A DRM System Supporting What You See Is What You Pay

Bin B. Zhu[1], Yang Yang[2], and Tierui Chen[3]

[1] Microsoft Research Asia, Beijing 100080, China
binzhu@microsoft.com
[2] Dept. of Elec. Eng. & Info Sci., Univ. of Sci. & Technol. of China,
Hefei, Anhui 230027, China
wdscxsj@ustc.edu
[3] Inst. of Computing Technology, Chinese Academy of Sciences, Beijing 100080, China
chentierui@software.ict.ac.cn

Abstract. We present a Digital Rights Management (DRM) system that supports what you see is what you pay. In our system, multimedia is compressed with a scalable codec and encrypted preserving the scalable granularity and multi-access capability. This paper focuses on the DRM modules enabling efficient key generation and management. We employ a light license server which stores only the master keys of content publishers, which are used to regenerate decryption keys for clients during license acquisition. All the remaining information needed in key generation is efficiently packaged in a DRM header of protected content. The DRM header is sent to a license server during license acquisition to allow the license server to generate a single key for a requested access, which is sent to the client in a license along with the acquired rights. The key is used by the client to generate all the remaining keys of subordinate accesses.

1 Introduction

With advances of digital technologies, more and more multimedia contents are released in or converted to digital formats. Wide access to high speed Internet makes distribution of digital multimedia efficient and easy. At the same time, the same technologies and Internet create rampant piracy of digital multimedia, which causes dramatic financial damage to the content owners and prevents content owners from releasing more contents in digital formats through the Internet as an efficient and cheap distribution channel. There is a great demand for technologies to protect digital contents from illegal access, copy, or sharing. Digital Rights Management (DRM) is a system to address such a need. A DRM system provides persistent management of all rights ranging from description, identification, trading, and protection to monitoring and tracking for digital contents from creation to consumption [1][2]. Such a system consists of many core technologies and essential parts such as rights expression language to describe rights to be managed, encryption and key generation and management to protect the content from unauthorized access and usage, and tamper-proof trusted DRM modules on the client side to ensure and manage the rights exactly as acquired.

R. Safavi-Naini and M. Yung (Eds.): DRMTICS 2005, LNCS 3919, pp. 341 – 355, 2006.

We have seen in recent years an increasing interest in DRM from both academia and the industry. Standardization of DRM systems has also been actively pursued. The Moving Picture Experts Group (MPEG) has adopted recently a DRM framework, eXtensions to the Intellectual Property Management and Protection (IPMP-X), for both MPEG-2 and 4 [3][4]. The Open Mobile Alliance (OMA) has also adopted a DRM system recently for mobile environments [5]. There are also several proprietary DRM systems available on the market. Typical commercial DRM systems include the Windows Media Rights Manager (WMRM) from Microsoft [6], Commerce and Rights|System from InterTrust [7], Electronic Media Management System (EMMS) from IBM [8], Helix DRM from RealNetworks [9], and the EBooks from Adobe [10]. A typical DRM system encrypts multimedia content which is distributed to consumers via distribution channels such as superdistribution. Superdistribution is a powerful distribution mechanism that treats ease of replication of digital content as an asset rather than a liability. Superdistribution actively encourages free distribution of digital content via any distribution mechanism imaginable to reach the maximum number of potential consumers. A DRM system enforces acquired rights of multimedia content through the trusted DRM modules on the client side and a license which contains the decryption key along with specifications of the rights a user has acquired. A license is usually individualized, typically encrypted with a key that is bound to the hardware of a user's player, so the license cannot be illegally shared with others. Control of content consumption rather than distribution is much more efficient in protecting digital assets in the digital world since modern networks, storage, and compression technologies have made it trivial to transfer digital content from one device or person to another.

The same multimedia content can be consumed with devices of a variation of characteristics and capacities such as mobile devices or PC. To enable different devices to play the same content, the traditional DRM approach is to compress and encrypt a single multimedia content into multiple copies, with each copy targeted at a specific application scenario such as a PC with high resolution display and computing power and storage, a 3G cellular phone with a small display and limited computing power and storage, etc. These multiple copies are all stored in a server to make them available for each individual user to select a copy that best fits his or her need. Another approach is to apply a transcoder at some node of the multimedia delivery path to generate a lower resolution or quality bitstream to fit in the targeted network condition or device capability. Decryption and re-encryption are typically used in performing such transcoding. A more elegant solution is to encode multimedia contents with a fine granularity scalability (FGS) codec. A scalable codec encodes a signal into a single codestream which is partitioned and organized according to certain scalable parameters or importance. Based on scalabilities offered by a codestream, each individual user can extract from the same codestream the best representation that fits his or her application. An FGS scalable codec offers near continuously optimal tradeoff between quality and rates over a large range. Unlike traditional approaches, a single scalable codestream is stored and used for all different applications, with possible simple adaptation manipulations such as truncations on the codestream. This capability of one-compression-to–meet–the–needs–of–all–applications is very desirable in many multimedia applications. Many scalable codecs have been proposed. Some have already been adopted by standard bodies. MPEG has adopted a scalable video

coding format called *Fine Granularity Scalability* (*FGS*) into its MPEG-4 standard [11]. The Joint Photographic Experts Group (JPEG) has adopted a wavelet-based scalable image coding format called *JPEG 2000* [12] and *motion JPEG 2000* [13]. Many schemes have been proposed in recent years to encrypt scalable codestreams such that fine granularity scalability is preserved in the encrypted codestream to enable direct truncations without decryption. Most of those schemes are described in the review paper [14].

One of the unique features offered by an FGS codec is multiple access types in a single scalable codestream. For example, a PC can show a high-fidelity full resolution video from a scalable video codestream, while a mobile phone can show a low quality video at a reduced resolution from the same codestream. Different accesses should be charged differently. It is natural to require a PC user to pay more for a high-fidelity full resolution video than a mobile phone user. A DRM system for FGS codestreams should preserve the property of multiple access types in a single DRM-protected codestream to enable the business model that charges different accesses differently. This means that a scalable codestream should be encrypted with multiple keys. Generation and management of multiple keys for different accesses of a scalable codestream are a challenge in the design of a DRM system.

We have been building a research prototype of a DRM system on top of the Microsoft Windows DRM system [15] to support scalable codestreams, esp. scalable encryption to enable direct truncations of encrypted codestreams and multiple accesses to support what you see is what you pay, as well as content and license roaming among devices of different characteristics in a digital home (eHome). An example application is to view multimedia at reduced quality and resolution, either free or at a small cost. If the content is good and a better version is desired, then the user can acquire a new license, and download the enhanced portion (i.e., the difference) of the encrypted content if needed. A typical case for content roaming is that a full version is downloaded to a PC, and then truncated to appropriate representations to fit other eHome devices. Appropriate licenses are also roamed to those devices. In this paper, we concentrate on the part of our DRM system related to the management of multiple keys to support multiple accesses with a single protected scalable codestream. The major contribution of this paper is that we propose and implement a DRM system to support a new business model of what you see if what you pay. In addition, we present an efficient key generation and management scheme to facilitate a light license server used in our DRM system. A license server does not need to remember the decryption keys for individual protected contents. Instead, only the publisher-specific master key is remembered by the license server, which is then used to generate content decryption keys. In typical DRM applications, the number of publishers is much less than the number of protected contents, therefore our license server is much cheaper to run and simpler to maintain. The system has a very small overhead on the file size. This design is very desirable in many DRM applications since license server is a single point of failure in a DRM system. To play a protected content, a player has to acquire a license from a license server if the license has not been acquired previously or has expired. Reliability and availability of the license server is essential in a DRM system. A light and simple license server enables deployment of many cheap yet secure servers to provide license granting servers for a DRM system, therefore increases reliability, scalability, and availability of the license

granting service. In addition, some of the low quality levels in a codestream can be unencrypted in our DRM system to enable free preview and content-based search with a single DRM-protected codestream.

This paper is organized as follows: In Section 2 the background of Microsoft's WMRM, JPEG 2000 and motion JPEG 2000 are briefly described. They are the basis in describing our DRM system. In Section 3 the detail of our DRM system is described. Experimental results are reported in Section 4 and the paper concludes with Section 5.

2 Background

2.1 Microsoft's Windows Media Rights Manager

Microsoft has developed a Windows based DRM system called *Windows Media Rights Manager* (WMRM). A developer can download the WMRM and format SDKs to build his or her own DRM applications. Fig. 1 shows the work flow of Microsoft's WMRM. The basic WMRM process is described as follows. More details can be found in [15].

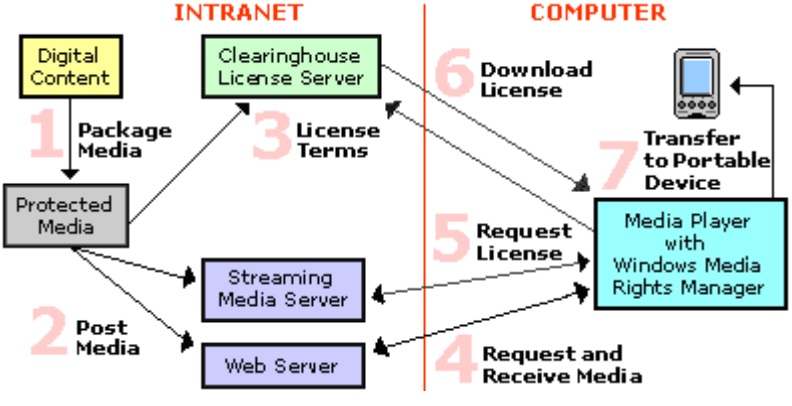

Fig. 1. Microsoft's windows media rights manager flow (from [15])

i. **Packaging.** The rights manager encrypts the digital media and then packages the content into a digital media file. The decryption key is stored in an encrypted license which is distributed separately from the media file. Other information such as a link to the license is added to the media file to facilitate license acquisition.

ii. **Distribution.** The packaged file is distributed to users through some distribution channels such as downloading, streaming, and CD/DVD. Superdistribution is a convenient distribution mechanism. There is no restriction on distribution of the packaged content.

iii. **Establishing a license server.** The content provider (referred to as the publisher in the following) chooses a license clearing house that stores the specific rights

or rules of the license and runs a license server which is used to authenticate the consumer's request for a license. Licenses and protected media files are distributed and stored separately to make it easier to manage the entire system.

iv. **License acquisition.** To play the protected content, a consumer first acquires a license which contains the decryption key and the rights the consumer has with the content. This process can be done in a transparent way to the consumer or with minimal involvement of the consumer (such as when payment or information is required).

v. **Playing the content.** A player that supports the DRM system is needed to play the protected content. The DRM system ensures that the content is consumed according to the rights or rules included inside the license. Licenses can have different rights, such as start times and dates, duration, and counted operations. Licenses, however, are typically not transferable. Each consumer has to acquire his or her own license to play the protected content.

Microsoft's WMRM is a complex and complete DRM system with a lot of advanced features such as revocation, license backup and restoration, obfuscation and other tamper-resistant mechanisms. By building our research DRM system on top of Microsoft's WMRM, we are able to leverage the existing modules and building blocks in WMRM and focus on the key DRM modules under studies. We believe that this approach is the easiest way to build a real and working DRM system for research purpose.

2.2 JPEG 2000/Motion JPEG 2000 and Scalable Encryption

For convenience, we use motion JPEG 2000 as the scalable codec to demonstrate our DRM system in this paper. Our DRM system is also applicable to other scalable codecs. JPEG 2000 [12] is the newest image coding standard based on the wavelet transform. In JPEG 2000, an image can be partitioned into smaller rectangular regions called tiles. Each tile is encoded independently. Data in a tile is divided into one or more components in a color space. A wavelet transform is applied to each tile-component to decompose it into different resolution levels. The lowest frequency subband is referred to as the resolution level 0 subband, which is also resolution 0. The image at resolution r ($r>0$) consists of the data of the image at resolution (r-1) and the subbands at resolution level r. Each subband is partitioned into smaller non-overlapping rectangular blocks called code-blocks. Each code-block is independently entropy-encoded. Bitstreams from code-blocks are distributed across one or more layers in the codestream. Each layer represents a quality increment. A layer consists of a number of consecutive bit-plane coding passes from each code-block in the tile, including all subbands of all components for that tile. JPEG 2000 also provides an intermediate space-frequency structure known as the precinct. A precinct is a collection of spatially contiguous code-blocks from all subbands at a particular resolution level. The fundamental building block in a JPEG 2000 codestream is called the packet, which is simply a continuous segment in the compressed codestream that consists of a number of bit-plane coding passes from each code-block in a precinct. Each packet is uniquely identified by the five scalable parameters: tile, component, resolution level, layer, and precinct. In motion JPEG 2000, each frame is

independently encoded as an image with JPEG 2000. Details on JPEG 2000 and motion JPEG 2000 can be found in [12][13].

Many scalable encryption schemes have been proposed for JPEG 2000 [16-20]. They can be used as a building block in our DRM system to encrypt motion JPEG 2000 codestreams. Multiple access control for a scalable codestream is equivalent to the access control of a partially ordered hierarchic set (poset). An efficient key scheme for a poset was proposed in [21], which is the basis of the key scheme of multiple accesses in our DRM system. Fig. 2 shows the key generation scheme in [21] that a parent node such as n_1 derives the key of its child node n_2 by using the parent's key k_1, the unique label l_2 of the node n_2, and the value $v_{1,2}$ of the edge linking the parent node n_1 to the child node n_2 : $k_2 = v_{1,2} + H(k_1, l_2)$, where $H(\cdot)$ is a cryptographic hash function.

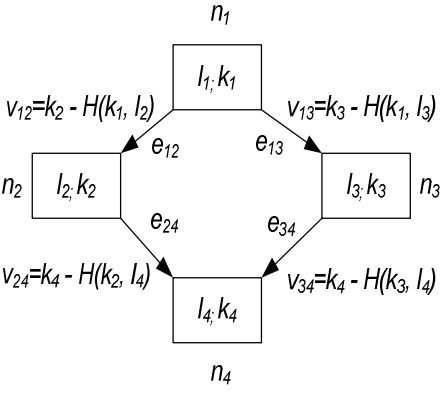

Fig. 2. Key scheme proposed in [21] which is the basis for the multiple access control of our DRM system. The arithmetic is modulo ρ which is a proper number.

3 Our Multi-access DRM System

In our DRM system, content is packed by the content owner or a publisher. The license terms of the content is then sent to a license server in a secure channel. Each publisher uses a publisher-specific master key in generating encryption keys to encrypt all the contents packed by the publisher. This master key has to be shared with the license server to enable the latter to generate decryption keys for clients. Symmetric encryption is used to encrypt the content so that the same key is used for both encryption and decryption. When a player plays a protected content, the DRM header packed with the content is extracted and the local license store and possibly the local secure storage of play statistics such as the number of times the content has been played are searched for a valid license of the content. If the search returns positive, the access key in the local license store is extracted along with the access node which is the subroot of all the accessible types and levels. The keys of all the lower access levels that the user has the right to access are derived and used by the

client DRM module to decrypt the corresponding data of the protected content. The decrypted data is then decoded and rendered to show to the user. The associated DRM parameters in a secure local storage such as the count of playing times are adjusted accordingly. If no valid license can be found from the local license store, the user is prompted to select a proper access type(s) and level(s) with possible payment, depending on the setting of the content owner. The information is sent to the license server along with the key generation information included in a DRM header packed with the protected content. The license server generates the key for the specific access type(s) and level(s) and returns to the client in a license which also contains the acquired rights by the user. The client receives the license and stores it in the local license store. The aforementioned process when a valid license is found in the local license store is repeated to play the protected content. The detail of the processes is described in the subsequent sections.

3.1 Content Packaging

A publisher must first generate a pair of public and private keys called content publisher public key $K_{P,Pub}$ and private key $K_{P,\mathrm{Pr}iv}$ and a publisher-specific master key $K_{P,M}$ before performing any content packaging. The publisher has also to obtain a certificate $C_{P,pub}$ for the public key $K_{P,Pub}$ from a certificate authority. The certificate will be used by a client to verify the publisher's public key in a DRM protected codestream. Armed with the above keys and the certificate, the publisher is ready to pack contents into DRM-protected codestreams.

To pack an individual piece of content, the publisher first generates a unique ID denoted as *KeyID* for the content. This *KeyID* is used to identify the license associated with the protected content in a local license store as well as to generate encryption keys. Since content is encrypted with a symmetric encryption primitive in our DRM system, decryption and encryption keys are the same. As we mentioned previously, multiple access control of a scalable codestream such as motion JPEG 2000 is equivalent to the access control of a poset with a single root node. The key of the root node is generated with the following equation in our DRM system:

$$k_{root} = MAC_{K_{P,M}}(KeyID), \tag{1}$$

where $MAC(\cdot)$ is a Message Authentication Code (MAC) which can be implemented with a secure keyed hash function. This equation means that the root key k_{root} of the multiple access control is a MAC of the *KeyID* with the master key as the key in generating the MAC.

To generate other encryption keys, the Hasse diagram representing the multiple access poset is first generated. Each node n_i except the root node $n_0 \equiv n_{root}$ is assigned a random key k_i. Those keys are used to encrypt the corresponding data for each frame. To avoid repetitively applying the same encryption parameters to encrypt different frames, each frame is inserted with a random initialization vector IV_{frame} which is used together with the above keys in encrypting the data for the

corresponding frame. A proper scalable encryption scheme is used in the encryption process. For motion JPEG 2000, any scheme described in [17-19] can be used.

To enable a node to derive all the keys of its descendants, the key scheme proposed in [21] for a poset is used in our DRM system. Each node n_i in the Hasse diagram is assigned a unique label l_i. Since an encrypted scalable codestream may be truncated to fit a certain application scenario, care has to be taken in generating the node labels. We want to ensure that the nodes generated by a truncated codestream match the original nodes without any truncation. This implies that the node labels should be invariant to truncations. In other words, truncation-invariant parameters that uniquely identify each node should be used in generating the node labels. For JPEG 2000 and motion JPEG 2000, canvas coordinates are such parameters and are used in our DRM system to generate truncation invariant node labels $\{ l_i \}$. The labels generated in this way are unique and therefore valid. A major advantage in generating the labels $\{ l_i \}$ in such a way is that the labels are not stored in a DRM-protected codestream. They can be regenerated once the Hasse diagram is generated. The file size overhead is therefore reduced. The value $v_{i,j}$ for each edge $e_{i,j}$ in the Hasse diagram that links a parent node n_i to its child node n_j is then calculated as:

$$v_{i,j} = k_j - H(k_i, l_j) . \tag{2}$$

A publisher packages the following information into a DRM-protected codestream:

- **KeyID:** This allows the proper license to be looked up in a local license store and requested from a license server. It is also used by a license server to generate the root key of the codestream. KeyID also contains information to identify the publisher.
- **License Server URL:** This allows a client to request a license from a proper license server.
- **Information of access types and levels:** This contains the information for the supported multiple access types and number of access levels for each type of the protected codestream packaged by the publisher.
- **Key generation information:** This contains the edge values of the Hasse diagram ordered in a certain order, and the information on how the edges are ordered and how the nodes are labeled. The information will be used to generate decryption keys by both the license server and the client.
- **Other DRM information:** This contains additional information about the DRM protection of the codestream such as DRM version, encryption scheme, etc.
- **Publisher's signature:** Everything above is signed with the publisher's private key $K_{P,\mathrm{Priv}}$.
- **Publisher's public key $K_{P,Pub}$ and certificate $C_{P,pub}$:** This part allows a client to check whether the DRM header has been tampered or not before requesting a license. This is important in preventing hackers from modifying

DRM header to point to a malicious server that a client machine might get viruses or other attacks when requesting license from the server.

3.2 Content Playing

Fig. 3 shows the sequence of steps a player executes in playing a codestream. In the first step, a player opens the codestream and checks whether the codestream is DRM-protected or not. If a DRM header is found, the DRM subsystem on the client side is called, and the DRM header is sent to the subsystem along with the requested action such as playing. When first launched, the DRM subsystem performs sanity checks to ensure that the subsystem is functioning well and there is no tampering to the system. It then extracts the publisher's public key and corresponding certificate from the DRM header and checks whether the DRM header has been tampered or not. If the checking is passed, *KeyID* is extracted from the DRM header and used to search the local license store to find any matching licenses. Our DRM system allows multiple licenses on a client side for the same protected content. Each license is assigned a priority level. All found matching licenses are ordered and checked for validity and to find out if there is any valid license that matches the requested action. Any invalid licenses are removed from the license store. If multiple valid and matching licenses are found, the default action is that the one that matches the client's characteristics and has the highest priority is used. For example, if the client is a powerful PC, the license containing the key of the highest access priority in the Hasse diagram, e.g., in case of motion JPEG 2000 the one with the highest resolution and best quality is used by default. A user can also set the DRM system to prompt the user to select from the set of licenses. If no valid license can be found from the license store, the user is asked to select the access type(s) and level(s) of which the decryption key is requested. An alternative approach is to select the access type(s) and level(s) automatically that best fits the client's hardware without user's input. A user can set up the DRM system to behave in either mode. A user may be requested to pay in this process, depending on the setting of the publisher. Once a license is acquired from a license server, the license is inserted into the local license store, and is used to address the current requested action. The detail of license acquisition is described in the next subsection.

In the next step, the DRM system extracts the information on access types and levels from the DRM header to generate the corresponding Hasse diagram of the multiple access control supported by the encrypted scalable codestream. The key generation information is also extracted from the DRM header to regenerate the unique label for each node and to assign the value for each edge in the Hasse diagram. The decryption key and the information of the corresponding node of the key are extracted from the license. The keys of all the descendants of the node are then derived in the following way: if node n_i is a parent of node n_j, and the edge linking the two nodes has a value $v_{i,j}$, then the key k_j of node n_j can be derived from the key k_i of the parent node n_i and the label l_j of the child node k_j:

$$k_j = v_{i,j} + H(k_i, l_j) . \tag{3}$$

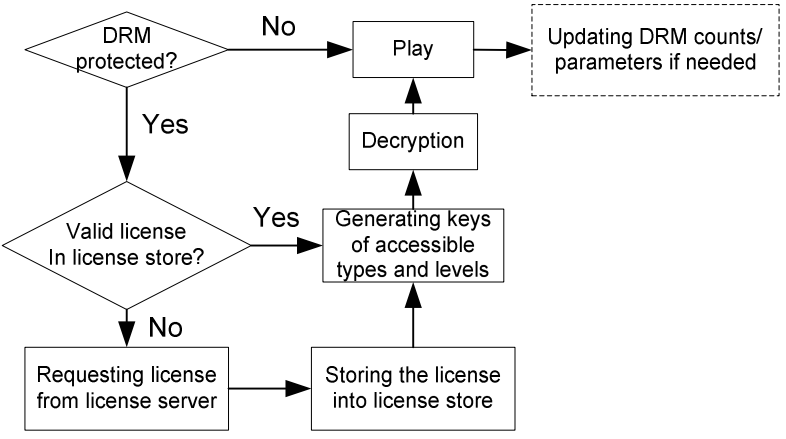

Fig. 3. Content playing flowchart

Eq. (3) is applied repetitively until the keys of all the descendants have been derived. Once the decryption keys that the user has rights to access are available, they are used to decrypt the encrypted data for each frame, together with the frame initialization vector IV_{frame}. Decrypted data is then decompressed and rendered for the user to view the content. At the last stage for DRM-protected content, the DRM parameters in the local secure storage such as playing counts are updated if needed to reflect the accomplished action requested by the user.

3.3 License and License Acquisition

License has to be acquired before the DRM-protected content can be played. Each license is individualized that only the targeted client can use it. This is achieved by generating a pair of public key $K_{C,Pub}$ and private key $K_{C,\mathrm{Pr}iv}$ for the client with a DRM key generation module at the DRM system installation phase. The private key $K_{C,\mathrm{Pr}iv}$ is tied with the hardware's unique IDs of the client's machine while the public key $K_{C,Pub}$ is signed by a trusted certificate authority. The certificate $C_{C,pub}$ of the client's public key $K_{C,\mathrm{Pr}iv}$ is stored at a local store, which will be used in communication with a license server during license acquisition time.

To acquire a license, the client DRM subsystem first extracts the license server's URL from the DRM header, and uses the client's public and private keys to authenticate with the license server through a public key based challenge and response protocol. In our DRM system, the license server also has a pair of public and private keys, with the public key signed by a certificate authority. After the mutual authentication, the client's DRM subsystem sends to the license server securely the *KeyID*, the information of access types and levels, the key generation information extracted from the DRM header along with the requested access type(s) and level(s)

selected by the user or automatically by the DRM subsystem that best fits the characteristics of the client's hardware, depending on the setting of the DRM subsystem by the user. The license server identifies the publisher from the received *KeyID* (recall that *KeyID* contains the unique identifier of the publisher), and extracts the publisher's master key $K_{P,M}$. The received information of access types and levels as well as the key generation information are used by the license server to regenerate the Hasse diagram and node labels $\{l_i\}$, and assign the edge values $v_{i,j}$. The root key k_{root} of the Hasse diagram is generated with Eq. (1), which is used in turn to generate the key $k_{req.node}$ of the node corresponding to the requested type(s) and level(s). The node is in fact the subroot of all the accessible types and levels the requesting user is entitled to access. This key is then packed into a license and sent to the requesting client. We note that in our DRM system, only a single key is sent in a license to a client. The remaining keys associated with the types and levels that are accessible to the client are generated by the client's DRM module based on the received node key $k_{req.node}$ and the information obtained from the DRM header of the protected content. Fig. 4 shows the process that a license server generates the key of the requested node.

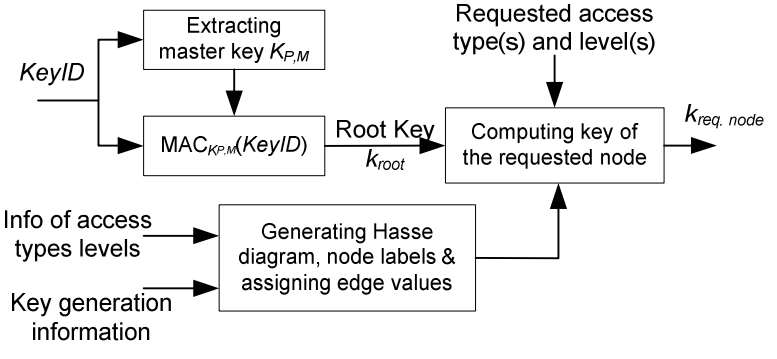

Fig. 4. Key generation by the license server

The rights a client requested and the info about the node of the key generated by a license server are also packed into the license to a client. A license is written in XML for flexibility and extensibility. Fig. 5 shows an example of a license. In the license, the node key $k_{req.node}$ is encrypted with the client's public key so that only the client with the corresponding private key can recover the node key. A client's private key is only available to the client DRM subsystem. This effectively prevents a client from sharing the content decryption keys with other clients. The data part of the license is signed by the license server and the chain of certificates is also provided in the license so that a client's DRM subsystem can check whether the license has been tampered or not. When a received license passes the checking, it is inserted into the local license store for future usage.

4 Experimental Results

Without loss of generality, we have implemented our DRM system for motion JPEG 2000 to test functionalities of the system and to conduct feasibility studies. Motion JPEG 2000 provides nice scalabilities ideal to test our DRM system. We should emphasize here that our DRM system is equally applicable to other scalable codecs and formats.

```
<?xml version="1.0" ?>
    <LICENSE version="1.0.0.0">
        <LICENSCONTENT>
            <DATA>
                <KID>...</KID>
                <ISSUEDATA>...</ISSUEDATA>
                <PRIORITY>...</PRIORITY>


                <ONSTORE>
                    <ACTION>
                        secstate.playcount = 5;
                    </ACTION>
                </ONSTORE>

                <ONACTION type=”Play”>
                    <CONDITION>
                        secstate.playcount > 0
                    </CONDITION>
                    <ACTION>
                        secstate.playcount --
                    </ACTION>
                </ONACTION>

                <KEYDATA>
                    <KEYALGORITHM type="SCALABLEDRM" />
                    <PUBKEY type="client">...</PUBKEY>  ← Client's public key.
                    <VALUE>...</VALUE>  ← The node key k_req.,node encrypted by the client's public key.
                    <NODEINFO>...</NODEINFO>  ← The node info associated with the node key.
                </KEYDATA>
            </DATA>

        <SIGNATURE>
            <HASHALGORITHM type="SHA" />
            <SIGNALGORITHM type="SCALABLEDRM" />  ← Signature signed by the license server.

            <VALUE>...</VALUE>
        </SIGNATURE>

        <CERTIFICATECHAIN type="SCALABLEDRM">
            <CERTIFICATE>...</CERTIFICATE>  ← The certificates for the license server.
        </CERTIFICATECHAIN>
         <CONTENTPUBKEY>...</CONTENTPUBKEY>

    </LICENSCONTENT>
</LICENSE>
```

Fig. 5. Example of a license to a client

As we mentioned previously, our DRM system was built on top of Microsoft's WMRM. We maximized reuse of the DRM modules offered by the SDKs of Microsoft's WMRM so that we could focus on the core parts we wanted to develop in our DRM system. In our experiments, Kakadu [22] was used as the frame encoder and JasPer 23was used as the frame decoder. A set of standard CIF sequences of first 100 frames were used in our experiments. Each frame was of the size 352 by 288 pixels. Each frame of the experimental sequences was encoded with 5 layers, 3 resolutions, 2 tiles, and 2 precincts. Layers were determined in such a way that each layer shows visible improvement in perceptual quality over the next lower layer. The nominal frame rate was set to 30 frames per second.

Table 1 shows the experimental results of the file size overheads and PSNR values for different layers for four MPEG standard CIF sequences. It can be seen that the file size overheads due to the DRM header for key generation is small, around 0.209% to 0.294%. Since the DRM header does not change with increasing number of frames, we would expect that the actual overhead for a typical length of video should be negligible. The sequence "foreman" at different accesses of resolutions and layers are shown in Fig. 6.

Table 1. Experimental results of file size overheads and PSNR values (in dB) for different layers. Each sequence consists of 100 frames.

Sequence (cif)	Bitrate (kbps)	Overhead (%)	PSNR Layer 5	PSNR Layer 4	PSNR Layer 3	PSNR Layer 2	PSNR Layer 1
crew	6436.16	0.260	42.110	38.328	31.219	28.175	23.901
foreman	7000.18	0.239	42.082	38.222	30.937	27.867	21.695
irene	5682.90	0.294	42.197	39.033	31.604	27.846	23.114
soccer	7975.84	0.209	42.124	37.359	30.081	27.145	22.592

Fig. 6. The sequence "foreman" at resolutions 1, 2, and 3 and layers 1 (top left), 2 (top right), 3 (bottom left), and 4 (bottom right)

5 Conclusion

We have described a DRM system that provides what you see is what you pay, where multimedia content is encoded and encrypted to enable multiple access types and

multiple access levels for each type with a single DRM-protected codestream. Different users can share the same protected content or download the codestream truncated to best fit the device. Different keys are acquired for different accesses. We presented in detail the parts of the DRM system that enables a light license server which stores only the publisher's master key. Such a system allows a wide deployment of cheap yet secure servers for license granting services, and therefore improves license service's reliability and availability, and the system's performance since license service is a single point of failure in a DRM system.

References

1. Iannella, R.: Digital Rights Management (DRM) Architectures. D-Lib Magazine, 7(6) (June 2001)
2. Eskicioglu, A.M., Town, J., Delp, E.J.: Security of Digital Entertainment Content from Creation to Consumption. Signal Processing: Image Communication, Special Issue on Image Security. 18(4) (2003) 237 – 262
3. ISO/IEC JTC1/SC29/WG11 13818-11:2003(E). Information Technology – Generic Coding of Moving Pictures and Associated Audio Information – Part 11: IPMP on MPEG-2 Systems (2003)
4. ISO/IEC JTC1/SC29/WG11 14496-13:2004(E). Information Technology – Coding of Audio-Visual Object – Part 13: Intellectual Property Management and Protection (IPMP) Extensions (2004)
5. Open Mobile Alliance. OMA DRM Specification Draft Version 2.0. http://www.openmobilealliance.org (March 2004)
6. Microsoft Windows Media Digital Rights Management. Available at http://www.microsoft.com/windows/windowsmedia/drm/default.aspx
7. Intertrust. Available at http://www.intertrust.com/main/overview/drm.html
8. IBM: Electronic Media Management System. Available at http://www-306.ibm.com/software/data/emms/
9. RealNetworks: Helix DRM. Available at http://www.realnetworks.com/products/drm/index.html
10. Adobe EBooks. Available at http://www.adobe.com/epaper/ebooks
11. Li, W.: Overview of Fine Granularity Scalability in MPEG-4 Video Standard. IEEE Trans. on Circuits and Systems for Video Technology. 11(3) (2001) 301 – 317
12. ISO/IEC: Information Technology – JPEG 2000 Image Coding System, Part 1: Core Coding System. ISO/IEC 15444-1:2000 (ISO/IEC JTC/SC 29/WG 1 N1646R) (March 2000)
13. ISO/IEC: Information Technology – JPEG 2000 Image Coding System, Part 3: Motion JPEG 2000. ISO/IEC 15444-3:2002
14. Zhu, B.B., Swanson, M.D. Li, S.: Encryption and Authentication for Scalable Multimedia: Current State of the Art and Challenges. Proc. of SPIE Internet Multimedia Management Systems V, Vol. 5601, Philadelphia PA (Oct. 2004) 157-170
15. Microsoft: Architecture of Windows Media Rights Manager. Available at http://www.microsoft.com/windows/windowsmedia/howto/articles/drmarchitecture.aspx.
16. Wee, S.J. Apostolopoulos, J.G.: Secure Scalable Streaming and Secure Transcoding with JPEG-2000. IEEE Int. Image Processing, 1 (Sept. 2003) I-205-208
17. Wu, H., Ma, D.: Efficient and Secure Encryption Schemes for JPEG2000. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, 2004 (ICASSP '04). 5 (May 2004) V869 — 872

18. Wu, Y., Deng, R. H.: Compliant Encryption of JPEG2000 Codestreams. IEEE. Int. Conf. on Image Processing 2004 (ICIP'04), Singapore (Oct. 2004) 3447-3450
19. Zhu, B.B., Yang, Y., Li, S.: JPEG 2000 Encryption Enabling Fine Granularity Scalability without Decryption. IEEE Int. Symp. Circuits and Systems 2005. (May 2005) 6304 – 6307
20. Zhu, B.B., Feng, M., Li, S.: A Framework of Scalable Layered Access Control for Multimedia. IEEE Int. Symp. Circuits and Systems 2005. (May 2005) 2703-2706
21. Zhong, S.: A Practical Key Management Scheme for Access Control in a User Hierarchy. Computer & Security. 21(8) (2002) 750-759
22. Kakadu. Available from http://www.kakadusoftware.com/welcome.html
23. JasPer. Available from http://www.ece.uvic.ca/~mdadams/jasper/