A LICENSE TRANSFER SYSTEM FOR SUPPORTING CONTENT PORTABILITY IN DIGITAL RIGHTS MANAGEMENT*

Qiong Liu, Reihaneh Safavi-Naini and Nicholas Paul Sheppard
Center for Information Security, School of Information Technology and Computer Science, University of Wollongong, NSW 2522, Australia

Abstract: Current digital rights management systems typically bind the right to use content to a particular device whose location cannot be changed easily. Users may find it difficult to acquire a new license for each device. We propose a license transfer system that allows the user to share a license between devices and uses transaction track files to ensure that only one device can use the license at a time. We analyze the security properties of the proposed system.

Key words: digital rights management; content portability; license transfer.

1. INTRODUCTION

Digital rights management (DRM) systems are used to prevent unauthorized access to digital content and to manage content usage rights. Rather than trading the content as in traditional physical methods of distribution, the subject of trade is a license that grants certain rights over the content. Typically, the content, which is sold by the content provider, is encrypted with cryptographic algorithms to protect it from illegal copying and consumption. Protected content can be obtained by the user through various delivery channels. However, without possession of a valid license, the content cannot be decrypted.[1] To access protected content, the user's

* This work was funded by the Smart Internet Technology Cooperative Research Centre, Australia.

device needs to contact the license server, which is used by the license issuer to issue licenses, to acquire a valid license.

We define content portability as the ability to access the same digital content on multiple suitable devices. Traditional content distribution models bind the right to use the content to the physical object that carries the content, such as a CD, which can be easily moved to wherever the owner wishes to go. One of the problems in current DRM implementations is that most solutions offer licenses that are bound to a playback device, which is often a desktop computer whose physical location cannot be changed easily. Usually digital content is locked to the device that it was downloaded to. Having purchased the right to use the content for one device, the user still needs to acquire a new license to access the same content on a different device,[2-3] which discourages the use of DRM. It would be welcomed by the users if the usage rights can be shared among multiple devices, so that the users do not need to re-acquire the rights when they have a new device, or when they want to use the content at different physical places.

In this paper we propose a license transfer system that allows the user to share a license among multiple devices while ensuring that only one device can use the license at a time. We present surveys of existing DRM models that support content portability in Section 2. Section 3 describes our proposed license transfer system. We analyze the security properties of the proposed system in Section 4. Section 5 describes our demonstration system that supports content portability using the proposed license transfer protocol. Finally, we give conclusions in Section 6.

2. RELATED WORKS

Several existing DRM systems or proposed schemes provide a limited degree of content portability. We identified three DRM models as follows.

2.1 Rights Locker Model

A rights locker[4] is a storage system that contains the digital rights purchased by a user. It is implemented as a central server to facilitate consumers' access to their rights anywhere anytime.

In a DRM rights locker model, permissions to use content are no longer bound to a particular device, but to the consumer himself. Each consumer has an account with the locker service that allows him to redeem his content usage rights from multiple locations using any DRM enabled device. Every time the user wants to access the content on his device, he needs to log on to his locker account via a web browser by typing in his username and

password. After the central server has verified the authenticity of the request, it grants the access rights for the content to the user. Digital World Services[5] and MP3.com[6] use such a model.

The rights locker model requires Internet connections between users' devices and the central server. Users, especially those who do not have permanent Internet access, may find it difficult to access their rights using this model. Another drawback of this approach is that the central server can easily become a bottleneck. There may be a single point of failure: if the server were crashed or compromised by an attacker, e.g. under denial of service attacks, all the consumers would not be able to access their rights. Moreover, cracking a DRM rights locker would violate user's privacy[4].

2.2 Rights-Sharing Model

In the rights-sharing model, content can be shared among a collection of devices, which is called an authorized domain. Usually, a domain represents a set of devices belonging to a consumer. The content can only be accessed in the domain for which it has been authorized. Several schemes have been proposed to address the need for content protection in the authorized domain, such as the Family Domain concept[7], the xCP Cluster Protocol[8] and OMA DRM domain sharing[9]. The disadvantage of such model is that it introduces the overhead of setting up the domain prior to using it. Only when the domain is formed and the devices are enrolled in the domain, can content be sharing among devices in the domain. If user wants to access the content using devices that are not part of the domain, he is required to register each of these devices as a member of the domain.

2.3 Rights Transfer Model

The rights transfer model allows the consumer to transfer the rights to use the content to machines on the condition that the original copy of the content cannot be used. A few content protection schemes use this model.

FlexiToken[10] proposed by NTT Laboratories is a generic copy prevention scheme for trading digital rights. In this scheme, a digital right is represented using two types of information: the rights description object and the token object. The token object represents the "genuineness" of the rights object and is stored and circulated using tamper-proof devices such as smart cards. The rights object can be held in any storage medium, but to redeem the rights, the user must present the token of the rights to the service provider. The rights transfer protocol proposed in this scheme takes place between two smart cards of the user, using public key cryptography. The token object must be deleted from the original card after the rights transfer procedure.

FlexiToken assumes that neither participant flees from the other, i.e. the sender will delete the token object after it receives the receipt from the receiver. However, this assumption may be violated if the operation of the rights transfer protocol is interrupted either intentionally or accidentally. For example, a dishonest user may cut power from the sender before it deletes the token object.

Aura and Gollmann[11] proposed a simple license transfer procedure for transferring software licenses between smart cards. Since this protocol is based on the assumption that there is no communication interruption between two the cards, it shares the same problem as FlexiToken.

The license transfer model has several advantages over the rights locker model and the license-sharing model: First, in the license transfer model, the digital rights portability among various devices does not depend on the availability of the central server. And it does not require Internet connection if licenses are transferred using local area network (LAN) or between PC and PDA. Second, the license transfer model is a low cost solution because it does not require domain set-up and management process, as the license-sharing model does. We consider using the license transfer model to support content portability in DRM.

3. ROPOSED SOLUTION

We assume that encrypted content can be obtained by the user in any way the user wishes. Access to the content will be provided if the device has a valid license and can extract the content key from the license. Suppose that the user has purchased a license for a content file using one of his devices. Encrypted content is allowed to be copied to any device. To access the content on a particular device, the user needs to transfer the license from the original device to this machine. We require that at any one time, a license should only be used by one device to consume the digital content. Rogue users may try to make copies of one purchased license, then exchange or resell the license copies, thus allowing many devices to consume the digital content at the same time with the cost of a single license. In our system, the management of a single copy is done by the player software by using a transaction flag to record the state of the license: there are four different transaction flags, only when the license has an 'Active' flag, can the content be played. When the transaction flag changes to one other than 'Active', playing stops. Each device keeps a transaction track file that records the current state of each license stored on the device. Only the player can read and update transaction flags. In our proposed license transfer protocol, suppose that license L is for user U to use content C on device D_1, L's usage

rules allow a new license to be created for the same user U to use the same content C on another device D_2, subject to the condition that after transfer of the new license from D_1 to D_2, L becomes invalid.

3.1 System Overview

Figure 1 shows the components of our system: The license database is a conceptual database, such as a file directory, on the user's device, which stores all the licenses that the user has purchased. The transaction track file is a digital data file that records the current state of these licenses. The digital library is a digital content repository on the user's device that stores protected content files obtained by the user. To decrypt and use the protected content, there must be a valid license in the license database and the transaction flag for that license must be 'Active' in the transaction track file. The player is a content viewer responsible for content decryption and playback, and for providing an interface with which the user can request/transfer a license from/to another device through a network.

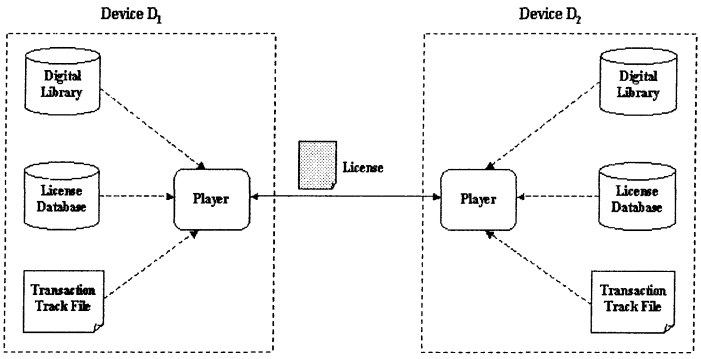

Figure 1. Components of license transfer system

Following is a use scenario of our system: The user has acquired a license from the license server and stored the license on his home PC. If he wants to consume the content on multiple devices, he must transfer the license to the appropriate device. Transfer can be through mobile phones or other handheld devices with wireless connections. The user carries the mobile phone wherever he goes. Using the mobile phone to transfer licenses to a DRM-enabled device is a convenient solution to the above scenario.

3.2 Assumptions

In DRM, end users cannot be assumed to be trusted. Player software executes in a hostile environment, taking control of how the content is used, by whom and under which conditions. Rogue users may try to modify the player software to circumvent the enforcement of usage rules.

We assume the player is trusted. The trusted player is responsible for rendering digital content while enforcing the permissions and constraints associated with the content. By "trusted", we mean trusted by the content provider. The integrity of the trusted player can be protected by using a trusted computing platform[12], for example, Microsoft's Next-Generation Secure Computing Base[13] (NGSCB). If a trusted computing platform is not available, there are techniques such as encryption or code obfuscation[14] exist to make it hard for rogue users to tamper with the player.

We assume each device has a certified public/private key pair. The corresponding private key and the transaction track file stored on the device are only accessible to the trusted player.

We assume that license transfers take place between two trusted players and that there exists a mechanism for players to authenticate each other before engaging in communications and transactions.

3.3 Requirements

Our license transfer system has the following requirements: R1: The content key in the license must be hidden from the user. R2: Players must be able to verify the authenticity and the integrity of the license, and extract the content key from the license. R3: The license transfer protocol must ensure that only the authorized user can access the license. R4: At one time, a user should only be able to use a license on one device. Using copies of the license on other devices does not give access to the protected content. R5: The license transfer protocol must satisfy the atomicity property, which is to ensure that exactly one device (D_1 or D_2) has a valid copy of the license at the end of the transfer procedure regardless of any communication failure between two devices.

3.4 License Transfer Protocol

In our system, the license transfer protocol is run between two trusted player programs. Consider the scenario there are two devices D_1 and D_2 with trusted players P_1 and P_2 respectively. Suppose the user uses device D_2 to request the license L stored on device D_1. The user should first authenticate himself to P_2. P_1 and P_2 will perform a mutual authentication protocol to

authenticate their mutual identities. P_1 only allows transfer of the license if the identity of the user matches the identity of the licensee stored in the license on D_1.

We use a flag mechanism to define license status. Each license is associated with a transaction flag that describes the status of the license. There are four different transaction flags: Active, Deactivated, Request and Recover. The meaning of these flags is as follows:

- Active: the player can use the license to decrypt the content.
- Deactivated: the license is deactivated, so the player cannot use it to decrypt the content.
- Request: the license is not physically stored on the device. The player can request it to be transferred from another device.
- Recover: the license is physically stored on the device. The player can request its status to be changed from 'Deactivated' to 'Active'.

We assume P_1 and P_2 share an authenticated session key K, which can be obtained by executing an authenticated key establishment protocol.[15] In the following license transfer protocol, Req(P_1, P_2, L) denotes the license request that P_2 sends to P_1 for license L. ID_L is L's identifier. Flg denotes the message type, which is to show that the message is for license request or license recovery. PK_1/SK_1 and PK_2/SK_2 are public/private key pairs of device D_1 and D_2 respectively. Let $ENC_K(X)$ denote encryption of a message X using a symmetric key K. T is the timeout value of the protocol. The license transfer protocol is described as follows:

Step1: P_2 sends to P_1: ENC_K(Req(P_1, P_2, L)), P_2 writes (ID_L, 'Flag=Request')

where Req(P_1, P_2, L) = P_1, P_2, ID_L, Flg.

Step2: If Flg = 'Request' and the transaction flag for L on D_1 is 'Active' or 'Deactivated by P_2', P_1 sends to P_2: $ENC_K(L)$, P_1 writes (ID_L, 'Flag=Deactivated by P_2'); else P_1 quits.

Step3: If L is valid, P_2 stores L and writes (ID_L, 'Flag=Active'); If L is invalid or P_2 does not receive L within time T, P_2 quits.

In step 1, P_2 sends a license request Req(P_1, P_2, L) to P_1, which is encrypted using the shared session key K. This encryption provides privacy for the user against eavesdropping and also possible theft and misuse of the license. The message type Flg in Req(P_1, P_2, L) is 'Request'. At the same time, P_2 writes (ID_L, 'Flag=Request') as the entry for L in the transaction track file on D_2, which indicates that L is being requested by P_2.

In step 2, P_1 uses the session key K to decrypt the license request Req(P_1, P_2, L) and checks the message type Flg in Req(P_1, P_2, L). If Flg is 'Request', P_1 checks transaction flag for L on D_1. If L has the 'Active' flag, P_1 sends license L to P_2 and updates L's transaction flag on D_1 from 'Active' to 'Deactivated by P_2'. If P_1 finds that L was deactivated by P_2, which indicates

that P_2 failed to receive the license in the previous license transfer procedure, P_1 will send L to P_2 again. Once L is deactivated, it cannot be used by P_1 anymore, although L is still physically kept on device D_1, i.e. P_1 will refuse to use L to decrypt the content if P_1 finds that L is marked as deactivated in the transaction track file. If P_1 finds that L does not have the 'Active' flag or was deactivated by other players, P_1 quits.

In step 3, P_2 receives L from P_1 and verifies L's integrity and authenticity using the public key of the license issuer. If this succeeds, P_2 stores L and sets the transaction flag for L as 'Active', i.e. L's entry in the D_2's transaction track file is changed from (ID_L, 'Flag=Request') to (ID_L, 'Flag=Active'). If the license verification fails or P_2 does not receive L from P_1 within time T after sending the license request, P_2 quits. To get the license L, P_2 needs to request the license again, starting from step 1.

The license recovery protocol is similar to the above approach. A license recovery scenario is that both D_1 and D_2 have a copy of the license L. The transaction flag for L is 'Active' on device D_2, but is 'Deactivated by P_2' on device D_1. P_1 requests L's transaction flag on D_1 to be set to 'Active'. In this procedure, P_1 sends the license recovery request to P_2 in which the message type Flg is 'Recover'. At the same time, P_1 writes (ID_L, 'Flag=Recover') as the entry for L in the transaction track file on D_1. The transaction flag 'Recover' indicates that L is physically stored on D_1 but cannot be used and P_1 requests reactivation of L. After P_2 receives and verifies the license recovery request, it sets the transaction flag for L on D_2 from 'Active' to 'Deactivated by P_1', so P_2 will not be able to use the license L. On receipt of the respond message from P_2, P_1 updates L's transaction flag on D_1, changing it to 'Active', so L can only be used by P_1 to decrypt the content. If P_1 does not receive the response from P_2 within time T, P_1 quits. To activate L on D_1, P_1 needs to send the license recovery request to P_2 again.

Using transaction flags rather than deletion of licenses guarantees atomicity property, i.e. at any one time, exactly one device can use the license to get access to the content. In comparison with the FlexiToken scheme, our license transfer protocol is robust against communication interruption between two devices.

3.5 Content Key Management in License Transfers

In current DRM implementations, a license includes the content identifier, the identity of the licensee, usage rules, and the encrypted content key. The content key is usually encrypted with the public key of the user's device. Only the device that possesses the correct private key is able to decrypt the encrypted content key and gain access to the content. The license is usually digitally signed by the license issuer to enable its integrity and authenticity

to be verified. A problem with this in the license transfer system is that the content key, which is encrypted with the original device's public key, in the transferred license cannot be accessed by the receiving device. We need to design a mechanism that protects the license in such a way that enables the content key to be successfully extracted by the target device in license transfer while ensuring that the license integrity can be verified. We consider using broadcast encryption and point-to-point encryption for content key encryption. Let K_C denote the content key and $PENC_{PK}(X)$ denote encryption of a message X using a public key PK. Let PK_1/SK_1, PK_2/SK_2, . . . PK_n/SK_n be public/private key pairs of devices D_1, D_2, . . .D_n respectively.

3.5.1 Broadcast Encryption

In broadcast encryption, the user needs to register all the devices he intends to use with the content provider. During license transfer, the sender does not need to modify the original license. Only legitimate devices can access the content key after receiving the license. We consider two types of broadcast encryption:

Public key broadcast encryption[16]: The content provider generates a public key (PK_G) for the group of devices that the user will be using. Each legitimate device has a different private key (SK_1, SK_2, … SK_n) stored in a secure storage on the device. The license contains the encrypted content key $PENC_{PKG}(K_C)$. Player P_i (i = 1, 2, … n) on device D_i uses D_i's private key SK_i to decrypt the content key K_C and then uses K_C to decrypt the content.

Using a public key infrastructure[17]: The content key in the license is encrypted with each device's public key: $PENC_{PK1}(K_C)PENC_{PK2}(K_C)$ … $PENC_{PKn}(K_C)$. When player P_i (i = 1, 2, … n) on device D_i receives the license, it uses D_i's private key SK_i to decrypt $PENC_{PKi}(K_C)$ and then uses K_C to decrypt the content.

The disadvantage of using broadcast encryption is that new devices must be registered to the content provider. When the user replaces old devices with new ones, he wants to continue to use the content he has purchased. The new devices must receive a private key. If a device is compromised, the content provider must change the public key and update the private keys of all devices. Thus, the content provider will have to save and periodically update a record of the user and the device set. Moreover, if the user wants to subscribe content from different content providers, the user has to register his devices with each content provider, which is inconvenient for the user.

3.5.2 Point-to-Point Encryption

Suppose that the license issuer's public key and private key pair is PK_{LI}/SK_{LI}. When the license is distributed to device D_1, the license issuer encrypts the content key K_C with D_1's public key, i.e. $PENC_{PK1}(K_C)$. In the license transfer procedure from D_1 to D_2, player P_1 on device D_1 needs to use D_1's private key SK_1 to decrypt the encrypted content key first and then re-encrypt the content key using D_2's public key, i.e. $PENC_{PK2}(K_C)$, so that player P_2 on device D_2 can decrypt the content key using D_2's private key SK_2.

A problem with this scenario is that the license integrity cannot be verified, because the encrypted content key in the license has changed from $PENC_{PK1}(K_C)$ to $PENC_{PK2}(K_C)$. When player P_2 on device D_2 checks the integrity of the license using the license issuer's public key, the verification will fail.

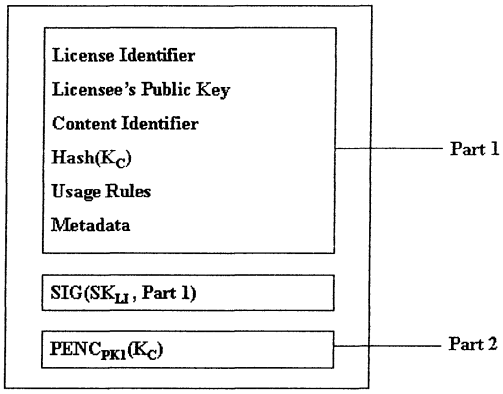

Figure 2. License format on device D_1

To solve the above problem, we propose a license format as shown in Figure 2. A license is split into two parts: the first part of the license includes the license identifier, licensee's identity, the content identifier, the hash value of the content key, usage rules and other related licensing information. The license identifier is used to link the license with its entry in the transaction track file. The licensee's identity shows the identity of the user to whom the rights are granted, using the user's public key. The content identifier, which may be in the form of a Uniform Resource Identifier[18] (URI) or a Uniform Resource Locator[19] (URL), is used to uniquely identify the protected content file associated with this license. The first part is digitally signed by the license issuer to enable its integrity and authenticity

to be verified. We use SIG(SK_{LI}, Part 1) to denote the license issuers signature on the first part of the license. The second part includes the content key encrypted with the public key of the device. The reason for constructing the license in this way is to prevent usage rules from unauthorized modification and to ensure that the license issuer's signature can be verified when the content key is encrypted with another device's public key during license transfer.

One may ask: what happens in the case of a dispute when the user claims that the license issuer put the wrong content key in the license? To avoid such dispute, the hash function has to be one-way and collision-free, so it would be infeasible for the license issuer to generate two content keys with the same hash value. When the player receives the license, it checks the signature of the first part of the license. If the verification succeeds, the player checks the transaction flag for the license in the transaction track file according to the license identifier specified in the license. If the transaction flag for the license is 'Active', the player verifies the user's identity, checks the content identifier, decrypts the encrypted content key using the device's private key and passes the resulting content key to the hash function. If the computed result is the same as the hash value contained in the license, the player will accept the license. Otherwise, the license will be rejected and the player will contact the license server for license reissue. If the license is accepted but the key cannot be used to decrypt the content, the license issuer needs to reissue a license that contains the correct content key.

3.6 Transaction Track File

Each device has a transaction track file that records the current state of each license stored on the device. There are two fields in a track entry: the license identifier and the transaction flag. If the license identifier in the track entry matches that in a license, the track entry is for the corresponding license. Each time the user wants to use the license to access the content, the player checks the transaction flag of the license. Access to the content can be granted only when the transaction flag for the license is 'Active'.

There is only one entry in the track file for a specific license. When a license is delivered to the user's device for the first time, the player adds an entry for the license to the track file after the license integrity is verified. The transaction flag for the license is set to 'Active'. When a license transfer happens, the player reads the license identifier specified in the transferred license and checks the track entry for the license according to the identifier. The player will update the transaction flag for the license after the license has been transferred to another device.

To prevent track entries from unauthorized manipulation, the transaction track file is only accessible to the trusted player. Rogue users may take a snapshot of the hard disk drive, perform one or more license transfers to another device and then restore the transaction track file to the state before license transfers happen. To prevent this attack, software tamper resistant techniques need to be deployed. We do not discuss this topic here, because it is out of the scope of this paper.

4. SECURITY ANALYSIS

This section analyzes the security properties of our system according to the requirements listed in 3.3.

Requirement R1 is satisfied, because the content key in the license is encrypted using the device's public key (or the group key of authorized devices). The corresponding private key of the device is only accessible to the trusted player, so the user cannot decrypt the content key.

Requirement R2 is satisfied. The license is digitally signed by the license issuer, so the integrity and authenticity of the license can be verified by the trusted player using the license issuer's public key. As discussed in 3.5, the proposed license format ensures that the license issuer's digital signature will work properly when a license transfer happens using point-to-point encryption. The content key in the license is encrypted using the device's public key (or the group public key of authorized devices), which can be decrypted by the trusted player using the device's private key.

Requirement R3 is satisfied. The license is transferred through a secure communication channel between two devices. The shared session key of the two trusted players is unknown to any third party, so rogue users cannot intercept and get access to the license during license transfers. Moreover, license transfers can only be allowed if the license requestor is the licensee.

Requirement R4 is satisfied. Unauthorized devices will not be able to use copied licenses to consume the protected content, because the content key in the license is encrypted using the authorized device's public key (or the group public key of authorized devices). Only the authorized device has the knowledge of its own private key and hence can decrypt the encrypted content key, which then can be used to decrypt the content.

Requirement R5 is satisfied. After a license transfer procedure takes place, exactly one device has the license with 'Active' flag. This property is analyzed on a case-by-case basis, as follows:

Case 1: There is no communication problem between P_1 and P_2. Exchanged messages are not disrupted by an attacker. The protocol runs

successfully. At the end of the license transfer, only P_2 gets the license with 'Active' flag.

Case 2: P_1 fails to receive a valid license request from P_2 in step 2. L is still kept on device D_1. P_2 does not get the license. The transaction flag for L on D_1 is kept unchanged, which is still 'Active'.

Case 3: P_2 fails to receive the license from P_1 in step 3. The protocol aborts after timeout. The transaction flag for L in the transaction track file on D_1 is marked as 'Deactivated by P_2'. However, P_2 can get the license from P_1 through a negotiation procedure, i.e. P_2 re-starts the license transfer protocol by sending the license request Req(P_1, P_2, L) to P_1 again. The message flag in the license request shows that the current transaction flag for L on D_2's transaction track file (i.e. 'Request'). From the message flag and L's transaction flag on D_1, P_1 gets to know that P_2 did not receive the license previously. Since L is still physically stored on D_1, P_1 sends L to P_2 again. Finally, P_2 gets the license L and changes the transaction flag for L on D_2 from to 'Request' to 'Active'.

Case 4: The trusted player P_3 on the third device D_3 initiates the license transfer protocol, requesting L stored on D_1. Upon on the receipt of the license request, P_1 checks L's transaction flag on D_1 that indicates that L was deactivated by P_2 due to the previous license transfer procedure that transfers L from D_1 to D_2. The deactivating device is D_2 rather than D_3, which tells P_1 that this is not a negotiation procedure due to the previous communication failure between P_1 and P_3. Therefore, P_1 refuses the license request from P_3 by terminating the license transfer protocol.

In conclusion, our license transfer system satisfies all the requirements listed in 3.3. Our license transfer protocol is robust against intentional or unintentional communication failures of the protocol. The license is not bound to a particular device so it can be easily transferred within a household if required. Finally, our scheme is an offline solution, so the user does not need to be connected on the Internet when he wants to share a license between devices.

5. IMPLEMENTATION

We have developed a functional DRM demonstration system that supports content portability among different devices using the license transfer mechanism defined in this paper. The player is implemented in Java, which allows the user to access protected content using the licenses stored on the device and supports transfers of licenses from one device to another using the license transfer protocol. In our system, licenses are generated based on the MPEG-21 REL data model. We use the Java cryptography

extension library provided by Cryptix Foundation Limited to implement cryptographic operations, such as encryption, hashing and digital signature.

Our implementation of license transfer includes a client program and a server program. The client requests licenses to be transferred from the server. We use the following software modules, as shown in Figure 3.

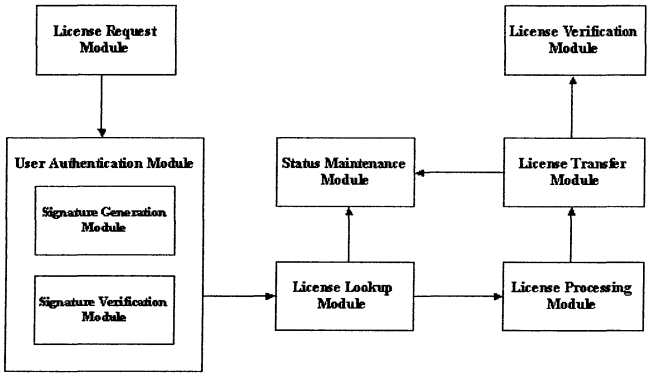

Figure 3. Software modules of license transfer

The license request module connects to the server and creates a secure communication channel between the client and the server using secure socket layer. The user authentication module uses a "challenge-response" mechanism to authenticate the user's identity. It includes signature generation and verification modules. The signature generation module takes the challenge and the user's private key as input and generates a digital signature using the user's private key. The signature verification module verifies the user's signature on the challenge using the user's public key. We implemented these modules using RSA digital signature scheme. The license lookup module is called by the server. It takes each license stored on the server machine and the user's public key as input and checks if the identity of the user matches the identity of the licensee (i.e. the public key of the key holder) in the license. It also consults the status maintenance module to check the status of the license. The license processing module is called by the server. It processes the requested license to make the content key inside the license accessible to the client machine. It uses the private key of the server machine to decrypt the content key in the requested license, and re-encrypts the content key using the public key of the client machine. The license transfer module accepts the output of the license processing module and transfers the license to the client through the secure communication channel created by the license request module. The license verification

module is called by the client. It verifies the integrity and authenticity of the license by checking the digital signature on the license using the license issuer's public key. The status maintenance module maintains status of all the licenses stored on the device. It updates transaction flags for the transferred licenses in the transaction track file.

6. CONCLUDING REMARKS

This paper describes the requirements for a digital rights management system that supports content portability. We propose a license transfer system that allows the user to share a license between multiple devices, which satisfies these requirements. We describe the license transfer protocol and analyze content key management problem in license transfers. We propose a new license format by which the integrity of the license can be verified. We use a transaction track file to ensure that at one time, exactly one device can use the license to decrypt content. We evaluate the security properties of our solution. The development of our DRM test bed shows a working implementation of possible DRM services that could be offered to the customers. Through implementation, we get better understanding about how DRM components work together to provide a secure DRM solution. We believe that DRM systems can be widely used only if customers find it easy to use.

Currently, our system only supports transfers of licenses that grant stateless rights to the user. Stateless rights are usage rights for which the device does not have to maintain state information. Permissions that require maintenance of state by the device, for example a limited number of plays or a maximum period of metered usage time, are considered stateful rights. To make our license transfer system correctly enforce stateful rights expressed in the license, we need to design a mechanism that keeps track of the uses of the associated content.

In the future, we will work on the loan application to make our system support the loan right. The loan right represents the right to lend the content to another user for a specific period of time. While the content is on loan, the original copy of the content cannot be used. At the end of the loan period, the loaner copy deactivates and the original copy reactivates. Currently, our system supports license transfers for a single user between different devices. To support the loan application, the license transfer protocol needs to handle license transfers between different users and take the loan period into account, which provides a challenge for us to conduct future work on this field.

REFERENCES

1. Q. Liu, R. Safavi-Naini, and N. P. Sheppard, "Digital rights management for content distribution," In Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003, pp. 49-58, Australian.
2. D. K. Mulligan, J. Han, and A. J. Burstein, "How DRM-based content delivery systems disrupt expectations of 'personal use'," In Proceedings of the 2003 ACM workshop on Digital rights management, pp. 77-89, ACM Press, 2003.
3. W. Eric, "Protecting digital music using DRM: example of a mobile service over WLAN (version 1.0)," 2002.
4. J. Feigenbaum, M. J. Freedman, T. Sander, and A. Shostack, "Privacy engineering for digital rights management systems," In Digital Rights Management Workshop, pp. 76-105, 2001.
5. Digital World Services, "Digital world services launches rights locker and content manager," http://www.dwsco.com/, 2001.
6. Cyber Patrol, "Free MP3.com technology takes cue from open-source movement,"http://news.com.com/2100-1023-251014.html?legacy=cnet, 2001.
7. T. S. Messerges and E. A. Dabbish, "Digital rights management in a 3G mobile phone and beyond," In Proceedings of the 2003 ACM workshop on Digital rights management, pp. 27-38, ACM Press, 2003.
8. F. Pestoni, J. B. Lotspiech, and S. Nusser, "xCP: Peer-to-Peer Content Protection," IEEE Signal Processing Magazine, 2004.
9. Open Mobile Alliance, "OMA DRM specification v2.0. draft version," http://member.openmobilealliance.org/ftp/Publicdocuments/BAC/DLDRM/, Jan. 2004.
10. M. Terada, H. Kuno, M. Hanadate, and K. Fujimura, "Copy prevention scheme for rights trading infrastructure".
11. T. Aura and D. Gollmann, "Software license management with smart cards," USENIX Workshop on Smartcard Technology, May 1999.
12. P. England, B. Lampson, J. Manferdelli, M. Peinado, and B. Willman, "A Trusted Open Platform," IEEE Computer Security, 2003.
13. Microsoft Corporation, "Next-generation secure computing base: the road to security," http://www.microsoft.com/ngscb, 2003.
14. M. Sosonkin, G. Naumovich and Nasir Memon, "Obfuscation of design intent in object-oriented applications," In Proceedings of the 2003 ACM workshop on Digital rights management, pp. 142 – 153, ACM Press, 2003.
15. W. Stallings, "Cryptography and network security: principles and practice," Prentice-Hall, Inc., Upper Saddle River, NJ 07458, USA, third edition, 2002.
16. D. Boneh and M. K. Franklin, "An efficient public key traitor tracing scheme," In Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology, pp. 338-353, Springer-Verlag, 1999.
17. R. Safavi-Naini and H. Wang, "Broadcast authentication for group communication," Theoretical Computer Science, 269(1-2): 1-21, October 2001.
18. The Internet Society, "Uniform Resource Identifiers (URI): Generic Syntax. http://www.ietf.org/rfc/rfc2396.txt, 1998.
19. The Internet Society, "Registration Procedures for URL Scheme Names", http://www.ietf.org/rfc/rfc2717.txt, 1999.