Efficient Aggregate Licenses Validation in DRM

Amit Sachan[1], Sabu Emmanuel[1], and Mohan S. Kankanhalli[2]

[1] School of Computer Engineering, Nanyang Technological University, Singapore
[2] School of Computing, National University of Singapore, Singapore
amit0009@ntu.edu.sg, asemmanuel@ntu.edu.sg, mohan@comp.nus.edu.sg

Abstract. DRM systems involve multiple parties such as owner, distributors and consumers. The owner issues redistribution licenses to its distributors. Distributors in turn using their received redistribution licenses can issue new redistribution licenses to other distributors and usage licenses to consumers. For rights violation detection, all newly generated licenses must be validated against the redistribution license used to generate them. The validation becomes complex when there exist multiple redistribution licenses for a media. In such cases, it requires evaluation of an exponential number of validation equations with up to an exponential number of summation terms. To overcome this, we propose a prefix tree based method to do the validation efficiently. Experimental results show that our proposed method can reduce the validation time significantly.

1 Introduction

Digital rights management(DRM)systems generally [4][5] involve multiple parties such as owner, distributors and consumers. The owner gives the rights for redistribution of contents to distributors by issuing redistribution license. The distributors in turn use their received redistribution license to generate and issue new different types of redistribution licenses to their sub-distributors and usage licenses to consumers. A redistribution license allows a distributor to redistribute the content as per the permissions and constraints [6] specified in it. Thus, as part of the rights violation detection, it is necessary to validate these newly generated licenses against the received redistribution licenses with distributor. A validation authority does the validation of all the newly generated licenses.

Both redistribution(L_D) and usage licenses(L_U) for a content K are of the form: $\{K;\ P;\ I_1, I_2, ..., I_M;\ A\}$, where P represents permissions (e.g. play, copy, etc. [5]), I_i represents i^{th} ($1 \leq i \leq M$) instance based constraint and A represents aggregate constraint. Instance based constraints in redistribution licenses are in the form of ranges such as region allowed for distribution, etc. Instance based constraints in usage licenses such as expiry date, region allowed etc. may be in the form of a single value or range[5]. The range/value 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 in the redistribution license [5]. Aggregate constraint decides the number of *counts* that can be distributed or consumed using a redistribution or usage

license respectively. The sum of *counts* in all the licenses generated using a redistribution license must not exceed the aggregate constraint $counts(A)$ in it.

For business flexibility, distributors may need to acquire multiple redistribution licenses for the same content[5]. In this case, if all instance based constraints' ranges/values in an issued license are within the respective constraint range in at least one redistribution license then the issued license is said to be instance based validated[5]. The problem of aggregate validation becomes harder in case a license can be instance based validated using more than one redistribution licenses(say a set S). This is because the validation authority needs to select a redistribution license from the set S for aggregate validation. Selecting a redistribution license randomly may cause potential loss to the distributors as shown in Sec. 2. Thus, we propose a better aggregate validation approach using validation equations in Sec. 2. But, the approach requires validation using exponential number of validation equations, containing up to exponential number of summation terms. This necessitates an efficient validation mechanism. So, we propose an efficient aggregate validation method using the prefix tree based structure[1][2][3]. The experiments show that our approach reduces the validation time and memory requirement significantly. To the best of our knowledge, the work presented in this paper is the first for the efficient aggregate licenses validation in DRM.

Rest of this paper is organized as follows. Section 2 discusses problem definition. Section 3 describes our proposed validation method. Section 4 presents the performance analysis. Finally, Section 5 concludes the paper.

2 Problem Definition

In case of multiple licenses, a newly generated license can be instance based validated using more than one redistribution license. For aggregate validation, selecting one redistribution license randomly out of multiple redistribution licenses may cause potential loss to the distributors as illustrated in example 1.

Example 1. Consider three redistribution licenses acquired by a distributor to distribute the play permissions according to two instance based constraints(validity period T, and region allowed R) and aggregate constraint A.
$L_D^1 = \{K; Play; I_D^1 : T = [10/03/09, 20/03/09], R = [X, Y]; A_D^1 = 2000\}$
$L_D^2 = \{K; Play; I_D^2 : T = [15/03/09, 25/03/09], R = [X]; A_D^2 = 1000\}$
$L_D^3 = \{K; Play; I_D^3 : T = [15/03/09, 30/03/09], R = [Y]; A_D^3 = 3000\}$

Now, the distributor generates the usage license $L_U^1 = \{K; Play; I_U^1 : T = [15/03/09, 19/03/09], R = [X]; A_U^1 = 800\}$. L_U^1 can be instance based validated using L_D^1 and L_D^2[5]. Let the validation authority randomly picks L_D^2 for aggregate validation then remaining *counts* in L_D^2 will be 200(i.e. 1000-800). Next, let the distributor generates $L_U^2 = \{K; Play; I_U^2 : T = [21/03/09, 24/03/09], R = [X]; A_U^2 = 400\}$. L_U^2 can only be instance based validated using L_D^2. The validation authority will now consider L_U^2 as invalid as L_D^2 now cannot be used to generate more than remaining 200 counts. 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 will result in both L_U^1 and

L_U^2 as valid licenses. Thus, the challenge is to do the aggregate validation such that the distributors can use their redistribution licenses in an intelligent way. We present a method to do the aggregate validation using validation equations.

A Method for Aggregate Validation: Let a distributor has N received redistribution licenses for a content and the set of redistribution licenses be represented as $S^N = [L_D^1, L_D^2, ..., L_D^N]$. Let $SB^r[S]$ denotes the r^{th} subset of the set S of redistribution licenses. Thus if a set S contains k received redistribution licenses then $r \leq 2^k - 1$. An issued license is said to belong to a set S if it can be instance based validated using all licenses in the set. E.g. L_U^1 in example 1 belongs to the set $[L_D^1, L_D^2]$.

Let $C[S]$ denotes the sum of permission counts in all previously issued licenses that belongs to the set S of redistribution licenses. Let $E^i[S]$ be the i^{th} redistribution license in the set S and $A(x)$ be the aggregate *count* in the received redistribution license x. For deriving the first equation, we use the fact that the aggregate of the *permission counts* in all previously issued licenses must not exceed the sum of the allowed *permission counts* in all the redistribution licenses with the distributor. Further, each valid issued license belongs to only one set of redistribution licenses out of the total $2^N - 1$ possible sets(due to N redistribution licenses). Therefore, in equation form we can write it as:

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

The LHS of equation 1 represents the sum of *counts* in the issued licenses that belongs to the set formed by any possible combination of all N licenses(each possible combination can be represented by a subset of the set S^N). The RHS denotes the summation of maximum allowed *permission counts* in all the redistribution licenses with the distributor(as S^N is the set of all N licenses).

Although equation 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 subset of the set S^N, as shown in example 2 below.

Example 2. Consider the redistribution licenses in example 1. The above inequality ensures only the sum of all the play *counts* in the licenses issued must be less than 6000 i.e. $C_{Play}[L_D^1] + C_{Play}[L_D^2] + C_{Play}[L_D^3] + C_{Play}[L_D^1, L_D^2] + C_{Play}[L_D^1, L_D^3] + C_{Play}[L_D^2, L_D^3] + C_{Play}[L_D^1, L_D^2, L_D^3] \leq 6000$. But, it may not be able to prevent the violation due to issuing of excess counts for other combination of licenses. Equation 1 can be satisfied even if aggregate of the *counts* generated for play permission using only L_D^1becomes more than 2000 i.e. $C_{Play}[L_D^1] > 2000$, or using only L_D^1 and L_D^3 becomes more than 5000 i.e. $C_{Play}[L_D^1] + C_{Play}[L_D^3] + C_{Play}[L_D^1, L_D^3] > 5000$, but both these conditions are invalid. Thus, if there are N redistribution licenses then violation can happen in $2^N - 2$ ways(all proper subsets of S^N). So, we require an inequality for each subset of S^N. For r^{th} subset of set S^N, $SB^r[S^N]$, the validation equation is given as:

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

where, $m = |SB^r[S^N]|$ is the cardinality of the set $SB^r[S^N]$. Equation 2 can be interpreted similar to equation 1 by replacing S^N by $SB^r[S^N]$. These inequalities ensure that in case of violation at least one inequality will not be satisfied.

Requirement of Efficient Aggregate Validation: If a newly generated license can be instance based validated using k number of redistribution licenses then the set formed due to k licenses will be present in $2^N - 2^{(N-k)}$ validation equations. Validation using such a large number of validation equations every time a new license is issued is computationally intensive. So, instead of doing validation online, we collect the logs of the sets of redistribution licenses(which issued licenses belong) and *permission counts* in issued licenses. We refer each entry corresponding to an issued license as a *record*. During the offline aggregate validation firstly different set *counts* can be aggregated and secondly the aggregated set *counts* can be applied to the validation equations. If M number of records are present, the time complexities for the first and second step would be $O(M * 2^N)$ and $O(3^N)$ respectively. The time complexities for both *set counts* aggregation and validation may be quite high for practical purposes. Thus, an efficient method is required to reduce the total validation time required.

3 Proposed Efficient Aggregate Validation Method

In this section, we present validation tree, a prefix tree based structure to do the validation of validation equations(equations 1 and 2) efficiently. The proposed structure and validation algorithm is based on the observation that validation equation for a set S aggregates the set counts of all the sets that are subset of the set S. The structure can compactly represent the log records for offline validation and use properties of prefix tree structure to do the validation efficiently.

Generation of *Validation Tree*: Initially, a *root* node is created. The tree is then expanded using the log records. Each node 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 current node). Redistribution licenses are indexed in the order they were acquired by distributor i.e. if L_D^j is acquired before L_D^k then $j < k$ and a redistribution license can act as a prefix only to the redistribution licenses having index greater than the index of redistribution license, as shown in Fig. 1. Child nodes of a node are ordered in increasing order of their indexes.

Records Insertion: Let the set of redistribution licenses in the record that needs to be inserted be given by:R=[r, R']and the *permission count* value be given by *count*, where, r is the first redistribution license and R' is the set of remaining redistribution licenses. Initially *root* node is allocated to T. Algorithm $Insert(T, R, count)$ is used to insert the records in the *validation tree*. Fig. 1 shows the *validation tree* designed based on the Alg. 1 for the records in Fig. 2.

Validation using *Validation Tree*: To do the validation efficiently, we 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. Thus, in a prefix tree based structure, if

Algorithm 1 *Insert*(T, R=[r, R'], *count*)

1. If T has a child T' such that $T'.L$=r then no action is taken.
2. Else add a node T' such that $T'.L$=r and $T'.C$=0 as the child node of T.
3. If R'=null set then $T'.C$= $T'.C$+*count*. Else, call Insert(T', R', *count*).

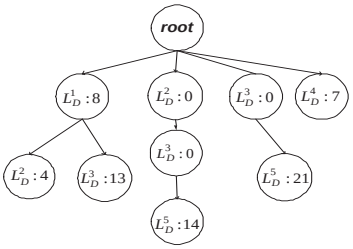

Fig. 1. Generation of *validation tree*

Fig. 2. Table of log records

Serial Number	Set of redistribution licenses	Count
1	$[L_D^1]$	8
2	$[L_D^1, L_D^2]$	4
3	$[L_D^2, L_D^3, L_D^5]$	14
4	$[L_D^1, L_D^3]$	13
5	$[L_D^3, L_D^5]$	9
6	$[L_D^4]$	7
7	$[L_D^3, L_D^5]$	12

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 to travel the child nodes of the node N_1. Another fact we use is that a set containing n licenses cannot be a subset of a set containing less than n licenses.

For N redistribution licenses, we can map each set of redistribution licenses corresponding to a validation equation into an N bits bit-vector. The bits from LSB to MSB correspond to a particular 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. E.g. If N=10 then the bit-vector for the set $[L_D^1, L_D^3, L_D^9, L_D^{10}]$ will be 1100000101. So, if we consider bit-vectors in integer format then all possible sets can be represented using the values from 1 to $2^N - 1$.

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<< (j − 1) AND i) ≠ 0
        <<: left shift operator
        then
            A_v=A_v+ A(j).
            licNumber=licNumber+1;
    Call C_v=VaLHS(T, i, licNum).
    if C_v ≤ A_v then
        Print(Valid Equation).
    else
        Print(Invalid Equation).
```

Algorithm 3 *VaLHS*(T,B,$licNum$)

```
Subroutine: Process(T,B,licNum) {
while (licNum > 0) do
    foreach child of T do
        Let the current child be T'.
        Temporary variables: i and j.
        i=index of license in node T'
        j=1<<(i − 1).
        if (B AND j ≠ 0) then
            C_v=C_v+T'.C.
            licNum=licNum-1.
            if (licNum > 1) then
                Call Process(T', B, licNum).
} Return(C_v).
```

The *validation tree* is used to compute the LHS of the equations 1 and 2. RHS(let A) is calculated directly using redistribution licenses E.g. the RHS(A) for the set $[L_D^1,L_D^3,L_D^9,L_D^{10}]$ is given by: A=$A(L_D^1)$+$A(L_D^3)$+$A(L_D^9)$+$A(L_D^{10})$. Let T represents the *root* node initially, and B represents the bit-vector for the set

of redistribution licenses for validation equation. The algorithm *Validation*(T) evokes the validation process for all possible validation equations. First, it calculates the RHS of each validation equation. Second, it calls *VaLHS*(T,B,$licNum$)to calculate the LHS. Finally, it compares the RHS and LHS to validate the equation. The algorithm *VaLHS*(T,B,$licNum$) traverse the *validation tree* for the set of redistribution license determined by the bit-vector B. $licNum$ is the number of redistribution licenses in the set corresponding to the current validation equation. For illustration, consider the validation using validation equation for the set [L_D^1,L_D^2,L_D^4] for validation tree in Fig. 1. B and $licNum$ for this set will be 01011 and 3 respectively. Let C_v denotes the LHS of the current validation equation for the set [L_D^1, L_D^2, L_D^4], it is initialized to 0 for every validation equation. The algorithm traverses the nodes *root*, *root*$\rightarrow L_D^1$, *root*$\rightarrow L_D^2$, *root*$\rightarrow L_D^4$ and *root*$\rightarrow L_D^1 \rightarrow L_D^2$. The final value of C_v for this case is 19.

4 Performance Analysis

We performed experiments on on Intel(R) core(2) 2.40 GHZ CPU with 2 GB RAM. To perform the experiments, first we 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 *permission counts* is saved in the log records. For experiments, each redistribution license is assumed to contain aggregate *permission counts* in between 5000 and 15000. Each issued license is assumed to contain permission counts in between 10 and 30.

We evaluate our proposal against the direct approach and a modified approach. In the direct approach, we scan the log records to find the subsets for the set corresponding to each validation equation and then aggregate their *set counts*. Whereas, in modified approach, we preprocess the log records to first aggregate the *set counts* for the sets containing the same redistribution licenses. Since, in practice many issued licenses may belong to the same set of redistribution licenses therefore scanning the modified log records would take lesser time. Table 1 summarizes the validation time performance. The experiments show that our proposed algorithm enhances the performance at least by 500 times and 10 times as compared to the direct and modified approach respectively. The large performance enhancement as compared to the direct method is mainly due the compact data representation in *validation tree*. However, in modified approach, the main reason for performance enhancement is the efficiency of our proposed subset search algorithm in the *validation tree*. Thus, it can be concluded that the *validation tree* gains efficiency both by compactly representing the log records and efficiently searching the subsets. Fig. 3 compares the performance in terms of memory requirement. The memory requirements for our approach is much less as the records can be stored in a much compact form in the *validation tree*.

5 Conclusion

Rights violation detection is an important security issue in DRM systems. In this paper, we presented a violation detection mechanism for distributors using the

Table 1. Comparison of validation time required(in milliseconds)

n	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Direct	.02	.05	.23	.80	2.12	5.80	15	39.33	93	188
Modified	0	.00031	.00094	.0034	.0122	.04	.11	.30	.74	1.64
Proposed	0	.00016	.00031	.0011	.0035	.01	.03	.06	.14	.33
n	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Direct	422	890	1953	4047	9078	21188	47906	108281	239953	524641
Modified	3.828	8.8	21.6	43.5	95	203	438	937	2063	4485
Proposed	.687	1.44	3.07	6.4	17	31	47	110	250	500
n	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
Direct	$1.2*10^6$	$2.6*10^6$	$5.4*10^6$	$1.2*10^7$	$2.6*10^7$	$5.6*10^7$	$1.3*10^8$	$2.9*10^8$	$6.1*10^8$	
Modified	9640	20922	43844	95640	195953	399281	840109	1761812	3810765	7648375
Proposed	1063	2219	4515	9563	19406	39422	79531	162938	328672	669360

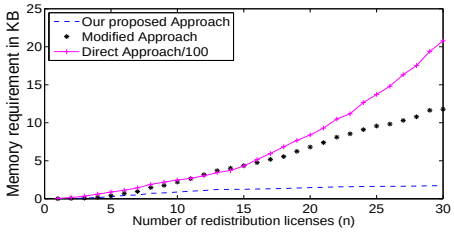

Fig. 3. Comparison of memory requirement

aggregate validation of licenses. However, large number of validation equations make the task difficult. Thus, we proposed '*validation tree*', a prefix tree based data structure to do the validation efficiently. The experiments show that *validation tree* performs better than the direct and a modified approach of validation in terms of validation time and memory requirements.

Acknowledgement: This work is supported by A*STAR Singapore, under project No: 0721010022, title:'Digital Rights Violation Detection for Digital Asset Management'.

References

1. Eavis, T., Zheng, X.: Multi-level frequent pattern mining. In: Database Systems for Advanced Applications(DASFAA), pp. 369–383. (2009).
2. Grahne, G., Zhu, J.: Fast algorithms for frequent itemset mining using FP-trees. Knowledge and Data Engineering, IEEE Transactions on, 17(10):1347–1362, (2005).
3. Han, J., Pei, J., Yin, Y., Mao, R.: Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data Mining and Knowledge Discovery, 8(1):53–87, (2004)
4. Hwang, S.O., Yoon, K.S., Jun, K.P., Lee, K.H.: Modeling and implementation of digital rights. The Journal of Systems and Software, 73(3):533–549, (2004)
5. Sachan, A., Emmanuel, S., Kankanhalli, M.S.: Efficient license validation in MPML DRM architecture, In: proceedings of the 9th ACM workshop on digital rights management, Chicago, pp. 73–82, (2009)
6. Safavi-Naini, R., Sheppard, N.P., Uehara, T.: Import/export in digital rights management. In: proceedings of the 4th ACM workshop on digital rights management, pp. 99–110, (2004)