a distributor has N number of received redistribution licenses $L_D^{r_1}$, $L_D^{r_2}$, ..., $L_D^{r_N}$ for the content K, as represented below:

$L_D^{r_n} = \{K, P_1^{r_n}, P_2^{r_n}, ..., P_M^{r_n}, C_1^{r_n}, C_2^{r_n}, ..., C_M^{r_n}\}$, $1 \leq n \leq N$.

and the distributor generates a new redistribution license $L_D^{i'}$ for the same content:

$L_D^{i'} = \{K, P_1^{i'}, P_2^{i'}, ..., P_M^{i'}, C_1^{i'}, C_2^{i'}, ..., C_M^{i'}\}$.

$L_D^{i'}$ is a valid redistribution license if each constraint in each the set $C_j^{i'}$ is subset of the respective constraint in the set $C_j^{r_k}$ for at least one n, where $1 \leq j \leq M$, and $1 \leq n \leq N$.

The validation process for a usage licenses is similar to the validation of redistribution licenses except constraints in usage licenses are in form of single values and we need to find at least one received redistribution license, for which all the constraint values in the usage license are within the respective constraint range in the received redistribution license.

Example 2: Let three received redistribution licenses ($N = 3$), $L_D^{r_1}$, $L_D^{r_2}$, and $L_D^{r_3}$, shown below are obtained by a distributor.

$L_D^{r_1} = \{K; Play; C_{Play}^{r_1} : T = [10, 30], R = [Asia, Africa], A = [10, 50]\}$

$L_D^{r_2} = \{K; Play; C_{Play}^{r_2} : T = [20, 40], R = [America], A = [10, 50]\}$

$L_D^{r_3} = \{K; Play; C_{Play}^{r_3} : T = [15, 30], R = [Asia], A = [15, 60]\}$

Now, if the distributor generates a new usage license $L_U^{i_1}$

$L_U^{i_1} = \{K; Play; C_{Play}^{i_1} : T = [15], R = [America], A = [26]\}$

Now, for validation purpose the validation authority checks all the constraints in the issued licenses one by one using the received redistribution licenses $L_D^{r_1}$, $L_D^{r_2}$, and $L_D^{r_3}$. Since all three constraints in the issued license are not within the respective constraints in any received redistribution license, hence the license $L_U^{i_1}$ is not valid and cannot be issued.

4. LICENSE ORGANIZATION FOR EFFICIENT VALIDATION

In this section, first we discuss about the requirement of an efficient method of license validation. Then, we propose the bit-vector transform based license organization approach and associated data structure to do the validation of newly generated licenses efficiently.

4.1 Requirement of Efficient Validation Method

As discussed in the previous section, a validation authority does the validation of all the licenses generated by all the distributors. In a geographically distributed DRM architecture like MPML DRM architecture there can be a large number of distributors. Each distributor might be generating a large number of redistribution and usage licenses per unit time. Every time a new license is generated by a distributor, a validation request is send by the distributor to the validation authority. Thus, validation authority needs to handle a large number of validation requests per unit time. It may be difficult for the validation authority to handle such a large number of validation requests per unit time as all the validations need to be done in real time.

The problem of validation becomes even more difficult when we consider the validation using multiple received redistribution licenses, as discussed in section 3.3, for the content by a distributor. Multiple received redistribution licenses complicate the problem in two ways. First, the validation time increases several times as each newly generated license, now, needs to be validated using multiple received redistribution licenses. Second, for validation purpose, the validation authority needs to store a copy of all the received redistribution licenses for each media. Hence, multiple received redistribution licenses for each media may increase the storage space requirements drastically.

Thus, we propose a bit-vector transform based license organization method. The license organization method helps to achieve low validation time and storage space required for storing redistribution licenses.

4.2 Bit-vector Transform

The bit-vector transform converts received redistribution licenses into the bit-vectors. The transform is applied separately for each permission. In the transform presented each constraint for a permission P_q can be seen as representing one dimension in a d dimensional hyperspace, where d is the number of constraints for the permission P_q(same as n_q). Each constraint dimension is divided into non overlapping *elementary constraint ranges* based on the respective constraint ranges in the received redistribution licenses, as we discuss in the sub-sections below. For example, in figure 2(a) there are three elementary ranges along constraint 1 dimension:$(0, 100)$, $[100, 200]$, $(200, \infty)$. Each elementary range represents a bit-vector. Let the bit vector for the e^{th} elementary range along the j^{th} constraint dimension for a permission P_q is represented by $B_{P_q}^{je}$. If there are N received redistribution licenses then the length of bit-vector is N bits, and the n^{th} bit in the bit-vector $B_{P_q}^{je}$ represents the n^{th} received redistribution license. A bit equal to 1 in a bit vector shows presence of the elementary constraint range and a bit equal to 0 shows absence of the elementary constraint range in the corresponding (to the bit) received redistribution license. Next, we discuss the process to obtain the bit-vectors from multiple received redistribution licenses. For better understanding, first we start with transform of a single received redistribution license. Later we generalize it for the case of multiple received redistribution licenses.

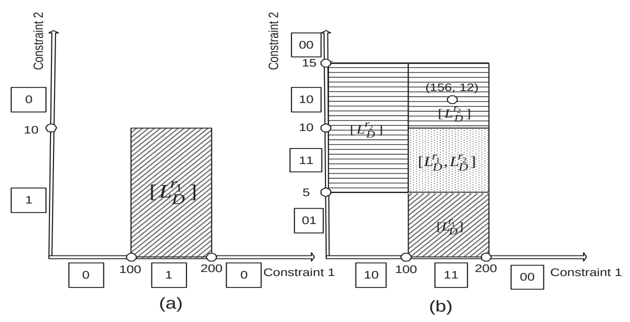


Figure 2: Representation of Redistribution licenses

4.2.1 Transform for a Single license

Initially, when there is no redistribution license, the original elementary range in each constraint dimension is assumed from 0 to ∞. When the distributor obtains the first redistribution license, the elementary range along each constraint dimension is divided into several elementary ranges according to the respective constraint range in the first received redistribution license. If the j^{th} constraint range for a permission P_q in the first license is represented by $[a_j, b_j]$ then the constraint range along j^{th} constraint dimension are divided into elementary ranges$(0, a_j)$, $[a_j, b_j]$, and (b_j, ∞). Each elementary range represents a 1 bit length bit-vector(as only 1 redistribution license is present). Since only the elementary range $[a_j, b_j]$ is present in the received redistribution license so the bit-vectors for the ranges $(0, a_j)$, $[a_j, b_j]$, and (b_j, ∞) would be 0, 1 and 0 respectively.

Example 3: Consider a received redistribution license with two constraints and respective constraint ranges as $[100, 200]$ and $(0, 10]$ along the constraint 1 and constraint 2 dimensions respectively. Figure 2(a) illustrates the insertion of new elementary ranges along constraint 1 and constraint 2 dimensions respectively.

4.2.2 Transform for multiple licenses

Let $L - 1$ received redistribution licenses are present and the L^{th} redistribution license is just obtained, two steps are needed. In the first step, new elementary ranges are inserted along each constraint dimension. In the second step, $L - 1$ length bit-vectors are modified to obtain L length bit vectors.

Let $[a_j, b_j]$ be a new constraint range in the j^{th} constraint dimension in a new license. First a search is made for a_j along the j^{th} constraint dimension. If a_j is present on the axis then no action is taken. Else, it is added in the numerical order on the axis (Each non-numeric constraint such as, region allowed for distribution, can also be easily converted into range of numbers). It forms two elementary constraint ranges by dividing an existing elementary constraint range, the same bit-vector as of the original range is assigned to both ranges. The similar process is used to insert b_j on the j^{th} constraint axis. In the next step, the L^{th} bit is appended (to LHS) to the bit-vectors corresponding to all the elementary ranges. A bit value equal to 1 is appended to all the elementary ranges between a_j and b_j along the j^{th} axis and bit 0 is appended otherwise. Figure 2(b) illustrates the transformed space after inserting a new received redistribution license $L_D^{r_2}$ having constraint ranges $(0, 200]$ and $[5, 15]$.

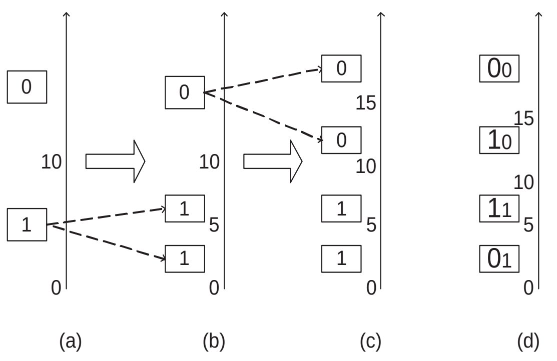


Figure 3: Transform for Multiple Licenses

Example 4: Let for the case of figure 2(a), the distributor acquired another license with constraint ranges (0, 200] and [5, 15] along the constraint dimensions 1 and 2 respectively. In figure 3 we show the steps 1 and 2 in above algorithm for the constraint dimension 2. According to above algorithm, first, the end point 5 is searched along the constraint dimension 2. Since it is not found so the point 5 is inserted along constraint dimension 2 as shown in figure 3(b). Since the point 5 divides the range (0,10] so a bit-vector same as that of the range (0,10] in figure 3(a) is assigned to both (0,5) and [5, 10]. Similarly, the point 15 is inserted, as represented in figure 3(c). Finally, in bit-vector modification step, a bit equal to 1 is appended to all bit vectors between 5 and 15, and a bit equal to 0 is appended to other bit-vectors. This step is illustrated in figure 3(d) with newly inserted bits in bigger font.

The transform gains efficiency,firstly, by taking advantage of redundancy in representation of licenses. If there is some common end point of a constraint range between multiple licenses then it is represented only once in the transformed domain. For example, in figure 2(b), end point 100 along constraint 1 axis is shared between $L_D^{r_1}$ and $L_D^{r_2}$, it is in both licenses but in the transform domain it is represented only once. Secondly, in contrast to the direct representation of licenses, data in the transformed domain can utilize efficient data structures, as discussed in the next sub-section.

4.3 Efficient Data Structure to Represent Constraint Dimensions

In bit-vector transform elementary ranges are present in increasing order along each constraint dimension. Thus, the elementary ranges along each constraint dimension can be represented with a search efficient data structure such as BST, AVL trees [15] [1], etc. AVL trees are the most efficient in searching. So, we use AVL trees to represent each constraint dimension.

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, if each A, B,..., and F represent a numerical value then A< B< C... <F is true. In our case, 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 represent an elementary range and store a bit-vector corresponding to the elementary range. As shown in figure 4, each leaf node stores a unique elementary range. The

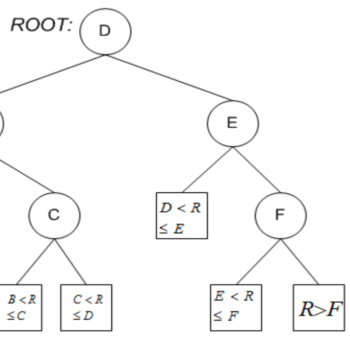


Figure 4: Representation of an AVL tree

readers may refer to [1] to get more details about AVL trees.

A separate AVL tree is designed for each constraint in the licenses, and 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 the received redistribution licenses. Whereas each leaf nodes represents a unique range R (i.e. elementary ranges) determined by the values stored in internal nodes and stores the bit vector corresponding to its elementary range. Next, we discuss the process of insertion of new licenses in AVL trees.

4.3.1 *Insertion of a New Received License*

Insertion of a new received redistribution license in an AVL trees is done in two steps. In the first step, AVL tree for j^{th} dimension AVL_j is modified by inserting j^{th} constraint range in the received redistribution licence in it. In the second step, bit-vector corresponding to each elementary constraint range is modified. Each step corresponds to the respective step(i.e. elementary range insertion and bit-vector modification) in section 4.2.2.

(A) Constraint Range Insertion. AVL_j is created when the first redistribution license for some content is received by a distributor. Let in the first received redistribution license the j^{th} constraint range is given by $[a^1_{j_r}, b^1_{j_r}]$. An internal node corresponding to $a^1_{j_r}$ is created along with an internal node with $b^1_{j_r}$ as its right child and a leaf node with the bit vector 0 as its left child. The node corresponding to $b^1_{j_r}$ has both left and right child nodes as leaf nodes. The left child node stores the bit-vector 1 and the right child node stores the bit-vector 0. Figure 5(a) shows an AVL tree formed by insertion of a constraint range $[0, 10]$.

For the insertion of j^{th} constraint range in an arbitrary k^{th} received redistribution license, $[a^k_{j_r}, b^k_{j_r}]$, except the first received redistribution license following process is followed.

1. A search is made for $a^k_{j_r}$ in the internal nodes of AVL_j. If an internal node containing $a^k_{j_r}$ is found then no modification is done in AVL_j.

2. If no exact match is found then an internal node with value equal to $a^k_{j_r}$ is inserted and both its child leaf nodes are created by copying the leaf node previously at the position, where the new node is inserted.

3. Rotation is performed, if needed to balance the tree.

4. Insertion of $b^k_{j_r}$ is done using a similar process that was used for the insertion of $a^k_{j_r}$.

5. Go to step B for bit-vector modification.

Insertion of the constraint range $[5, 15]$ in the AVL tree in figure 5(a), using the above process is shown in figures 5(b) to 5(d).

(B) Bit-vectors modification. All the bit vectors in the in the modified AVL tree need to be modified by appending them with a bit. So, we perform in-order traversal (Nodes are processed recursively by processing the left subtree, then processing the root, and finally the right sub-tree) in the AVL_j and process each internal node traversed one by one. If the value stored in the traversed internal node is in the range $(0, a^k_{j_r}), [a^k_{j_r}], (a^k_{j_r}, b^k_{j_r}), [b^k_{j_r}]$, and $(b^k_{j_r}, \infty)$ then the node is processed using one of the steps from the step 1 to 5 respectively.

1. If the node has any child node(s) as a leaf node(s) then the bit 0 is appended (to the LHS) to the bit-vector(s) corresponding to the child node(s). If the node traversed has no child node as leaf node then no action is performed.

2. If the node has a left child as leaf node then append 0 to the bit-vector corresponding to the left child. If the node has a right child as leaf node then append 1 to the bit-vector corresponding to the right child.

3. If the node has any child node(s) as a leaf node(s) then the bit 1 is appended (to the LHS) to the bit-vector(s) corresponding to the child node(s). If the node traversed has no child node as leaf node then no action is performed.

4. If the node has a left child as leaf node then append 1 to the bit-vector corresponding to the left child. If the node has a right child as leaf node then append 0 to the bit-vector corresponding to the right child.

5. If the node has any leaf nodes then append a value 0 to the bit-vector corresponding to that node.

According to above algorithm, in figure 5(d), the nodes containing value 0, 5, 10, and 15 are processed using the steps 1, 2, 3, and 4 respectively for modification of bit-vectors to obtain the AVL tree shown in figure 5(e).

4.3.2 *Search in AVL Trees*

For validation purpose, as we discuss in section 5, we need to search bit-vectors corresponding to the constraint values in newly generated licenses. Following steps are required for searching the bit-vector corresponding to a value v in AVL_j.

1. Make the *ROOT* node as current node.

2. Compare the value v with the value stored in the current node. If v 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.

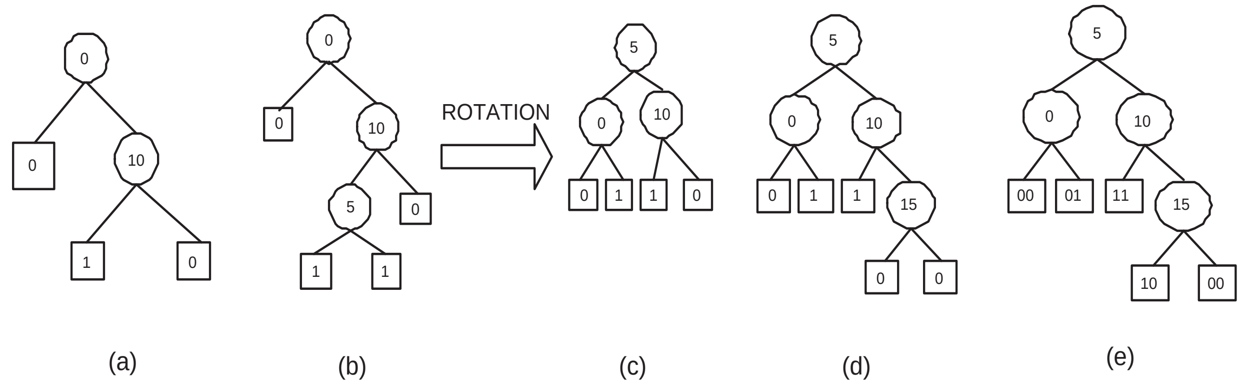


Figure 5: Operations on AVL tree

5. VALIDATION IN TRANSFORMED DOMAIN

In this section, we discuss our method of validation of issued usage and redistribution licenses in the bit-vector transformed domain. To make a better understanding, first, we discuss the basis we are using for validation of licenses.

5.1 Basis of Validation

In the transformed domain, we define a similar problem along each constraint dimension. For the j^{th} constraint dimension for a permission P_q, the problem is to determine the set S_j of received redistribution licenses such that the j^{th} constraint in the newly generated license is a subset of j^{th} constraint range in each redistribution license in the set S_j. And once we determine the set of received redistribution licenses for all d constraints for the permission P_q, the next step is to determine the set S; where S is the set of received redistribution licenses such that all the constraints in the generated license are subset of the respective constraint in all the redistribution licenses in the set S. It can be obtained by taking intersection of the individual sets obtained for all d constraint dimensions i.e.

$$S = S_1 \cap S_2 \cap .... \cap S_d. \qquad (1)$$

The generated license will be a valid license if the set S contain at least one received redistribution license. This basis of validation can be efficiently applied using bit-vectors, as shown below.

5.2 Validation of Usage Licenses

Let the value of j^{th} constraint for a permission P_q in a usage licenses be C_j . For validation, first we find the elementary constraint range along the j^{th} constraint dimension to which the constraint value C_j belongs, and this is done for all values of j i.e. all d constraints. Let the constraint value C_j belongs to the e_j^{th} elementary range with bit-vector $B_{P_q}^{je_j}$. In the bit-vector transformed domain each bit-vector has a unique position for each particular redistribution license. So, performing the logical $'AND'$ operation between them for all d values of j is equivalent to intersection operation performed between S_j values in equation (1). The resultant bit-vector (B_{P_q}) after $'AND'$ operation is given as:

$$B_{P_q} = B_{P_q}^{1e_1} \wedge B_{P_q}^{2e_2} \wedge ... \wedge B_{P_q}^{de_d} \qquad (2)$$

The bit corresponding to a received redistribution license in the bit-vector B_{P_q} will be 1 in equation (2) if that received redistribution license can be used to generate the new license. Thus, if no received redistribution license can be used to generate the given license then all bits will be 0 in B_{P_q}. Hence for validation purpose, we calculate numerical value of the resultant bit-vector B_{P_q}; if it is 0 then the license issued is not valid, else it is a valid license. For instance, a new license with the constraint values 156 and 12 is generated using the redistribution licenses L_D^{r1} and L_D^{r2}. In this case, as we can observe from figure 2, the point 156 belongs to the elementary range (100, 200]along constraint dimension 1, thus $B_{P_q}^{1e_1} = 11$. The point 12 belongs to elementary range (10, 15] along constraint dimension 2, thus $B_{P_q}^{2e_2} = 10$. According to equation (2), $B_{P_q} = B_{P_q}^{1e_1} \wedge B_{P_q}^{2e_2} = 11 \wedge 10 = 10$. As the numerical value of B_{P_q} is not zero, therefore it is a valid license. And, the value equal to 1 of the bit corresponding to the license L_D^{r2} shows that the issued license can be generated using the received redistribution license L_D^{r2}.

5.3 Validation of Redistribution Licenses

Let the range of j^{th} constraint in a received redistribution license L_D^r be $[a_{j_r}, b_{j_r}]$. A newly generated redistribution license belongs to L_D^r if each constraint range in the generated license is a subset of respective constraint ranges in L_D^r. Let the j^{th} constraint range in the issued redistribution license be $[a_{j_i}, b_{j_i}]$, it will be a subset of the constraint range $[a_{j_r}, b_{j_r}]$ if $a_{j_i} \leq a_{j_r}$, and $b_{j_i} \leq b_{j_r}$. Thus the bit corresponding to the license L_D^r will be 1 in the bit-vectors corresponding to the both end points of j^{th} constraint range in the license (a_{j_i} and b_{j_i}). So, we can perform an $'AND'$ operation between the bit vectors corresponding to both end points (a_{j_i} and b_{j_i}) to obtain the bit-vector $B_{P_q}^{je_j}$. Once we obtain the bit-vector for all d constraint dimensions, equation (2)can be used to validate the redistribution license in the same manner as used for validation of usage licenses.

6. PERFORMANCE OF THE METHOD

The proposed validation method using bit-vector transform outperforms the direct approach of validation, in which each issued license needs to be validated through received redistribution licenses one by one, significantly because of following reasons:

1. The proposed method utilizes the fact that most of the constraint dimensions can have fixed maximum number of elementary ranges in practical scenarios. If there are large number of possible values for a constraint then the owner can limit the maximum possible elementary ranges for each constraint using a granularity value. For example, the owner may limit the range of a constraint between 10 and 60 in further redistribution licenses with a granularity of 5 (i.e. end points of this constraint in further issued redistribution licenses must be a multiple of 5). So, in this case maximum 12 elementary ranges are possible, including 0 to 10, and 60 to ∞. Further a distributor might only be interested in constraint range value between 10 and 30. This further reduces number of elementary ranges for the distributor.

2. The proposed method can utilize the AVL trees for representation of constraint dimension. AVL trees have a logarithmic searching time [19][7][11] and hence can search faster along each constraint dimension during the validation process.

6.1 Mathematical Performance Analysis

We analyze the performance on the basis of validation time complexity and Storage space complexity.

(A) Validation Time Complexity..

According to figure 4, r_j elementary ranges along the j^{th} constraint dimension (for a permission P_q) can be represented with the help of $r_j - 1$ internal nodes in an AVL tree. So, the average searching time in the AVL tree will be proportional to $\log_2(r_j - 1) + 0.25$ along the j^{th} constraint dimension (if there are n nodes present in an AVL tree, then the average number of comparisons [19] is $\log_2(n) + 0.25$). This leads to average number of operations equal to $\sum_{j=1}^{d}(\log_2(r_j - 1) + 0.25)$ for searching in the elementary ranges along all d constraint dimensions, where d is the number of constraints for the permission P_q. Then we need to perform $'AND'$ operations between the bit-vectors obtained for all constraint dimensions (equation (2)). For performing one $'AND'$ operation between two bit-vectors of length N (represented in computer using k bit integers), the complexity is $C = \lceil N/k \rceil$, where $\lceil\rceil$ is the ceiling operator. And a total of $d - 1$ $'AND'$ operations ($'AND'$ operation is performed between d bit vectors) are needed to do the validation. So, the total validation time complexity will be proportional to $\sum_{j=1}^{d}(\log_2(r_j - 1) + 0.25) + (d - 1) * C$ number of operations. The method gains efficiency due to the use of logarithmic and ceiling operators, whose value increases relatively much slower as compared to N, as N grows.

(B) Storage Space Complexity..

Along the j^{th} constraint dimension there are r_j elementary ranges. k bits are needed to represent each elementary range, and each elementary range represents a bit-vector of length N (In computer, it will take $k * \lceil(N/k)\rceil$ bits). Furthermore, considering that we represent each dimension with an AVL tree so 2*k number of bits are needed to represent left and right child pointers in each internal node. This leads to storage space requirement of $\sum_{j=1}^{d} r_j * (k * \lceil(N/k)\rceil + 3 * k)$ bits, where d is the number of constraints for the permission P_q.

6.2 Results and Discussion

We performed experiments (in MATLAB) for different values of number of received redistribution licenses, N, to compare the performance of our approach with the direct approach for validation time and storage space complexity. For each value of N, we performed experiments 10,000 times; in each experiment we generated N number of received redistribution licenses, and issued licenses using them. The received redistribution licenses were generated by randomly choosing a random number of constraints(5 to 10 from total 15) for each permission(number of permissions is chosen randomly between 3 to7). All the constraints were defined with a minimum value, maximum value and granularity according to practical scenario. The range of constraints in received redistribution licenses were chosen randomly from all possible ranges. We selected value of $k = 32$, as most of the modern computers use 32 bit integer format. The approaches used to calculate the performance of direct method, and our method are discussed below.

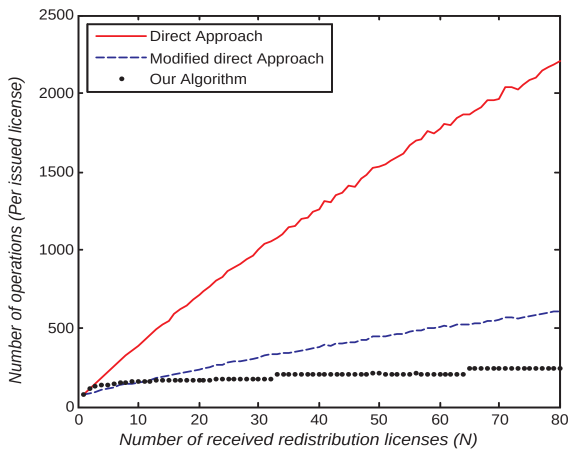
A) Direct Method. In direct method, as discussed in section 3, all the constraints in a newly generated licenses need to be compared with the respective constraint ranges in the received redistribution licenses one by one until we find a received redistribution such that all the constraints in the generated license are within the respective constraints the received redistribution license.

For comparison purpose, we modify the direct method to make it more search efficient by storing a pointer to the next redistribution license license with all the constraint in all the distribution licenses. Using the pointers, we can terminate the comparison with a redistribution license as soon as we find a constraint value such that its value is not within the respective constraint range in the redistribution license. And can go to the next redistribution license for comparison with the help of the pointer stored.

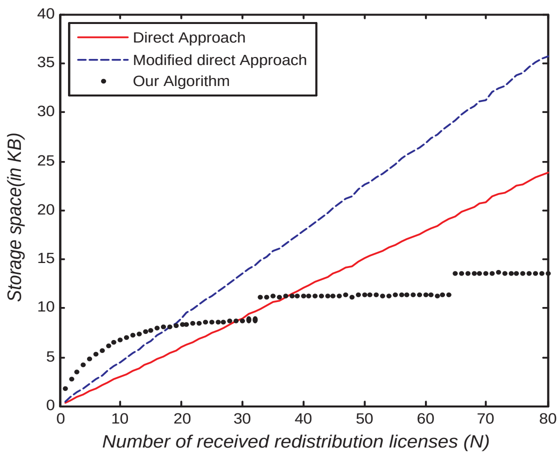
In both direct method and modified direct method, each comparison with a constraint range is counted as two operations. And for each N, number of operations are averaged over all the issued licenses for that N. Computation of storage space is done by averaging the storage space taken by all the received redistribution licenses generated during all 10,000 experiments for each N. Storage space for each redistribution license is the space required to store permissions, constraints, and pointers to the next redistribution license.

B) Our Method. We calculated the number of elementary ranges for all constraints using N redistribution licenses generated in each experiment. Thus, in case of $N = 3$, if the j^{th} constraint ranges for a permission in three licenses are [10, 20], [15, 25], and [20, 25] then r_j will be 5. Then, we simulate the equations: $\sum_{j=1}^{d}(\log_2(r_j - 1) + 0.25) + (d - 1) * C$, and $\sum_{j=1}^{d} r_j * (k * \lceil(N/k)\rceil + 3 * k)$ for validation time storage space complexity using different values of elementary ranges (these act as random variables) calculated in each experiment.

The comparison between direct method, modified direct method and our method for validation time (per issued license) and storage space (for N licenses) is shown in figure 6(a) and 6(b) respectively. We can observe that for our method both validation time and storage space requirements does not change by big amount as N grows to a large value, which is one of the most important requirement for smooth running of system. Our method gives validation time perfor-



(a)



(b)

Figure 6: Case of single range for each constraint

mance comparable to the direct and modified direct methods for small number(N<10 in this experiment) of redistribution licenses and outperforms both the methods for most of the practical values of N for business requirements. Storage space requirement becomes constant (except abrupt change at N=32, and N=64) for higher values of N because of the fixed maximum possible elementary ranges for each constraint. As N increases, the number of elementary ranges for each constraint tend towards its fixed maximum value and thus the probability of increase in elementary rages becomes lesser. And, once the number of elementary rages reaches its maximum value then there is no further possibility of increase in elementary ranges. Abrupt changes in both the curves observed at N=32, and N=64 are due to the ceiling operator.

In the above analysis, we considered only single range for each constraint in the redistribution licenses. But, in reality there may exist multiple ranges for the constraints. The relative performance of our method with respect to the direct and modified direct approach will become even better in this case. As the number of ranges for each constraint increase, the validation time required for the direct and modified direct methods would also increase proportionally. But in our case, for each individual N, value of the term containing ceiling operator $((d-1)*C)$ in the validation time complexity equation does not change and for the other term $(\sum_{j=1}^{d}(\log_2(r_j-1)+0.25))$ the increase will only be logarithmic as compared to linear in case of direct method. We performed experiments for the case of multiple licenses by generating by generating multiple ranges for each constraints with the following probabilities- single range: 0.40, two ranges: 0.25, three ranges: 0.20 and four ranges: 0.15. Comparison between all three approaches in case of multiple ranges for constraints is shown in figure 7(a) and 7(b). We can observe, our method always performs much better than the direct and modified direct method in terms of validation time and gains much more efficiency relatively in terms of storage space too.

7. CONCLUSION

In this paper, we proposed the bit-vector transform based license organization technique and a method to do the license validation in bit-vector transform domain. We performed mathematical and experimental analysis for the proposed bit-vector transform based validation method. Both mathematical and experimental results show that the our proposed method performs better than the direct method and modified version of direct method in terms of both validation time and storage space. This makes our proposed Bit-vector transform based license organization method a good choice for representation of licenses.

Acknowledgement. Thanks to the Agency for Science, Technology and Research ($A*STAR$), Singapore for supporting this work under the project 'Digital Rights Violation Detection for Digital Asset Management' (Project No: 0721010022).

8. REFERENCES

[1] A. Andersson. General balanced trees. *Journal of Algorithms*, 30(1):1–18, 1999.

[2] A. Arnab and A. Hutchison. Fairer usage contracts for drm. In *Proceedings of the 5th ACM workshop on Digital rights management*, pages 1–7, 2005.

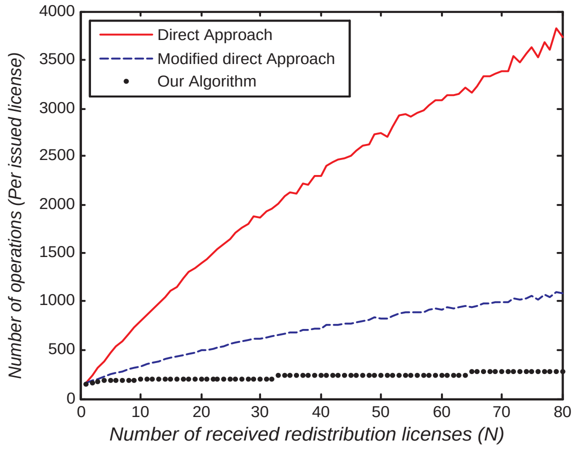
[3] A. Arnab and A. Hutchison. Drm use license negotiation using odrl v2. 0. In *Proceedings of the 5th International Workshop for Technical, Economic and Legal Aspects of Business Models for Virtual Goods*, 2007.

[4] L. Chiariglione. A walkthrough in the dmp phase ii specification, 2006. Available at: http://www.chiariglione.org/docs/idp-2_overview.htm.

[5] S. Emmanuel and M. S. Kankanhalli. Digital rights management issues for video. *Multimedia Security Handbook*, 8(6):759–787, 2003.

[6] S. Emmanuel and M. S. Kankanhalli. A digital rights management scheme for broadcast video. *Multimedia Systems*, 8(6):444–458, 2003.

[7] C. C. Foster. Information retrieval: information



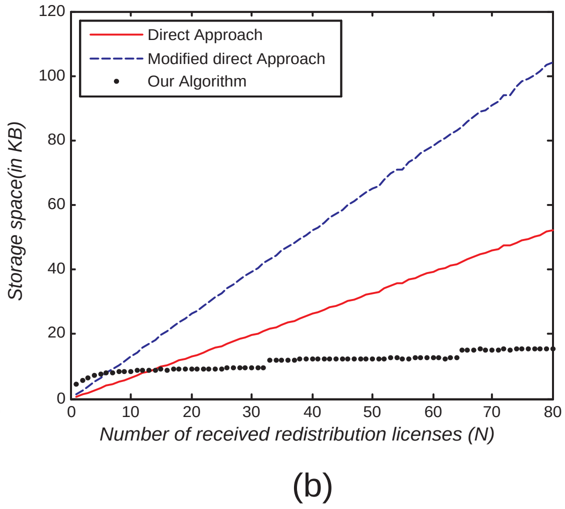


Figure 7: Case of multiple ranges for each constraint

storage and retrieval using avl trees. In *Proceedings of the 1965 20th national conference*, pages 192–205. ACM New York, NY, USA, 1965.

[8] 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*, 73(3):533–549, 2004.

[9] P. A. Jamkhedkar and G. L. Heileman. The role of architecture in drm vendor economics. In *International Conference on e-Commerce*, Porto, Portugal, 2005. IADIS.

[10] P. A. Jamkhedkar and G. L. Heileman. Digital rights management architectures. *Computers and Electrical Engineering*, 35(2):376–394, 2009.

[11] B. Jon Louis and H. F. Jerome. Data structures for range searching. *ACM Comput. Surv.*, 11(4):397–409, 1979.

[12] Q. Liu, R. Safavi-Naini, and N. P. Sheppard. Digital rights management for content distribution. pages 49–58. Australian Computer Society, Inc. Darlinghurst, Australia., 2003.

[13] X. Liu, T. Huang, L. Huo, and L. Mou. A drm architecture for manageable p2p based iptv system. In *IEEE International Conference on Multimedia and Expo, 2007*, pages 899–902, 2007.

[14] S. K. Nair, A. S. Tanenbaum, G. Gheorghe, and B. Crispo. Enforcing drm policies across applications. In *Proceedings of the 8th ACM workshop on Digital rights management*, pages 87–94, 2008.

[15] B. Pfaff. Performance analysis of bsts in system software. *ACM SIGMETRICS Performance Evaluation Review*, 32(1):410–411, 2004.

[16] A. Sachan, S. Emmanuel, A. Das, and M. S. Kankanhalli. Privacy preserving multiparty multilevel drm architecture. In *Workshop on Digital Rights Management, 6th IEEE Consumer Communications and Networking Conference, 2009.*, pages 1–5, 2009.

[17] R. Safavi-Naini, N. P. Sheppard, and T. Uehara. Import/export in digital rights management. In *Proceedings of the 4th ACM workshop on Digital rights management*, pages 99–110. ACM New York, NY, USA, 2004.

[18] T. Sans, F. Cuppens, and N. Cuppens-Boulahia. Opa: Onion policy administration model-another approach to manage rights in drm. *International Federation for Information Processing*, 232:349, 2007.

[19] A. M. Tenenbaum and M. J. Augenstein. *Data structures using Pascal.* Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1986.

[20] Y. Zheng, D. He, H. Wang, and X. Tang. Secure drm scheme for future mobile networks based on trusted mobile platform. In *Wireless Communications, Networking and Mobile Computing, 2005. Proceedings. International Conference on*, volume 2, 2005.