Specifications for a Componetised Digital Rights Management (DRM) Framework

Alapan Arnab

Supervised by:
Prof. Andrew CM Hutchison

Draft
September 4, 2005

Version: 0.1.1

Data Network Architecture Laboratory
Department of Computer Science
University of Cape Town
Private Bag, RONDEBOSCH
7701 South Africa
e-mail: aarnab@cs.uct.ac.za

Abstract

This document lays out the specifications for a componentised DRM system. Requirements for a general DRM system are discussed, and we detail a set of components that address these requirements. This document also details the specific services that should be offered by each component and specifies the communication protocols and contents of these messages.

Each of the components of the DRM system are fully fledged web services, and thus some of these components can be used in areas other than DRM. Furthermore, we envisage existing services, such as Certificate Authorities, easily fitting into our proposed framework.

Version History

0.1.1: September 2005 - Added versioning

0.1: Released May 2005.

Contents

List of Figures

List of Tables

1 Introduction

Digital Rights Management (DRM) is a relatively new field in Computer Science. The aim of DRM is to create a mechanism of *persistent access control* [24]. This means that regardless of where data protected with DRM resides, users of the data will have to abide by the access control mechanisms placed on the data or not be able to access the data all. This is different from encrypted data in that there is no control on what a legitimate user does with the data once he/she has unencrypted the data. While the current focus of DRM is to solve the problems associated with piracy of media (esp music), DRM can also be used to protect any type of sensitive data like health records.

In [20], Park et al. described three ways of distinguishing various access control mechanisms for data. In the first level, there is a "*virtual machine*" or a "*DRM controller*" as described by Rosenblatt et al. [23] that enforces the access control rules for the DRM work. In the second level are the various mechanisms to define the access control rules or "*control sets*". Control sets can be fixed to the DRM controller, embedded with the protected data or come as a separate "use license". Finally the authors distinguished between the actual distribution mechanism of the data. In [10], Bartolini et al. discussed the various roles and players in an "*Electronic Copyright Management System*". We use these roles and players as a base for most of the components in our DRM system.

In [24], Rosenblatt et al. also discussed a wider definition of DRM as "*everything that can be done to define, manage and track rights to digital content*". Under this definition, technologies such as watermarking and fingerprinting can also be considered as part of a DRM solution.

In these specifications, we aim to define a set of components and related data formats for the creation, administration and distribution of DRM protected data. The primary aim of the system is to achieve persistent protection, but the system will also allow for components that allow for additional technologies such as watermarking and fingerprinting. We will also define the control set formats and mechanisms as well as the interface between the DRM controller and the rest of the system. The enforcement of DRM will need to differ in different platforms and operating systems and thus we will not define the specifics of how a DRM controller should be implemented. However, we will define the requirements of the DRM controller.

1.1 Motivation

DRM can potentially be used for a variety of uses but the core requirements remain the same. For example, DRM can be used by individuals to protect personal information and data (financial details, love letters etc), by enterprises to protect data (from trade secrets to market plans to office memos about the christmas party) as well as the current use of DRM – a means to control the usage of widely distributed intellectual property.

Regardless of the use, DRM must also allow for the enforcement of the legal rights of all the parties involved. However, the legal rights do depend on the circumstances of use. For example, if the protected data happens to be the design documents of a company's new product, the company is rightfully expected to track very carefully the users who have access to the document as well as when and where they access the informations. However if the data is a pop song being sold to the public, the public has an expectation (and in most countries a right) not to be monitored where, when and how the listen to the song.

For the above reason, there can be an argument that DRM systems for enterprises and consumers need be separate. However separate systems also end up creating duplications – the underlying protection mechanism needs to be the same and the cryptographic fundamentals are the same. Thus, users would end up needing multiple identities, multiple key pairs and multiple mechanisms to enforce DRM.

The aim of the specifications is to provide a system that is general enough but at the same time allows flexibility for any type of use. Thus, one DRM system would allow the users to access protected documents for their work as well as buy and listen to music while they work.

1.2 Requirements

In this section, we list the requirements of a general DRM system. Specialised DRM systems like enterprises DRM systems could have additional requirements to the ones detailed below.Many of these requirements have been presented before by Bartolini et al. [10], Park et al. [20] and Mulligan et al. [19]. We have also presented some of these requirements in a previous paper [7].

1. **Persistent protection:** A DRM system must guarantee persistent protection of the secured objects. This means that regardless of the location of the digital data [1], the access controls that are imposed by the rights holder must either be enforced or the device should not be able to read the file at all. If persistent failure is not achieved, the system can be considered a failure.

2. **Portability:** Portability can have a number of different meanings, and not all aspects of portability are equally important. We have divided portability into four types, and discuss them in increasing importance:
   (a) **Time Shifting:** Time shifting refers to the ability of the user to access the work when he or she wants to. While the freedom is critical in the consumer space, this is not the same in an enterprise. In fact, it could be the case that an access to protected data in “odd” hours is indicative of misuse. Time shifting is conceptually easy to declare and implement, but time is measured by different means (is device’s clock using GMT or local time) and proper synchronisation is required.
   (b) **Space Shifting:** Space shifting refers to the ability of the user to freely access the work in whichever device he or she wants. In most cases, enterprises would like to restrict the number of devices that can access protected data while consumers would like to use any of the devices they own.
   (c) **Format Shifting:** Referring to online music and video stores, Mulligan et al. argued that format shifting is also an important portability issue [19]. Format shifting allows the user to change the format of the data file (without necessarily affecting the access control rules). Format shifting could be important in an enterprise for a variety of reasons – for example, the enterprise could keep internal data stored in a certain format and in a different format when released to other companies or even to the public (in the case of financial statements for example). Similarly in the consumer space, format shifting is also important to consumers – for example, format shifting would allow users to write audio CDs from digital music downloads. Format shifting should also allow for easier integration between different applications across different platforms.
   (d) **Platform Shifting:** Platform shifting refers to the ability of the user to use different operating systems and devices to access the protected data. In an enterprise, this is probably the most important requirement. Even small businesses are likely to make use of a multitude of different devices – PDA’s, desktop computers, laptops

[1]This only applies to the digital format of the data. For example, for an electronic text file, the controls must apply to the text and not to the paper copy if the text can be printed. This is an important distinction, as there will always be analogue bypasses to DRM – for example taking photos with a camera when a screenshot is not allowed.

and servers. Even if an enterprise decides to make use of one vendor for all their devices (e.g. Linux or Microsoft), the devices are likely to run different versions of the operating system and applications and thus portability is extremely important. In the consumer space, platform shifting is also important, and while mobility between a PDA and a computer may not be important, other devices are. Many mobile telephones already allow for music playback and home entertainment systems allow for the convergence of digital media to the living room. Thus, seamless platform mobility may become even more important than in enterprise DRM systems.

3. **Integration with existing applications:** This requirement could also be another portability requirement – the ability of using different applications to access protected data. Currently all DRM systems require specialised applications to handle protected data – but a DRM system that is application independent is ideal. Off course this would depend on whether the application itself supports the data file format (for example, Microsoft Word does not support Open Office file formats).

4. **Excerpting:** Excerption allows a user to take a certain segment of data from one source for inclusion in another data file. Excerpting can range from a simple "copy and paste" to complex uses such as mixing multiple audio and video streams to form a new creative work. Fair use clauses in most copyright laws allow for excerption under certain circumstances. Excerption is also useful in an enterprise to allow different data sources to be merged to be presented in one document for example.

   There are two major problems in dealing with excerption. Firstly, as discussed in [19], applications that are used to render the protected data are usually unsuitable for complicated excerption tasks. At the current moment, only specific applications can access DRM protected data thus not allowing excerpting. Secondly, it is difficult to control how much of a protected document can be be excerpted. For example, fair use allows the use of a video clip for the purpose of review (use of a trailer to review a movie for instance). However, the reviewer cannot take the entire work and use it as part of the review. The first part of the problem is part of the portability issue, and we do address it in the specification. The second part is a more complicated issue and is not addressed.

5. **Transfer of Rights:** The ability to transfer rights of usage is very important in both the consumer space and enterprise DRM systems. Transfer of rights would allow a user to lend a work to friend or family member. However transfer of rights does need to be controlled in two respects:

   (a) In these specifications we position DRM to be used for general protection of data and not only for music and e-books. Transfer of rights for confidential data need to be controlled by an authorised entity.

   (b) Because of the ease of replication of digital data, transfer of rights also means that the work should not be accessible after the right has been transfered. For example, if a user sells his right to listen to a music file, his use license should expire and remain invalid until he purchases the right again.

   The revocation of rights also falls under transfer of rights.

6. **Allow for changes to access and usage rights after distribution:** The initial rights assigned to a user might not be enough (because they are too restrictive for example) or maybe more than necessary (because the employee was re-assigned to a different department in a enterprise DRM deployment for example). This requirement does allow indirectly for the expression of fair uses, although in a more regulated and constricted environment.

7. **Privacy vs. Usage Tracking:** In the consumer space, users expect limited interaction with the rights holders after purchase [19]. User privacy is thus very important and many governments have laws governing the protection of user privacy. User privacy includes the protection of personal data collected from the user as well as not monitoring the usage of DRM enabled works by the right holder. Privacy becomes important in a distributed service environment as different services should not be able to access data that they are not entitled to. However, monitoring of usage does not necessarily extend in a corporate environment. For confidential data, a corporation would probably retain the right to monitor the usage of the data, and should confidential data be leaked, the source of the leak should be easily identifiable.

   Privacy and usage tracking are mutually exclusive – an increase usage tracking leads to a lower user privacy and vice versa.

8. **Offline Usage:** Communication networks are not perfect, and there are many situations where users may not have access to the Internet (for example using a laptop on a aeroplane). Also, some consumer electronic products (such as music players like iPods) do not have Internet access. Thus offline usage is desirable; but does have its drawbacks for rights holders – offline usage reduces monitoring and tracking capabilities. For example, in an enterprise DRM deployment, if an employee is fired and the employee has protected data that can be accessed offline, the employee could still retain access to the protected data.

9. **Easy identification:** In [10], the authors identified the identification of digital works as a crucial component of a DRM system. A DRM system must be able to uniquely identify digital works on the Internet, and have a mechanism to correctly associate the users that have rights to use/access the work as well as mechanisms to associate the right holders of the work.

10. **Easy Verification:** Another criteria given by Bartolini et al. is to allow honest users to easily prove that they have legitimate access to the protected work [10]. This extends in general to all objects and transactions in a DRM system; integrity and verification should be easy to proove.

11. **Correctly collect revenue for right holders:** Should an end user require to pay for the right to use a work, the DRM system must be able to collect the correct amount, and record the transaction such that the correct right holders are compensated appropriately. This requirement is mainly for the consumer space.

In [9] we looked at some of the security considerations for a DRM system. A DRM system must address the 5 services identified in ITU's X.800 specifications. These services are:

1. **Authentication:** The process of verifying an identity claimed by or for a system entity [25].

2. **Access Control:** Protection of system resources against unauthorized access [25]

3. **Data Confidentiality:** Service that protects data against unauthorized disclosure [25].

4. **Data Integrity:** Protects against unauthorized changes to data, including both intentional change or destruction and accidental change or loss, by ensuring that changes to data are detectable [25]

5. **Non-Repudiation:** Provide protection against false denial of involvement in a communication [25].

Availability is not explicitly stated in X.800 but many consider it as another essential security service [27].

6. **Availability:** A system or a system resource being accessible and usable upon demand by an authorized system entity, according to performance specifications for the system [25].

In our view, the ideal DRM system must be able to cater for all the above requirements.

1.3 Layout

Before we detail the specifications however, in section 2 we detail three scenarios where we see the possible use of DRM. Following the scenarios, we detail the overall system architecture as well as an overview of the communication and security architecture in section 3. Following these two chapters, the specific modules of the architecture are detailed.

The layout of the specifications is as follows: In section 4, we detail the PKI infrastructure used in the DRM system. Following that, in section 6 we discuss identity management in the DRM system as well as service details of the UNI.

1.4 Glossary of Terms

Use License: A set of terms and conditions, like a contract, that allow a user to access and use a DRM enabled work.

User: The user is a person or entity making use of the DRM system to create or access DRM protected work. It could be a human, a computer program or legal persons such as companies.

2 Scenarios

In this section, we present three scenarios intended to capture the variety of uses of DRM as a security mechanism. In the first scenario, we demonstrate the use of DRM in a scenario where 2 or more different organisations need to share access to the data. In the second scenario, we examine the case of the traditional DRM use – protecting copyrighted material. In the last scenario, we discuss the use of DRM totally within a single organisation. Following the description of the scenarios, in section 2.4 we present a summary of the important requirements linking back to the requirements set out in section 1.2.

2.1 Scenario A: Exam Papers

Carl is the teaching assistant for the 3rd year undergraduate Computer Science course. One of his duties is to typeset the final exam. This process involves collecting the questions from the various lecturers of the course (including a part time lecturer who works for an IT company in the city). After typesetting, the paper is then sent to all the relevant lecturers for comments. The lecturers email the comments back to Carl, who then makes the relevant changes and emails the final exam to the eternal examiner. All communication between Carl and the lecturers take place via e-mail. After the exam has been set, it is sent to the printers for printing.

Traditional encryption is not a full solution to the problem. Ideally, Carl should be the only person with rights to modify the exam. The printers should only have rights to reading and printing the exam and never write and modify permissions. Traditional access control mechanisms such as active directory are also unsuitable – external examiners and printers will not have access, while a part time lecturer would probably like to use a personal machine not connected to the university network.

The integrity of a degree requires that the exam questions are not leaked. Thus confidentiality is of highest importance. Tracing the source of a leak, if there is one, is also important, and thus tracking usage of the exam is also important. Portability between platforms is also important as the lecturers, Carl, the external marker and the printers could be using different platforms.

2.2 Scenario B: Selling Music

Eric, James and Oswald play in a band, but are not affiliated with any music label. They have produced an album which they would like to sell online through their website. Other local bands that Eric and co have played with have also expressed interest in such a venture and thus Eric would like to set up an online music store for all the bands. Eric eventually hopes that other bands will sign up with the idea, and use the music store to sell their music.

Eric would like the music store to have similar restrictions as the iTunes music store, except allow people more freedom on where they can listen to the music they buy, as well as sell and lend the music to their friends. They would also like to ensure their fans' privacy and not track the usage of their music files. The bands would like certain permissions, like the ability to excerpt, to be granted on a request basis.

2.3 Scenario C: Patient Data in a Hospital Group

Netcare [2] is a large private hospital group operating a number of hospitals, clinics and medical centres across the country. For the convenience of their clients, patient data (personal data, medical history

[2]Netcare is a real private hospital group in South Africa. This example is for illustrative purposes only and the example does not have any correlation with the operations of the Netcare group.

etc) is accessible from any hospital, clinic or medical centre. Thus, should a patient be checked into hospital for any reason, the full patient data set is available to the attending doctors and nurses. Data can also be updated across the system, allowing full tracking of medical history for the patient.

This however has a counter problem – the patient data can be accessed by any nurse, doctor or even administrator who has access to the system. This is a major privacy problem, as a patient would expect their medical data to be available to only the doctors involved with the treatment. Thus the system poses a dual challenge - making data available to doctors when needed but at the same time restricting the use of the data. It is also a case where DRM can be used to protect the privacy, which is contrary to many claims about DRM [4, 6].

This scenario can be solved to a certain extent using technologies such as LDAP and similar directory services. However this would also mean that, ultimately the administrators of the systems would have unlimited access to the private data, which is still a privacy risk. Ideally however, you would like the patient's personal doctor, if not the patient him(her)self controlling access to the data. However, safe guards do need to be taken in certain scenarios where the patient is not capable of granting access and his/her doctor is unavailable (for example a car accident and the personal doctor is away on holiday).

2.4 Requirement Analysis

In the following table, we examine which of the requirements set out in section 2.4 apply to the scenarios above. A **Y** represents a definite requirement while a **N** represents a definite non requirement. A **M** represent a requirement that would be nice to have but is not crucial. In the case of *Privacy v.s. Tracking*, a **P** represents a case where user privacy requirements override tracking requirements, while a **T** represents the case where user privacy is not a concern as much as tracking the data.

	Requirements	**Scenario A**	**Scenario B**	**Scenario C**
01	Persistent Protection	Y	Y	Y
02	Portability: Time Shifting	Y	Y	Y
03	Portability: Space Shifting	Y	Y	M
04	Portability: Format Shifting	Y	Y	M
05	Portability: Platform Shifting	Y	Y	Y
06	Integration with existing applications	Y	Y	Y
07	Excerpting	N	Y	M
08	Transfer of Rights	N	Y	Y
09	Allow for changes to access and usage rights after distribution	Y	Y	Y
10	Privacy vs. Usage Tracking	T	P	T
11	Offline Usage	Y	Y	Y
12	Easy identification	Y	Y	Y
13	Easy verification	Y	Y	Y
14	Collection of revenue	N	Y	N

Table 1: Requirement analysis of the three scenarios

3 System Architecture

In this section we give an overview of the entire system, an overview of the communications architecture and discuss where various security services detailed in section 1.2 are handled.

3.1 Components of the DRM System

In Figure 1, we give the overall architecture of the DRM system. Each of the components are derived from a set of roles described by Bartolini et al. in [10]. In this systems architecture, we do not detail how each of the components communicate, and instead we give our proposed communications architecture in section 3.4.

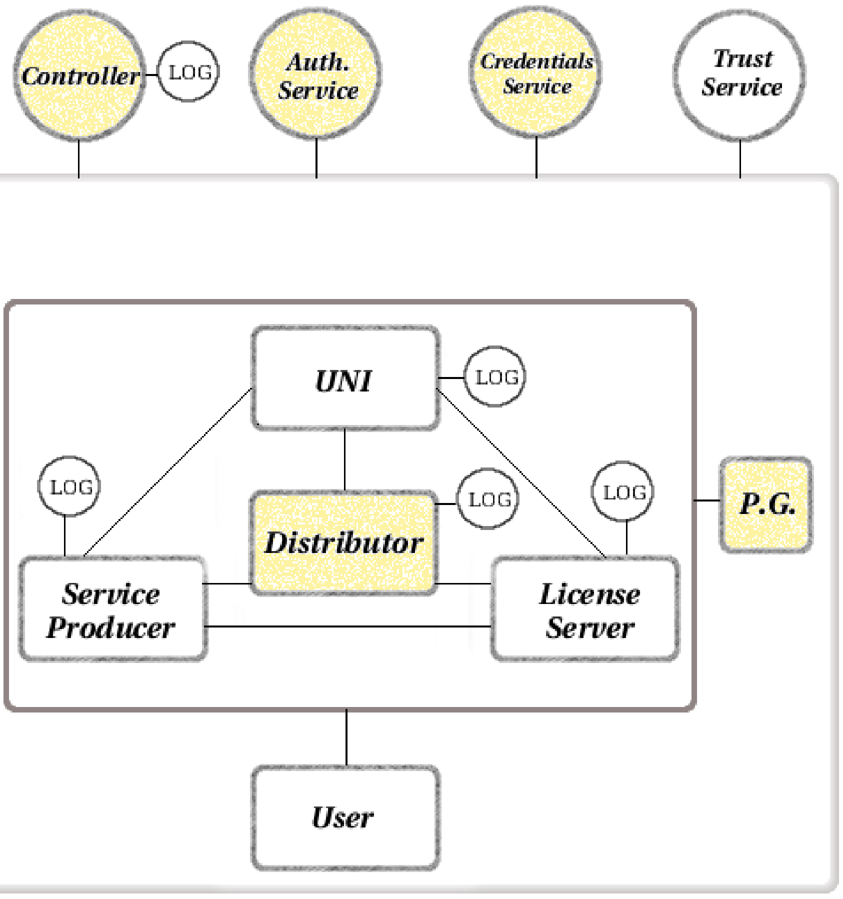

Figure 1: Architecture of the Componentized DRM System (shaded components are optional)

In [7] we noted that the roles described by Bartolini et al. did not have a role defined for the end user. Bartolini et al. did however, cater for the creators and the right holders of the DRM work. In our architecture, we consolidate them into one – as the users of the DRM system. We also add a *Payment Gateway (P.G.)* module, which is not considered as a role in [10]. While authentication and credential services are implied requirements in [10], they did not appear as firm roles in the system. In our system these services are optional, as there are other mechanisms to implement these functions. We would also like to separate our proposed components into required and optional (which are shaded in figure 1); with the Authentication Service, Credentials Service, Payment Gateway (P.G.), the Controller and the Distributor being optional roles.

1. **Trust Service**(*Required*)**:** The trust service is the main trusted third party (TTP) that is trusted by all the involved parties. At the moment, the trust service would be a certificate authority (CA) who will be involved in certifying public keys of the participants. Similar trust services could also be involved in system such as the TPM manufacturers in the trusted computing initiative [12]. The role of PKI is discussed in more detail in section 4.

   Involved in addressing requirements: Offline Usage, Easy Verification

2. **UNI** (*Required*)**:** The Unique Number Issuer (UNI) will be responsible for managing identity for users, services and every DRM enabled work. The identifiers should be globally unique, and as such one UNI should be able to serve a number of service producers. The UNI is based on the handle service and is discussed in more detail in section 6.

   Involved in addressing requirements: Easy Identification

3. **Service Producer** (*Required*)**:** The service producer packages the work in a security envelope, together with the unique identifier from the UNI and creates a template for the rights that can be granted to an end user. The service producer and the author roles can be the same; but there could be situations where a separate service producer is desired (e.g. the rights holder of the work is not the same as the author).

   Involved in addressing requirements: Persistent Protection, Format Shifting, Platform Shifting

4. **License Server** (*Required*)**:** The license server serves two functions in our framework. Firstly, the license server hands out use licenses to the end user. The use licenses specify the rights that the user has on a DRM protected work. Should it be required, the license server can make use of the Payment Gateway for the end user to pay for the rights.

   The second function of the license server is to handle requests from users for additional rights. Some of these requests could be granted automatically (for free or for a price) which can be predetermined by the rights holder. The rights holder could also setup a set of requests to deny automatically. Otherwise the license server should communicate with the rights holder the requests from the end user; and the rights holder can then communicate back with the license server granting or denying the request. The use of a request system should allow for a work around for fair use; which is difficult to express in a REL.

   Involved in addressing requirements: Transfer of Rights, Allow for changes to access and usage rights after distribution

5. **User** (*Required*)**:** The user can be either the creator of a DRM work or be the end user. As an author, the user uses the service producer to create a DRM protected work. The user can then make use of a *distributor* or can distribute the work on his/her own. The author will also require a mechanism to receive requests from end users (forwarded by the license server) and respond to these requests. As an end user, the user retrieves use licenses from the license server and then can use the DRM protected work as defined by the license. The user component is responsible for the enforcement of the terms and conditions laid out in the use license.

   Involved in addressing requirements: Persistent Protection, Time Shifting, Space Shifting, Platform Shifting, Format Shifting, Integration with existing applications, Excerpting, Offline Usage

6. **Authentication Service** (*Optional*)**:** Authentication of users and services can be handled in a variety of ways and is discussed in more detail in section 10.

   Involved in addressing requirements: Privacy vs. Usage Tracking, Easy Verification, Offline Usage

7. **Credentials Service**(*Optional*)**:** Authentication gives proof that a user is who he or she claims they are. Credentials is aimed at access control – is the user allowed to access a service once they are authenticated. However, our use of credentials for users is slightly different, and is geared more towards enabling *fair use*. The credential authority and its functions is discussed in more detail in section 11.

   Involved in addressing requirements: Allow for changes to access and usage rights after distribution, Transfer of Rights

8. **Distributor** (*Optional*)**:** A DRM protected work is required to have persistent protection regardless of where the work resides. For this reason, the distributor is not a requirement for the system, as DRM protection must work if the work is transferred over peer-to-peer networks, made available for download on the Internet etc. However the distributor does provide some interesting possibilities for DRM enabled work distribution.

   The main use of a distributor is in an electronic store; as demonstrated in the Apple iTunes Music Store. In the Apple iTunes Store, the music file is encrypted using the end user's public key and thus only the end user is supposed to be able to decrypt the file. Similarly, the use of a distributor allows for the possibility of personalising the DRM enabled work for every user. In this case, the Distributor would need to make use of the Service Provider for every "sale" of the DRM enabled work.

   Without the use of a distributor, a shared key is required. In this scenario, the use license of the work would specify the shared key to decrypt the DRM package. This solution is most useful in an intra-enterprise deployment.

   Involved in addressing requirements: Persistent Protection, Correctly collect revenue for right holders

9. **Payment Gateway** (*Optional*)**:** Like the Distributor, the Payment Gateway is an optional component. The Payment Gateway is only required where the framework is used for delivering commercial products, such as an online store.

   Involved in addressing requirements: Correctly collect revenue for right holders

10. **Controller** (*Optional*)**:** In [10] Bartolini et al. put the Controller as a very important component of a DRM system. While the controller plays a very important role in commercial DRM systems (co-incidentally none of the current DRM system deployments use a controller), its use in an intra-enterprise scenario is not that important. For this reason, we have decided to cater for the Controller as an optional component in our framework. The use of a controller does increase the overall overhead and this could be a major factor against the use of a controller.

    Involved in addressing requirements: Easy verification

11. **Logging** (*Configurable*)**:** While the use of a controller is optional, the use of logs in a DRM system is of great importance. However the use of logs has its drawback. If used excessively,

logs can be used to monitor the usage of DRM enabled work by end users. For this reason our framework must strike a balance between what actions should be logged (issue of a license, revocation of a license) and what should not be logged.

Involved in addressing requirements: Privacy vs. Usage Tracking

3.2 Requirement Analysis Summary

Table 2 gives a summary of which components are involved in addressing a particular requirement, as discussed in section 3.1.

	Requirements	**Components/Role**
01	Persistent Protection	Service Producer, User, Distributor
02	Portability: Time Shifting	User
03	Portability: Space Shifting	User
04	Portability: Format Shifting	Service Producer, User
05	Portability: Platform Shifting	Service Producer, User
06	Integration with existing applications	User
07	Excerpting	User
08	Transfer of Rights	License Server, Credentials Service
09	Allow for changes to access and usage rights after distribution	License Server, Credentials Service
10	Privacy vs. Usage Tracking	Authentication Service, Logging
11	Offline Usage	Trust Service, User, Authentication Service
12	Easy identification	UNI,
13	Easy verification	Trust Service, Authentication Service, Controller
14	Collection of revenue	Distributor, Payment Gateway

Table 2: Summary of components that address a particular requirement

3.3 Scenario Analysis

	Components	**Scenario 1**	**Scenario 2**	**Scenario 3**
01	Trust Service	Y	Y	Y
02	UNI	Y	Y	Y
03	Service Producer	Y	Y	Y
04	License Server	Y	Y	Y
05	User	Y	Y	Y
06	Authentication Service	M	M	Y
07	Credentials Service	N	M	Y
08	Distributor	N	Y	M
09	Payment Gateway	N	Y	N
10	Controller	M	Y	Y
11	Logging	Medium	Low	High

Table 3: Scenario Analysis of the components

Table 3 examines the components required to address the scenarios discussed in section 2. A **Y** represents a component that is definitely required, a **M** represents a component that could be used to provide additional functionality but could be left out while a **N** represents a component that is not needed to solve the problems posed by the scenario. The values in row 11 addressing logging, looks at the degree of logging that is needed by the scenario.

3.4 Communication Architecture

We suggest that each of the components be developed as standalone Web Services, and use SOAP as the communication protocol. At the moment, SOAP does not offer any security services, but it is hoped that WS-Security proposals will be standardised. As standalone Web Services, each module can also be used for other functionalities that are not necessarily DRM related. For example, if the trust service is a certificate authority, it can also be used to certify any certificate and not just for DRM systems.

Thus, each module is effectively a *service* and the framework can then follow a Service Oriented Architecture (SOA). The *service provider* and *service requestor* in our framework will be the respective modules. A *service description* module is not included in our architecture as it is not specific to the system. However, a service such as *service description* was regarded as a neccessity by Bartolini et al. in [10].

3.4.1 Use of SOAP

SOAP is a communication protocol standard designed to allow two Web Services to communicate regardless of the platform on which the Web Service is running. SOAP does not provide any security advatages, but does allow for the use of an established, multi platform, language independent communication protocol. We thus strongly recomend the use of SOAP for communication between the services.

The payload is carried in a SOAP envelope, and two security services, non-repudiation and data integrity, can be handled at the payload level. Each Web Service in the framework has an associated XML schema to describe the payload for the communication. A root element of each schema[3] is the *SignedCommUnit* element. Figure 2 shows an example of one such element.

This element encapsulates the data required (contents) for communication with the service, and then attaches a XML digital signature. The digital signature must sign the contents, and thus offers non-repudiation (only the owner of the private key can sign the contents) and data integrity (a feature of digital signatures themselves).

3.5 Security Architecture

Our framework must cater for the security services identified in 1.2. Of the 6 services identified, Authentication and Access Control need to be addressed in finer detail at the individual service levels.

3.5.1 XML Security

Web Services make extensive use of XML for communication because XML files are portable across different system types. Use licenses also make use of XML for the same reason. In our system, we also allow for the DRM data package to be stored as a XML file.

The system makes extensive use of two major XML Security recommendations – XML Digital Signa-

[3]Most schemas have more than one root element – one root element handles communication, while other root elements handle any data produced by the service.

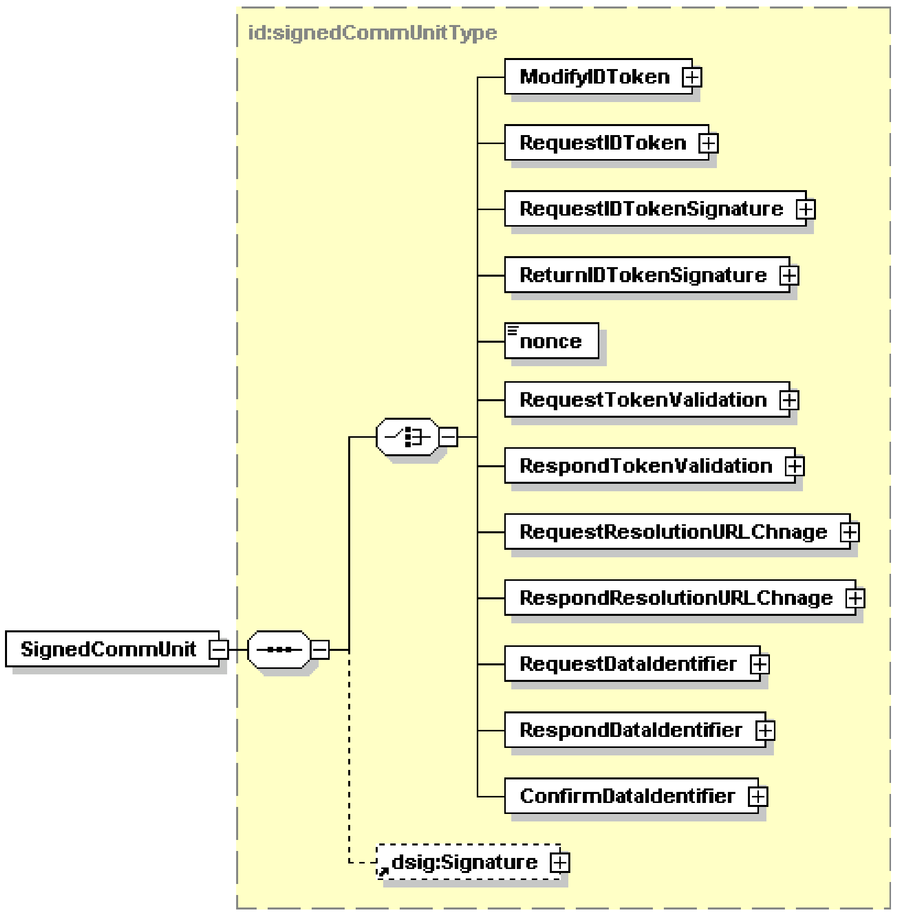

Figure 2: XML schema for SignedCommUnit *element for the UNI service (See chapter 6)*

ture (XML-DSIG) [29] and XML Encryption (XML-ENC) [28]. Both specifications allow part of a (or an entire) XML file to be signed or encrypted respectively.

3.5.2 Authentication and Access Control

Authentication can be handled separately by individual services, and if this is the case, they should make use of the XML-Security extensions to SOAP [?] for communication. However, authentication can also be provided by an external service, like through the use of a federated identity management system. Similarly, access control can be be implemented by individual services or through the use of a credentials service.

3.5.3 Data Confidentiality

As discussed in [9], data confidentiality has to be addressed at two levels:

1. during communication between services
2. during processing or storage by a service

In most respects, secure communication is a "solved" problem on the Internet. Internet commerce can only take place with the presence of secure communication through the use of SSL sessions between the web server and the client.

We propose the same – all communication between various Web Services must take place in SSL sessions. This will ensure data confidentiality during communication.

If data needs to be protected in a DRM enabled form, then the storage should be in the DRM package itself. This applies to services like the distribution service. However there is other data that would need to be stored securely in a database (such as identifers) and maybe user data. Although database files themselves can be secured using DRM, weak access controls on the database are more likely to be the source of security breaches. Thus strong access control to the databases and isolating the database behind a firewall etc. are good steps towards securing the database.

Secure processing is another concern as confidential data can be in an unencrypted form during processing, and lead to a breach in confidentiality. This may not be a problem in the services dealing with the production of DRM protected data, but is of high important in the virtual machines of the end users that render the data in an usable form. The first hack of Apple's iTunes music service managed to grab the unencrypted copy of the music file from memory while the song was being played with iTunes for Windows application. Such a breach is more due to programmer error, but should be kept in mind. Trusted computing also aims to address the problem through the use of trusted devices which can handle encrypted data on the devices themselves.

3.5.4 Data Integrity & Non-Repudiation

As mentioned in section 3.4.1, each message from a service is signed using the service's private key. Digital signatures provide both data integrity and non-repudiation services. Should the communication protocol be changed from SOAP to another protocol (like using a traditional RPC mechanism), the security services offered by the payload is not affected.

3.5.5 Availability

Availability is a difficult requirement to address in the case of a denial of service (dos) attacks. However in a componentized system, with multiple components that serving a single function, the effects of a dos attacks can be reduced. Furthermore, to take down the complete system; every component needs to be unavailable, which is less likely to occur than the case of a single monolith system.

4 Public Key Infrastructure (PKI)

Public Key Cryptography will be used extensively in the DRM system. To allow for different components to talk to each other they must make use of a common PKI system. For that there needs to be a common certificate system, and there are three such systems available currently:

1. X.509
2. SPKI/SDSI
3. PGP

4.1 PKI Systems

4.1.1 X.509

Currently the most popular PKI system used [27], and is well entrenched in many web based services. The main drawback of a X.509 is the number of different extensions available for version 3. However the extensions are not really required in the system; and any version 2 certificate should be able to do the required job.

4.1.2 SPKI/SDSI

The Simple Public Key Infrastructure (SPKI) was set up by The Internet Engineering Task Force (IETF) "*to address the overcomplications in the X.509 world*" [15]. SPKI certificate structure was designed from scratch; and instead of designing on the premise that the identity of a person implies the authorisation for an action as in X.509, SPKI rather uses the idea of the user of a particular certificate has the authorisation for an action. There can then be a separate mapping between a user and the certificate.

This abstraction is very useful for a DRM system. In the consumer space, current DRM systems potentially allow for the monitoring of users through their use of the DRM protected work. By moving the authorisation to a certificate level, this is removed. As long as the publishers cannot make an association between the certificate and the actual user - there is a reduced privacy concern[4].

4.1.3 PGP

Pretty Good Privacy (PGP) is already well entrenched as a PKI infrastructure for secure email [27]. PGP's open source nature and the "web of trust" certification mechanims also makes it attractive to end users since the hierarchichal certification mechanism of X.509 certificates can be quite expensive.

Similar to the SPKI specifications, PGP is more lightweight when compared to X.509. It can also be used in an anonymous scenario although it is recomended to use an email address as part of the user id field in the certificate for identifying a PGP certificate [26].

There are potentially two problems with PGP. Firstly, certain businesses could be reluctant to rely on the web of trust mechanism. For example, in a web of trust mechanism, it is possible that the user and a service do not have a common trust partner. This scenario will not occur with a set of certificate

[4]Privacy concern is not removed as the publisher can still potentially monitor the use - but this becomes more at a superficial level. Now the publisher cannot make associations between the age group of their consumers and consumer's product preferences for example

authorities common to the system. The second potential problem could be the user id's defined in the PGP certificate format. Since there is no specific format it is not guaranteed to be unique between two users (for example if two users called John Smith both use their names only and not their email addresses).

4.2 Usage in DRM system

We are leaning towards the use of PGP as the main PKI system. PGP is already well entrenched for use in secure email [26]. PGP is very user centric and actively promotes user privacy. The problem of trust can be solved by using a CA as one of the main trusted parties. Using the email address only as the identifier for the certificate, the CA can sign the certificate and return it via email. This will get round the second problem posed above.

For the sake of flexibilty, any of the three systems should be usable. However, for the sake of compatability all components must implement PGP system. Thus any components can communicate using PGP, but should two components prefer a different mechanism and they both support that mechanism, they should be able to use it. The certificate should be a standard PGP certicate with the identifier as only the user's email address. Before it can be used however, it must be certified by a CA trusted by the DRM system.

4.3 The Trust Service

Currently, certificate authorities are the only examples of a trust service available. As a CA, the service can sign public key certificates of users and other services. A CA *can* fit within a *web of trust*, by simply becoming a member of the the web of trust. In this manner, a PGP certificate can be used in both hierarchichal trust environments and its traditional web of trust environment. The CA should also sign device identifier tokens (see Chapter 10.3.4).

Manufacturers of TPM modules can also be seen as providers of a trust service. TPMs are discussed in more detail in section 10.4.

5 The Controller

Trust is a major factor in any transaction. DRM systems have to take this into account, especially when serving the public. The consumer expects that the service provider will not abuse any data that they collect, nor make any illegal side transactions. For example, the DRM system should not monitor the usage of the DRM enabled media unless the consumer agreed to such monitoring. Similarly, the service provider trusts the consumer not to break the DRM protection. Trust does not need to be absolute but there needs to be a mechanism to measure the degree of trust. In [17], the authors discussed setting the security level of a DRM package based on the consumer's trustworthiness. In our system, the controller is a TTP that can independently measure the trustworthiness of any other participant in the system.

The idea of a controller was first raised in [10] by Bartolini et al. who defined the controller as "*a Trusted Third Party (TTP) responsible for monitoring all transactions have been carried out legally.*". Bartolini et al. felt that the establishment of TTPs such as the controller and certificate authorities will have a large role in the success of a DRM system. The implementation of a controller as described by Bartolini et al. however implies a big overhead in network routing and traffic cost. The presence of a controller also raises the issue of privacy, as to monitor the legality of a transaction, the controller would also require to know all the details of the transaction.

However as discussed earlier, the controller has the ability to provide an independent trust valuation for all the parties in a DRM system. Thus instead of a per-transaction monitoring system, it would be better to describe the controller as an auditing system. The controller should be able to audit a component to verify that the component caries out its tasks legally, and implements the requirements of the component correctly. Thus it can audit the service producer on how well they protect consumer data, and can audit the consumer on how well the DRM controller protects the rights holders. This is important, even in open source implementations, as the implementation can differ from the original project.

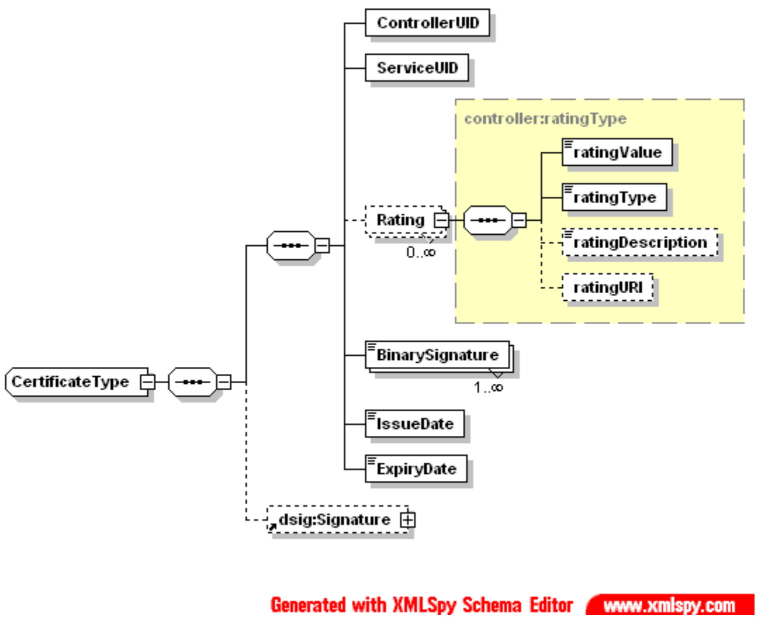

Figure 3: XML schema of a Controller Certificate

Thus the role of a controller is to audit a component, and certify that the component carries out all its tasks legally and maybe give a score on how trustworthy a system is. To certify a component, a digital signature of the binaries can be taken (individually if there are more than one binary) and any changes to a binary would require a re-certification. The controller then issues a certificate, which can be reviewed by any other party if they wish to. Figure 3 shows the XML schema of a controller certificate which can handle multiple binaries. It is not our intention to show how trust can be measured in a DRM system, but rather to provide a mechanism to describe the result of the measurement.

However, certification of a set of binaries does not imply that the component is using the certified binaries. This is an open problem and open to further research.

5.1 Scenario Analysis

In table 3 (section 3.1), we indicated that the controller is required only in scenarios 2 and 3 and while it can be used in scenario 1, it is not really required.

For scenario 2, the controller provides the rights holders a measure how much they can trust the consumer. Similarly, the consumer can gauge the trustworthiness of the music service and whether the consumers are safe in revealing any personal information (like a credit card number while purchasing music) to the service providers.

For scenario 3, the stakeholders in the trustworthiness of the system are not only the consumers and the hospital, but also the government. In many countries, there are laws that protect patients on how their medical data is used. The controller can be used to express the results of an audit of the computer system of the hospital group. This can be used to re-assure both patients and other interested stakeholders.

6 Identity and the UNI Service

6.1 Introduction

Every digital object needs a globally unique identifier if they are to exist on the Internet. Globally unique identifiers usually compose of a protocol identifier (eg. http), followed by a server or a host, followed by the actual identifier of the object.

The Unique Number Issuer (UNI) is the component responsible for issuing unique identifiers to to all parties and objects involved in a DRM transaction, and is one of the core components of the system. Because of the number of parties and digital objects involved in a DRM system, we have decided to divide them into different categories to allow for easier administration, and further classes can be added if required. In our system, all copies of the same digital object have the same identifier. The identifier (discussed in more detail in section 6.2) also has versioning support allowing two versions of a data file to have similar identifiers (and thus also allow use licenses to specify what versions the license applies to).

At the moment, our categories are:

Data: Data that is being protected by DRM need to be uniquely identifiable. The use licenses granted by the license servers need to match the data they provide access to. Different versions of data must have different identifiers. Details on versioning is detailed in the next subsection.

Licenses: Licenses used in the system should be identifiable (e.g. serial number) for auditing purposes. This would also allow for easier revocation of licenses or changing existing conditions in a license.

Users: An user of the system could be an author or publisher (creator) using DRM to protect their works or they could be a consumer (end user) wishing to access the DRM protected work. For an end user to access a DRM protected work, the user must be able to produce a valid use license for the DRM product. Likewise it is necessary to validate that the use license belongs to the end user in question. If requested, it is also necessary to link payments or usage data of a protected work for the creator.

Services: It is necessary for non-repudiation purposes to identify the web-services involved in a DRM transaction. Thus a license issued by a license server will contain at least four identifiers: the identifier of the license, the identifier of the object that the license is issued for, the identifier of the end user and the identifier of the license server itself.

Receipts and Invoices: Services that charge a fee for a transaction (e.g. an user buying a digital music from a distributor) require to document proof of payment or the invoice for the fee. Similarly, the user requires a proof that he/she has paid the fee. Furthermore, license servers would probably also require to link the use license with a payment receipt or invoice for auditing purposes (as in the previous example of an user buying music online, the use license should not be granted unless the user has paid for the right to use the music file).

Identifiers are handled by the Unique Number Issuer (UNI) service. The UNI's schema caters for two different root elements – SignedToken and SignedCommUnit. As discussed in section 3.4, the SignedCommUnit caters for communication with the UNI. The SignedToken element contains the identity token created by the UNI. Figure 2 shows the schema diagram for the SignedCommUnit while Figure 4 shows the schema diagram for the SignedToken element.

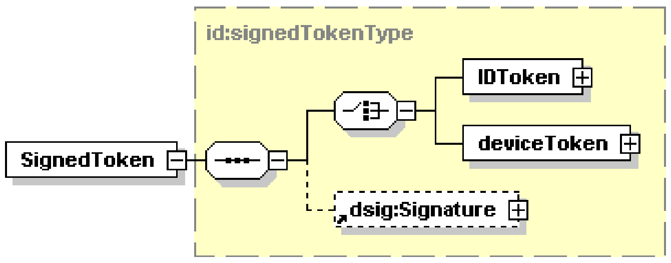

Figure 4: XML schema for SignedToken *element for the UNI service*

The Digital Object Identifier (DOI) has often been cited for use in a DRM system [22, 1, 21]. However, while the DOI format achieves the main objectives of an identifier, it does not have a verification feature. Thus there is no way to disprove that the identifier claimed by an object is not the actual identifier of the object. For this reason we consider the DOI in its present form to be unsuitable for DRM systems, but the UNI does build on the principles of the Handle service, which is what the DOI system is built on [21].

In this chapter, we detail the services that the UNI must provide as well as give details on the identity mechanisms of the framework. Each of the sub-elements in Figure 2 and Figure 4 are discussed in more detail.

6.2 Identifier Format

The format for our identifier follows the specifications of the Handle format. The identifier is split into two major parts: the prefix and the suffix. The prefix consists of the *directory identifier* and the *registration server* identifiers. The suffix consists of the *object class id*, the id and the version of the identifier. In general, the identifier should look like:

uni://directory_id.registration_server/object_class/id/version

Object class is represented by integers; specifically:

- 0 representing data,
- 1 representing licenses,
- 2 representing users,
- 3 representing services and
- 4 representing receipts and invoices.

This scheme allows for the addition of further object classes by extending the mapping.

The registration server is responsible for the allocation of the actual identifier. The directory identifier represents the server that allocated the identity of the registration server(for example, 10 represents the DOI foundation). The directory identifier is handled by the Handle system.

The *id* is generated by the UNI and can be in any alpha-numeric scheme desired. It is left up to the UNI to make sure that the id is unique. Combining the unique id with the rest of the identifier guarantees global uniqueness. Like the *id*, the version scheme does not have a prescribed format. A suggestion is to use **MajorVersion.SubVersion.MinorVersion** format. Thus two objects of different identifiers may have the same id, but by using different version numbers have globally unique identifiers. It is recommended that objects with different versions keep the existing identifiers (or maybe change identifiers at major versions). The major advantage is the flexibility in licensing, as licenses could implement a wildcard scheme for access to objects.

Identifiers must be case insensitive. Although the handle system does prescribe the use of case sensitive identifiers, the DOI foundation have commented on the complexity of such a system [21].

6.3 Identifying different unique identifier

The identifier scheme detailed in section 6.2 is not the only globally unique identifier. E-mail addresses, IM identifiers, phone numbers (with international and regional dialing codes) are all globally unique identifiers. There is a need to cater for these identifiers, esp. in handling user identities. Thus, we introduce a second variation of the UNI format to cater for such identities.

uni://identifier_type/identifier

The *identifier_type* refers to the type (or protocol) of identifier represented. This should be an alpha-numeric string that should be standardised to allow for inter-operability. Below, we list a few types that could be considered for such a list. The identifier is the actual identifier for the listed type, with the protocol reference stripped out (for example, for the Internet web address of W3C, the identifier_type is http, and the identifier is www.w3c.org). Similarly other identifiers including other handle types like DOI can be handled. This identifier does not need a UNI to issue it, and thus does not need the standard handle formatting. This also means that there is no way to verify the alternate identifier, but we think that the main use of the alternate identifier would be for user identification, which have their own verification mechanisms.

- **http:** representing Internet web addresses. Similar strings for other Internet protocols like ftp.
- **email:** representing email addresses.
- **aim:** representing AOL Instant messenger. Similar strings for other instant messengers like Jabber, ICQ or MSN.
- **passport:** representing Microsoft Passport.
- **liberty:** representing Liberty Alliance's federated identity management system.

6.4 Data, License, Receipt and Invoice Identification

Conceptually, the identifiers for these classes do not differ. The respective services that require an identifier (requestor) should make use of the UNI for creating unique identifiers. Creating identifers for users is more involved and is discussed in section 10. Issuing an identifier requires the digital signature of the object. This can be used to verify an identifier later. The process for creating an unique identifier for data, license, receipt and invoice classes is shown in figure 5 and discussed below. The nonces are used to correlate requests and responses correctly and counter replay attacks.

Step 1: The requestor sends a request for an identifier to the UNI. The request must include the following components:

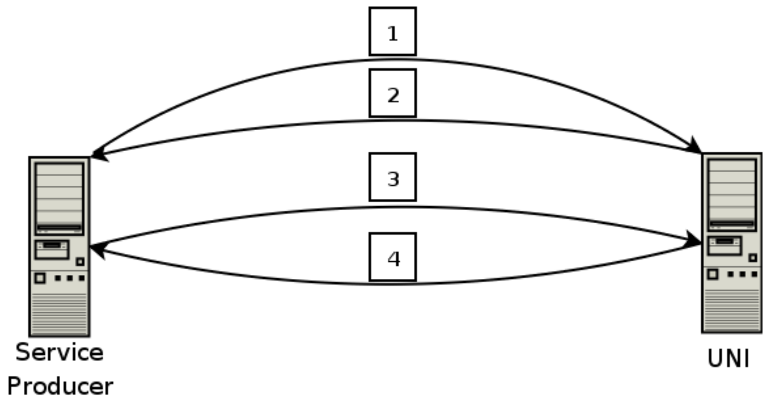

Figure 5: Fetching an identifier from the UNI

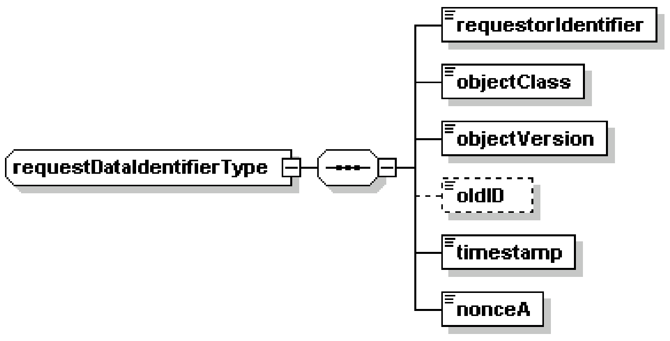

Figure 6: XML schema for requestDataIdentifierType *type for the UNI service*

1. The identifier of the requestor
2. The class of the object
3. The version of the object
4. If the object had a previous version with an identifier, the existing identifier and the digital signature of the old object. Note this only applies if the old identifier was issued by the current service in question. A different UNI service cannot guarantee that the older id field is unused.
5. A timestamp of the request
6. A randomly generated nonce to identify the request.

The request token's schema is shown in Figure 6.

Step 2: The UNI generates an identifier and returns it to the requestor. The identifier is not yet issued – just set aside. The UNI policy must set a lifespan of the identifier. The UNI must return a signed SOAP envelope to the requestor with the following details:

1. The identifier of the UNI
2. The identifier of the object
3. The identifier of the requestor
4. The timestamp of the identifier generation
5. The nonce sent in Step 1, as well as a challenge nonce.

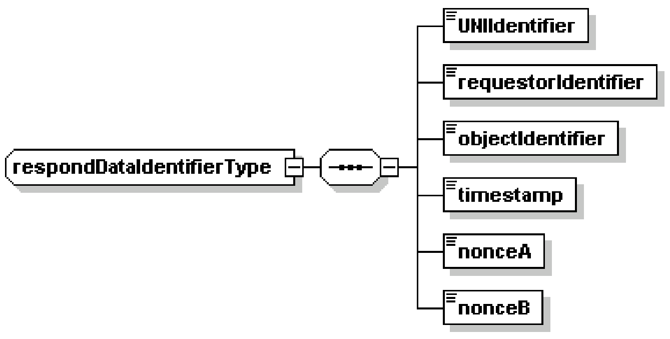

Figure 7: XML schema for respondDataIdentifierType *type for the UNI service*

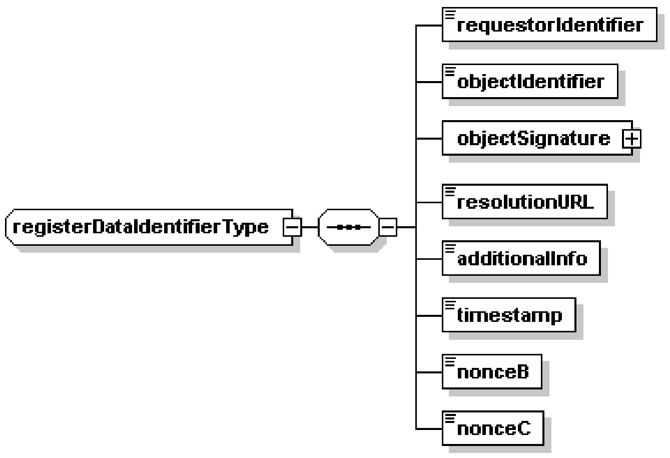

Figure 8: XML schema for registerDataIdentifierType *type for the UNI service*

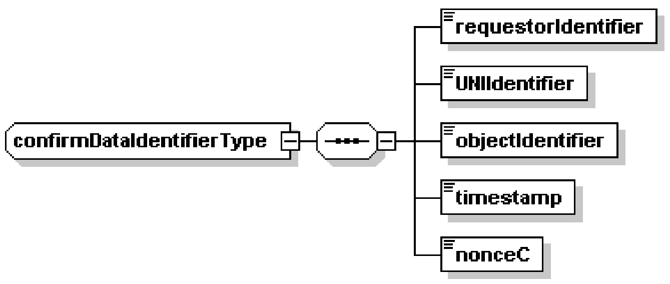

Figure 9: XML schema for confirmDataIdentifierType *type for the UNI service*

The UNI's response token's schema is shown in Figure 7.

Step 3: Once the requestor has assembled the data package that required the identifier, it sends the UNI a signed message with all the details that are required for registering an identifier. This message must contain teh follwoing details:

1. The identifier of the requestor
2. The identifier of the object
3. The digital signature of the object
4. The resolution address of the identifier
5. Any additional information that may be required
6. The timestamp of the response generation
7. The challenge nonce and another nonce from the requestor

This token can be represented in an XML schema as demonstrated in figure 8.

Step 4: Once the UNI registers the identifier, the UNI sends a confirmation message to the requestor. The confirmation message must contain the following details:

1. The identifier of the requestor
2. The identifier of the UNI
3. The object identifier
4. The timestamp of the response generation
5. The nonce generated by the requestor in Step 5.

This message can be described by an XML schema as shown in figure 9.

The identifier of the object needs to be generated first because the identifier might be needed as part of the object (for example, the identifier for a receipt could be part of the receipt itself). Separating the two creates a problem with generating the digital signature of the object. The UNI must not get a copy of the actual data.

6.5 User Identification

Identification and authentication are strongly linked. For this reason, the details of user identification is discussed in detail in section 10.

6.6 Service Identification

Service identification is essentially the Internet address used to access the service. Services can provide authentication and verification through the use of digital certificates as commonly used in current day e-commerce sites. Alternately, services can meake use of the user identity tokens described in section 10.

6.7 Unique Number Issuer (UNI)

As discussed earlier, the UNI service is an adaptatation of the Handle service. Thus the UNI needs to provide the services required by the Handle service. The UNI can be public or private; with private UNI's serving only specific clients. All UNIs must:

1. Provide a service for issuing unique identifiers for data, licenses, receipts and invoices. The requestor needs to provide a signed SOAP message with the digital signature of the object, the class of the object (data, license or player), the address for the handle resolution, the version of the object and if a previous version exists, the existing identifier. The UNI must then generate a globally unique identifier. Further details on the format of the identifier is given in section 6.2 and the communication protocol is detailed in section 6.4.
2. Provide a service for issuing unique identifiers to users and services. This protocol is described in section 10.3.2. The UNI must implement an email channel for communication with the requestor. Other channels are optional. The handle for service and user identifiers should resolve to the UNI if a specific URL is not selected.
3. Provide a handle resolution service. The handle should ideally resolve to a license server or distributor for the identifier of a data object, the license server for a license and the appropriate web service provider for an invoice or receipt. This is a requirement of the handle service.
4. Ensure persistence of the identifier even if the ownership of the license/data changes. For example, if the right ownership of a book is transferred between two parties, the identifier of the book should not change. This is a requirement of the handle service.
5. Provide a mechanism for verification. Given an identifier and a digital signature, the UNI must be able to return a "valid" or "invalid" response. Whould the identifier be not issued by the UNI, the UNI may route the request to the appropriate UNI and forward the request back to the user. Responses must be signed by the UNI. Verification must be provided to any requestor.
6. Provide a mechanism to change the URL for handle resolution.
7. Web Service Description. Given a request, the UNI must service the requestor with at least the following details:
   (a) The owner of the web service.
   (b) A human readable description of the service. This description can include other data that might be useful to the requestor.
   (c) Whether the service is public or private.
   (d) The web service's public key certificate.
   (e) If public, the classes of identifiers serviced by the web service.
   (f) If public, the payment details per class. If there is no such details, the web service can be deemed to be free.

6.8 Notes

6.8.1 Caching Optimisations

A possible missing service is the “request all tupples” which would return the identifiers and digital signatures stored in a UNI server. Such a system would allow for aggressive caching for a network of UNI servers and could make verification faster. However, should such a mechanism be allowed there needs to be adequate access control with only “trusted” UNI servers allowed to access such a service.

6.8.2 Redundancy and Resilience

The main advantage of using the handle system is the handle resolution mechanism. The resolution service should ideally resolve to a license server that would distribute end user licenses. The handle service also acts as a persistent identifier and can handle multiple resolution addresses for load balancing.

Thus even if the rights holders change the license server, there will be no need to change the identifier. This removes the complexity from the end user if there is a change in rights holders, license servers etc. It also for scalability – older distributed files do not need to be changed to use newer servers.

6.9 Scenario Analysis

The UNI is a required component and is used by all the scenarios. However the type of UNI deployment can vary between the three scenarios.

In scenario 1, the UNI service is used in a limited scope (can basically serve just the university). However, protected data is often distributed to users who are not part of the university. Thus, while the UNI itself needs to only serve the university, it needs to be accessible by the public.

In scenario 2, the UNI service needs to be publicly accessible. In fact, a business model where a person or enterprise operates and maintains a UNI service is possible. Thus the music store could make use of this service for handling the data identities.

In scenario 3, the UNI service is entirely private and public access can be reduced. This increases the security of the data, as any requests to the UNI service could be retraced to track potential leaked data.

7 Creating Protected Data

Data protected using DRM entails creating an encrypted package, a set of access rules and maybe some identification mechanisms for the data such as watermarking or fingerprinting.

The *Service Producer* component is responsible for creating DRM packages. The set of access rules, or rights are handled by the *License Server*. In this section we detail process of creating a protected package.

7.1 The Service Producer: Creating a DRM Package

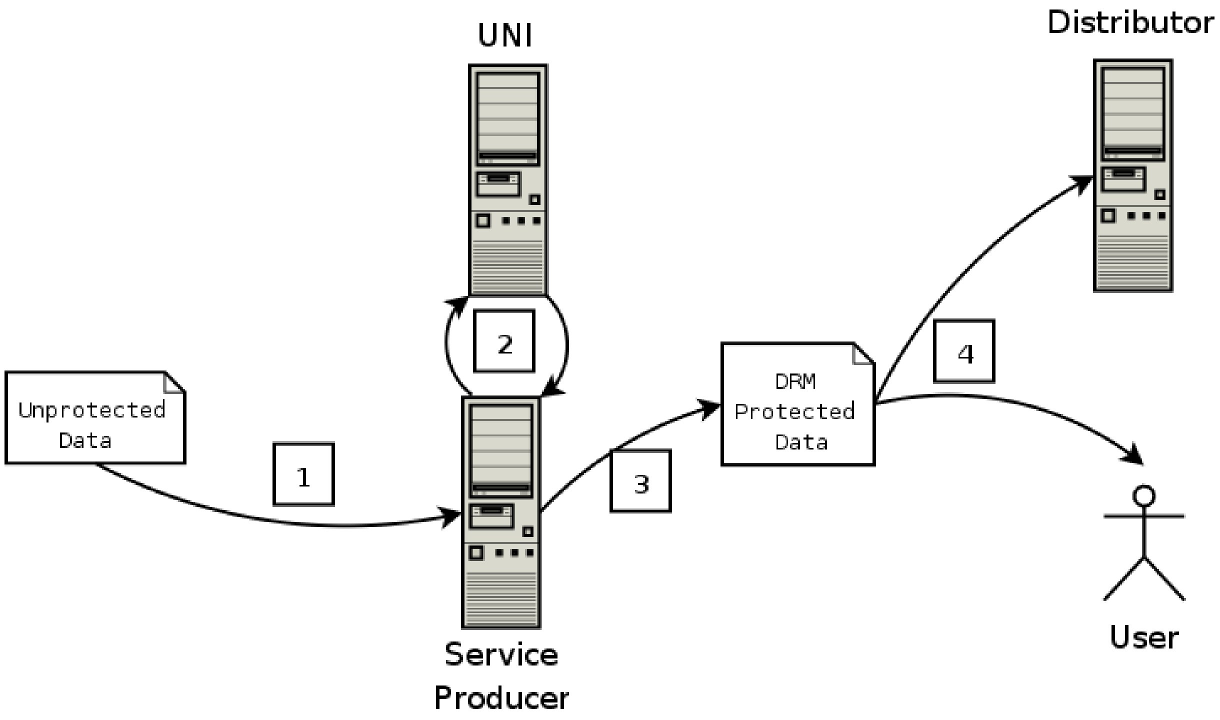

Figure 10: Creating a DRM Package

The process of creating a DRM package is shown in figure 10 and explained below.

Step 1: The user transfers the data file(s) to the service producer over a secure connection (for example through a SSL tunnel). The user also selects the options he/she would like on the DRM package, including encryption format, watermarking, compression etc. These options are further discussed in section 7.2. We recomend the use of the File Transfer Protocol (FTP) for transfering files. This protocol is fairly lightweight, widely supported and can be implemented through a secure communication tunnel. We do recognise that FTP is an insecure protocol, but the use of a secure connection does overcome the security problems of FTP. Figure 11 shows the XML schema that could be used to communicate these options.

Step 2 & 3: The service producer gets an identifier from the UNI and then assembles the DRM package. The process of getting an identifier is detailed in section 6.4, and assembly of the DRM package is discussed in section 7.2.

Step 4: The service producer can then forward the DRM package back to a distributor or transfer it back to the user. This option can be chosen by the user in step 1, or could be the part of

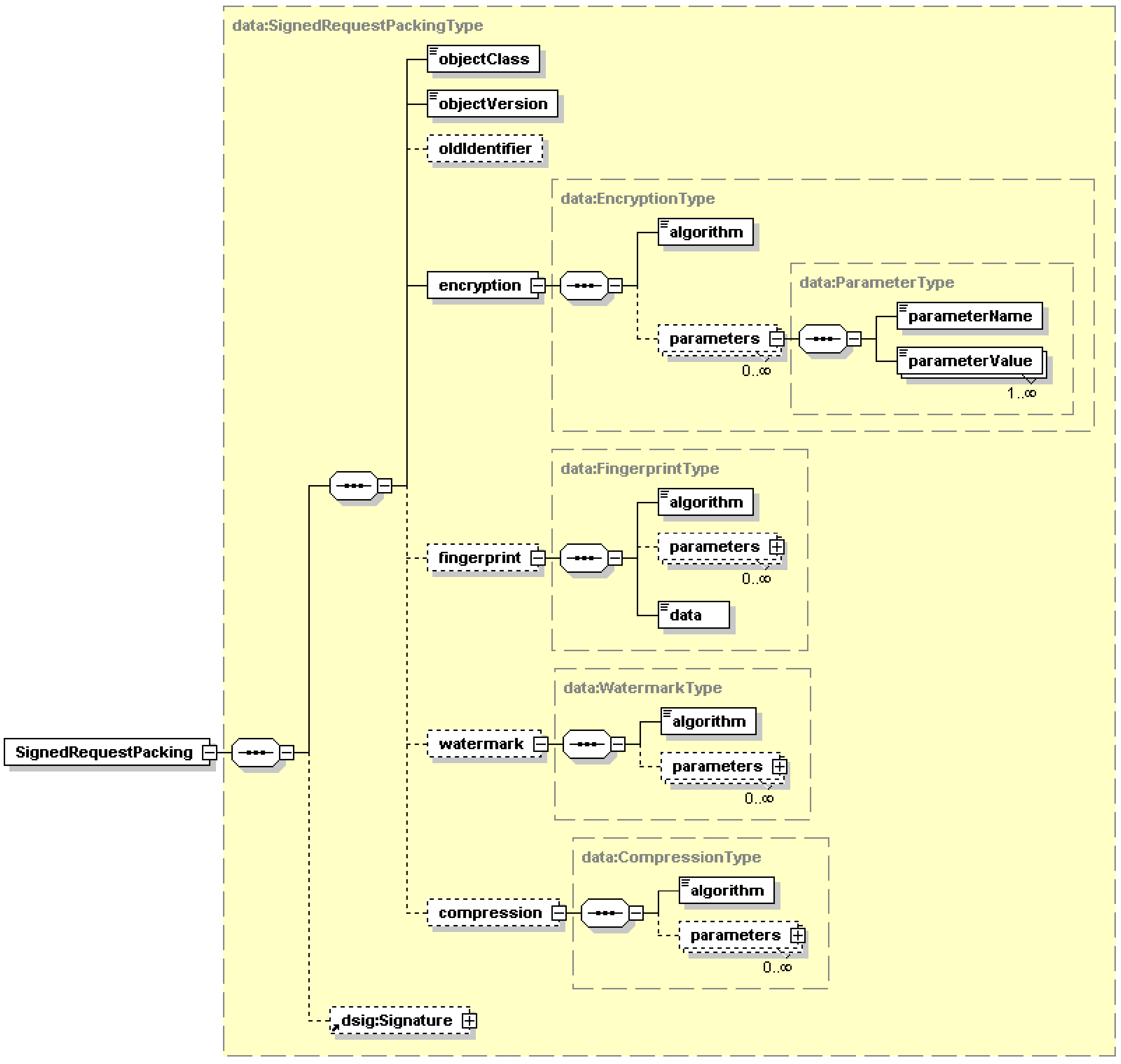

Figure 11: XML schema of a signed request to create a DRM package

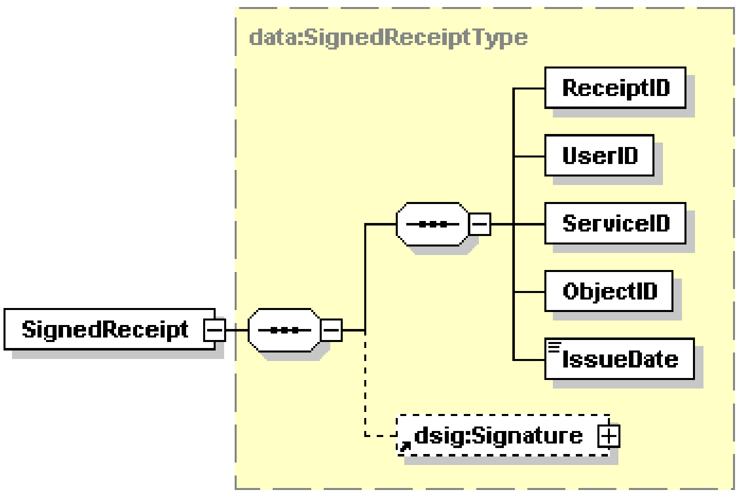

Figure 12: XML schema of a signed receipt from the service producer

the service producer policy. The user is given a "receipt", which ties the object to the user and the service producer. This token can then be used to prove to the license server that the user creating the license terms for the package is the rights holder of the package. Figure 12 shows the XML schema of such a token. The service provider could also forward the user to a selected license server.

7.2 Package Format

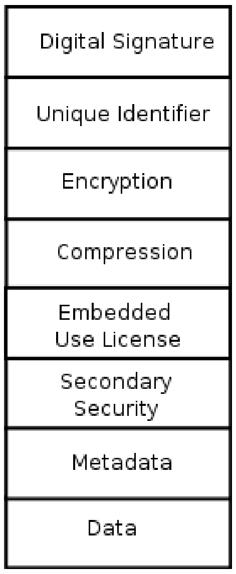

Figure 13: Proposed Layered approach for a DRM package

A DRM enabled data file is essentially an encrypted file with some metadata. It is the range of possible metadata that makes it more suitable than a simple data encryption. Current encrypted packages normally make use of an encrypted file and its digital signature as the package. However there are other security features that could also be useful – data could be fingerprinted for additional protection, or in some media, a watermark could be added to the package for a different identification mechanism. Current DRM systems however do not offer such flexibility and most of them do not offer watermarking or fingerprinting options. In this section we propose a flexible package format that we hope to be a base for a standardised package format.

A standardised package format is crucial for inter-operability. However fingerprints, encryption standards etc. must be flexible to suit the needs of the rights holder. For this reason we propose a layered approach to the problem – with each layer containing information about the technology used (for example, the encryption layer details the encryption algorithm and parameters for the algorithm). The layered approach is shown in figure 13. Currently, the layers we have are:

1. **Data:** The bottom layer, and represents the raw data that needs to be protected. The package allows for any type of data, and the end application needs to determine the type before using the data.
2. **Metadata:** Metadata is data about the data. Metadata is optional, and could be in any format.
3. **Secondary Security:** This layer allows for the service producer to add "extra" security features such as watermarks to the data or take a fingerprint of the data. These measures do not necessarily add to the protection of the data, but do carry other benefits. For example, watermarks can allow easy identification of images even when converted to an analogue form. Secondary security features are optional.
4. **Embedded Use License:** The embedded use license is optional and can be used to specify simple rules. The terms of the use license will be overwritten by any external use license. If an embedded use license is used, a separate key to unlock the encryption is required.
5. **Compression:** This layer can optionally compress the separate files from the previous layer as one compressed file. The package needs to describe the algorithm used as well as any parameters. Compression allows for faster encryption and hashing operations.
6. **Encryption:** The encryption layer describes the encryption algorithm and parameters used. All the layers preceding the encryption layer are encrypted with the same algorithm and parameters. The remaining layers remain unencrypted. Strictly speaking, encryption is not mandatory, and can be ignored – although this fails to provide any real protection to the data.
7. **Unique Identifier:** The identifier is issued by the UNI as described previously and all copies of the object will have the same identifier. The identifier should not be encrypted as any DRM controller needs to access the identifier to match with the use license. The identifier is mandatory.
8. **Digital Signature:** The digital signature provides support for the integrity of the data. The digital signature is a signed one way hash of the encrypted data concatenated with the identifier. The data package must detail the algorithm, the parameters and attach the signature to the final package.

The layers show how the service producer can process the data to create the DRM package, and similarly show the reverse steps for the DRM controller to access the package.

Using scenario 2, if a band wants to create a package with watermarking but no fingerprinting, no embedded use license, zip compression, AES 128 bit encryption with SHA-1 hashing, it is easy to specify and create. Similarly the end user's DRM controller can easily decipher the package and access the file once an appropriate use license is secured.

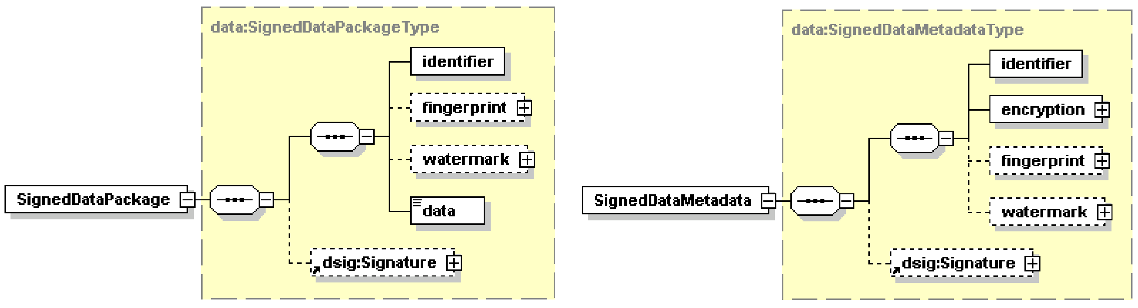

Figure 14: XML schema diagrams for DRM package and DRM package metadata

NEED TO WORK ON THE XML SCHEMAS IN THIS SECTION The package (excluding the compr) can be fully captured in XML. However, because of the processing power required to process XML files, it could be easier to separate the data and meta-data. Figure 14 shows the XML schema diagrams for both approaches.

7.3 Scenario Analysis

As discussed earlier, the Service Producer component is required by all deployments. But, like the UNI service, the service producer component can be shared by a number of deployed systems. We expect however enterprises to deploy their own service producers, while consumers could make use of public service producers instead of deploying their own. The service producer is not a complicated component, and thus it is also possible for every user to run their own personal service producer services.

8 Distributing DRM protected data

The core requirement of a DRM system is to protect data regardless of the location of the data. Thus the manner of distribution must not affect the security of the protected data. Furthermore, we specify the separation of the use license and the actual data, and thus a central distribution point for the data does not have any real benefits. In fact the use of peer-to-peer (P2P) distribution networks such as bit-torrent can lead to more efficient distribution when compared to a centralised approach[5]. For these reasons, the *distributor* is an optional component of the system.

The distributor can take different roles depending on how it is deployed. In a corporate environment, the role of the distributor can easily be a central file server. For a media store (like Apple's iTunes Music store for example), the store can be a traditional online store. In the later scenario, the store can also deploy a license server to serve the licensing needs of the user.

It is possible to use a distributor to "personalise" every distributed copy, for example through the use of a different encryption key or by using different settings. While this type of distribution is more computationally expensive, it does provide better security – a breach of a encryption key minimises the potential damage, and leaks are also easier to track. While this mechanism is probably not useful for distributing media to the public in the near future, it could be very useful in protecting very sensitive data that is not meant to be widely distributed.

The distributor essentially has three types of interactions – uploading data to the distributor, downloading data to the user, and browsing, listing etc. the data stored by the distributor. For intra-enterprise usage, these functions are provided by most file storage systems. For inter-enterprise or consumer systems, these functionalities can be easily provided by a FTP server. We do not prescribe any approach to a distributor; but if the distributor makes use of an open standard, it would make it easier for users of the distributor to make use of the service.

In section 7, the specified that the Service Producer can forward a newly assembled DRM package to the distributor. This function can only be provided if the component can interact with the distributor. The use of FTP protocol is probably the easiest in this regard, as this is the recommended protocol for the Service Producer.

8.1 Scenario Analysis

Scenario 2 is the only scenario which we think needs a distributor, and it is a simple online store. Scenario 3 deals with very sensitive data that is not widely distributed and the data is most likely to be stored in a central storage area and not regularly accessed. Thus it is possible to use a distributor to personalise the data for every different request (and also store the data in a DRM envelope). However, the nature of the deployment means that data needs to be accessed at high speed, and thus this form of distribution may not be practical.

[5]P2P networks are more resilient to denial of service attacks and can usually handle larger traffic volumes without compromising the quality of service. However, the performance of the networks do differ according to the P2P protocol used.

9 Accessing Protected Data

The DRM protected data is distributed within an encrypted package. To open the package, the end user requires a key that can decrypt the contents. Ideally, this key should be distributed along with the use license which specify the conditions for access to the data.

In this section we detail the process through which a DRM enabled work can be accessed by an application. Since DRM aims to protect data regardless of where the data actually resides, the distribution of DRM protected data does not concern us. Therefore we assume that the end user has got a DRM protected data and wishes to access it.

9.1 Components

The user role in our framework (see figure 1) can comprise of a few components. The main components are outlined below, but there can be other modules that fit in the user space, like a *Rights Interpretation Layer* as discussed by Jamkhedkar et al. in [17]. A module that is very important but not discussed is the *key storage* mechanism. The efficient and secure storage of keys is important for the effectiveness of the system, but this functionality depends on a lot of other factors such as the platform.

1. **DRM Controller:** This component is enforces the rules laid out by the use license. Ideally this would be an operating system kernel module or a hardware level implementation. However, the controller could be at an application level also, but the protection offered by an application level DRM controller is lower. The DRM controller is discussed in more detail in section 9.4.

2. **Communication Module:** This component would be used to communicate to the license server, authentication services and any other communication required.

3. **License Store:** This component stores, indexes and manages use licenses for the user. The store is used primarily by the DRM controller, but the store could offer mechanisms for offline uploading of use-licenses or even automatic updating of use-licenses etc. through the communication module. Ideally, the license store will be a tamper-resistant hardware module, but such a device will probably not be available for some time.

4. **Revocation List:** There is a need to keep track of invalid use licenses. Use licenses could become invalid for a variety of reasons like the end user upgrading the terms of the license or the end user transfering the right to a different party. Ideally the revocation list gets updated at regular intervals, but it could be possible to distribute revocation lists with use licenses. However, even though revocation lists need only be the use-license identifiers, the list could become very large, and separation of the use-license and revocation list would be preferred. Alternatively, the license server could also allow online verification for the the use-license's validity.

9.2 Process – Accessing DRM enabled data

The main responsibility of a DRM system is to enforce the access controls for that system, and that responsibility lies with the DRM controller module. The remaining components are involved in supplying all the required data to the DRM controller. In this subsection we shall show the access paths for scenario A (section 2.1, and an overview of the process is shown in figure 15. The access pattern will be the same regardless of the scenario and thus we do not repeat the access path for all the scenarios.

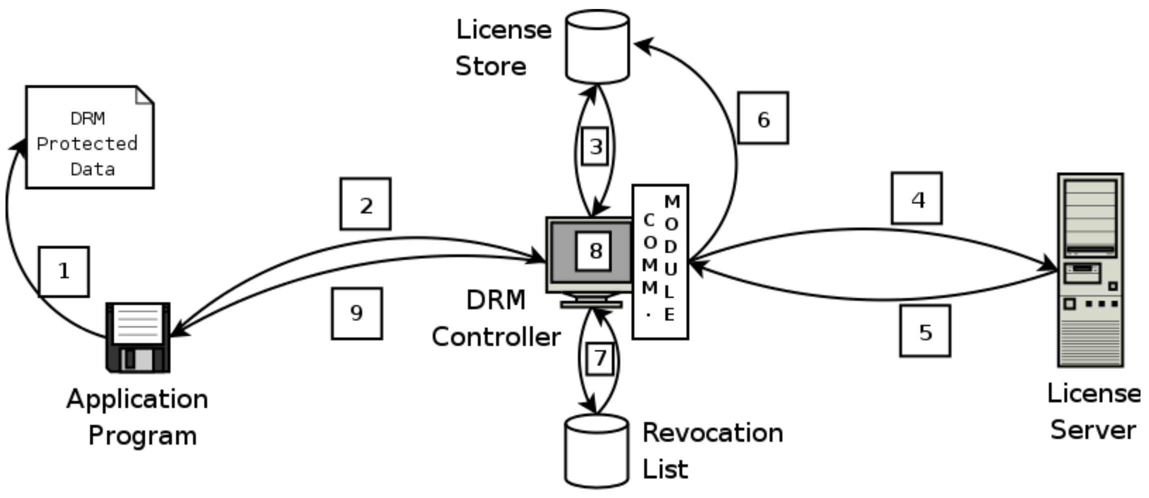

Figure 15: Steps required to use a DRM enabled work

Carl has distributed to the protected exam papers to the lecturers concerned. We describe the process that takes place when a lecturer, James, wants to view the exam.

Step 1: The application recognises that the file is DRM enabled and thus cannot open the file.

Step 2: The application asks the DRM controller if it can open the file. If the DRM controller is an operating system or hardware module, it could have intercepted the application's initial attempt at opening the file and thus handled it automatically. The DRM controller is discussed in more detail in section 9.4.

Step 3: The DRM controller looks up whether an use license exists for the data file in question. The license store gives the DRM controller the license if it exists or a message indicating a license for the data file does not exist yet. If the license exists, the DRM controller moves to step 7. The license store could have an expired license[6], in which case the DRM controller continues with step 4.

Step 4: The DRM controller contacts the license server (by resolving the UNI of the data) for a use license. The user authenticates themselves to the license server, pays for the license if required and should also be allowed to negotiate with the license server for specific terms if the rights holder allows for such an option. The authentication process is detailed in section 10 while the license server is discussed in more detail in section 12.

Step 5: If a license is granted, it is transferred to the DRM controller. If a license is not granted, an appropriate error message is sent to the DRM controller. If a license is not granted, the user should be prevented from accessing the file.

Step 6: The DRM controller stores the use license in the license store.

Step 7: The DRM Controller checks the validity of use license against the revocation list. As explained previously, this action could be done online with the license server. If the license is invalid, the user could negotiate a new license with the license server (Step 4).

[6]A license expires once the last date of the license passes. An invalid license is a license that is put on the revocation list but not necessarily expired. Once an invalid license expires, it can be removed from the revocation list.

Step 8: The DRM controller authenticates the user (James) against the user specified in the use license. More detail on user authentication is given in section 10. If authentication fails, the user should be prevented from accessing the file.

Step 9: If authentication is successful, and the use license allows the action the application wishes to perform, a positive message is sent to the application. For every subsequent action that the application wishes to perform (for example, if James wishes to print a copy of the exam), Steps 2 and 9 are performed if the DRM controller can cache a copy of the use license in use. Should the license expire during use, steps 5 – 8 need to be redone. Authentication of the user is not necessary for every action, but can be scaled according to the use license – for example the license may stipulate that the user needs to authenticate themselves after every hour of use.

9.3 Time Synchronisation

Time synchronisation is a solved problem, and most modern operating systems offer time synchronisation protocols. Having UNI and license servers as time server could be very beneficial, as most end users are likely to make use of these servers regularly.

9.4 DRM Controller

As discussed earlier, the DRM controller can be implemented at various levels in a computer system – at an application level, as an operating system module and at a hardware level. The level of protection is lowest at the application level – there is no way for the DRM controller to stop operating system functions (such as file copy). Currently, all DRM systems make use of application level DRM controllers.

A DRM controller implemented as an operating system module can be far more effective. Now, applications no longer need to be modified to access DRM enabled files – the kernel catches the request to access the file, makes a descision on the request and then allows or disallows an application to carry out that request. Thus the application should not even need to know whether the file is DRM protected or not. However, kernels can be patched, and it is still software control. Thus, while the security and usability offered by a kernel level DRM system is very promising, the protection offered is not absolute. Curently, Microsoft's Rights Management Services (RMS) has the closest kernel level DRM controller implementation but it still requires application level changes to function.

A DRM controller as a chip is ultimate, as hardware level circuits are difficult to bypass. However, hardware level DRM does also have a counter problem – flexibility becomes harder to achieve, and while software can be easily patched or upgraded, hardware is harder to upgrade.

9.5 Notes

The crux of the security lies in the authentication of the user against the use license. The model described above allows for portability – the user can carry their data and license to a different device (with a compatabile DRM controller and applications) and access the data as long as they can authenticate themselves to the DRM controller. These authentication processes are detailed in section 10.

The DRM controller can also have other functionalities. All the current RELs strive for generality, and thus even though they offer standardised grammar, the semantics of the grammar is not standardised. However, standardising the semantics can lose the generality of the REL – for example the term *render* requires different interpretations for image, video and audio files. In [17], the authors proposed the idea of a "*Rights Interpretation Layer*", a module that can interpret the different semantics of the license. Such a module should be part of the DRM controller.

It would also be useful for the DRM controller to allow loading of licenses as an additional function. Thus the license does not have to be downloaded from the license server, but from a file on disk. This feature will be particularly useful in providing mobility for the user and for devices not connected to the Internet (for example portable music players).

10 User Authentication

There are two different times when an user would require authentication in the system. Firstly, there is a need for the user to authenticate themselves to the various Web Services – the license server, the distributor, the service producer etc. This authentication can be done using various Web Service enabled authentication mechanisms including federated identity management schemes such as Liberty or WS-Federation, proprietary solutions such as Microsoft Passport or even custom authentication services that are linked to the corporate network, like to an LDAP service. Except for the license server, authentication for the other services depends on the particular deployment of the system.

The second authentication occurs at step 8 of the access process detailed in section 9. In Step 8, the user is required to authenticate themselves to the DRM controller. Here the user provides authentication to prove that the use-license belongs to the user that wishes to access the protected data. This authentication can be done through an online authentication protocol (which would require the user to have a connection to the authentication server) or through offline mechanisms.

Online authentication requires an active network connection to the server and is mostly beneficial to rights holders as they can retain a larger degree of control over their work. Thus in a rental scheme, the rights holders can suspend an account if the user has not paid his/her bills. Online authentication however also allows the rights holders to actively monitor the frequency of usage through how often the user authenticates themselves online. Offline authentication on the other hand is more beneficial to the user. The rights holders loose most of their control on the user as the user does not need any interaction with the license server after purchase. In this section, we look at how online and offline authentication can be achieved.

10.1 User and their Use License

Like every other component in the system, the user also requires a globally unique identifier. However, the UNI service does not have to be the only source of this identifier – other global identifier schemes such as Liberty or Microsoft Passport can also be used for this purpose. The DRM controller however must be able to interpret and process this identifier. The user may have multiple unique identifiers, both online and offline identifiers, and the DRM controller must be able to handle such a variety.

In section 6.3, we detailed an alternate identifier format that caters to other identifier formats. The main use of the alternate identifier formats will be in use licenses as a mechanism to identify the user. Thus it will be possible for the use license to specify virtually any identifier for the user, and the DRM controller will be able to interpret the identifier. Modularising the authentication mechanisms for the DRM controller should allow for the DRM controller to support multiple authentication schemes.

10.2 Online Authentication

All current DRM systems employ online authentication. In systems such as Apple iTunes, the iTunes store authenticates the computer running the iTunes software (which is essentially the software and DRM controller) as a trusted machine; and the machine is then able to play audio files bought from the music store. Other systems such as Microsoft's RMS platform, the RMS client authenticates the user against the company's active directory server, and then retrieves a use license from the RMS server. This use license is usually time locked and expires after a specific period.

Online authentication is only needed in cases where the protected data requires a high degree of protection or where the end user is not trusted enough to use offline authentication. Online authentication provides a high degree of security, as use licenses can be easily revoked if the user breaches the license agreement or is no longer trusted by the rights holder. However, there are also privacy concerns, as online authentication allows the right holder to easily track the user's usage.

Alternatively, online authentication can be used to generate a second use license that is time locked, or locked with a particular device. The use license could then direct the DRM controller not to conduct authentication in step 8. This type of use license could also be generated by the license server as the initial license.

Any Web Service based authentication system can be used for online authentication. An obvious system would be the federated identity management systems like Liberty Alliance or implementations of the WS-Federation. As discussed in section 6.3, use license can make use of the alternate identifier format to identify the protocol for authentication. Thus we do not specify any particular authentication mechanism.

10.3 Offline Authentication

Offline authentication is highly desirable, especially if DRM is to succeed in the consumer space. However, offline authentication is difficult to achieve – users and services do not have secure hashes that can be easily verified. An intermediate token can be used to solve this problem; the token has an identifier and the possession of the token can be used to prove authentication.

Ideally, the token should be hardware based, unique, mobile and tamper resistant. One such token is described in section 10.4. Another token is the smartcard based system used in the Estonian Digital Signature Project [5]. The smart card doubles up as an identity document, and provides a public/private keypair, a digital certificate and an offline authentication mechanism. However, neither token is available in mass circulation – the latter available in Estonia while the former only on certain laptops. For this reason we discuss an alternative software based mechanism in this section.

Digital certificates provide the requirements of a token based system, and they have an additional advantage in providing a PKI interface. However digital certificates have two main drawbacks:

1. Number of different formats
2. Expiry (and revocations) of digital certificates

The identifier lifetime is expected to be longer than the lifetime of a digital certificate. In section 4.2 we advocated using PGP as the standard PKI system. PGP certificates do have the advantage of having an optional expiry date, but revocation of a PGP certificate remain a problem. If a certificate is revoked, the owner of the certificate should not loose access to their data since revocation of a certificate means that the identifier associated with the certificate is also revoked. While it could be argues that one can re-use the same identifier with a new certificate, this mechanism is not ideal as revocation lists of digital certificates works by associating the identifiers of the certificates.

Yet another problem with a certificate based identity is portability. Because current certificate systems are software based, they can easily be replicated. A good example is the OSS Sound Drivers for Unix platforms available from 4-Front Technologies. 4-Front uses a PGP certificate to grant use licenses to the end users. However, these use licenses are easily available online as individuals who buy a license can easily distribute the license afterwards (although it is illegal). However portability is also a key requirement of a DRM system. End users must be able to move the data they have rights to between machines they own. Ideally, an identification system for end user must take this into account.

We propose a 3 token system that provides authentication, identification and mobility. These tokens are:

A user/service identity token which can be used to identify the user. The identity token must be as permanent as possible, provide anonymity but at the same time allow the UNI to verify the user of the token by some means. It would also be useful to have the identity token provide a verification mechanism itself.

A device identifier list which lists the devices owned by the user or service. Use licenses that only allow a limited number of devices to be used should use the list to assign the devices to the use license.

Digital certificates which will be used for PKI purposes. Both the device identifier list and identity tokens must be signed by their owners, and the digital certificate can be used to link the tokens together as only the digital certificate can verify the signature.

In the following two sections we describe the specifications of the tokens as well as detail the protocol for creating the tokens.

10.3.1 User Identity Token

The xml schema diagram of the proposed document is given in Figure 16. This token is part of the *SignedToken* element. We recommend following the PGP certificate style and have a minimum of two signatures for SignedToken – the signature of the token's owner, and the signature of the UNI that releases an identifier.

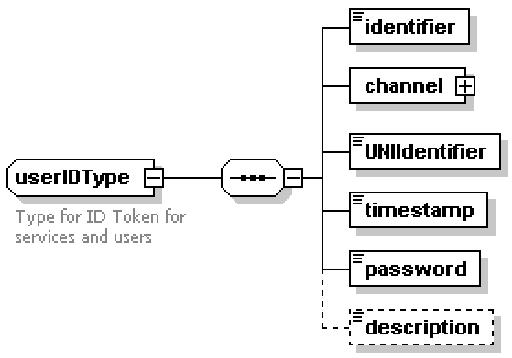

Figure 16: XML schema diagram for the userIDType *type*

The contents of the token have five mandatory elements and one optional element. The identifier element is the identifier as given by the UNI. A possible transaction mechanism between the UNI and the user is described later. The second element is a channel element. The channel will be used as a means of authenticating the user; and only one channel is required. As shown in Figure 17, the ID token currently provides for email, cellular phone, pager and common instant messenger as channels. A custom channel type and ID is also available and thus even the traditional letter can be used as a channel. The third mandatory element is the identifier of the UNI. This effectively identifies the issuer of the identifier.

The next two fields are the timestamp and password elements. We have used the password storing mechanism of UNIX [27] for the token. The password should be hashed using SHA-1 and then stored in the token. To add complexity to the hashing, the timestamp should be concatenated with the password before hashing. This would ensure to some extent, that even if the passwords chosen by

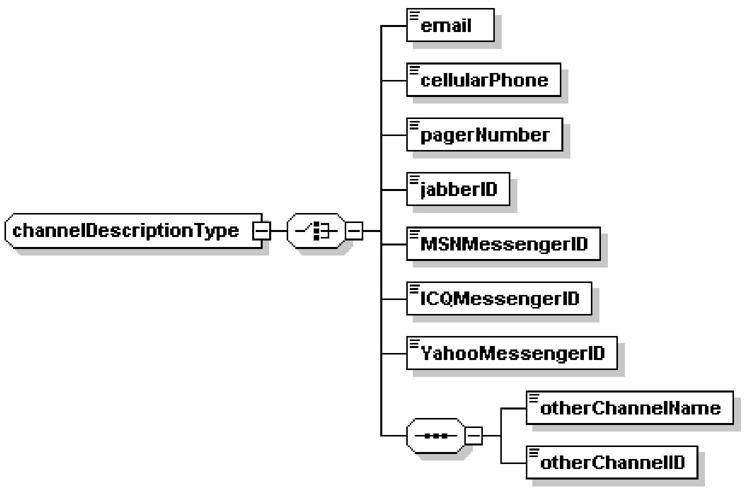

Figure 17: XML schema diagram for the channelDescriptionType *type*

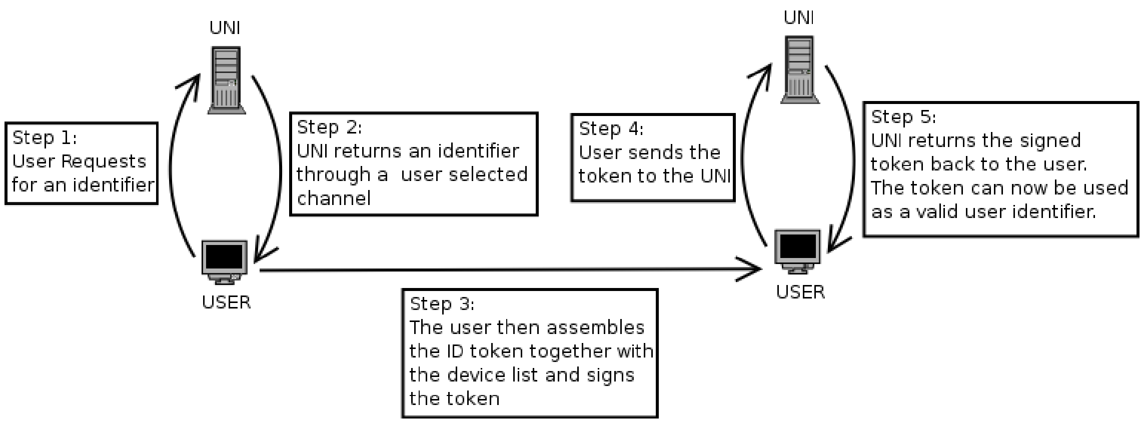
UNI
Step 1: User Requests for an identifier
Step 2: UNI returns an identifier through a user selected channel
USER
Step 3: The user then assembles the ID token together with the device list and signs the token
UNI
Step 4: User sends the token to the UNI
Step 5: UNI returns the signed token back to the user. The token can now be used as a valid user identifier.
USER

Figure 18: Steps in generating a Global User Identity Token

two users are the same, the resulting hashes are different. This mechanism ensures that the the ID token is self validating; if a user needs to authenticate themselves, they can present the token and the password.

The optional description field can be used for any other information that the user would like to have on the token. There is no field for name or any other identification mechanism in the token to protect the user's privacy, but this information can be put in the description field if required. Another option is to create a custom token for personal data, and use the identifier issued by the UNI to link the custom token with the identifier token.

10.3.2 Creating a User ID Token

There are 5 simple steps in creating a User ID token, as shown in Figure 18. These steps will be discussed in more detail below.

Step 1: The end user requests the UNI for an unique identifier. This can be in the form of a web based interface, or through a web service with a client application. As part of the request, the user chooses a channel for a response and the appropriate channel identifier. This ensures that

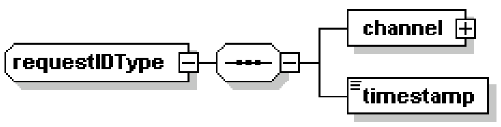

Figure 19: XML schema diagram for the requestIDType *type*

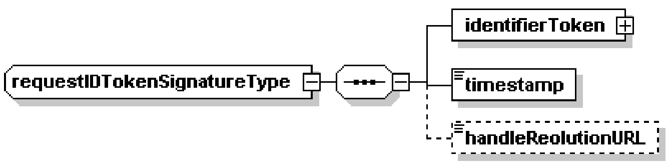

Figure 20: XML schema diagram for the requestIDTokenSignature *type*

there is a "real" person who will own the user token and the email based system is currently used frequently in many online registration systems for activating online accounts. The channel response is not limited to an email, and should the UNI support other mechanisms, like a sms to a cellphone, the user could make use of it. The UNI must implement the email channel. A timestamp is also present, and the UNI can discard all requests that are "stale". The amount of time before a request is deemed to be stale can be decided by the UNI. The XML schema for such a request token is shown in Figure 19

Step 2: The UNI creates an unique identifier and sends the identifier through the channel selected by the user.

Step 3: The user assembles the identity token using the identifier, the channel that was used to communicate the identifier, the UNI's identifier and the list of devices owned or used by the user. The user then signs the content block with his private key. As in PGP certificates, signing an own public key is very important [26] – it ensures that no third party including the UNI has made any changes to the identity token.

Step 4: The user then sends the identity token back to the UNI. As in step 1, a timestamp field is also present in the request as well as an optional handle resolution URL. If the handle resolution URL is not present, the identifier should resolve to the UNI. The XML schema for the request token is shown in Figure 20

Step 5: The UNI signs the identity token and then sends it back to the user. The XML schema for the response token is shown in Figure 21. The identifierToken is an instance of a SignedToken. By signing the identity token, the UNI certifies that the identifier in token was:

(a) was generated by the UNI, and that

(b) the channel id matches the identifier

The UNI also stores the signature with the identifier in its database and set the appropriate handle resolution address. This will allow the UNI to provide a verification service. Details on the verification service can be found in section 6.7.

10.3.3 Changing a User Identity Token

If the owner of an ID token wishes to change the token (i.e. change the details of the channel type and ID, the description or the password) then it would require a modification of the token. The user's ID

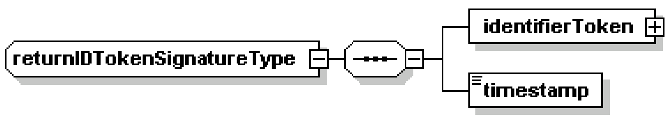

Figure 21: XML schema diagram for the returnIDTokenSignature *type*

cannot be modified; as the user can easily get a new token if they desire. Token modification is quite simple. As in token creation, the requestor assembles their new token and sign it. The new token and the old token is then sent to the UNI (XML schema shown in Figure 22). The UNI then sends a nonce along the new channel to the token owner. The owner would then need to send an accept message back to the UNI thereby verifying the validity of the channel. The UNI can then sign the new token, update its database and send the signed token back to the requestor.

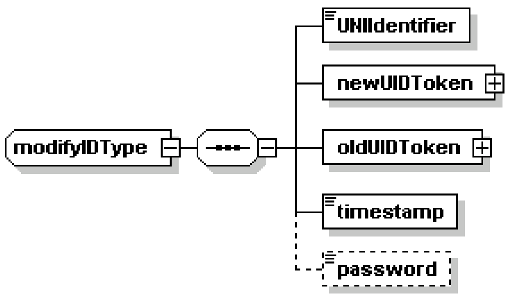

Figure 22: XML schema diagram for the modifyIDType *type*

The user should use a password for authentication, but this feature might not be necessary in closed environments. This is the only scenario where an authentication service is required for the UNI. All other services should be open, although access control can be employed should there be a need to restrict the users of the UNI.

10.3.4 Device Identifier List

Unlike the identifiers discussed previously, device identifiers cannot be independently verified. In fact creating unique hardware identifiers themselves are a challenge, and there are no standards on uniquely identifying electronic devices. Although it is possible to construct an unique identifier using the serial numbers of the components in the device (like the processor, the network interface card, motherboard etc) there does not seem to be any standardised algorithm to extract such information. Furthermore, except for network interface cards, hardware devices do not have a standard mechanism for serial numbers. There is also the added complexity of changes in the hardware profile of a device due to upgrading components.

In these specifications we do not give a mechanism to identify hardware. License servers that wish to restrict usage to specific devices need to:

1. Find a scheme to uniquely identify a device, and
2. create a mechanism to verify that the identifier has not been tampered with.

The second part could be implemented through the use of an application that extracts the hardware

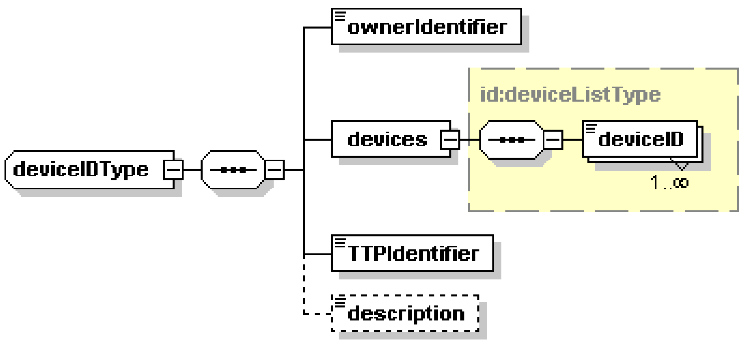

Figure 23: XML schema diagram for the deviceIDType *type*

identifier, creates the identifier token and then signs the token. A XML schema to represent the device list token is shown in Figure 23. The *TTPIdentifier* field identifies the trusted third party (or application) that generated the token (or signed the token). The *OwnerIdentifier* field containts the identifier of the owner of the token. The device list token forms part of SignedToken and thus the request and return signature XML schemas (Figure 20 and Figure 21) discussed above can also be used for requesting a TTP to sign the token through a web service. The token does not have an UNI issued identifier and thus there must not be any URL to resolve to.

10.4 Trusted Platform Module (TPM)

The Trusted Computing Group (TCG) is an industry consortium that aims to develop and promote an open industry standard for trusted computing [12]. The consortium consists of leading hardware vendors like IBM, Intel and Hewlett-Packard. The Trusted Platform Module is an unique, trusted, tamper resistant hardware module from the TCG. Currently, only available on laptops from a few vendors, the TPM provides certified RSA key pairs, as well as a protocol to provide online authentication without compromising user privacy [12]. Such a module would be ideal for authentication in a DRM system even with the lack of mobility. If the TPM can be produced in a mobile format (similar to USB flash drives for example), mobility can also be provided.

The TPM (and any other hardware based system) is also compatible with the system proposed in section 10.3. The hardware modules all have unique identifiers, and thus can be listed in the device list. Thus a user can have portability if the use license allows the user to use a subset of the devices listed on the device list. The digital certificate however can become unnecessary if the hardware based system provides public key cryptography.

10.5 Summary

In this section, we looked at the authentication requirements for the user. The TPM promises to provide a great solution to user identification and authentication, as it supports both online and offline authentication support. Furthermore, it provides mechanisms for user privacy and thus is ideal for DRM. Mobile TPM would also allow usage in multiple devices like PDAs. However, until TPM becomes commonly available, we provide an alternative (but possibly less secure) mechanism for user authentication and identification.

10.6 Scenario Analysis

As discussed by Mulligan et al. [19], consumers should have limited interactions with the rights holders after they complete their purchase. Thus in scenario 2, offline authentication is the best solution. Offline authentication also satisfies the requirement of end user privacy for scenario 2. However, in scenarios 1 and 3 tracking is a requirement and thus online authentication is the solution. Both scenarios already have possible online authentication mechanisms such as the user management systems used in the existing computer network.

11 Credentials Authority

Copyright law in most countries allow users certain rights depending on the use of the material. For example, under South African copyright law, teachers and academics are allowed to make reproductions of copyrighted material if it is used for teaching purposes [2]. Similarly journalists are allowed to excerpt quotes for the purpose of reviews. A difficulty in current DRM systems is that there is no mechanism to distinguish between users, like whether a user is a teacher, and thus all users are given the same rights. This results in conflicts with fair use allowances in copyright laws.

Credentials can be used to solve this problem, if a trustworthy structure is set-up to facilitate such a transaction. For example, professional organisations such as ACM or IEEE can provide credential authorities who can certify whether a user is a member of a the organisation or not. Thus, provided trusted authorities can set up credential authorities, they can prove to the license server that the user is a certain "type" of user.

Figure 24 shows the two ways the credentials authority can be used. Firstly, when the user can request a license with fair use requirements, and forward the credential service that can provide the credential for the fair use. The license server can then contact the credential authority on the user's behalf and get the credential. Alternatively, the user can get the credential from the credential authority and forward it to the license server. If the credential authority is trusted by the license server, the license server can provide the user with the fair use requests.

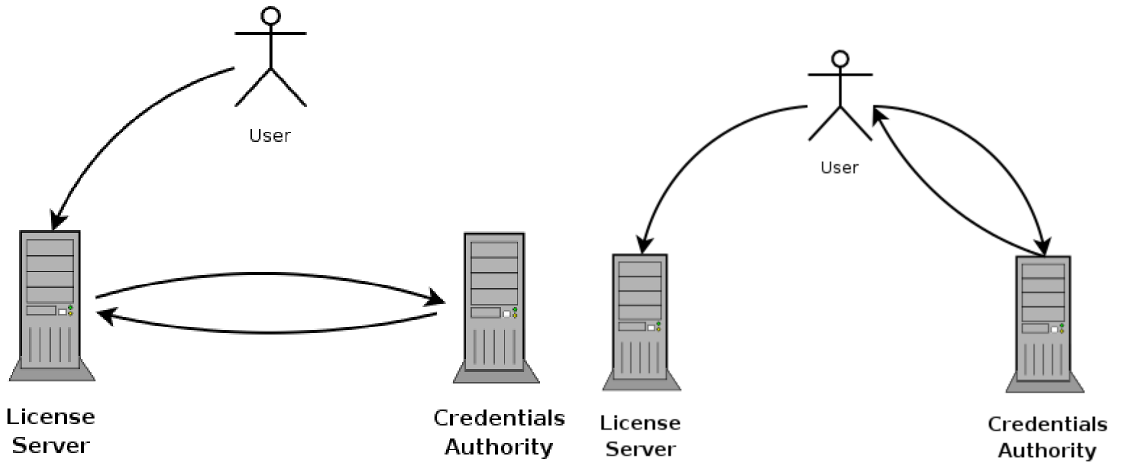

Figure 24: Using the Credentials Authority

Another function that can be performed by the credentials authority is that of access control. Access control does not require any modification to how the service is defined above. Regardless of the use, the credentials authority is strongly linked to the authentication service, and both services may be provided by the same web service.

11.1 Representing Credentials

Currently there are no web services standards or recommendations for credentials. Use of WS-Trust or WS-Security has been suggested for the use of credentials. Iny [14], the authors suggest the use of Universal Description, Discovery and Integration (UDDI) for credentials. UDDI is currently used by the *service registry* in the web services model. It provides service requestors means to find services provided by service providers. The authors propose an extension to the function of the service registry to provide credential services.

11.2 Scenario Analysis

Credentials could be used in both scenario 2 and 3. As discussed above, credentials could be used as a mechanism to negotiate fair use, and thus useful for scenario 2.

In scenario 3, the use of credentials would be different. Ideally, medical records of an individual should only be accessible by the persons authorised by the patient, like his/her doctor. However, a situation may arise where the patient is in a critical condition and no authorised personnel is available to authorise access. In a credentials model, the patient can choose to allow doctors and nurses of a certain level access to the data (for example, surgeon on call can open file). The credentials model will still allow for confidentiality, and also allow to track cases where data is accessed when there was no need (the patient is perfectly healthy but data is still accessed by the surgeon for example).

12 Licensing

The use license specifies the conditions for accessing the DRM enabled data; and usually caries the decryption key for the DRM package. The issuing of use licenses is handled by the *license server* component. In this section, the protocol for issuing a use-license is discussed as well as the format of the use license.

The process of the getting a use-license can be broken up into three parts:

1. Authentication,
2. Negotiation and
3. Creation and issuing of the license.

The license server is also used by the creators (or license holders) to create the licenses. The creators can create various versions of licenses, and should be allowed to have multiple types of licenses for the same object. The user will always communicate with the license server and acts as the proxy rights holder. In this section, the majority of the communication will be between the end-user and the rights holder with the rights holder's role handled by the license server. The process of creating use-license templates can be broken up into two parts:

1. Authentication, and
2. Creation of templates

12.1 Authentication

Authentication of a user has two functions in the license server. Firstly, authentication allows the user to prove that they have the right to access the license server. This is important in a corporate environment as the corporation may decide only certain users should have the right to create or use protected document. This authentication should ideally take place using a federated identity system to maintain consistency across different components. The second purpose of authentication is to create a link between the use license and user (either as a creator or as the end user). This link is not necessary in the later case, as license servers are likely to provide "gift services" where the purchaser of the license is not the consumer of the product. In such a case, the purchaser can ask the license server to issue a license against a different user identifier.

Because of the distributed nature of the system, it is possible for anyone to create license templates for a DRM work. To restrict only the rights holders to create license templates, we introduced a receipt token in section 7.1. The receipt token can be used to prove that the user creating the license template is the license holder.

There is a complication however, when the user creating the package is not the license holder (for example, in a corporate environment, the rights will usually belong to the company and not the employee). This problem can be solved in two ways – either issue the token with a different user identifier from the package manager, or have a separate signed token transferring the rights from one user to another. The former approach is simpler while the later approach has the advantage of being useful after the DRM package has been created. However, with the later approach, it does not stop the first user from creating his/her own license template with a different license server.

12.2 Negotiation

The use license is in effect a contract – the end user agrees to a set of rules that they are going to adhere to in order to get access to the data. In current systems, these rules are fixed by the rights holders and none of the current DRM systems allow for a change in these rules. However there are lots of cases where a change in the terms of the contract are desirable and maybe even necessary. Furthermore, because there is no interaction between the end user and the rights holder, the terms of the use license are usually the same for every end user. However, as discussed in [17], it would be advantageous to tailor the use licenses according to the end user – for example, if an end user is trusted by the rights holder, the rights holder is likely to give him/her less restrictive rights than an untrusted end user.

There are three requirements before negotiation can take place between the end user and the rights holder; there needs to be a standardised language to conduct negotiations, a Rights Expression Language (REL)[7] that is capable of expressing various templates of use licenses that the rights holder is willing to provide and a program good enough to conduct a negotiation and pick the correct use license.

12.2.1 Expressing Negotiations

Current REL standards can only express the rules from the rights holders perspective [18]. In this regard, in an earlier work [8], we discussed extensions to two common RELs – XrML and ODRL – that would allow for end users to communicate requests to the rights holder. The use of a REL to conduct negotiations is simpler than creating a separate language for conducting negotiations. This is because there will not be a need to convert the negotitaed contract from the "negotiation language" to the REL. Thus we believe, extending a current REL is a better option than creating a negotiation language.

In our scheme, we extended the standards with 3 requests from the user – to add additional rights, to remove existing rights and to replace existing rights with new rights. There are also two extetions for the rights holder – to grant a request and to deny a request.

12.2.2 Creating Templates

Before negotiations can take place, the rights holder must decide the terms of the licenses he/she is willing to allow. Thus the rights holder must set a number of different "*offers*"[8]. Ideally it would be useful to generate patterns for common offers, like templates for educators would be very similar in a consumer environment. In closed environments, the template could be generated around roles – for example templates in a corporation can be designed according to employee positions (managers, secretary etc). In the extreme case, templates could be generated around user identifiers – if the rights holder knows the specific requirements of certain users, then the templates can be generated along those lines.

On the other hand, in a consumer environment, the rights holder could require payment for certain rights. In such an environment, the rights holder can charge the user for the amount of flexibility the customer requires – for example, in a music store, the basic right could allow no mobility; but the customer could buy the mobility right for an additional charge, or maybe in place of another right.

As mentioned earlier, it would be desirable to express license templates with a REL. However, even

[7]A REL is a language used to express rights in use licenses. RELs are discussed in more detail in section 12.3

[8]This is a term from the ODRL REL. In ODRL, the rights holder offers a set of terms to the end user, which if the end user accepts generate a use license agreement.

though ODRL offers an “offer” syntax, we are not sure that any current REL has the syntax to express the options described above. This is discussed further in section 12.3.

12.2.3 Conducting Negotiations

Negotiations for licenses can take place at two points – when a new license is being generated, or when an end user wants to change the terms of an existing license. In the later case, the negotiations should continue from the existing license conditions.

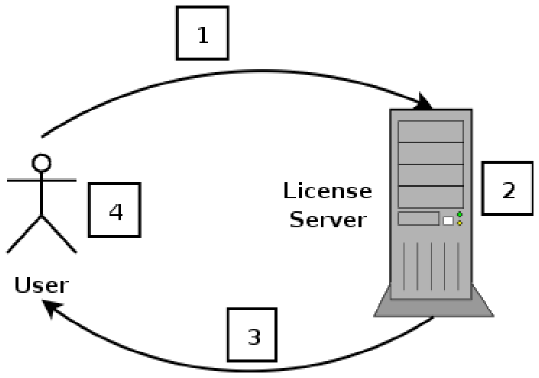

Figure 25: Negotiating for a use license

Negotiations essentially comprise a set of “requests” from the end user and responses from the license server on the requests. This process can be broken into 5 steps and as shown in figure 25 and detailed below.

Step 1: The user assembles the license request. The interface for the user could be the DRM controller component, or alternatively an online web site. As discussed earlier, ideally the request is created using the REL used for the use license.

Step 2: The license server analyses the license request and identifies one or more licenses that match close to the user’s request.

Step 3: The license server sends these as separate “offers” to the user.

Step 4: The user can then either choose a license or make additional requests, in which case Step 1 - 4 are repeated.

Step 5: Once the user has chosen an offer, the license server can generate the license matching the offer.

12.2.4 Generating Use Licenses

Once the terms of the license have been negotiated, generating the use license should be trivial as long as the negotiation language is the same as the REL. The use license must be signed by the license server. The use license should also include a key that can be used to decrypt the DRM package. This key should ideally be encrypted using the end user’s public key.

12.3 Expressing the Use License – Rights Expression Languages (RELs)

Rights expression languages (RELs) allow for the definition of the constraints, permissions etc. that the rights holder gives to the user, and thus is used to express use-licenses. RELs are one of the cornerstones of a DRM system and is seen as the main component that can be used to drive interoperability between different DRM systems [23].

The foundations of modern RELs were laid by Marc Stefik at Xerox Parc in the 1990s with Digital Property Rights Language (DPRL) [13]. DPRL was subsequently developed into eXtended rights Markup Language (XrML) by ContentGuard, a company founded by Microsoft and Xerox. Other languages like Open Digital Rights Language (ODRL) and Creative Commons (CC) have arisen to create open rights language specifications. Most modern RELs, like XrML and ODRL, are expressed in XML, using XML Schemas to define the syntax and grammar of the language; enabling portability across operating systems and platforms.

In the XrML 2.0 specifications [3], the requirements of a REL are given as:

- **Comprehensive:** A language that shall be capable of expressing simple and complex rights in any stage in a workflow, lifecycle or business model.
- **Generic:** A language shall be capable of describing rights for any type of digital content or service (an ebook, a file system, a video or a piece of software)
- **Precise:** a language shall communicate precise meaning to all players in the system.

Many of the criticisms of DRM systems stem from the current inability of RELs to express all the legal rights expected by the consumer [16, 4]. Felten has argued that some legal rights like fair use of copyrighted material, can never be expressed using RELs [16]. In [11] Bechtold countered that, while RELs cannot express fair use cases in general; they can do so in individual cases. However, as Mulligan et al. in [18] pointed out, current RELs do not have any syntax or grammatical capabilities to express communication between the end user and the rights holder. This limits the usage of RELs for contract negotiations, and thus current specifications of XrML, ODRL and other RELs are not comprehensive.

12.3.1 Feature Requirements for a REL

In addition to the requirements of a REL as discussed above and in [3], there are key features that should be provided by a REL. Below we list these features:

1. **Negotiation Support**: As discussed in section 12.2.1, it is very useful to have negotiation language the same as the REL. No current REL standard offer these constructs.
2. **License Templates**: As discussed in section 12.2.2, there is a need for rights holders to describe the terms they are willing to offer to the end user. ODRL does have an “offer” syntax, but we are not sure if the mechanisms offered in the current ODRL specifications allow the full range of requirements for license templates.
3. **Extensibility**: It is impossible to predetermine all the terms that could be used to express an use license. Thus it would be useful to allow mechanisms to add further terms to the base REL. Both XrML and ODRL have this feature.

12.3.2 The REL

Presently, the extended XrML and ODRL discussed in [8] are the only RELs we know of that support bi-directional communication. Furthermore ODRL is currently the only ODRL to have an "offer" construct. Thus for these reasons, an extended ODRL with bi-direction support is probably the most complete REL available in terms of the features we desire from a REL.

12.4 Transfer of Rights

The transfer of rights is not only a crucial cornerstone of many fair uses in copyright law, but also useful in other applications, but, currently no DRM system supports transfer of rights.

Transfer of rights allow users to pass the rights to access and use DRM protecetd data to another user temporarily or permanently. This could be a user selling their right to another person, or it could be a user lending their access rights to a friend or family. In corporate environments, this feature is particularly important – for example, when a person leaves the employment of a corporation, their access to protected data must not only be removed, but should also be transfered to another person.

The main problem obstacle to transfer of rights in a digital environment, is that unlike the physical world, creating perfect digital replicas of the object in question is easy. Thus even if the right is transfered permanently, the user would most likely retain copies of the "originals".

12.4.1 Permanent Transfer of Rights

Permanent tranfer of rights is relatively simple to handle if done through a third party. The idea is simple – the third party, the license server, issues a new license to the new users with the current terms and conditions, and invalidates the original license, using the revocation list. Thus, once the original license has been revoked, the first user can no longer access the DRM protected work. For this to work however, it must be assumed that all users update their revocation lists regularly.

12.4.2 Temporary Transfer of Rights

Unlike permanent transfer of rights, in a temporary transfer of rights, the use license cannot be permanently invalidated. Thus after the lending period has passed, the rights must revert back to the original user. Thus if the revocation list is maintained on a daily basis, this would work quite well.

12.5 Scenario Analysis

Scenarios 1 and 3 do not really need negotiations support, and thus the use of templates is not that important. However, bi-directional support is still required – in scenario 1, a lecturer may want more time to review the paper for example. Scenario 2 does require the use of templates and negotiation support. Since negotiations can be used to support fair use in DRM, there is a high chance that a negotiations system will be utilised.

13 Payment Gateway

The Payment Gateway represents a service that can process payments. We do not intend specifying how such a service is implemented, rather just a protocol that would allow the user to pay various Web Services for services rendered without requiring the Web Services themselves to process the transaction. The accounting for the services however need to be handled by the Web Services themselves. This system would also potentially allow for any type of online payment service including Paypal, credit cards and even debit cards.

To ensure privacy, the Payment Gateway does not require to know the details of what is being paid for – just how much is being paid. Similarly, the Web Service does not require to know how the user is paying for the services, just that the service was purchased successfully. Common payment gateways could be very useful for general Web Services, as small businesses could pool together for a common payment gateway that can handle multiple payment mechanisms.

13.1 Invoices

Web Services should create an invoice for the user after a user purchases a service. The details of the invoice can be left to the Web Services themselves, but must contain an invoice identifier (from an UNI), the ammount, the identifier of the service and the identifier of the user. Invoices must have a digital signature to maintain integrity. A XML schema is shown in figure 26.

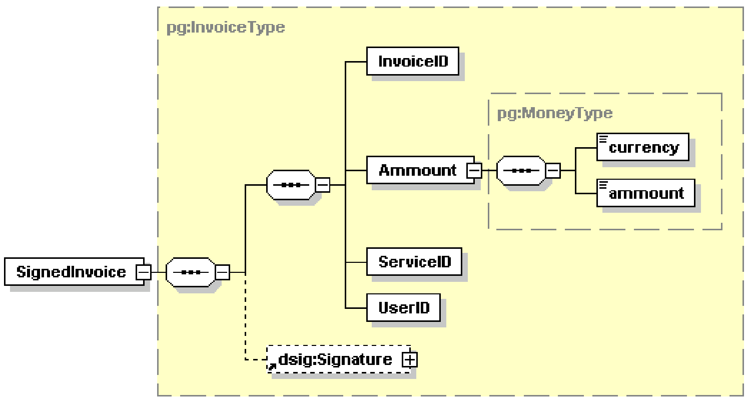

Figure 26: XML schema for an Invoice

13.2 Receipts

The user uses the invoice's identifier to make a payment. The mandatory details detailed above are required by the payment gateway. After a successful transaction, the payment gateway creates two receipts – one for the user and one for the service. The receipt must contain the identifier of the user, the amount paid, the identifier of the service that the payment is for, the identifier of the payment gateway, the identifier of the invoice (to reconcile an invoice and the payment) and the identifier of the receipt. The receipt must be signed to maintain integrity. A XML schema is shown in figure 27.

13.3 Scenario Analysis

The payment gateway is required only for scenario 2 and provides an easy interface for the users to prove that they have paid for their music. The concept of a payment gateway is also useful to the

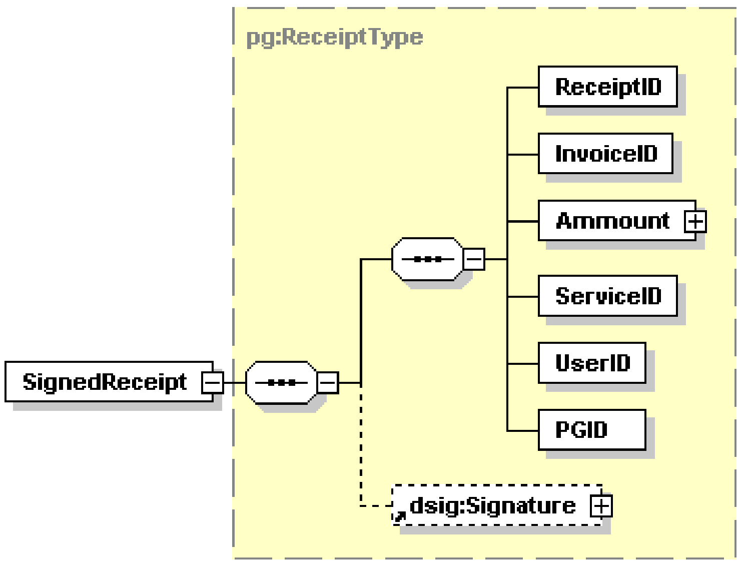

Figure 27: XML schema for a Receipt

vendors – a number of different services can make use of one payment gateway for their financial transactions. Thus the payment gateway can potentially have a larger role to play in the broader e-commerce landscape.

14 Requirement Analysis

In section 1.2, we outlined the requirements of a DRM system. In this section, we examine if these requirements are addressed in our system.

1. **Persistent protection:** As in current DRM systems, protection is provided through the use of encryption. The separation of the use license and the data allows for any manner of distribution; and we believe that the requirements of persistent protection has been met.
2. **Portability:** Four types of portability were discussed in seciotn 1.2:
   (a) **Space Shifting:** Because there is a separation of the use license and the data, space shifting is easier to achieve. However, implementations could chose to restrict this, by tying down the machines that could be used to access the protected data in the use license.
   (b) **Time Shifting:** Depends on the restrictions of the use license, but there is no restrictions in the framework to time shifting.
   (c) **Format Shifting:** Because the DRM controller can be at an operating system or hardware level, it can allow programs that promote format shifting to operate, as long as the use license grants this permission.
   (d) **Platform Shifting:** The design of the system is geared towards portability between different operating systems and platforms. As long as two systems have compatible DRM controllers[9], the user should be able to access the DRM package.
3. **Integration with existing applications:** If the DRM controller is implemented at an operating system or hardware level, the application would not even know that the file is protected. However, some complex protections would probably require application level support.
4. **Excerpting:** If the DRM controller is implemented at an operating system or hardware level, and the use license allows excerpting, any application that allows excerpting can be used to excerpt.
5. **Transfer of Rights:** In section 12.4, we detailed mechanisms that can be used to transfer rights. Independent transfer of rights without the use of a TTP is however not solved.
6. **Allow for changes to access and usage rights after distribution:** The separation of the data and the use license allows for the changes in license terms.
7. **Privacy vs. Usage Tracking:** User privacy does depend on the use licenses, but our system can operate in an offline environment (as long as the use license allows it). The use of receipt and invoice tokens also allow for privacy of data between the various services – the license server does not need to know the user's credit card details and the payment gateway does not need to know about the user's buying habits.

   If the use license requires an online connected environment, this can be accommodated. However the exact user actions when they are accessing the data are not necessarily monitored[10].
8. **Offline Usage:** Offline usage is catered for as long as the use license allows it. Some online connectivity is required however to get the use license.

[9]Compatibility depends on whether the DRM controller can interpret and execute the restrictions. For example, a DRM controller for a DVD player is unlikely to be compatible with a e-book reader. Compatibility could also depend on auxiliary services, for example authentication to a Novell Netware account may not be implemented by all DRM systems.

[10]This would depend on the DRM controller implementation

9. **Easy identification:** The UNI provides easy identification for all the players and objects in the system. Furthermore, the numbering scheme also allows for non-UNI generated globally unique identifiers.

10. **Easy Verification:** Verification and integrity is provided through the use of digital signatures on all documents and tokens. The use of signed receipt tokens also allow the user to prove their legitimate access as required by Bartolini et al. in [10].

11. **Correctly collect revenue for right holders:** The payment gateway allows users to pay for their services. The distribution of the collected payments depnds on the payment gateway.

15 Scenario Analysis

In section 2, we presented three scenarios where the DRM system can be utilised. In this section, we review how the system addresses the needs presented in the scenarios.

15.1 Scenario A: Exam Papers

The DRM system in this scenario needed to handle different authentication systems. The full time lecturers and the teaching assistant would most likely use the authentication system used to access university computer facilities, while the printers, the part time lecturers and the external examiner would not necessarily have such accounts. This is supported in our system.

The terms of the use license need to be limiting, and maybe even make use of short term licenses that expire after a day or even few hours. The UNI server is best deployed at the university level, but the service producer and the license server can be deployed at departmental levels.

15.2 Scenario B: Selling Music

The DRM system in this scenario requires to handle portability and fair uses. The license server in our system meets these requirements – negotiation of use license terms allow for end user to request fair uses, as well as preventing misuse of fair use. The use of a payment gateway allows for revenue collection, and can be integrated with an online store quite easily.

15.3 Scenario C: Patient Data in a Hospital Group

The main challenge in this scenario is to protect patient data from irregular access, but at the same time allow access to the data during emergencies. Thus authentication is the most crucial element in this system. While biometrics would be a solution, this is impractical in a medical emergency. But, a token based system is still the most secure authentication system that could be used.

Many people aleady make use of physical tokens for emergency purposes – bracelets that have the wearer's blood type and allergies for example. This could be extended to be used in a DRM system, though having a codeword engraved on a bracelet is definitely insecure. Usage of mobile TPMs could solve this problem, and could be the ideal solution. However, all token based approaches have the same problem – the patient must have the token near him/her for it to be effective. Ultimately, the use of DNA based biometrics is probably the most secure approach, but the priavcy questions associated with such approaches have yet to be solved.

16 Conclusion

In this document, we have laid out the specifications of a componentised DRM system. We have detailed the components that would make up a complete DRM system, how these components interact as well as how they satisfy the requirements for a generalised DRM system. We have also detailed three different scenarios discussing how our systems can help solve the problem.

17 Acknowledgements

This work is partially supported through grants from the UCT Council and the National Research Foundation (NRF) of South Africa. Any opinions, findings, and conclusions or recommendations expressed in this paper/report are those of the author(s) and do not necessarily reflect the views of UCT, the NRF or the trustees of the UCT Council.

XML Schema diagrams produced using XMLSpy 2004.

References

[1] The digital object identifier system – frequently asked questions.
URL: http://www.doi.org/faq.html.

[2] Copyright act 98 of 1978. 2000 ed., vol. 2 of *Statutes of South Africa.* Juta, 2001, pp. 2–214–2–234. As ammended by (acts/year): 56/1980, 66/1983, 52/1984, 39/1986, 13/1988, 61/1989, 125/1992, 38/1997, 9/2002.

[3] *eXtensible rights Markup Language (XrML) 2.0 Specification*, 2001.

[4] DRM From the Viewpoint of the Electronic Industry. *Slashdot* (2003).
URL: http://slashdot.org/article.pl?sid=03/11/25/1821218.

[5] The estonian id card and digital signature concept – principles and solutions. White paper, AS Sertifotseerimiskeskus, Estonia, 2003.
URL: http://www.id.ee/file.php?id=122.

[6] Linux and DRM? *Slashdot* (2004).
URL: http://ask.slashdot.org/article.pl? sid=04/02/10/2329229&mode=thread.

[7] Arnab, A., and Hutchison, A. Digital Rights Management - An overview of Current Challenges and Solutions. In *Proceedings of Information Security South Africa (ISSA) Conference 2004* (2004).

[8] Arnab, A., and Hutchison, A. Extending ODRL and XrML to enable Bi-directional communication. Departmental Technical Report, No. CS04-28-00, University of Cape Town, 2004.

[9] Arnab, A., and Hutchison, A. Security Considerations for an Idealised DRM Framework. In *Proceedings of the South African Telecomunication Networks and Applications (SATNAC) Conference 2004* (2004).

[10] Bartolini, F., Cappellini, Piva, A., Fringuelli, A., and M, B. Electronic Copyright Management Systems: Requirements, Players and Technologies. In *Proceedings of the Tenth International Workshop on Database and Expert Systems Applications* (1999), IEEE, pp. 896–899.

[11] Bechtold, S. Digital Rights Management in the United States and Europe. IVir, Buma/Stemra - Copyright and the Music Industry: Digital Dilemmas.

[12] Brickell, E., Camenisch, J., and Chen, L. Direct anonymous attestation. In *Proceedings of the 11th ACM Conference on Computer and Communications Security* (2004), B. Pfitzmann and P. Liu, Eds., ACM, pp. 132 – 145.

[13] Coyle, K. Right Expression Languages, A report for the Library of Congress. Tech. rep., Library of Congress, USA, 2004.

[14] de Mes, A., and Rongen, E. Technical note - web service credentials. *IBM Systems Journal 42*, 3 (2003).

[15] Ellison, C. SPKI/SDSI Certificate Documentation [Carl Ellison], 2004.
URL:http://world.std.com/ cme/html/spki.html.

[16] Felten, E. Skeptical view of DRM and Fair Use. *Communications of the ACM 46*, 4 (2003), 57–59.

[17] Jamkhedkar, P. A., and Heileman, G. L. DRM as a Layered System. In *Proceedings of the Fourth ACM Workshop on Digital Rights Management* (2004), A. Kiayias and M. Yung, Eds., ACM, pp. 11 – 21.

[18] Mulligan, D., and Burstein, A. Implementing Copyright Limitations in Right Expression Languages. In *Proceedings of the 2002 ACM workshop on Digital Rights Management* (2002), ACM.

[19] Mulligan, D., Han, J., and Burstein, A. How DRM Based Content Delivery Systems Disrupt Expectations of "Personal Use". In *Proceedings of the 2003 ACM workshop on Digital Rights Management* (2003), ACM, pp. 77–89.
URL: http://doi.acm.org/10.1145/947380.947391.

[20] Park, J., Sandhu, R., and Schifalacqua, J. Security architectures for controlled digital information dissemination. In *Proceedings of the 16th Annual Computer Security Applications Conference* (2000).

[21] Paskin, N. *The DOI Handbook*, 4.0.0 ed. International DOI Foundation, 2004.

[22] Rosenblatt, B. Solving the dilema of copyright protection online. *The Journal of Electronic Publishing 3*, 2 (1997).
URL: http://www.press.umich.edu/jep/03-02/doi.html.

[23] Rosenblatt, B. DRM for the Enterprise, 2004.

[24] Rosenblatt, B., and Dykstra, G. Integrating content management with digital rights management - imperatives and opportunities for digital content lifecycles. White paper, Giantsteps Media Technology Strategies, 2003.
URL: http://www.giantstepsmts.com/drm-cm_white_paper.htm.

[25] Shirey, R. Rfc 2828 – internet security glossary, 2000.
URL: http://www.faqs.org/rfcs/rfc2828.html.

[26] Stallings, W. *Protect your Privacy – A guide for PGP users*, first ed. Prentice Hall, 1995.

[27] Stallings, W. *Network Security Essentials – Applications and Standards*, international second ed. Prentice Hall, 2003.

[28] W3C. *XML Encryption Syntax and Processing*, 2002-12-10 ed., 2002.
URL: http://www.w3c.org/TR/2002/REC-xmlenc-core-20021210.

[29] W3C. *XML-Signature Syntax and Processing*, 2002-02-12 ed., 2002.
URL: http://www.w3c.org/TR/2002/REC-xmldsig-core-20020212.