DRM convergence

interoperability between DRM systems

Asghar, M.R.

Award date:
2009

Eindhoven University of Technology

Department of Mathematics and Computer Science

Master's Thesis

DRM Convergence: Interoperability between DRM Systems

by

Muhammad Rizwan Asghar

Tutor: Dr. Daniel Catrein (Ericsson)
Supervisor: Dr. Jerry Den Hartog (TU/e)

Eindhoven, August 2009

Abstract

The vision of the multimedia world is that digital content can be freely shared and moved between all kinds of devices. This would be possible for both, unprotected and protected contents. Currently, various *Digital Rights Management (DRM)* systems are used to control access on the protected content. The most famous examples include Apple Fairplay, Windows Media DRM, *Open Mobile Alliance (OMA)* DRM, and Marlin DRM. The protected content format, license management, and trust management differs from one DRM system to another. Therefore, it is necessary to mediate those differences to make DRM systems interoperable. The aim of this thesis is to investigate the approaches for DRM interoperability.

Each DRM system has its own *Public Key Infrastructure (PKI)* domain to manage the trust between different entities like devices and license servers. However, if the entities belong to two similar DRM systems but each having different trust anchor then there is a need of trust establishment that can be achieved through *Interoperability in PKI*.

After analysis of the requirements and studying existing solutions like OMArlin and Coral, a new concept of *Generic Interoperability Solution* is proposed to achieve the interoperability between different DRM systems. The proposed solution does not require changes to the implementations at the device end. Instead, it introduces a new entity named *Content Format Translation (CFT)* that communicates with license servers in both DRM systems. A CFT may be a Web service or a device in the home environment to transfer the translated content from one device to another. Moreover, the proposed solution also requires the interaction between both license servers to move the rights from one DRM system to another. This interaction is accomplished through interoperability in PKI.

Dedicated to
My parents and brothers for their utmost care, endless love, and support throughout the course of my studies.

Acknowledgements

First of all, I am extremely thankful to Dr. Daniel Catrein for giving me an opportunity to carry out the research at Ericsson Research Aachen, Germany and his excellent guidance and comments throughout the thesis. I am also very thankful to Dr. Jerry den Hartog from Eindhoven University of Technology for his vital input to supervise this thesis.

I would like to thank my second supervisor Dr. Frank Hartung from Ericsson for his support in the initial phase of this thesis. Furthermore, thanks go to Dr. Dmitri Jarnikov from Eindhoven University of Technology for taking part in my assessment committee.

Finally, a special thank to my family, friends, and colleagues at Ericsson for their utmost support and motivation to achieve my goal.

Muhammad Rizwan Asghar
Eindhoven, The Netherlands
August 2009

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Problem Description . . . 1
- 1.3 Approaches to Achieve DRM Interoperability . . . 2
- 1.4 Thesis Goal and Contribution . . . 3
- 1.5 Thesis Organization . . . 3

2 Overview of DRM Systems **5**
- 2.1 Introduction . . . 5
- 2.2 A DRM System . . . 5
  - 2.2.1 What is Digital Rights Management . . . 5
  - 2.2.2 The Fundamental Principle of a DRM System . . . 7
  - 2.2.3 DRM as an End-to-end Security System . . . 8
- 2.3 DRM Components and Functions . . . 9
  - 2.3.1 Content Delivery Server . . . 10
  - 2.3.2 DRM Content Format . . . 10
  - 2.3.3 License Server . . . 11
  - 2.3.4 DRM License . . . 11
  - 2.3.5 DRM Client Device . . . 13
- 2.4 Trust Model . . . 14
  - 2.4.1 What is Trust Model . . . 14
  - 2.4.2 Why a Trust Model . . . 14
  - 2.4.3 Components of a Trust Model . . . 15
- 2.5 Basic Use Cases . . . 16
  - 2.5.1 Basic License Acquisition . . . 16
  - 2.5.2 Basic Content Transfer . . . 17
  - 2.5.3 Basic Stream Transfer . . . 17
- 2.6 Standardized DRM Systems . . . 18
  - 2.6.1 OMA DRM . . . 19
  - 2.6.2 Marlin . . . 20
- 2.7 Proprietary DRM Systems . . . 21
  - 2.7.1 Microsoft DRM . . . 22
  - 2.7.2 Apple Fairplay . . . 22

3 Interoperability in PKI **23**
3.1 Introduction . . . 23
3.2 Cross-Certification Approach . . . 25
3.2.1 Advantages . . . 27
3.2.2 Disadvantages . . . 28
3.3 Super CA Approach . . . 28
3.3.1 Advantages . . . 29
3.3.2 Disadvantages . . . 29
3.4 Proxy Server Approach . . . 29
3.4.1 Advantages . . . 31
3.4.2 Disadvantages . . . 31
3.5 Interoperability between two OMA DRM v2.1 Systems . . . 31
3.5.1 ROAP and CMLA PKI System . . . 31
3.5.2 Applying Cross-Certification Approach . . . 33
3.5.3 Applying Super CA Approach . . . 36
3.5.4 Applying Proxy Server Approach . . . 38
3.6 Synopsis . . . 40

4 DRM Interoperability Solution **41**
4.1 Introduction . . . 41
4.2 Requirements on an Interoperability Solution . . . 41
4.2.1 Security Aspects . . . 42
4.2.2 User Acceptance . . . 42
4.2.3 Content Provider Acceptance . . . 42
4.2.4 Supported Use Cases . . . 43
4.2.5 Cost . . . 43
4.2.6 Extendibility . . . 43
4.2.7 Performance Scalability . . . 43
4.2.8 Completeness . . . 44
4.2.9 Implementation Complexity . . . 44
4.3 Existing Interoperability Solutions . . . 44
4.3.1 OMArlin . . . 44
4.3.2 Coral and DECE . . . 45
4.4 Drawbacks in Existing Solutions . . . 47
4.4.1 OMArlin Limitations . . . 47
4.4.2 Coral Limitations . . . 48
4.5 The Proposed Interoperability Solution . . . 49
4.5.1 Components and Functions . . . 50
4.5.2 Sequence of Messages and their Description . . . 54
4.5.3 Formal Model . . . 58
4.5.4 Possible Optimizations in the Proposed Solution . . . 65
4.5.5 Streaming Case . . . 66
4.6 Evaluation of the Proposal . . . 67
4.6.1 Security Aspects . . . 67

4.6.2 User Acceptance 69
4.6.3 Content Provider Acceptance 69
4.6.4 Supported Use Cases 70
4.6.5 Cost 70
4.6.6 Extendibility 70
4.6.7 Performance Scalability 70
4.6.8 Completeness 71
4.6.9 Implementation Complexity 71
4.7 Interoperability between OMA DRM v2.1 and Marlin DRM . 71
4.8 Synopsis 72

5 Conclusions **75**
5.1 Introduction 75
5.2 Summary 75
5.3 Conclusions and Recommendations 76
5.4 Future Research 77

Bibliography **84**

A Rights Expression in ODRL **85**
A.1 ODRL Permission and Constraint Sets 85
A.2 ODRL Example 86

B Rights Object **89**

C The 4-Pass Registration Protocol **91**
C.1 Device Hello 92
C.2 RI Hello 92
C.3 Registration Request 93
C.4 Registration Response 94

D RO Acquisition Protocol **95**
D.1 RO Request 96
D.2 RO Response 96

List of Figures **100**

List of Tables **101**

Glossary **104**

Chapter 1

Introduction

1.1 Motivation

The digital content industry is using DRM systems to regulate how digital content can be consumed. A number of DRM systems are being used to provide content protection for electronic products ranging from portable to network devices. For example, portable music players support Microsoft Windows Media DRM [Micb] or Apple Fairplay [Ano07] while mobile phones adopt OMA DRM [OMA]. Consumers want to easily share the digital content amongst different devices in their home environments while content providers are interested to offer new business models (for example, subscription based) by using various types of DRM technologies [Wan05]. Currently, content providers are using different DRM systems which are neither standardized nor interoperable with each other. As a result, consumers often find that they cannot share digital content between a number of devices [TCG06].

A survey by INDICARE [IND05] shows that European consumers are willing to pay a higher price for the content that plays on any device rather than just on a single device. With DRM interoperability, content providers can target significantly a large number of audience. The lack of interoperability is considered as unfair to customers and it prevents true competition between different music services [Ars07]. In short, in order to share the digital content between different kinds of devices, it is necessary to have DRM interoperability.

1.2 Problem Description

DRM interoperability problem occurs on three layers:

Protected Content Format There is no unique standardized format to protect the content. Each DRM system uses its own protected content format. For example, OMA DRM uses *DRM Content Format (DCF)* while Windows Media DRM uses *Widows Media Format (WMF)*. Moreover, the content is encrypted with a symmetric key algorithm such as *Advanced Encryption Standard (AES)* or *Data Encryption Standard (DES)*. Each DRM system has freedom to use any symmetric key algorithm with any mode of operation and can apply that in multiple ways to encrypt the content. It is important for an interoperability solution how to resolve this issue.

License Management License format[1] and management varies in different DRM systems. Moreover, every DRM system has its own *Rights Expression Language (REL)* to express the rights granted to play the content. *Open Digital Rights Language (ODRL)* [Ian01] and *eXtensible Rights Markup Language (XrML)* [Gua02] are two examples of RELs. A DRM interoperability solution is responsible to convert the license (including the rights) from one format to another.

Trust Management Every DRM system has its own trust model. For example, OMA DRM v2.0 (or later version) typically follows *Content Management License Administrator (CMLA[2])* trust model while Marlin follows *Marlin Trust Management Organization (MTMO)* trust model. A trust model defines the compliance and robustness rules and specifies cryptographic trust anchors, that is, typically who operates root *Certification Authority (CA)*.

1.3 Approaches to Achieve DRM Interoperability

A variety of DRM interoperability approaches have been proposed. Generally, there are the following approaches to achieve DRM interoperability [KLMM03, PJ07, Con]:

One Standardized DRM System The most obvious solution is to develop a standard DRM system so that all devices can implement it.

Currently, the content providers have implemented different DRM systems and it is very unlikely that they could change this situation by shifting to a standard DRM system. The reason is that this shift would require high investments to adopt new technology.

Devices Support Multiple DRM Systems Another approach is to implement multiple DRM systems inside a device.

[1] For example, *Rights Object (RO)* is a license format in OMA DRM system.

[2] `http://www.cm-la.com/`

Since a device has limited storage and processing capabilities, this is not welcomed by *Consumer Electronics (CE)* industry. Moreover, it implies an additional licensing, implementation, and testing cost.

Content Providers Support Multiple DRM Systems A content provider may support multiple DRM systems and can deliver the content and corresponding license in all required formats simultaneously.

This approach requires the number of DRM systems times as much bandwidth. Therefore, it may not be a profitable business for a content provider to follow this approach. Moreover, the users still cannot share the content between their devices.

Translators One approach is to translate the formats of the protected content and the license while sharing the content from one DRM system to another.

This approach has the potential to be economically sustainable and provide benefits to all participants [KM05].

All approaches described above look at different aspects of the problem and make different trade-offs when it comes to security issues, reliance on networking, and implementation complexities.

1.4 Thesis Goal and Contribution

As a first step, this thesis addresses the interoperability in PKI to resolve the interoperability between similar DRM systems each having different trust anchor and extends the idea to achieve the goal. The goal of this thesis is to investigate the existing interoperability solutions and propose a user friendly solution which ideally does not require any change at the device end, supports sharing of both content downloading and streaming use cases, and can be applied to as many DRM systems as possible. This thesis focuses on translator approach because it has the potential to be economically sustainable and provide benefits to all participants. Simply implementing a translator is not enough because it is also important how to manage the trust across a DRM system. Therefore, this thesis also addresses the trust management issues across multiple DRM systems.

1.5 Thesis Organization

The remainder of this thesis is organized as follows: Chapter 2 provides background information about a DRM system and gives overview of different DRM systems. Chapter 3 highlights the importance of interoperability in PKI and looks at three different approaches to resolve the interoperability

between similar DRM systems each having different trust anchor. Chapter 4 lists the requirements on an interoperability solution. It explains two existing interoperability solutions and describes their problems. Furthermore, it proposes a solution to resolve the interoperability between different DRM systems and evaluates the proposed solution. Finally, Chapter 5 concludes this thesis with some recommendations and directions for further research.

Chapter 2

Overview of DRM Systems

2.1 Introduction

This chapter provides an overview of DRM systems. This chapter is organized as follows: Section 2.2 briefly explains a DRM system by considering its abstract level view. It highlights the fundamental principle of a DRM system. It also addresses how a DRM system provides end-to-end security. Section 2.3 describes core components and functions of a DRM system. Section 2.4 introduces the trust model and discusses its role in a DRM system. Section 2.5 looks at basic use cases to evaluate the different existing DRM systems and the options for convergence. Finally, Section 2.6 and Section 2.7 provide examples of *Standardized* and *Proprietary* DRM systems, respectively.

2.2 A DRM System

2.2.1 What is Digital Rights Management

According to [ZYL06], DRM refers to technologies that support legal distribution of the digital contents while protecting appropriate property rights imposed by a content provider. DRM is more than a technology and can function only in a legal framework that includes legislation, conformance, and enforcement [PJ07].

Figure 2.1 shows an abstract level view of a DRM system. A *Content Delivery Server* is a server responsible to distribute the *Encrypted Content*. A content delivery server is discussed in more detail in Subsection 2.3.1. A *DRM Client Device* is an entity that can play the content. It obtains the encrypted content from a content delivery server. The content is encrypted with a cryptographic key called *Content Encryption Key (CEK)*. A sym-

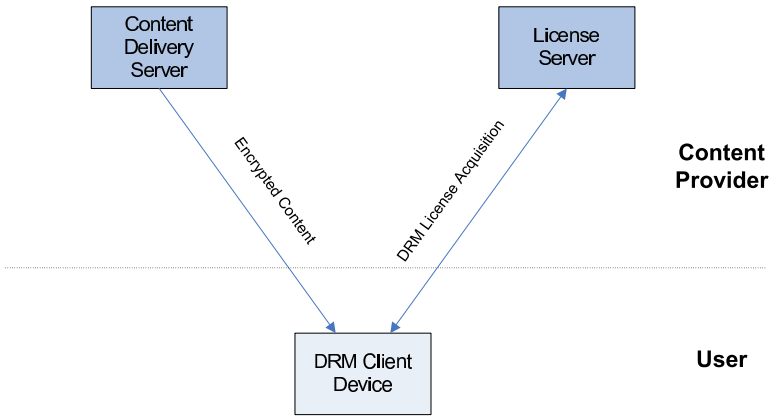

Figure 2.1: Abstract level view of a DRM system

metric encryption algorithm, for example AES or 3DES, is used to encrypt the content. The encrypted content format is discussed in more detail in Subsection 2.3.2. A *License Server* is responsible to issue *DRM License* to the requesting entity. A license server is discussed in more detail in Subsection 2.3.3. The dotted line in Figure 2.1 separates a content provider and a user. A content provider manages a content delivery server and a license server.

A DRM license contains the rights[1] and the key, that is CEK, to decrypt the encrypted content. A DRM license is discussed in more detail in Subsection 2.3.4. A DRM client device is responsible to enforce the rights mentioned in the license. A DRM client device is discussed in more detail in Subsection 2.3.5.

A DRM client device acquires the license from a license server. Before issuing a license to the client, a license server needs to ensure that:

- The rights will be enforced by the client.
- The client will expose the key or the decrypted content only to the allowed components.

In short, a client must be trusted. A client is considered to be trusted if, and only if it conforms to the compliance and robustness rules defined by a trusted entity. A server only authenticates the trusted clients. Just like a server, a client also needs to ensure that the content is issued by a valid server supposed to be trusted. The main reason behind server-to-client authentication is to avoid *White-washing*. This is a legal framework regulation to prevent the content piracy. This regulation ensures that the content delivery servers sell what they own. In this way, a device can trust

[1]For example, the right to play a file three times and the right to burn a file to an audio CD.

that the content is legal. Therefore, a DRM system requires mutual trust establishment mechanism between the client and the server. A trust model plays a key role to perform such sort of trust establishment. Even though DRM is not only technology but this report mainly focuses on the technology part. The trust model captures much of the legislation, conformance, and enforcement on (the application of) this technology. The trust model is discussed in more detail in Section 2.4.

A DRM system enables a content provider to control the content. For example, a content provider can limit the number of copies, play counts, and the devices on which the content can be transferred to. DRM technology is used in a number of media, but most commonly it is found in video and audio music files. Many online music stores, such as iTunes[2], and other major brands throughout the entertainment industry are using this technology to protect their audio and video files.

2.2.2 The Fundamental Principle of a DRM System

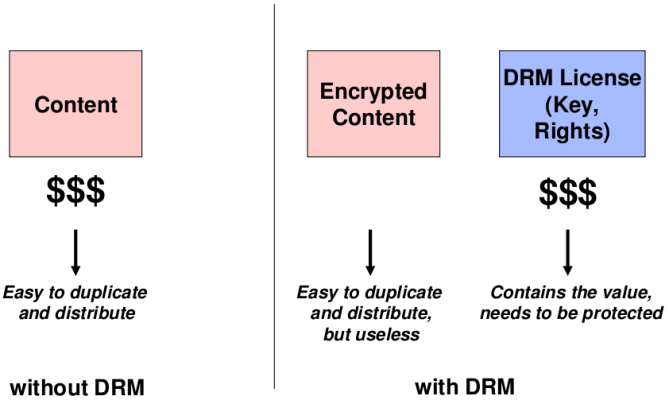

Figure 2.2: The fundamental principle of a DRM system [Har04]

The fundamental principle of a DRM system is separation of the content and the rights. As illustrated in Figure 2.2, without DRM, the content is the asset. While in a DRM system, a license is the asset and is typically protected by public key of the requesting entity. A license contains the rights and a key. The rights define how the content can be used while the key (that is, CEK) is used to decrypt the encrypted content.

The encrypted content and corresponding license can be sent, received, or requested together or separately. This feature provides applicability for a range of different business models. For example, *Super-distribution*, where users can send the encrypted content to each other as they like. However,

[2] `http://www.apple.com/itunes/`

the content can only be accessed with a valid license. Hence, the user needs to acquire a license from the license server.

2.2.3 DRM as an End-to-end Security System

A DRM system provides end-to-end security. That is, all the way from the content provider over the communication channel to the end user device.

The key ingredients, used to accomplish this function, are:

- *CEK:* It is generated by a license server.
- *PKC: Public Key Certificate (PKC)* is used for the purpose of authentication. PKCs are managed by a PKI.
- *Compliance and robustness rules:* These rules are specified by a trusted entity in a trust model. For information about the trust model, see Section 2.4.

How a DRM system protects content at different levels, is explained below:

Content and License Protection on Content Provider

A content provider manages a content delivery server and a license server. Physical security policies are applied for their protection. Both servers conform to the compliance and robustness rules defined by a trusted entity in the trust model (to know further about a trust model, see Section 2.4). Therefore, a content provider does not leak the content or the keys.

Content Protection on Communication Channel

The content is usually encrypted with CEK using symmetric encryption algorithm. For example, 3DES or AES. The encrypted content is sent to the device via a communication channel. A DRM system assumes that state of the art symmetric encryption algorithms are secure enough to provide confidentiality to the content. Hence, the content is always under protection on the communication channel.

License Protection on Communication Channel

The license plays an important role to provide the content protection. A license is sent to the requesting entity during license acquisition phase. The license contains the secret information, like CEK. Therefore, the license needs to be protected on the communication channel. Usually, the parts of a license are signed and encrypted with public key of the requesting entity. Hence, the license is protected on the communication channel.

Content and License Protection on End User Device

In addition to the protection during the communication phase, the content and the license need to be protected on the end user device. A DRM system provides such sort of protection by issuing the license only to the devices conforming to the compliance and robustness rules. A device obtains a certificate from a trust authority if it conforms to the compliance and robustness rules defined by a trusted entity in a trust model. This certificate proves to the server that the device is trusted and it will not leak the content and the license information. Hence, the content and the license are protected on the end user device.

2.3 DRM Components and Functions

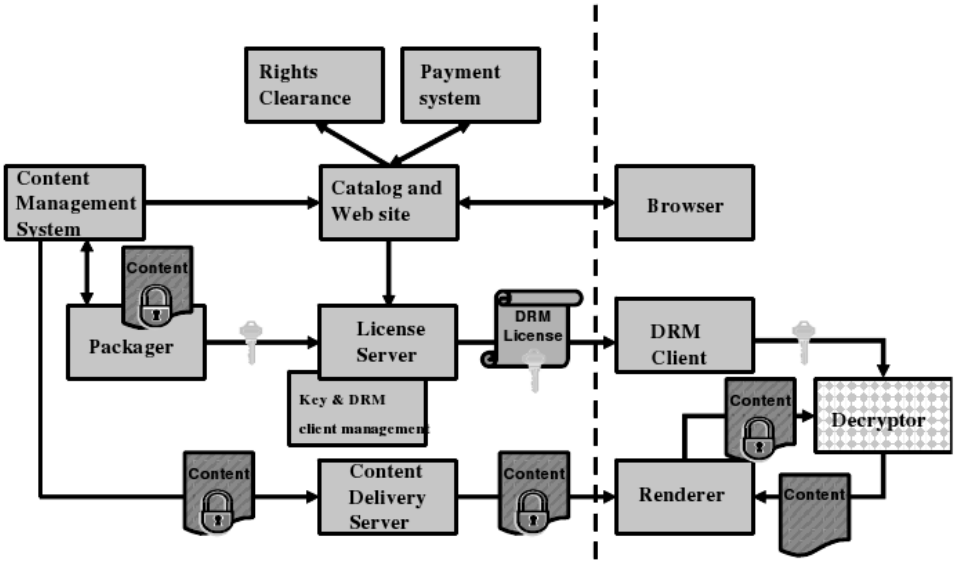

Figure 2.3: A generic DRM architecture [PJ07]

Figure 2.3 illustrates a generic DRM architecture with additional components which include payment system, catalog and Web site, and browser. The dotted lines separates the server side and the client side components which are on the left and the right side of Figure 2.3, respectively. The content is pre-encrypted with a CEK and packaged through the packager; and distributed by a content delivery server. In a typical scenario, a user interacts with the browser to select a specific content item from the catalog. The user obtains the encrypted content directly or indirectly[3] from a content delivery server.

[3]In case of super-distribution

When a specific content item is purchased, which is handled by a payment system, the license server is instructed to prepare a license for the requested item. The CEK is sent to the license server. A license server is responsible to generate the license which includes the rights, a CEK, and optionally a *Message Authentication Code (MAC)* key used to protect the integrity of the content. The license is usually encrypted with public key of the requesting entity. The delivery manager and the license manager send the encrypted content and the license, respectively, to the client. The client uses its private key to decrypt the license and extracts the CEK. Then the decryptor uses that CEK to decrypt the encrypted content. Finally, the content is rendered by the renderer and is played by the device.

The core components of a DRM system are described below. For further information about DRM components, an interested reader is referred to [PJ07].

2.3.1 Content Delivery Server

The content delivery server is responsible to distribute the encrypted content and is managed by a content provider.

2.3.2 DRM Content Format

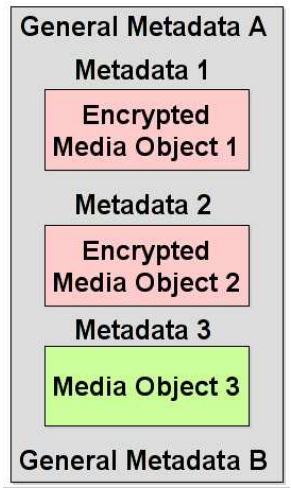

Figure 2.4: DRM content format [Har04]

Typically, a media object is a basic building block of a DRM content format. A DRM content format may contain multiple media objects. A media object may be protected (that is, encrypted media object) or unprotected (that is, media object). Figure 2.4 is an example containing two pieces of protected and one piece of unprotected content items.

At the beginning of a DRM content format, there is *General Metadata A*. It identifies and describes the content. It typically contains:

- *Content ID:* A unique identification number to identify the content
- *URL:* The download location of this content
- *License Acquisition URL:* The location to download a license
- *Cryptographic Parameters:* It includes the parameters[4] such as content encryption scheme and mode, information about *Initial Vector (IV)*, block length, and information about the padding

After general metadata A, several protected or unprotected media objects with their own metadta may be included in the DRM container. The media metadata identifies and describes the media object. It typically contains:

- *Media Type:* MIME type
- *Content Information:* Description of the content, for example the title and information URL

DRM container may end with *General Metadata B*. It holds the validation information. It typically contains, for example a hash or MAC of (the rest of the information in) the container object.

2.3.3 License Server

A license server is responsible to issue the license to the requesting entity. Before issuing a license, a license server needs to check whether the client is trusted or not. A client is considered to be trusted if it has a valid certificate (that is, PKC) from a trust authority. The client sends this certificate in the license request during license acquisition phase. A license server validates the client certificate. On successful validation, a license server issues a license (for further information about DRM licenses, see Subsection 2.3.4) encrypted with client public key (obtained from the client certificate).

2.3.4 DRM License

A DRM license is based on XML or binary format. Usually, it is generated individually for every purchase request. It contains metadata, a CEK, and the rights. Typically, the metadata holds the following information:

- *License ID:* A unique identification number to identify the license
- *Content ID:* A unique identification number to identify the content

[4]For more information about cryptographic parameters, see [All08b]

- *Content Provider ID:* A unique identification number to identify a content provider
- *License Acquisition URL:* The location to download a license
- *Cryptographic Parameters:* (Where needed/applicable)
- *Subscription/Purchase Information*
- *Hash of DRM Container:* (Optional)
- *Hash or MAC of DRM License*
- *Signature*

The CEK is a symmetric key used to encrypt or decrypt the media object. In addition to CEK, there may be an additional key, that is *MAC key*, to protect the integrity of a DRM license.

Rights Expression

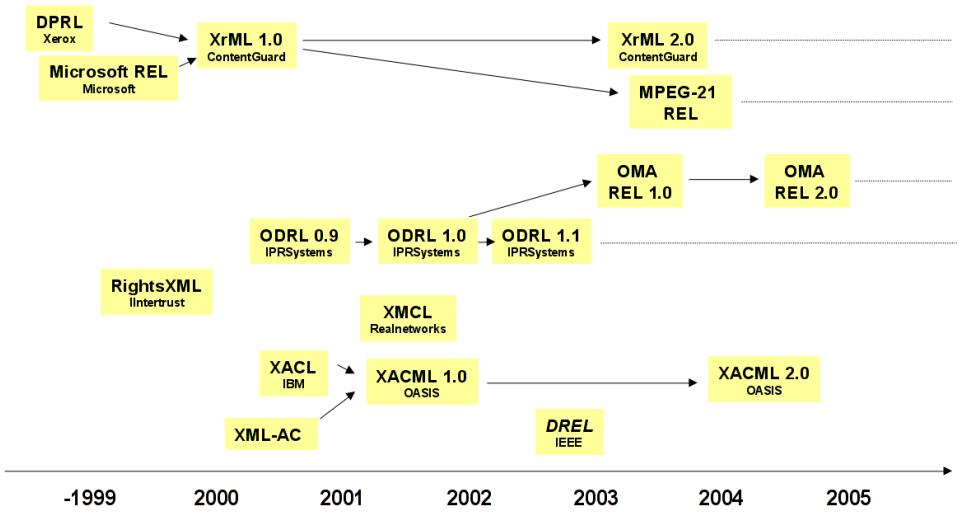

Figure 2.5: REL landscape [Har04]

In a DRM license, the usage rights are typically expressed in a *Rights Expression Language (REL)*[5]. There are many different RELs for the rights expression. Mostly RELs do not have much technical difference. It is more a political issue than any technical difference to use a specific REL for a DRM system. Figure 2.5 depicts how different RELs are evolved with the passage of time.

[5]In Marlin, the usage rights are expressed with a control program.

A REL has capability to express usage rights within a license. These rights include what actions are allowed (permission) and under which conditions (constraint). For better understanding about the available permission set and constraint set in a REL, see Appendix A.1.

In a DRM system, a user has the following rights which are explicitly granted:

- Everything not mentioned in the license, is not allowed.
- Except if default rights are implicitly assumed in the system.

Usually, the rights are textual description or expressions using XML-based languages. Copy control information and copy flags are a simple form of rights expression. To know how to express the rights in a REL, see Appendix A.2.

For further information about different RELs, an interested reader is referred to [Bar06].

DRM License Types

Generally, there are following two types of DRM license.

Stateful License A license whose state changes depends on how often (that is, count) or how long (that is, time) it is used [OW07]. For stateful license, it may be necessary to manage license state on the server. For example, a right to play the content within a specified time.

Stateless License A license whose state does not depend on how often or how long it is used. For stateless license, the license server does not need to manage any license state on the server side. For example, *unrestricted content*, which can be played any time but only by the trusted devices.

To see the structure of a typical DRM license, see Appendix B.

2.3.5 DRM Client Device

A DRM client device is an entity that can play the content. It obtains the encrypted content from a content delivery system. It acquires the license from a license server after providing client certificate in the license request. This client certificate is issued by a trust authority. After acquiring the license, it can play the content. It is also responsible to enforce the rights mentioned in the license.

The enforcement of rights and the prevention of the attacks are the main challenges at the client side [PJ07]. A trusted entity is necessary at the

client. This may either be a specific software, a hardware security module, or a combination of both. The keys and the decrypted media objects must not be available to any application, which is not trusted, or a user even when hacking the software or the hardware. The end user device is usually the main point of attack because it is under the hostile environment. The aim of attack may be to disclose the content, CEK, or to circumvent (enforcement of) the rights on the end user device. Therefore, the end user device must support trusted environment [Gro] and there must be a secure storage to store the keys and the rights.

For further information about DRM, an interested reader is referred to [BBGR03, ZYL06].

2.4 Trust Model

2.4.1 What is Trust Model

A trust model specifies full processes and implementation necessary to complete a DRM system. In a trust model, a trusted entity offers agreements for the adopters and the content participants. These agreements include compliance and robustness rules for the implementation. A trust model specifies a root CA in a DRM system. A root CA certificate is a self-signed CA certificate that is used by the devices and the license servers as a trust anchor to validate the certificates within a PKI domain. In a trust model, the CA within a PKI domain is entitled to issue certificates to entities whose implementation conforms to the rules specified in the agreement. It is also responsible to revoke a certificate in case of a security breach.

2.4.2 Why a Trust Model

A DRM system completely specifies the communication between the client and the server. It partly specifies the internal actions inside the client. For example, what happens if a signature verification fails. But it does not specify the security of the implementation. That is, how securely private keys are stored or who operates CA in a PKI system.

In short, a DRM system provides only the functionality but does not specify:

- Legal agreements
- Compliance rules
- Robustness rules
- Security related services

The above items are what a content provider really requires. It is the trust model which specifies all of them.

2.4.3 Components of a Trust Model

The main components of a trust model are described below:

Legal Agreements

In order to implement any DRM specification, a license from a trust authority is necessary. This license is available through an agreement. There is a yearly fee for making these agreements. There is an additional fee usually a few cents per device. It also includes liability in case of hacks. The legal agreement plays an important role in acceptance of a trust model by content providers.

Compliance Rules

The compliance rules define that implementation must not circumvent the content protection through any way. It defines the device behavior dictated by design. For further information about compliance rules, see [Mica, Clo].

Robustness Rules

The robustness rules specify the security requirements. These security requirements include confidentiality and integrity of the content and the keys. It defines the secure storage of secrets and the secure data paths. For example, the decrypted content must not be accessible to anyone except allowed interfaces. For further information about robustness rules, see [Mica, Clo].

Security Related Services

These services provide mechanisms to obtain and manage certificates. It also includes a mechanism to revoke a certificate usually in case of a security breach. For the revocation, different mechanisms are possible. However, the most commonly used mechanisms are *Certificate Revocation List (CRL)* and *Online Certificate Status Protocol (OCSP)*.

The trust models of different DRM systems are given in Table 2.1.

Table 2.1: The trust models of different DRM systems

DRM System	**Trust Model**
OMA DRM v2.1	*CMLA (optional)*
Marlin	*MTMO*
Microsoft WMDRM	*(Microsoft proprietary)*
Apple Fairplay	*(Apple proprietary)*

2.5 Basic Use Cases

This section defines basic use cases which are used to evaluate the different existing DRM systems and the options for convergence. The use cases are presented in two different versions. One is the content downloading version while other is the content streaming version. The use cases are deliberately kept simple. That is, the detail of the communication between the client device and the server is not considered.

2.5.1 Basic License Acquisition

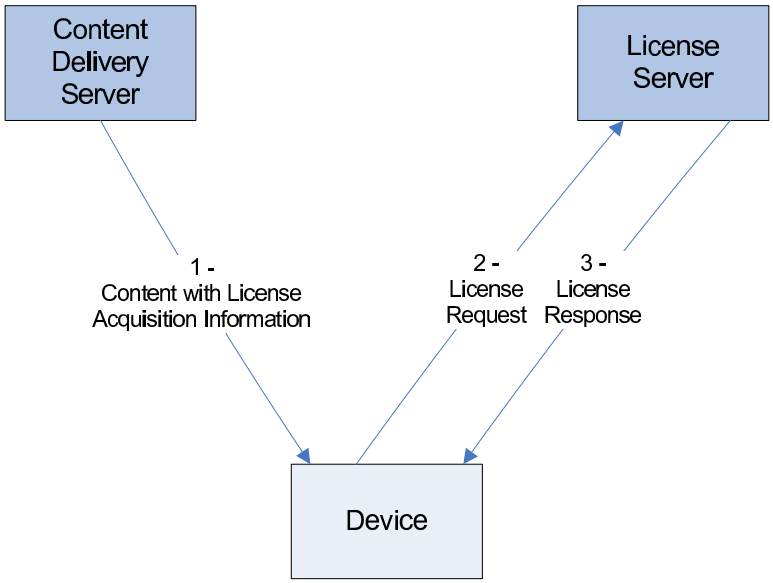

Figure 2.6: Basic license acquisition

This is the very basic use case where a single device obtains content (with license acquisition information) from a content delivery server. The device acquires a corresponding license from a license server using license acquisition information embedded into the content. To acquire a license, the device sends *License Request* message to the license server and receives corresponding license in *License Response* message as shown in Figure 2.6. This use case can be splitted in two versions, one for downloading and other for streaming. The sequence of messages for this use case is as follows:

1. Content (downloading or streaming) with license acquisition information
2. License request
3. License response

2.5.2 Basic Content Transfer

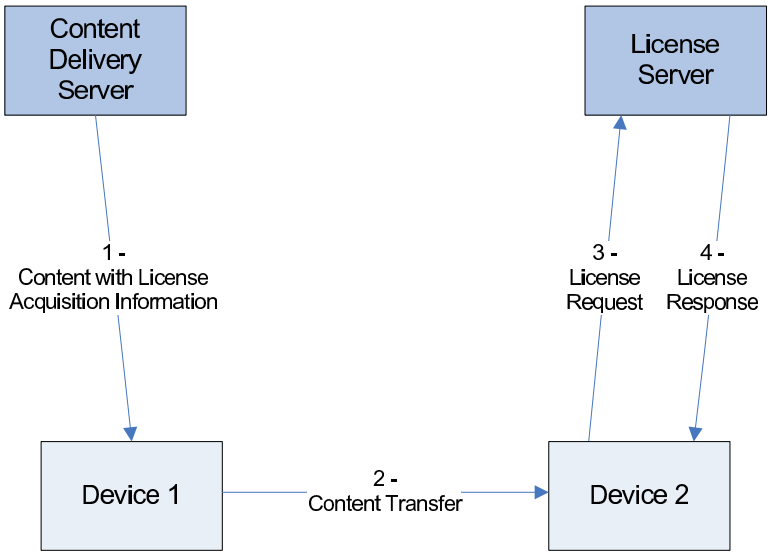

Figure 2.7: Basic content transfer

This is the basic content transfer use case for downloading content, often referred to as *Super-distribution*. A first device (that is, *Device 1*) downloads the content from a content delivery server as shown in Figure 2.7. Next, the content is moved to a second device (that is, *Device 2*). The second device acquires a license for the content using license acquisition information embedded into the content. The second device acquires the license in the same way as acquired in *Basic License Acquisition* use case (described in Subsection 2.5.1). The sequence of messages for this use case is as follows:

1. Content downloading with license acquisition information
2. Content transfer
3. License request
4. License response

2.5.3 Basic Stream Transfer

This use case governs stream transfer from one device to another (that is, from device 1 to device 2). The stream transfer is based on bookmark exchanged between the devices. This use case is almost same as *Basic Content Transfer* use case (discussed in Subsection 2.5.3) except it is based on bookmarks.

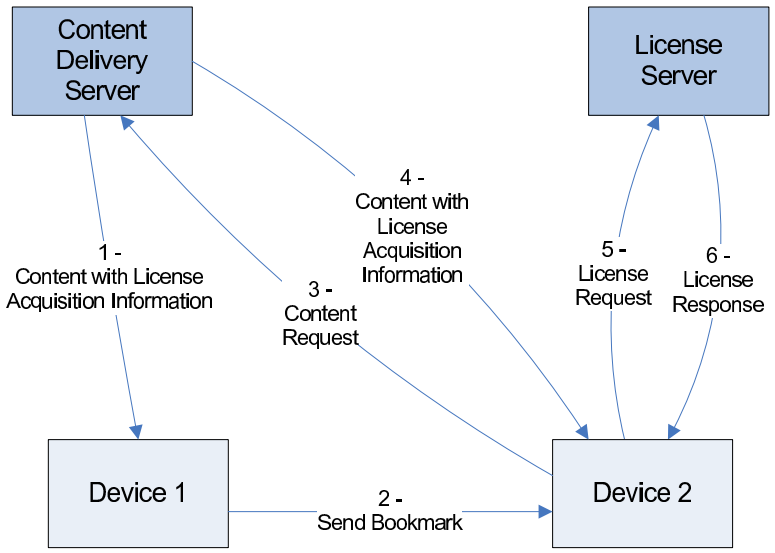

Figure 2.8: Basic stream transfer

Initially, the content is streamed to a first device (that is, device 1) as shown in Figure 2.8. Next, a bookmark is sent to a second device (that is, device 2). The second device uses this bookmark to obtain the content stream from the content delivery server. To obtain the content stream, the second device sends *Content Request* message to the content delivery server and receives corresponding content in return. In order to access the content, the second device acquires the license from the license server. The sequence of messages for this use case is as follows:

1. Content streaming with license acquisition information
2. Send bookmark
3. Content (streaming) request
4. Content streaming with license acquisition information
5. License request
6. License response

2.6 Standardized DRM Systems

This section gives examples of complete DRM systems. It covers two different standardized DRM systems described in Subsection 2.6.1 and Subsection 2.6.2. Subsection 2.6.1 describes OMA DRM system which targets mobile while Subsection 2.6.2 describes Marlin DRM system which targets CE.

2.6.1 OMA DRM

OMA DRM is a standardized DRM system. OMA members represent the entire value chain which includes mobile phone manufacturers (Sony-Ericsson, Nokia, LG, Motorola, and Samsung), mobile system manufacturers (Ericsson, NSN, and Huawei), operators (Vodafone, O2, Cingular, T-Mobile, and Orange), and IT companies (Microsoft, IBM, and Sun). Currently, there are different OMA specifications which include OMA DRM v1.0 [Allb], OMA DRM v2.0 [Allc], and OMA DRM v2.1 [Alld]. Some domain specific extensions of OMA DRM system include *Extension for Broadcast Support (XBS)* v1.0 [Alla], *Secure Removable Media (SRM)* v1.0 [Alle] and *Secure Content Exchange (SCE)* v1.0. OMA DRM v2.0 and v2.1 are briefly described in this report. However, this report concentrates on OMA DRM v2.1.

OMA DRM v2.0

OMA DRM v2.0 standard is an extension to its successor OMA DRM v1.0. It was released in 2006. It has improved security by employing PKI. Each of the device and the *Rights Issuer (RI*[6]*)* has a pair of keys and corresponding PKC. The PKCs play an important role during *Rights Object Acquisition Protocol (ROAP)*. They are used for authentication between the device and the RI. They are also used to protect the integrity of exchanged information. An RO is signed by an RI while CEK is encrypted with public key of the device. For the rights expression, OMA DRM v2.0 uses a subset of ODRL version 1.1. For further information about OMA DRM v2.0, see [Allc].

OMA DRM v2.1

OMA DRM v2.1 [Alld] is an extension to OMA DRM v2.0. It further introduced several useful features such as content usage metering, RO upload, and RO installation confirmation. Figure 2.9 depicts OMA DRM v2.1 architecture. Within this DRM system, a *Content Issuer (CI*[7]*)* issues protected content while an RI issues RO (containing usage rules) to the *DRM Agent*. A DRM agent manages permission for media objects on the device. A DRM agent can forward protected content to other DRM agents. A DRM agent can also store the protected content on removable media or network store. A user controls the operations of a DRM agent. For example, to forward the protected content. The CI and RI are managed by a content provider. The license acquisition information in OMA DRM system can be found in Appendix D.

[6]In OMA DRM, an RI (an entity distributing RO) is a license server.
[7]In OMA DRM, a CI is a content delivery server.

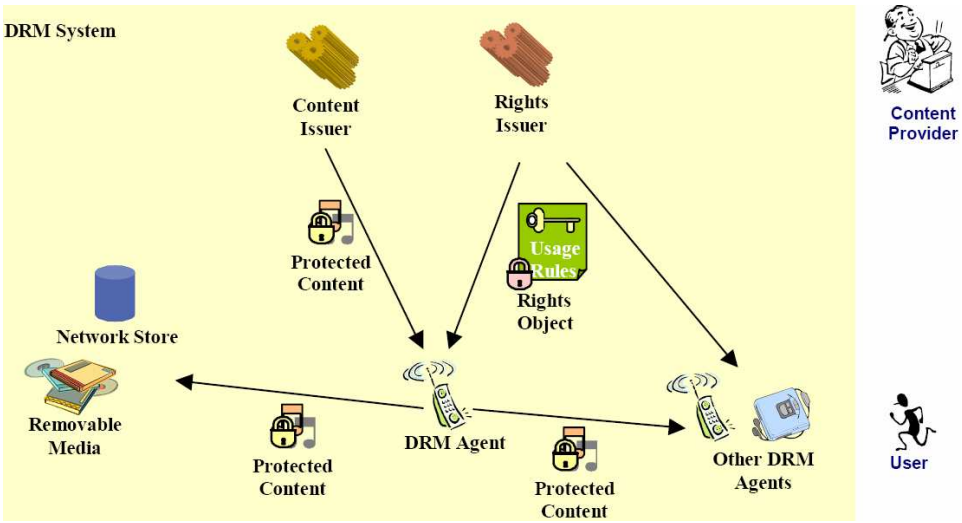

Figure 2.9: OMA DRM v2.1 architecture [All08a]

2.6.2 Marlin

Marlin standards are maintained and developed by *Marlin Development Community (MDC)* which was founded by Intertrust, Sony, Philips, Samsung, and Panasonic. It is an open standard and further companies can join it. In addition to the founding members, MDC has nine other members including Ericsson which is MDC member since November 2007. MDC has developed different standards. *Internet Protocol Television - Endpoint System (IPTV-ES)* and *Broadband (BB)* with *Broadband Network Service (BNS)* profile [Com08] are commercially used and are of practical importance. This report briefly describes and pays attention on the later one.

BB with BNS Profile

Marlin Broadband (MBB) is a standard for a complete DRM system. It mainly consists of Marlin core specification plus a signaling and registration protocol to trigger actions at the client side to obtain a license via tokens. Figure 2.10 depicts a broader picture of MBB network and provides an overview of BNS. In a typical scenario, a user interacts with the *Store Web Site* to purchase the content. A DRM client acquires actions token and configuration token from the store web site. These tokens enable the DRM client to make a license request to the *DRM Server*[8]. A DRM client acquires the corresponding encrypted content from the *Content Server*[9] using communication protocol such as HTTP. A content server, a DRM server, and a

[8]In Marlin, a DRM server is a license server.
[9]In Marlin, a content server is a content delivery server.

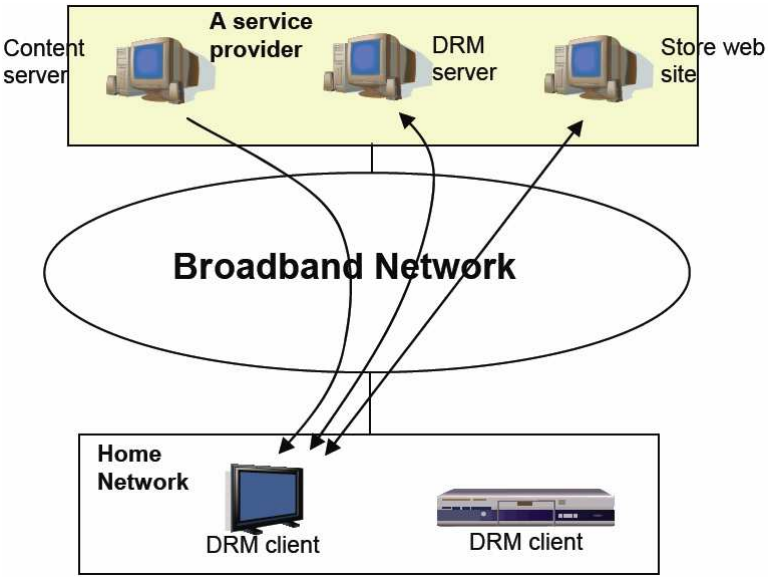

Figure 2.10: BNS overview [Com08]

store web site are managed by a service provider. For further information about MBB, an interested reader is referred to [Com08].

The core components of MBB are:

NEMO *Networked Environment for Media Orchestration (NEMO)* is the messaging framework for secure communication between different components of MBB. It provides end-to-end message protection to ensure integrity, confidentiality, freshness, authentication, and role-based service authorization. Authentication and authorization are ensured by NEMO messaging keys and certificates. To know more about NEMO, see [Com06d, Com06b].

Octopus In Marlin, usage rights are expressed with a *Control Program*. A control program is executed on a *Virtual Machine (VM)* named *Plankton* which is implemented on the device. The Plankton VM is part of the client engine *Octopus*. The output of a control program determines whether the client is entitled to access CEK or not. To know more about Octopus, see [Com06e, Com06c].

For further information about MBB, see [Com06a].

2.7 Proprietary DRM Systems

This section gives examples of complete DRM systems. It covers two different proprietary DRM systems described in Subsection 2.7.1 and Subsection

2.7.2. Subsection 2.7.1 describes Microsoft DRM system which targets both PC and mobile while Subsection 2.7.2 describes Apple Fairplay DRM system which targets iPod/iPhone.

2.7.1 Microsoft DRM

Microsoft is the biggest provider of proprietary internet DRM solutions. It offers two separate DRM solutions, one to target PCs and portable unconnected music players while other to target mobiles. Both of them are briefly described below:

Windows Media DRM 10

Windows Media DRM (WMDRM) is targeted at PCs and portable unconnected music players. Its technical details are publicly unknown. It contains an update mechanism which is used to recover from hacks. This update mechanism seems to be working quite well because reported hacks have been fixed very quickly. For further information about WSDRM, see [Mic04].

Playready

Playready was announced in February 2007 and its first version of its suite was released in June 2008. It extends the concept used in WMDRM to target mobiles. It includes the concept of domain and embedded license. It is only compatible with Microsoft solutions such as Windows Media DRM 10. For further information about Playready, see [Mic08].

2.7.2 Apple Fairplay

The Apple iTunes DRM is called *FairPlay* which only supports downloading. In this DRM system, protected files are encrypted MP4 container files. The content is encrypted with *Master Key*[10] which is stored inside container file along with the encrypted content. For each track, a new random key known as *User Key*, is generated and used to encrypt the master key. The user Key is stored on the server and on the receiving device[11] in an encrypted repository. That is why, this repository is the main attacking point for many of the known hacks. For playback, the user key is retrieved from the key repository and used to decrypt the master key. For further information about Apple Fairplay, see [Ano07].

[10] In this DRM system, a master key is a CEK.
[11] The device may be an iPod or iPhone.

Chapter 3

Interoperability in PKI

3.1 Introduction

The motivation behind the interoperability between DRM systems is not only because of the content providers implementing different DRM systems (for example, Marlin and OMA DRM v2.0) but it is also important for the content providers implementing similar DRM systems. For example, *Telefonica*[1] and *France Telecom*[2], both implement OMA DRM v2.0 (or later version) but their root CAs are different. Thus, each DRM system has its own root CA that is responsible to manage the trust between PKI entities.

Let's first have a look at PKI. A PKI enables users to exchange data securely through the use of a public and a private key pairs. After registering a user, a trusted authority in PKI provides the public key in a digital certificate. A digital certificate can identify an individual or an organization. A PKI directory service stores and, when necessary, revokes the certificates. Throughout this report, X.509 PKI standard is considered. To know further about PKI, see [AFKM05, Cur97].

Every DRM system has a PKI domain which contains different entities like CA, license server, and device. In a PKI domain, a CA may certify a CA, a license server, or a device. Formally, a PKI domain can be described as follows:

Let's assume the presence of a set of agents which range over a, b, c, and d having roles from $\{CA,\ license\text{-}server,\ device\}$. The $certifies$ action is defined as:

$$certifies(a, b) \tag{3.1}$$

[1] `http://www.telefonica.com/`
[2] `http://www.orange.com/`

meaning a certifies b, where a and b are agents having roles from $\{CA\}$ *and* $\{CA,$ *license-server, device*$\}$*, respectively.*

In a PKI domain, a root CA has a self-signed certificate and is defined in terms of (3.1) as:

$$certifies(a, a) \tag{3.2}$$

where a is an agent having role CA.

The certificates are issued by a root CA or an intermediate CA having certificate chain ending up with a root CA. The *certifies* action in (3.1) can be combined as a transitive rule:

$$certifies(a, b) \wedge certifies(b, c) \rightarrow certifies(a, c) \tag{3.3}$$

meaning if a certifies b and b certifies c implies that a certifies c, where a, b, and c are agents having roles from $\{CA\}$*,* $\{CA\}$*, and* $\{CA,$ *license-server, device*$\}$*, respectively.*

In a PKI domain, entities can authenticate each other. Formally, it is defined as:

$$authenticates(a, b) \tag{3.4}$$

meaning a and b authenticates each other; where both a and b are agents having roles from $\{$*license-server, device*$\}$*.*

A root CA certificate is a common trust anchor for all entities within a PKI domain. The entities can mutually authenticate each other through exchange of the certificates if they have certificates issued by a common CA. Formally, it can be expressed as a following rule in terms of (3.1) and (3.4):

$$certifies(a, b) \wedge certifies(a, c) \rightarrow authenticates(b, c) \tag{3.5}$$

meaning if a certifies b and a certifies c implies that b and c can mutually authenticate each other; where a, b, and c are agents having roles from $\{CA\}$*,* $\{CA,$ *license-server, device*$\}$*, and* $\{CA,$ *license-server, device*$\}$*, respectively, provided b and c have installed same root CA certificate.*

However, if two entities belong to different PKI domains then both would have certificates having certificate chain ending up with different root CAs, say root CA1 and root CA2. In such a case, there is a need of trust establishment between both root CAs (that is, between root CA1 and root CA2). This sort (that is, between different PKI domains) of trust establishment is achieved through *Interoperability in PKI*. In short, as a first step to the resolve the interoperability between DRM systems, there is a need of interoperability in PKI. The interoperability between PKI can be achieved in three different ways:

1. *Cross-Certification* approach
2. *Super CA* approach
3. *Proxy Server* approach

This chapter explains different approaches to achieve interoperability in PKI and contributes by applying each approach to the resolve the interoperability between two OMA DRM v2.1 systems. This chapter is organized as follows: Section 3.2 describes how to perform cross-certification between two root CAs with its advantages and disadvantages. Section 3.3 suggests to introduce a new root CA, named as *Super CA* on the top of existing root CAs, with its advantages and disadvantages. Section 3.4 considers a proxy server which is sort of a bridge between two PKI domains. It implements two roles at the same time. One role is to act as a server for the source PKI domain while other role is to act as a client for the target PKI domain. Section 3.5 gives a concise overview how to establish trust between a device and an RI within OMA DRM v2.1 system and applies above mentioned approaches for interoperability between two OMA DRM v2.1 systems.

3.2 Cross-Certification Approach

The establishment of a trust relationship between two CAs through signing of public keys of each other, is known as *Cross-Certification*. Cross-certification is performed to extend the trust from one PKI domain to another PKI domain. By extending the trust, the users in one PKI domain would recognize the users of another PKI domain [Tur00].

According to [BHR99], a cross-certificate is an attribute pair consisting of two elements, the forward and the reverse. The forward element contains the certificates issued to this CA by other CAs. The reverse element contains the certificates issued by this CA to other CAs. The cross-certification may be:

- In the forward direction if the forward element is present.
- In the reverse direction if the reverse element is present.
- In both directions if both elements are present.

The structure of a cross-certificate is specified in [BHR99].

Without performing the cross-certification, each root CA is an autonomous CA within its PKI domain. But after performing cross-certification (in both directions) between two root CAs, both can act as a root CA for each other. Figure 3.1 illustrates a typical scenario showing the cross-certification relationship in both directions. CA1 and CA2 are the root CAs in PKI domain 1 and PKI domain 2, respectively.

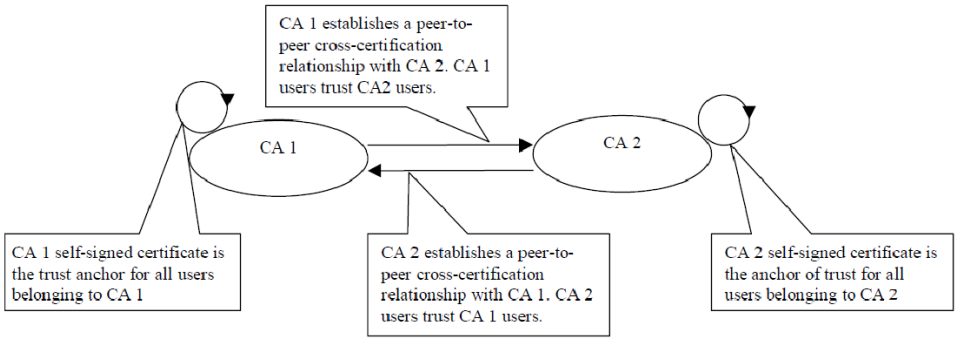

Figure 3.1: Cross-certification relationship between CA1 and CA2 [Tur00]

Formally, a cross-certificate in one direction, that is forward for a and reverse for b, is defined in terms of (3.1) and (3.2) as:

$$certifies(a, a) \wedge certifies(b, b) \wedge certifies(a, b) \wedge a \neq b \tag{3.6}$$

where a and b are agents both having role CA.

In above definition, a and b are root CAs and *a certifies b* is a cross-certification in one direction. Formally, a cross-certificate in both, forward and reverse, directions is defined by extending (3.6) as:

$$\begin{aligned} &certifies(a, a) \wedge certifies(b, b) \wedge \\ &certifies(a, b) \wedge certifies(b, a) \wedge a \neq b \end{aligned} \tag{3.7}$$

where a and b are agents both having role CA.

In above definition, a and b are root CAs; *a certifies b* and *b certifies a* represent the cross-certification in both directions. If two agents have installed different certificates then both can authenticate each other by combining rule (3.5) and (3.7):

$$certifies(a, c) \wedge certifies(b, d) \rightarrow authenticates(c, d) \tag{3.8}$$

meaning if a certifies c and b certifies d implies that b and d can mutually authenticate each other; where both agents a and b cross-certify (in both directions) each other, see (3.7), while both agents c and d have roles from $\{$*license-server, device*$\}$.

As the number of the cross-certified PKI domains grows, the number of the relationships between them grows exponentially [CDH+05]. It might be required to restrict the unintended transitive trust relationship between the PKI domains. According to [Tur00], there are three ways of constraining it:

Path Length Constraint Path length constraint is used to control the transitive trust relationship between CAs. For example, root CA1 has unilaterally cross-certified (in the forward direction) with root CA2 and root CA2 has cross-certified (in both directions) with root CA3, as shown in Figure 3.2. Root CA1 limits the trust to root CA2 by specifying a path length constraint of one in the cross-certificate. Consequently, root CA1 does not trust root CA3.

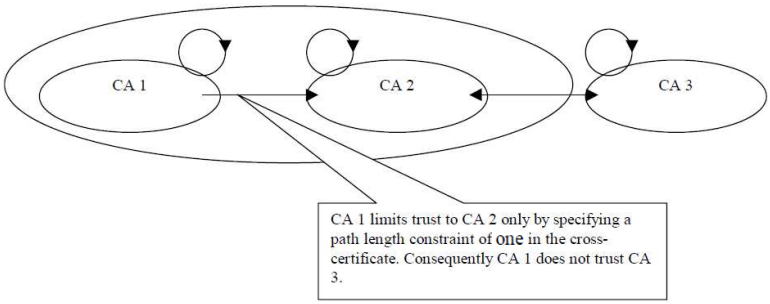

Figure 3.2: Path length constraint [Tur00]

Name Constraint Name constraint can be used to limit the trust to a sub-group of the cross-certified CA based on their *Distinguished Name (DN)*.

Policy Constraint This constraint is used to limit the trust only to those cross-certified CA users who have specific policy values within their certificates.

For further information about cross-certification, see [Tur00, STDD06, ITU00, Cur97, BNP].

3.2.1 Advantages

- Every company[3] would have freedom to perform the cross-certification with the company of her own interest.
- This approach is beneficial with respect to business point of view because two companies can make agreement as they want.
- It provides flexibility to establish and revoke[4] the trust relationship with other companies as business needs change.
- This approach does not require to establish any new entity.

[3]For example, Microsoft and Apple

[4]In cross-certification approach, the revocation requires CRLs (or OCSP responders) to be synchronized in both PKI domains.

3.2.2 Disadvantages

– If there are N number of DRM systems each having its unique trust anchor then to make all of them interoperable, the required number of the cross-certifications would be:

$$\begin{pmatrix} N \\ 2 \end{pmatrix} = \frac{N(N-1)}{2}$$

3.3 Super CA Approach

The trust relation among multiple CAs can be established by introducing a new CA that can work as a bridge. This new CA is known as *Super CA*. In the super CA approach, users in a CA would be able to trust the users belonging to any CA, provided both CAs are under the umbrella of the super CA. For further information about the super CA, see [STDD06].

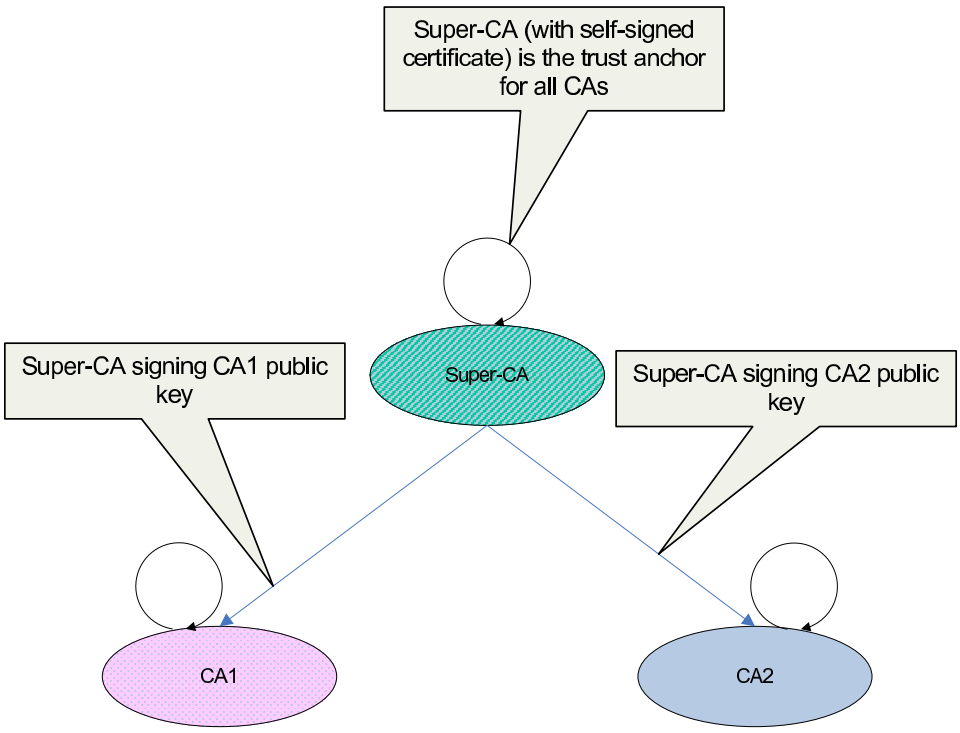

Figure 3.3: Super CA signing CA1 and CA2

Once a root CA is signed by a super CA, the users in that CA can trust the users belonging to any CA signed by the super CA. The signing of CA1 and CA2 by the super CA is shown in Figure 3.3.

Formally, a super CA can be expressed in terms of (3.1) and (3.2) as

follows:

$$certifies(a,a) \wedge certifies(b,b) \wedge certifies(c,c) \wedge \\ certifies(a,b) \wedge certifies(a,c) \wedge a \neq b \wedge a \neq c \wedge b \neq c \tag{3.9}$$

where a, b, and c are agents all having role CA.

In above definition, a is a super CA while b and c are root CAs. Only a can certify b and c but not vice versa (that is, b and c cannot certify a).

3.3.1 Advantages

- A super CA can sign and revoke its immediate subordinate CAs (that is, CA1 and CA2) at any time.
- The seamless integration of all DRM systems under the umbrella of a super CA.

3.3.2 Disadvantages

- The super CA certificate needs to be installed on the device.
- This approach is not a preferable choice with respect to business point of view because the companies have to pay additional licensing fees to the authority managing the super CA.
- Every DRM system has to trust on all other DRM systems which are under the umbrella of the super CA. This trust might be crucial in cases where some DRM vendors are not ready to rely on DRM systems complying the less strict robustness and compliance rules.
- The super CA is the root trust anchor for all CAs. Therefore, the maximum physical security policies and practices are required for its protection.
- This approach requires the establishment of a new CA.

For further information about super CA, see [STDD06, Tur00].

3.4 Proxy Server Approach

A proxy server is a bridge between a client and a server. It acts as an entity in the middle to connect different PKI domains. It implements two roles at the same time. The first role is to act as a server for the client (of the source PKI domain) while the second role is to act as a client for the server (of the target PKI domain).

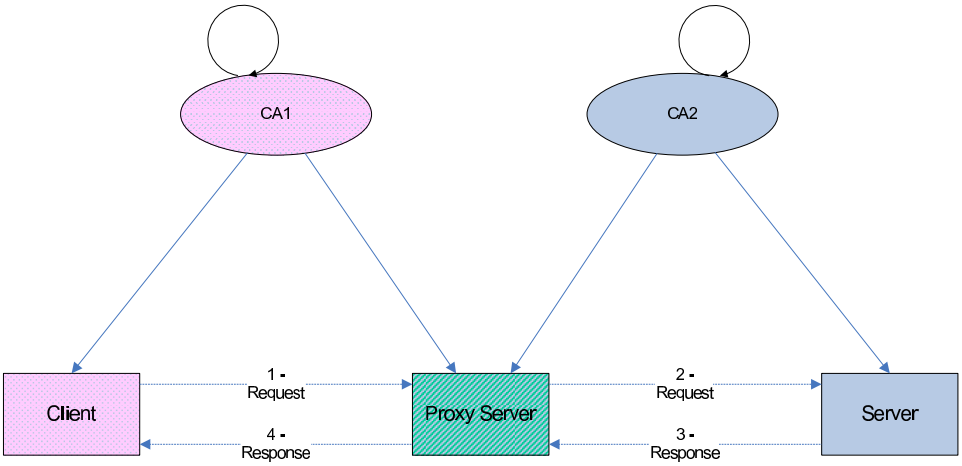

Figure 3.4: Proxy server approach

Formally, a proxy server can be expressed in terms of (3.1) and (3.2) as follows:

$$certifies(a,a) \wedge certifies(b,b) \wedge certifies(a,c) \wedge certifies(b,c) \wedge a \neq b \wedge a \neq c \wedge b \neq c \quad (3.10)$$

where a, b, and c are agents; both a and b have role CA while c has roles license-server and device.

In above definition, c is a proxy server. Both a and b are the root CAs. One of them certifies c as a *license-server* while other certifies c as a *device*. Therefore, c has simultaneously two roles, a *license-server* role and a *device* role.

Figure 3.4 shows a client which connects to the proxy server by requesting a service (that is, *1 - Request* message). The proxy server evaluates the request from the client according to its filtering rules. If the request is validated by the filter, the proxy server connects to the relevant server and requests on behalf of the client (that is, *2 - Request* message). In response, the relevant server replies to the proxy server (that is, *3 - Response* message). Finally, the proxy server forwards that response (after processing according to its rules) to the client (that is, *4 - Response* message). In this communication, a mutual authentication is performed two times. First time for the authentication between a client and a proxy server while the second time for the authentication between a proxy server and a (target) server. In other words, a client performs mutual authentication indirectly with a (target) server in this approach. Formally, it can be expressed a rule that combines (3.10) and (3.4) as follows:

$$authenticates(a, b) \wedge authenticates(b, c) \rightarrow authenticates(a, c) \quad (3.11)$$

meaning if a authenticates b and b authenticates c implies that a and c can mutually authenticate each other; where a, b, and c are agents having roles from {license-server, device}.

For further information about the proxy server, see [AV99, CD01].

3.4.1 Advantages

- A proxy server is easy to implement because an existing server can be extended to implement the client functionality.
- All advantages of cross-certification approach are also applicable to this approach (see Subsection 3.2.1).

3.4.2 Disadvantages

- There is a possibility of information leakage or compromise when a proxy server acts as a client.

3.5 Interoperability between two OMA DRM v2.1 Systems

This section discusses interoperability between two OMA DRM v2.1 systems having different trust anchors. Before applying the interoperability approaches (discussed in Section 3.2, Section 3.3, and Section 3.4), it is important to know the license format, the registration protocol to register a device, and the license acquisition protocol in OMA DRM v2.1 system. An RO is the license format in OMA DRM system. It is encrypted with the device public key. For further information about an RO, see Appendix B. The registration protocol performs the mutual authentication between a device and an RI. It is necessary to register a device by performing this protocol before acquiring a license. For further information about the registration protocol, see Appendix C. In the RO acquisition protocol, a device acquires an RO from an RI. An RO is encrypted with the device public key. For further information about the RO acquisition protocol, see Appendix D.

3.5.1 ROAP and CMLA PKI System

Figure 3.5 illustrates CMLA PKI system. *CMLA Root CA* certificate is a self-signed certificate, that is *certifies(CMLA root CA, CMLA root CA)*, and is used as a trust anchor by the devices and the RIs in CMLA PKI

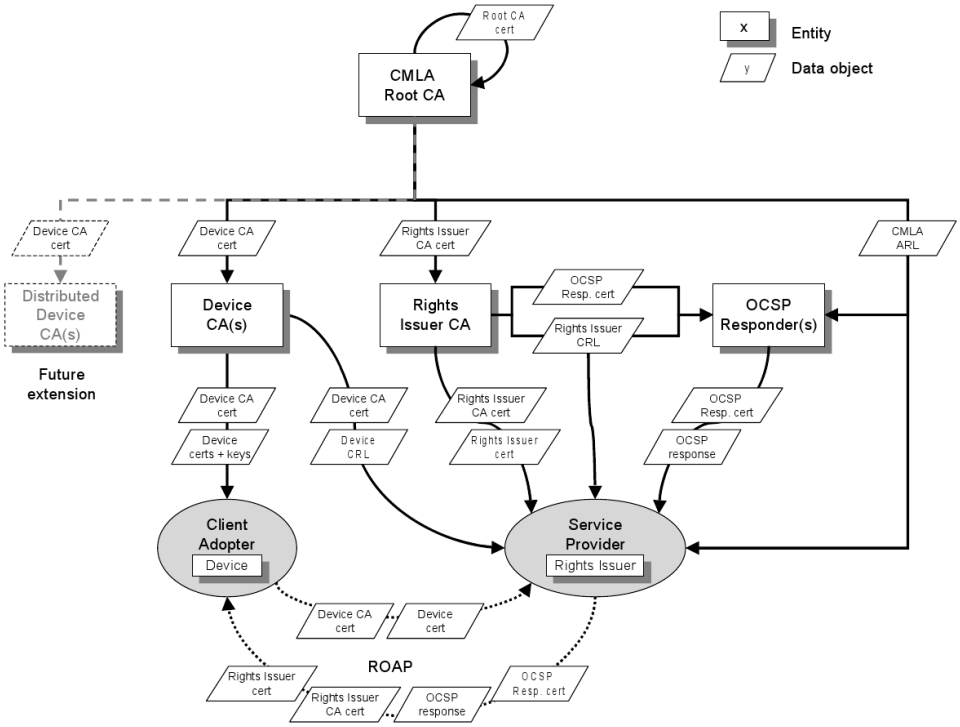

Figure 3.5: CMLA PKI system [CML05]

system[5]. The root CA certificate needs to be pre-installed on the device. The CMLA root CA authorizes the *Device CA* to issue the certificates to the devices. This authorization is granted by providing a device CA certificate which is signed by CMLA root CA, that is *certifies(CMLA root CA, device CA)*. In a distributed environment, there may be a number of device CAs to serve the purpose. A device obtains its certificate and corresponding keys from a device CA, that is *certifies(device CA, device)*. A device CA is also responsible to update the RI with information of the revoked devices. For this purpose, a device CA provides a device CRL to the RI.

Similarly, CMLA root CA authorizes *Rights Issuer CA* to issue the certificates to the RIs. This authorization is granted by providing an RI CA certificate which is signed by CMLA root CA, that is *certifies(CMLA root CA, RI CA)*. An RI obtains its certificate from an RI CA, that is *certifies(RI CA, RI)*. An RI CA is also responsible to keep the record of the revoked RIs by managing a *Rights Issuer CRL* and an *OCSP Responder*. An RI CRL is issued by an RI CA and it contains the revocation status information of the RI certificates. It is delivered to and used by OCSP responder. An RI CA issues a certificate to OCSP responder. An OCSP responder signs *OCSP Response*. An OCSP response is delivered to the device through the RI as a

[5]CMLA root CA certificate is delivered out-of-band.

proof to show the revocation status of the RI and to synchronize the device time.

The CMLA root CA also issues a CMLA *Authority Revocation List (ARL)*. A CMLA ARL is used by the RI to verify the revocation status of the device CA.

For the mutual authentication between the device and the RI, the most important element in the registration protocol (see Appendix C) is the *Certificate Chain*. It is used to exchange the certificates from the device to the RI and vice versa. Through this element, a device sends the device certificate and the device CA certificate to the RI. The RI verifies the chain of certificates starting from the device certificate up to CMLA root CA certificate. An RI already stores the trusted CMLA root CA certificate. Therefore, the verification process ends before verifying that certificate. Additionally, the RI also checks the CMLA ARL and the device CRL to verify the revocation status of the device CA and the device, respectively.

Similarly, An RI sends the RI certificate and the RI CA certificate to the device. Optionally, an RI may send the OCSP response and the OCSP responder certificate to the device. The device verifies the chain of certificates starting from the device certificate up to root CA certificate. A device already stores the trusted root CA certificate. Therefore, the verification process ends before verifying that certificate. The OCSP response and the OCSP responder certificate are validated in the similar fashion. For further information about CMLA PKI system, see [CML05].

In Subsections 3.5.2, 3.5.3, and 3.5.4, it is assumed that there are two OMA DRM v2.1 system, OMA1 and OMA2, having root CAs, root CA1 and root CA2, respectively but each using its own trust model. Furthermore, a device of OMA2 sends the encrypted content to the device of OMA1 (that is, *D1*) as mentioned in *Basic Content Transfer* use case discussed in Subsection 2.5.2. Now, D1 wants to acquire RO from the RI of OMA2 (that is, *RI2*). For this purpose, D1 needs to register with RI2. This registration is accomplished using the 4-pass registration protocol described in Appendix C.

3.5.2 Applying Cross-Certification Approach

As already mentioned in Section 3.2, the bilateral cross-certification can be performed by signing the root CAs public keys of each other. That is, root CA1 signs root CA2 while root CA2 signs root CA1 as shown in Figure 3.6. Formally, it can be expressed as:

$$\begin{aligned} &certifies(root\ CA1, root\ CA1) \wedge certifies(root\ CA2, root\ CA2) \wedge \\ &certifies(root\ CA1, root\ CA2) \wedge certifies(root\ CA2, root\ CA1) \wedge \\ &\qquad\qquad root\ CA1 \neq root\ CA2 \end{aligned} \tag{3.12}$$

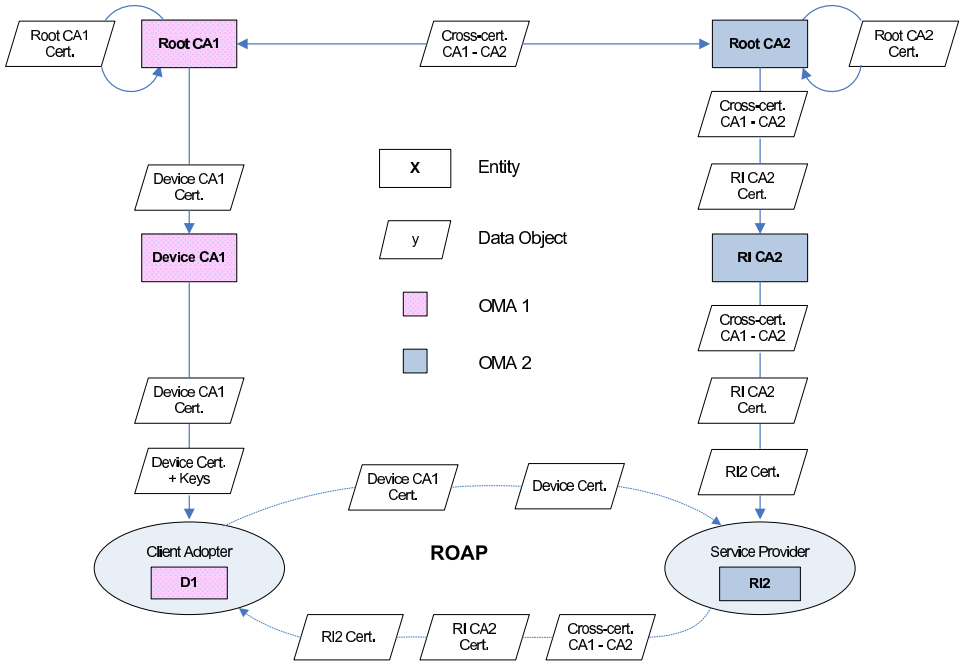

Figure 3.6: Cross-certification hierarchy

Figure 3.6 shows that in OMA1, root CA1 certifies device CA1 while device CA1 certifies D1. In OMA2, root CA2 certifies RI CA2 while RI CA2 certifies RI2. Formally, it can be expressed as follows:

$$certifies(root\ CA1, device\ CA1) \tag{3.13}$$

$$certifies(device\ CA1, D1) \tag{3.14}$$

$$certifies(root\ CA2, RI\ CA2) \tag{3.15}$$

$$certifies(RI\ CA2, RI2) \tag{3.16}$$

To accomplish the registration, D1 sends *Device Hello* message to RI2 as shown in Figure 3.7. In response to device hello message, RI2 replies with *RI Hello* message containing a list of trusted device authorities recognized by RI2 (that is, CA1 and CA2). D1 terminates the registration protocol if its trust anchor is not in the list of trusted device authorities recognized by the RI2. Therefore, RI2 has to add CA1 in its list of trusted device authorities by installing root CA1 certificate.

After that D1 sends *Registration Request* message containing D1 certificate chain (that is, D1 certificate and device CA1 certificate) and a list of trusted RI authorities recognized by D1 (that is, CA1). RI2 terminates the protocol if its trust anchor is not in the list of trusted RI authorities recognized by D1 and RI2 does not have the relevant cross-certificate (that is, a

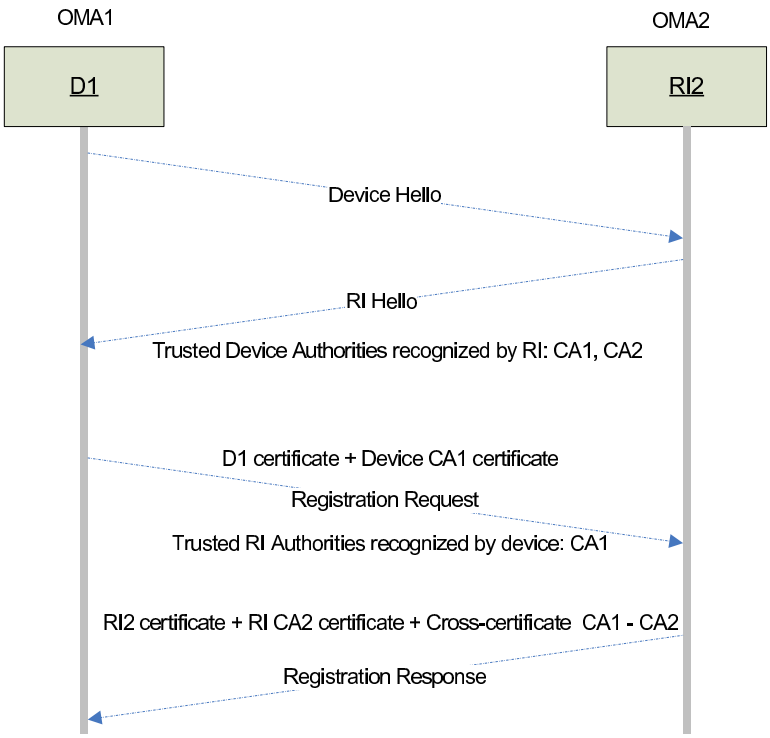

Figure 3.7: Registration protocol between D1 and RI2

cross-certificate between root CA1 and root CA2). Otherwise, RI2 identifies the trust anchor of D1 and includes the cross-certificate (between root CA1 and root CA2) in the start of the certificate chain before its validation.

After the certificate chain validation, RI2 sends *Registration Response* message containing RI2 certificate, RI CA2 certificate, and the cross-certificate (between root CA1 and root CA2). Optionally, a registration response message may contain OCSP response, OCSP responder certificate, and the cross-certificate (between root CA1 and root CA2). Next, D1 ends the registration protocol by validating the certificate chain sent by RI2. In this way, D1 registers with RI2.

Formally, RI2 and D1 performs mutual authentication as follows. First of all, combining (3.13) and (3.14) using rule (3.3) yields:

$$certifies(root\ CA1, D1) \tag{3.17}$$

and combining (3.15) and (3.16) using rule (3.3) yields:

$$certifies(root\ CA2, RI2) \tag{3.18}$$

Now, (3.17) and (3.18) can be combined using rule (3.8), which is applicable as (3.12) holds, to yield:

$$authenticates(RI2, D1) \tag{3.19}$$

After the registration protocol, D1 may acquire an RO by performing the RO acquisition protocol (that is described in Appendix D). In RO acquisition protocol, D1 requests an RO from RI2 (that is, *RO Request*). RI2 replies by issuing an RO encrypted with D1 public key (that is, *RO Response* message). Finally, D1 obtains an RO, decrypts it using its private key, and plays the content.

3.5.3 Applying Super CA Approach

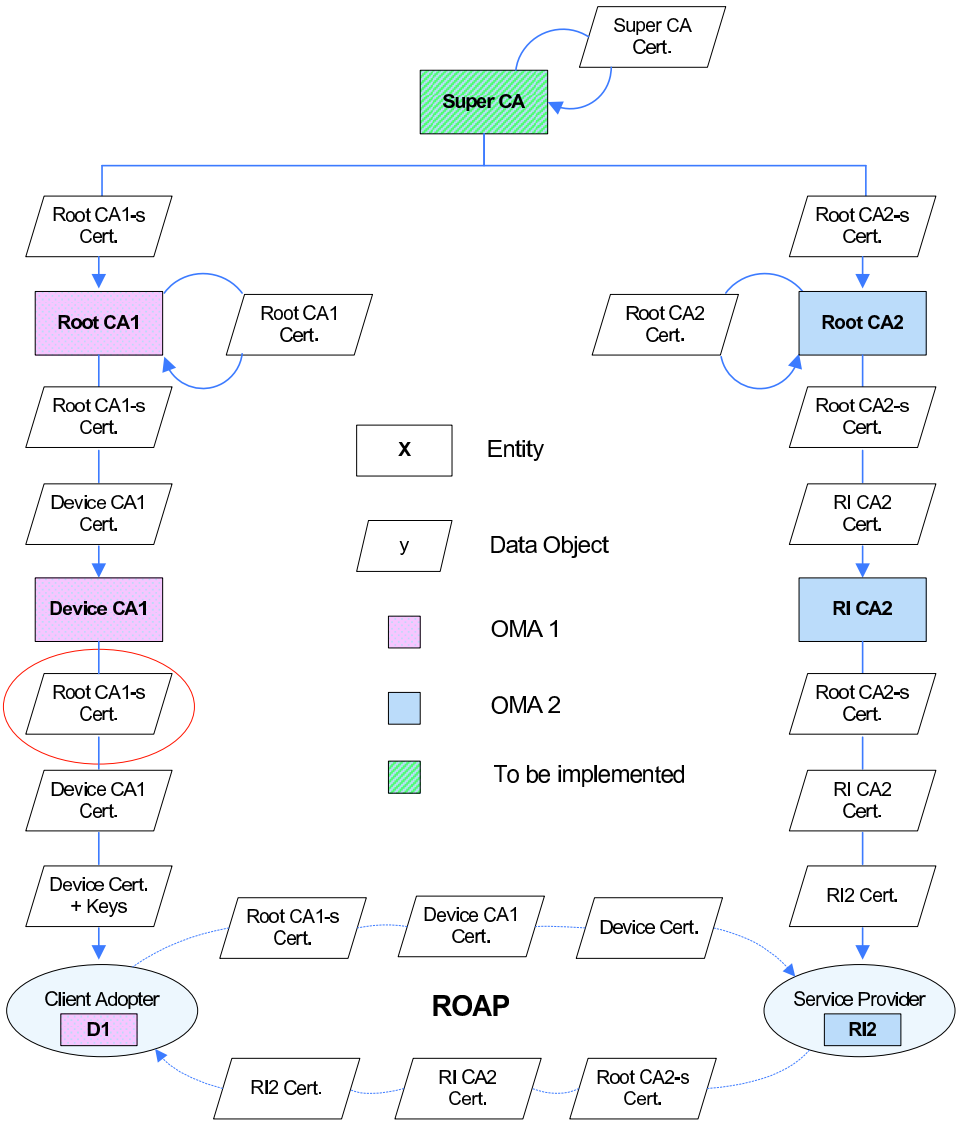

Figure 3.8: Super CA hierarchy

As already described in Section 3.3, there is a need to setup a new CA,

that is super CA. The super CA signs public keys of root CA1 and root CA2 as shown in Figure 3.8. As a result, root CA1 and root CA2 get root CA1-s and root CA2-s certificates, respectively from super CA. In this case, the devices and the RIs need to install an additional super CA certificate for the registration. Installing a new root certificate on the device is not always possible. For example, in case of a mobile device, the root CA is already fixed. That is why root CA1-s certificate is circled red in Figure 3.8.

A super CA certifies root CA1 and root CA2; root CA1 and root CA2 certifies device CA1 and RI CA2; device CA1 and RI CA2 certifies D1 and RI2, respectively. Formally, it can be expressed as follows:

$$certifies(super\ CA, root\ CA1) \tag{3.20}$$

$$certifies(super\ CA, root\ CA2) \tag{3.21}$$

$$certifies(root\ CA1, device\ CA1) \tag{3.22}$$

$$certifies(root\ CA2, RI\ CA2) \tag{3.23}$$

$$certifies(device\ CA1, D1) \tag{3.24}$$

$$certifies(RI\ CA2, RI2) \tag{3.25}$$

To accomplish the registration, D1 sends the certificate chain containing D1 certificate, device CA1 certificate, and root CA1-s certificate to RI2. In this case, RI2 verifies the chain of certificates starting from D1 certificate up to root CA1-s certificate.

Similarly, RI2 needs to send the certificate chain containing RI2 certificate, RI CA2 certificate, and root CA2-s certificate to D1. Optionally, RI2 may send OCSP response, OCSP responder certificate, and root CA2-s certificate to D1. D1 verifies the chain of certificates starting from RI2 certificate up to root CA2-s certificate. In this way, D1 registers with RI2.

Formally, RI2 and D1 performs mutual authentication as follows. First of all, combining (3.22) and (3.24) using rule (3.3) yields:

$$certifies(root\ CA1, D1) \tag{3.26}$$

and combining (3.23) and (3.25) using rule (3.3) yields:

$$certifies(root\ CA2, RI2) \tag{3.27}$$

Now, combining (3.20) and (3.26)u sing rule (3.3) yields:

$$certifies(super\ CA, D1) \tag{3.28}$$

and combining (3.21) and (3.27) using rule (3.3) yields:

$$certifies(super\ CA, RI2) \tag{3.29}$$

Now, (3.28) and (3.29) can be combined using rule (3.5) to yield:

$$authenticates(RI2, D1) \tag{3.30}$$

After the registration protocol, D1 may acquire an RO by performing the RO acquisition protocol (that is described in Appendix D). In RO acquisition protocol, D1 requests an RO to RI2 (that is, *RO Request*). RI2 replies by issuing an RO encrypted with D1 public key (that is, *RO Response* message). Finally, D1 obtains an RO, decrypts it using its private key, and plays the content.

3.5.4 Applying Proxy Server Approach

As already mentioned in Section 3.4, there is a need to extend the existing RI (of the source PKI domain) to implement a proxy server.

In a proxy server approach, root CA1 certifies device CA1 and RI CA1, while root CA2 certifies device CA2 and RI CA2. Furthermore, device CA1 and RI CA2 certifies D1 and RI2, respectively. However, both RI CA1 and device CA2 certifies proxy server. Formally, it can be expressed as:

$$certifies(root\ CA1, device\ CA1) \tag{3.31}$$

$$certifies(root\ CA1, RI\ CA1) \tag{3.32}$$

$$certifies(root\ CA2, device\ CA2) \tag{3.33}$$

$$certifies(root\ CA2, RI\ CA2) \tag{3.34}$$

$$certifies(device\ CA1, D1) \tag{3.35}$$

$$certifies(RI\ CA2, RI2) \tag{3.36}$$

$$certifies(RI\ CA1, proxy\ server) \tag{3.37}$$

$$certifies(device\ CA2, proxy\ server) \tag{3.38}$$

First of all, combining (3.31) and (3.35) using rule (3.3) yields:

$$certifies(root\ CA1, D1) \tag{3.39}$$

and combining (3.32) and (3.37) using rule (3.3) yields:

$$certifies(root\ CA1, proxy\ server) \tag{3.40}$$

Similarly, combining (3.33) and (3.38) using rule (3.3) yields:

$$certifies(root\ CA2, proxy\ server) \tag{3.41}$$

and combining (3.34) and (3.36) using rule (3.3) yields:

$$certifies(root\ CA2, RI2) \tag{3.42}$$

It is also important to know that proxy server performs two roles. Therefore, it installs two root CA certifies, root CA1 certificate and root CA2 certificate.

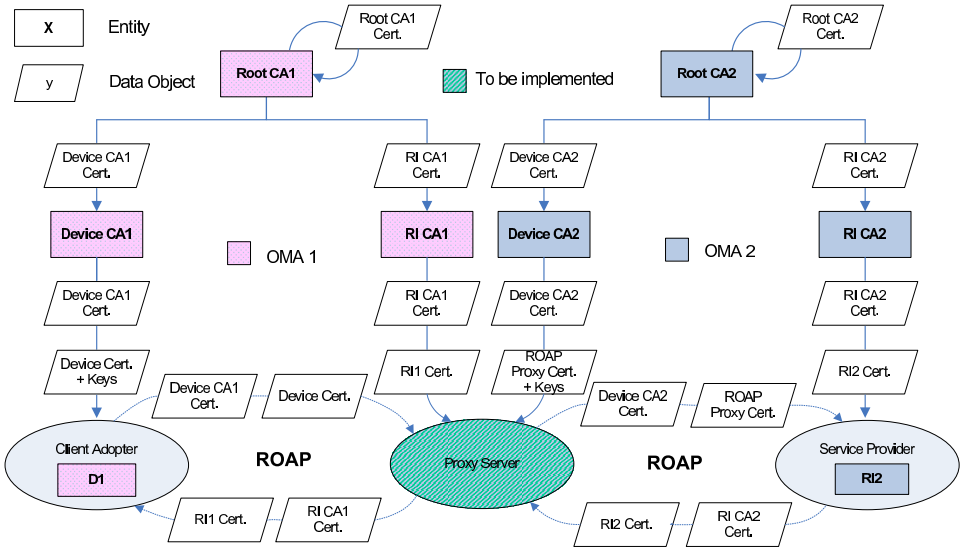

Figure 3.9: RI1 working as a proxy server

In the first step, D1 sends the certificate chain containing D1 certificate and device CA1 certificate to *ROAP Proxy Server*. After validating it, ROAP proxy server creates another certificate chain containing ROAP proxy certificate and device CA2 certificate and sends it to RI2.

RI2 validates the certificate chain sent by ROAP proxy server and replies with a certificate chain containing RI2 certificate and RI CA2 certificate. ROAP proxy serer validates it and replies to D1 with another certificate chain containing RI1 certificate and RI CA1 certificate. In this way, D1 indirectly registers with RI2.

Formally, RI2 and D1 performs mutual authentication as follows. First of all, combining (3.41) and (3.42) using rule (3.5) yields:

$$authenticates(RI2, proxy\ server) \tag{3.43}$$

and combining (3.39) and (3.40) using rule (3.5) yields:

$$authenticates(proxy\ server, D1) \tag{3.44}$$

Finally, (3.43) and (3.44) can be combined using (3.11) to yield:

$$authenticates(RI2, D1) \tag{3.45}$$

After the registration protocol, D1 may acquire an RO by performing RO acquisition protocol (that is described in Appendix D). In RO acquisition protocol, D1 requests an RO to ROAP proxy server. A ROAP proxy server forwards this request to RI2 (as discussed in Section 3.4). RI2 replies by issuing an RO encrypted with public key of ROAP proxy server. A ROAP proxy server decrypts the RO using its private key, re-encrypts it with D1 public key, and sends it to D1. Finally, D1 obtains an RO, decrypts it using its private key, and plays the content.

Now, the question is where to put ROAP proxy server. A ROAP proxy server works both as a client as well as an RI. As a client, it can be put anywhere either in the home environment or on the internet. But as an RI, the robustness rules do not allow to put it in the home environment. Therefore, it should be put somewhere on the internet and there should be physical security policies and practices for its protection.

3.6 Synopsis

In this chapter, three different approaches are described for interoperability in PKI. It has been seen that cross-certification approach and proxy server approach are simple and flexible approaches. Both are more suitable to adopt because of flexible business model and no need to setup any new entity as necessary in case of super CA approach. In cross-certification approach, only the disadvantage is to perform a large number of cross-certifications to create the full mesh between all PKI domains. Practically, there are limited number of DRM systems. Hence, there are limited number of PKI domains to be cross-certified.

In proxy server approach, there is a possibility of information leakage or compromise of the server when a proxy server acts as a client. This can be prevented by implementing the client with the restricted functionality so that the client could not cause information leakage or compromise of the server. This approach is even simpler than cross-certification approach because it only requires an existing server to implement a client with the restricted functionality.

In super CA approach, the seamless integration of different DRM systems is possible but there is a need to install a new certificate (that is, a super CA certificate) on the device. However, in the web environment, the root certificate is dynamic and can be changed any time. On the other hand, in the mobile world, the root certificate is fixed and cannot be changed once it is installed. Thus, super CA approach cannot work for interoperability between two OMA DRM v2.1 systems.

Chapter 4

DRM Interoperability Solution

4.1 Introduction

As a first step to resolve the problem, the previous chapter has discussed the interoperability between similar DRM systems but having different trust anchors. This chapter extends the idea to resolve the interoperability between different DRM systems (for example, the interoperability between OMA DRM v2.1 and Marlin). A content translator and a rights translator are required for the interoperability between different DRM systems.

This chapter first addresses the requirements on an interoperability solution. Next, it identifies the drawbacks of existing solutions and proposes a solution.

This chapter is organized as follows: Section 4.2 lists the requirements on an interoperability solution. Section 4.3 briefly describes the existing interoperability solutions. Section 4.4 looks at the problems in the existing interoperability solutions. Section 4.5 proposes the *Generic Interoperability Solution* and covers the necessary details to deploy the solution. Section 4.6 evaluates the proposed solution based on the requirements given in Section 4.2. Section 4.7 applies the proposed solution to resolve the interoperability between OMA DRM v2.1 and Marlin DRM systems. Finally, Section 4.8 concludes this chapter.

4.2 Requirements on an Interoperability Solution

The requirements identify the necessary attributes, capabilities, and characteristics in order to formulate the basis for the evaluation of a solution. A

DRM interoperability solution is evaluated based on the following requirements:

4.2.1 Security Aspects

Security is a primary requirement supposed to be fulfilled by an interoperability solution. It includes:

Threat Analysis

From security point of view, the proposed solution should withstand following security threats:

- It should not be possible to obtain a license without sharing or purchasing the content.
- The proposed solution should not allow to play the content without a license.
- The content should not be played more than allowed limit.
- The rights circumvention should be prevented.
- The entities in proposed solution should not expose the content or the keys.

Misbehave Identification

It should be possible to identify the misbehaving entities. For example, the entities exposing the content or the keys to any component except for allowed once.

Revocation

There should be a way to revoke the license of a misbehaving entity.

4.2.2 User Acceptance

It should be easy for the end user to share the content between different devices. Ideally, the interoperability solution should not require any change at the device end. For example, a change may be to install a new software on the device. Such sort of changes at the device end are not welcomed by a user.

4.2.3 Content Provider Acceptance

A content provider accepts the solution based on following parameters:

Device End Changes

Ideally, an interoperability solution should not require any implementation change at the device end because it is most likely that the devices are already in the market.

Backward Compatibility

An interoperability solution should not cause any trouble for the existing functionality. That is, the content and the license must be acquired as usual. Moreover, it should support the existing business models to increase the profitability.

Integration

An interoperability solution should be easy to integrate with the existing infrastructure. That is, the new entities (if introduced) should be easy to integrate with the existing architecture.

Deployment

An interoperability solution should be easy to deploy.

Maintenance and Troubleshooting

An interoperability solution should be easy to maintain and troubleshoot.

4.2.4 Supported Use Cases

An interoperability solution should support both use cases which are content downloading and streaming.

4.2.5 Cost

An interoperability solution should require as low cost as possible to deploy the solution.

4.2.6 Extendibility

An interoperability solution should be extendible. That is, it should be applicable to as many DRM systems as possible.

4.2.7 Performance Scalability

The performance scalability is an important requirement to evaluate a solution. It includes:

Distributed

An interoperability solution should be applicable in distributed environments.

Response Time

The time taken by the system to share the content from one device to another should be as small as possible.

Availability

The time a system is available to provide content sharing service. Ideally, a system should be available all the times.

4.2.8 Completeness

An interoperability solution should be complete and independent. That is, it should not depend on any other system.

4.2.9 Implementation Complexity

The implementation complexity of the interoperability solution should be as low as possible.

4.3 Existing Interoperability Solutions

This report mainly considers two practically relevant solutions for the interoperability between DRM systems. One solution is *OMArlin* for the interoperability between OMA and Marlin DRM systems. The other solution is an interoperability framework named *Coral* which requires an ecosystem such as DECE. It provides the interoperability between different DRM systems. OMArlin and Coral are described in Subsection 4.3.1 and Subsection 4.3.2, respectively.

4.3.1 OMArlin

OMArlin solution is provided by MDC for the interoperability between OMA v2.0 (or later version) and Marlin DRM systems. In OMArlin, a device first needs to register with an OMArlin service provider and then may join the user domain in order to obtain a license. If the device belongs to OMA DRM system then it registers with OMArlin service provider using the 4-pass registration protocol (see Appendix C). And if the device belongs to Marlin DRM system then it uses Marlin broadband protocols. OMArlin content format is supported by both OMA and Marlin devices. One of

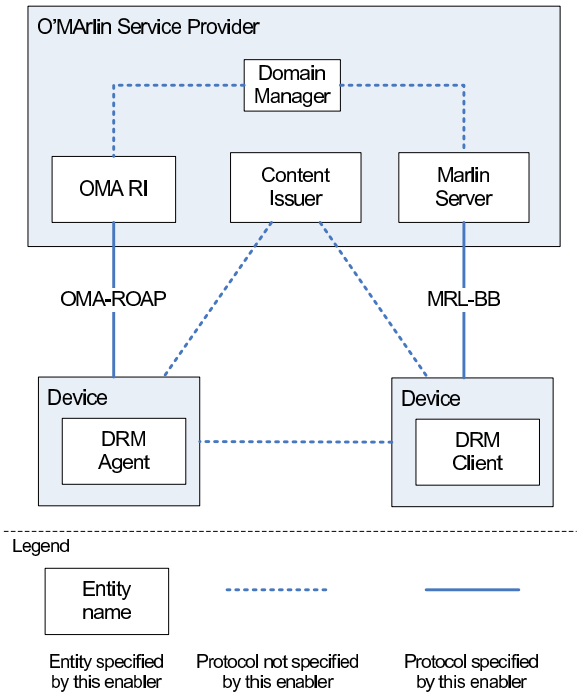

Figure 4.1: OMArlin architecture [All09]

the important aspects of OMArlin is that the content files contain meta information to obtain OMA and Marlin licenses.

In order to share the content, different devices register with a domain. A domain is controlled by a *Domain Manager* as shown in Figure 4.1. A domain can consist of different types of devices. For example, mobile phones, portable players, or PC players. Typically, a service provider puts a limit on the size[1] of a domain. Therefore, a limited number of devices can join a domain. The devices in a domain can freely exchange OMArlin content files with each other. Domain bound OMA RO and a Marlin licenses are embedded within an OMArlin content file. The devices of a domain access the embedded OMA RO or Marlin license and consume the content. For further information about OMArlin, see [All09].

4.3.2 Coral and DECE

Coral is a standardized framework for interoperability between DRM systems. Coral interoperability specifications are developed and maintained by *Coral Consortium*. Coral Consortium was founded in 2004 by HP, Intertrust, Philips, Panasonic, Samsun, Sony, and Twentieth Century Fox.

Coral provides the technical framework necessary to support DRM in-

[1]The size of a domain is defined as the maximum number of devices in a domain.

teroperability. It only works if there is an ecosystem. An ecosystem is a legal framework that defines DRM business models, policies, and implementation details. For example, how to map different license options between DRM systems. So far, the only publicly announced ecosystem is *Digital Entertainment Content Ecosystem (DECE)* developed by DECE Consortium. For further information about DECE, see [Con07].

In Coral, a license is derived from DRM independent *Rights-Tokens (RTs)*. The mapping of a rights-token to a license only depends on the rules and regulations of an ecosystem used along with Coral. A *Rights-Registry* is responsible to manage rights-tokens. It stores the granted rights-tokens upon request from authorized nodes and provides the functionality to check whether or not the usage rights are granted to a *Principal*. A principal represents a domain, a user, or a device in a particular DRM system. A *Domain Manager* manages *Principal Chains* according to the rules of a particular ecosystem. For example, to limit the maximum number of devices in a domain. A *Rights-Mediator* is the central entity in a Coral system. It receives the request from a client and invokes the appropriate identity service to map principal and content item names.

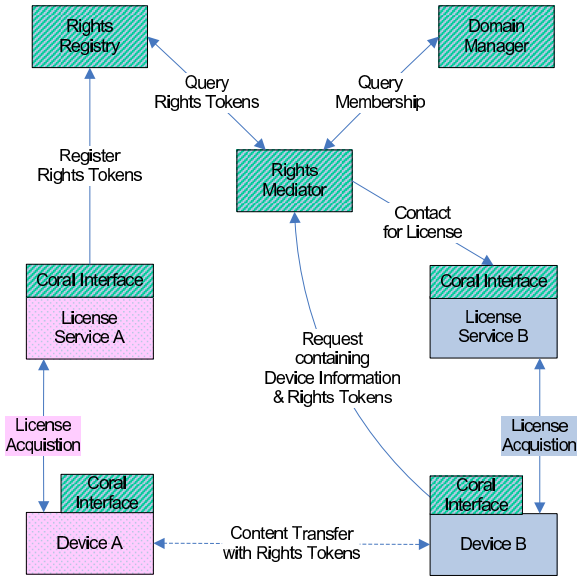

Figure 4.2: Coral interoperability framework architecture

An abstract level architecture of Coral interoperability framework is shown in Figure 4.2. Suppose a DRM system A issues the encrypted content and the license. It generates rights-tokens from the rights in a license. It transfers the encrypted content, the license, and the rights-tokens to a device (that is, *Device A*) belonging to its DRM system (that is, *DRM System A*). If a user desires to use the content on a device (that is, *Device B*) belonging

to other DRM system (that is, *DRM System B*) then the following actions happen: Device B obtains the encrypted content and the rights-tokens from Device A. Device B coral client role then contacts a rights-mediator. Device B coral client sends device information, the requested outcome, and the location of the rights-tokens to a rights mediator. A rights-mediator contacts with a domain manager to check membership of the requesting entity. After that it verifies the rights-tokens by querying a rights-registry. A rights-mediator contacts a DRM system B based license distribution service to request for a license for device B. That license distribution service generates and transfers the license to device B. Finally, device B can play the content. For further information about Coral, see [Con06, Con08a, Con08b].

4.4 Drawbacks in Existing Solutions

Now the question is whether the existing interoperability solutions fulfill the requirements (mentioned in Section 4.2) or not. The answer is that they do but partially. That is, they do not fulfill the requirements completely. This section identifies the requirements not fulfilled by the existing solutions. The problems in OMArlin and Coral are listed in Subsection 4.4.1 and Subsection 4.4.2, respectively.

4.4.1 OMArlin Limitations

This subsection evaluates OMArlin based on the requirements listed in 4.2. It only discusses the requirements those are not completely fulfilled by OMArlin.

User Acceptance

It requires the device to register with an OMArlin server provider. Typically, a user may not find it convenient to perform such sort of actions.

Supported Use Cases

OMArlin specification [All09] only discusses how to share downloading content but does not elaborate how to share streaming content.

Extendibility

In fact, it is not interoperability solution instead it is merger of OMA and Marlin DRM systems. Both DRM systems use common file format which cannot be generalized for every DRM system. Therefore, OMArlin is only limited to OMA and Marlin DRM systems.

4.4.2 Coral Limitations

This subsection evaluates Coral based on the requirements listed in 4.2. It only discusses the requirements which are not completely fulfilled by Coral.

User Acceptance

It requires a user to install a Coral client on the device. Again, a user may not find it convenient or may not be able to perform such sort of actions.

Content Provider Acceptance

So far, there is no publicly announced trial to use Coral by any content provider. It seems not to be accepted due to the following reasons:

Device End Changes It requires the device to install a Coral client.

Integration It does not define the implementation details. Therefore, it is hard to say whether the new entities are easy to integrate with the existing architecture. For example, Coral [Con08a] does not define who will perform content format translation. It may be an ecosystem which defines it but at least no such details are found in [Con07].

Deployment It is a complicated task to deploy a Coral framework along with an ecosystem. The main reason is that there are multiple Coral entities which need to be deployed. Moreover, the difficulty level of deployment may depend on how an ecosystem defines the rules.

Supported Use Cases

Coral [Con08a] only describes how to share downloading content but does not elaborate how to share streaming content.

Cost

It is a very costly solution because it needs to manage multiple entities. More entities mean more security risks and require to provide protection against those risks. Therefore, a higher cost is supposed to be spent to secure Coral entities. Moreover, a content provider has to pay an additional licensing fee to the authorities managing Coral and also needs to make a contract for an ecosystem.

Completeness

Coral itself is not a complete solution. It requires an ecosystem to define further details. For example, it provides implementation details and defines policies to share the content.

Implementation Complexity

If there are N number of DRM systems then Coral requires N number of translator to generate rights-tokens from the rights. It also requires N number of translators to translate rights-tokens back to the DRM-specific rights. So, the total number of required translators are twice the number of DRM systems. That is, $2 * N$. Therefore, the implementation complexity of Coral is on average.

4.5 The Proposed Interoperability Solution

After analyzing the existing interoperability solutions, it can be concluded that they have some drawbacks (mentioned in Section 4.4). So, there is a need of an interoperability solution which can fulfill all of the requirements (mentioned in Section 4.2). The architecture of the proposed interoperability solution is shown in Figure 4.3.

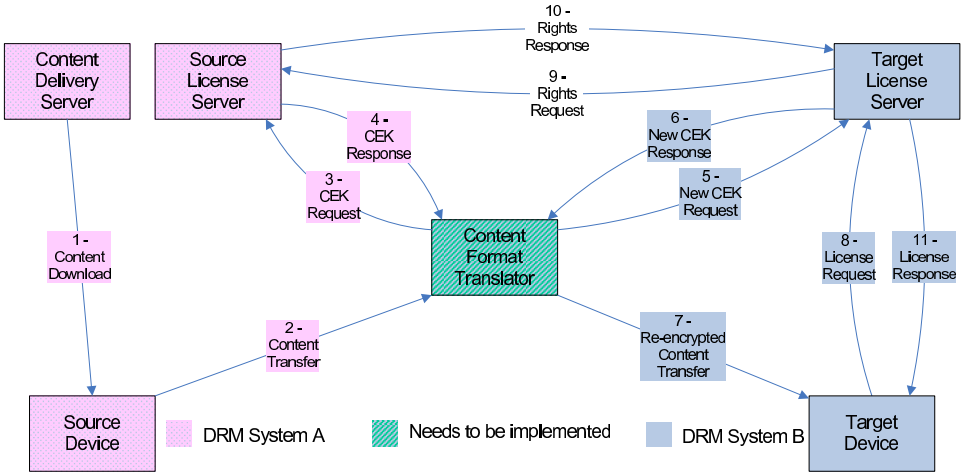

Figure 4.3: Generic Interoperability Solution

The proposed interoperability solution considers *Basic Content Transfer* use case (discussed in Subsection 2.5.2) to share the content between different devices. The idea is to introduce a new entity named *Content Format Translator (CFT)* to translate the content format from one DRM system to another. In this way, the devices can share the content with each other. That is, the content transfer from the source device to the target device. Once a target device obtains the translated content, it needs to acquire the corresponding license from its license server. Therefore, both license servers also needs to interact with each other to acquire the rights. The components and functions of the proposed interoperability solution are discussed in Subsection 4.5.1. Subsection 4.5.2 discusses the details of the messages

necessary to share the content between the devices belonging to different DRM systems.

A CFT has device certificates from the source as well as from the target DRM systems. Therefore, the source and the target devices share the content through a CFT considering that they are sharing the content with a device belonging to their own DRM system. A CFT also interacts with the source and the target license servers. Both DRM systems trust a CFT. The system is formally modeled in Subsection 4.5.3 that describes some rules and validates the security requirements. Subsection 4.5.4 looks at the possible optimizations in the proposed solution. Subsection 4.5.5 elaborates the additional elements in the current design to share the streaming content between different devices.

4.5.1 Components and Functions

Content Delivery Server

A content delivery server delivers the content to the devices. The proposed interoperability solution does not require any change at the content delivery server side.

Source Device

A source device is the device that transfers the content through a CFT to the devices belonging to other DRM systems. A source device obtains the content either directly from the content delivery server or from other devices using super-distribution. The proposed interoperability solution does not require any change at the source device end.

Target Device

A target device is the device that obtains the content from a source device through a CFT. The proposed interoperability solution does not require any change at the target device end.

Content Format Translator

A CFT is a central entity in the proposed interoperability solution. A CFT works as a bridge between the source and the target devices. A CFT conforms to the compliance and robustness rules from both DRM systems and has device certificates from the source as well as from the target DRM systems. Therefore, the source and the target devices share the content through a CFT considering that they are sharing the content with a device belonging to their own DRM system. A CFT translates the content format from the source to the target DRM system.

The symmetric key algorithms such as, AES, DES, or 3DES used to encrypt the content may be same or different in the source and the target DRM systems. They will also be treated as different if they either use different modes of operation or are applied in a different way. The proposed interoperability solution considers that they are different. However, the possibility of using same symmetric algorithm is covered in Subsection 4.5.4 under the heading *Same Algorithm for Content Encryption*.

If a source device desires to share the content, it sends the content with purchased information to a CFT. This sharing can be accomplished using super-distribution. Alternatively, a CFT and a device can share the content with each other by registering on a domain. The other option includes content sharing with a license embedded in the content file. In former case, a CFT needs to interact with the source license server to acquire the CEK while in later case, it does not require the CEK. It would also be interesting if a CFT can pay and obtain the rights from the license servers to transcrypt the content.

Let's suppose that the source DRM system uses AES while the target DRM system uses DES to encrypt the content. In such a case, a CFT has to transcrypt the content. That is, it needs to obtain CEK (that is, AES key) from the source DRM system to decrypt the content and also needs to obtain the new CEK (that is, DES key) in advance from a target DRM system to re-encrypt the content. A CFT can obtain both keys through licenses from the corresponding license servers because it has device certificates from both DRM systems.

To acquire both licenses, a CFT needs to know the locations of both license servers. A CFT knows the source license server location as it is already provided within the content file. That is, license acquisition URL. However, a CFT is assumed to know the target license server location. This can be managed by providing a mechanism to update the locations of target license servers inside a CFT. After acquiring licenses, a CFT transcrypts the content and sends it to the target device.

A CFT may be a combination of both software and hardware:

1. It can be a new standalone device used only for the purpose of transcryption.
2. It can be implemented in new devices as a value added feature.

In both cases, it should be made tamper proof using tamper proof hardware.

A CFT may be a piece of software:

1. It can be made available online through Web service provided by a new web server or may be offered by any (the source or the target) license server. In any case, it should be under secure environment.
2. It can also be installed as a plugin on the devices if they have capability to do so. In such a case, it should be made tamper proof using software

tamper proof technologies. The software tamper proof technologies include code obfuscation and Cloackware technology[2].

Source License Server

The proposed interoperability solution requires slight modifications at the source license server side. A source license server handles a CFT differently than other devices because it only has to provide the CEK instead of issue a complete license to a CFT. Additionally, it implements a mechanism to receive rights-request from a target license server and replies with the requested rights.

Target License Server

To acquire a license, a target device interacts with the target license server. A target license server requires to store the new CEK which is issued to the CFT in advance. Moreover, a target license server also needs to implement a mechanism to acquire the right from a source license server. To manage the new CEK, a target license server implements a table containing three columns to store content ID, new CEK, and source license server URL. Once a CFT sends a request containing content ID and source license server URL to the target license server, the target license server stores content ID, new CEK, and source license server URL in the table. On a license request from a target device, a target license server first checks content ID in the table. If it is found in the table then the target license server treats that the content is from any other DRM system. Otherwise, the target license server generates the license as usual.

A target license servers needs to acquire the corresponding rights from the source license server. To acquire the rights, a target license server contacts the source license server whose location can be obtained from the corresponding entry against content ID in the table.

Before acquiring the rights, both license server needs to perform mutual authentication. They can perform mutual authentication in three different ways:

1. Both license servers or their root CAs performs the cross-certification with each other.
2. A super CA works as a bridge by issuing certificates to both license servers or their root CAs.
3. A target license server implements a client of a source license server (recommended). In this case a target license server is working as a proxy server between a target device and a source license server.

[2] `http://www.cloakware.com/`

All three ways are already discussed in Section 3.2, Section 3.3, and Section 3.4, respectively.

Rights Translator

Every DRM system has its own REL to express the rights. The rights expression in one DRM system may differ from others. Therefore, there is need to perform a license translation before putting the rights in a license intended for the target device. The proposed interoperability solution requires to implement a rights translator either by the source or the target license server (recommended). For further information about a rights translator, an interested reader is referred to [CM05, JHMO06].

Key Generator

To generate a new CEK, a key generator needs to be implemented. It can be implemented by a target license server or a CFT. If a target license server implements it then a CFT can request and obtain a new CEK from the target license server. Otherwise, a CFT generates a new CEK and sends an approval request to the target license server and gets the response.

Before deciding where a key generator should be implemented, let's have a look at content encryption and signing[3]. A CFT cannot sign the content because it does not have a license server role. However, it re-signs the content by interacting with a target license serer. It is necessary to know how the content is encrypted and signed. There are two possibilities to encrypt and sign the content:

Encrypt then Sign Mostly the content is encrypted then signed. In this case, it is recommended to implement a key generator at the CFT end. Because a CFT can generate a new CEK, encrypts the content, calculates the hash on the encrypted contents, and sends the new CEK and the calculated hash to the target license server. A target license server stores the new CEK, replies with approval of the new CEK, and also sends the signed hash back to the CFT. So, it requires only a single interaction between a CFT and a target license server if a key generated is implemented by a CFT.

A key generator may also be implemented at the target license server. In such a case a CFT needs to interact two times with the target license server. First time to acquire the new CEK so that it can encrypt the content. While second time to get the signed hash on the encrypted content.

[3]The content re-signing may also be required for interoperability between similar DRM systems.

Sign then Encrypt The content may be signed then encrypted. In this case, a key generator may either be implemented at the CFT or the target license server. In former case, a hash on the content is calculated, the new CEK is generated, and a request containing calculated hash and the new CEK is sent to the target license server which stores the new CEK and replies by signing the calculated hash to the CFT. In later case, a hash on the content is calculated and sent to the target license server which receives the request, generates a new CEK, signs the calculated hash, and sends the response to the CFT.

This report considers that the content is encrypted then signed and a key generated is implemented by a CFT.

4.5.2 Sequence of Messages and their Description

The proposed interoperability solution considers *Basic Content Transfer* use case (discussed in Subsection 2.5.2) to share the content between different devices. The proposed interoperability solution assumes that:

1. A rights translator is implemented by a target license server.
2. A key generator is implemented by a CFT.
3. A proxy server approach is implemented by a target DRM system. That is, a target license server implements the source DRM client with the restricted functionality.
4. A CFT knows the location of the target license server and implements a mechanism to update that.

This subsection lists the messages in a sequence required to share the content from one device to another. Each message is numbered as depicted in Figure 4.3 and described in detail.

Message 1: Content Download

A source device downloads the content with license acquisition information from a content delivery server. This message is specified by a source DRM system. Generally, it contains the following parameters:

- *Metadata:* It holds the following information:
  - *Content ID:* A unique identification number to identify the content
  - *License Acquisition URL:* The location[4] to download a license

[4]Alternatively, there may be a trigger to acquire a license.

  - *DRM Specific Parameters to Acquire License:* (Where needed/applicable)
  - *Signature:* (Sometimes)

- *Content Object:* It is the content encrypted with a CEK.

Message 2: Content Transfer

A source device transfers the content to a CFT. This content transfer follows super-distribution (discussed in Subsection 2.5.2). This message contains same parameters as already listed in message 1.

Message 3: CEK Request

A CFT requests for CEK to the source license server. This message is specific to the source DRM system. Generally, it contains the following parameters:

- *Content ID:* A unique identification number to identify the content
- *Device Certificate:* It may be a PKC, a PKC ID, or a URL to download PKC. That is, a CFT certificate issued by a source DRM system
- *DRM Specific Parameters to Acquire License:* (Where needed/applicable)

Message 4: CEK Response

In response to the CEK request, a source license server replies with a CEK. This message is a customized message (that is, a license response without the rights[5]) but it is still specific to the source DRM system. Generally, it contains the following parameters:

- *Metadata:* It is signed and contains the following parameters:
  - *License ID:* A unique identification number to identify the license
  - *Content ID:* A unique identification number to identify the content
  - Signature
- *CEK:* It is encrypted with the CFT public key (of the source DRM system)

[5]Or may be with the rights to *transcrypt* the content.

Message 5: New CEK Request

A CFT requests for a new CEK to the target license server. A new CEK is generated by a CFT (or a target license server) in advance. A target license server stores it and provides it in the license when a target device requests for that license. This message[6] is specific to the target DRM system. Generally, it contains the following parameters:

- *Metadata:* It is signed and contains the following parameters:
  - *Content ID:* A unique identification number to identify the content
  - *Device Certificate:* It may be a PKC, a PKC ID, or a URL to download PKC. That is, a CFT certificate issued by a target DRM system
  - *Hash:* Hash of the transcrypted content
- *New CEK:* It is encrypted with public key of the target license server

Message 6: New CEK Response

In response to the new CEK request, a target license server replies after storing the new CEK. This message is a customized message (that is, a license response without the rights) but it is still specific to the target DRM system. Generally, it contains the following parameters:

- *Metadata:* It is signed and contains the following parameters:
  - *License ID:* A unique identification number to identify the license
  - *Content ID:* A unique identification number to identify the content
  - *Signed Hash:* It is the signed hash (that is, the hash sent in the new CEK request is now sent back to the CFT by the target DRM system)
  - *DRM Specific Parameters to Acquire License:* (Where needed/applicable)
  - Signature
- *CEK:* It is encrypted with the CFT public key[7] (of the target DRM system)

[6] This message is new for a target license server.
[7] It could also be encrypted with a domain key.

Message 7: Re-encrypted Content Transfer

A CFT transfers the content to the target device. This content transfer follows super-distribution (discussed in Subsection 2.5.2). This message contains same parameters as already listed in message 1.

Message 8: License Request

A target device requests for a license to the target license server. This message is specific to the target DRM system. Generally, it contains the following parameters:

- *Content ID:* A unique identification number to identify the content
- *Device Certificate:* It may be a PKC, a PKC ID, or a URL to download PKC. That is, a certificate issued by a target DRM system
- *DRM Specific Parameters to Acquire License:* (Where needed/applicable)

Message 9: Rights Request

After receiving a license request, a target license server searches the content ID in the stored table. If it is found in the table then it picks the corresponding source URL and the new CEK. A target license server uses this source URL to request for the rights. This message is specific to the source DRM system. Generally, it contains the following parameters:

- *Content ID:* A unique identification number to identify the content
- *Device Certificate:* It may be a PKC, a PKC ID, or a URL to download PKC. That is, a certificate issued by a source DRM system
- *DRM Specific Parameters to Acquire License:* (Where needed/applicable)

Message 10: Rights Response

In response to the rights request, a source license server replies with the rights. This message is specific to the source DRM system. Generally, it contains the following parameters:

- *Metadata:* It is signed and contains the following parameters:
  - *License ID:* A unique identification number to identify the license
  - *Content ID:* A unique identification number to identify the content
  - Signature
- *Rights:* The rights to play the content signed by a source license server

Message 11: License Response

Finally, a target license server replies with the license containing the new CEK (picked from the table) and the rights (received from the source license server). This message is specific to the target DRM system. Generally, it contains the following parameters:

- *Metadata:* It is signed and contains the following parameters:
  - *License ID:* A unique identification number to identify the license
  - *Content ID:* A unique identification number to identify the content
  - Signature
- *CEK:* It is encrypted with the target device public key
- *Rights:* The rights to play the content signed by a target license server

4.5.3 Formal Model

This subsection formally models the system and shows how the proposed solution works. It identifies set of agents, domain objects, and actions[8]. Next, it lists the assumptions and then set of rules allowed in the system. Finally, it applies those rules to share the content between different devices.

Set of Agents

Let *a* and *b* range over a set of agents each having a role from {*device, CFT, license-server, content-delivery-server, CA*}.

Set of Domain Objects

Let *d*, *d2*, and *d3* range over a set of domain objects each having a type from {*license, content*}. To formalize the system, a license and the content objects need to have some properties. A license object includes the following properties:

- *license.CEK:* This property either stores a CEK or may be *NULL*.
- *license.rights:* This property either holds the rights object or may be *NULL*.

The content object includes the following properties:

- *content.purchased:* If the content is purchased then its value would be *TRUE* otherwise *FALSE*.

[8]Although a portion of this model is already defined in Chapter 3 but for the ease of understanding a complete model, this section repeats some agents and actions.

– *content.purchasedInfo:* If the content is purchased then it would hold an object containing the content purchased information. Otherwise, it would be *NULL*.

Set of Actions

To model the system, all necessary actions are defined as follows:

$$requests(a, d, b)$$

meaning a requests b to acquire d; where a and b are agents each having a role from {license-server, CFT, device} and {content-delivery-server, license-server}, respectively, while d is a domain object having a type from {license, content}.

$$delivers(a, d, b)$$

meaning a delivers a d to b; where a and b are agents each having a role from {content-delivery-server, license-server, CFT, device} and {license-server, CFT, device}, respectively, while d is a domain object having a type from {license, content}.

$$allows(a, d, b)$$

meaning a is willing to send d to b; where a and b are agents each having a role from {CFT, device} and {device}, respectively, while d is a domain object having type content.

$$certifies(a, b)$$

meaning a certifies b; where a and b are agents having a role from {CA} and {CA, license-server, CFT, device}, respectively.

$$authenticates(a, b)$$

meaning a and b authenticates each other; where both a and b are agents having roles from {license-server, CFT, device}.

$$transcrypts(a, d, d2, d3)$$

meaning a decrypts d using d2 and re-encrypts it using d3; where a is an agent having role CFT while d, d2, and d3 are domain objects having types content, license, and license, respectively.

$$pays(a, d, n\$)$$

meaning a pays n$ for d; where a is an agent having device role while d is a domain object having type content.

$$plays(a, d)$$

meaning a plays d; where a is an agent having device role while d is a domain object having type content.

The following function is implemented by a device to check if a license is valid.

$$islicensevalid(license)$$

If license is valid then it returns TRUE*, otherwise it returns* FALSE.

Additionally, there are two functions implemented by license-servers:

$$issuelicensecount(content)$$

If content.purchased is TRUE *then it checks content.purchasedInfo object to verify the play count and returns 0 if the content cannot be played any more or a positive value indicating play count. This play count is managed by a source license server.*

$$storeslicense(content)$$

If content.purchased is TRUE *then it checks content.purchasedInfo object to verify the presence of the license at the license server and returns* TRUE *if the license is present otherwise it returns* FALSE*. This function is implemented by a target license server.*

Assumptions

The model assumes that:

1. A CFT is a device in the home environment and transcrypts the content under the trusted environment using tamper-proof hardware.
2. A CFT gets a license without the rights. That is, it can only acquire the CEK so that it can re-encrypt the content.
3. A CFT delivers the transcrypted content to a target device as instructed by a source device.
4. A device is robust. That is, it will never reveal any key and only releases the content according to the policy in a license.
5. A device is compliant. That is, it will enforce the rights within a license.
6. Every DRM system has its own root CA.
7. A device can share the content with another device delegating only one time rights to play the content.

8. A target license server acquires the rights from the source license server. Before acquisition of the rights, they authenticate each other using cross-certificate, super CA, or proxy server approach already discussed in Chapter 3.
9. In both DRM systems, a license is encrypted with the device public key while the content is encrypted with a CEK and state of the art encryption algorithms are secure enough to provide confidentiality and integrity to the CEK and the content on the communication channel.
10. Both license and content delivery servers are robust.
11. Both license servers allow content sharing.

Set of Rules

Before describing the rules, it is necessary to define different agents and domain objects in the system. They are defined as follows:
ca1: A root CA in a source DRM system
ca2: A root CA in a target DRM system
d1: A source device having device certificate from $ca1$
d2: A target device having device certificate from $ca2$
ls1: A source license server
ls2: A target license server
cft: CFT (to transcrypt the content) having device certificates from $ca1$ and $ca2$
cds: A content delivery server of a source DRM system
l1: A license containing CEK (but without the rights, that is $l1.rights = NULL$) issued by $ls1$ to cft
l2: A license containing new CEK (but without the rights, that is $l2.rights = NULL$) issued by $ls2$ to cft
l_r*:* A license containing only the rights (but without CEK, that is $l_r.CEK = NULL$) issued by $ls1$ to $ls2$
l: A license issued by $ls2$ to $d2$ where $l.rights = l_r$.rights and $l.CEK = l2.CEK$
l0: A license issued by $ls1$ to $d1$
c: The content issued by *cds*

Following is a minimal set of rules to describe how the system works:

$$pays(d1, c, n\$) \wedge requests(d1, c, cds) \rightarrow delivers(cds, c, d1) \wedge c.purchased = TRUE \wedge issuelicensecount(c) > 0 \quad (4.1)$$

meaning if d1 pays n$ for the content c and requests cds to deliver it then cds delivers purchased c to d1 where c can be played 1 or more times.

$$allows(d1, c, d2) \wedge c.purchased = TRUE \wedge certifies(ca1, d1) \wedge certifies(ca2, d2) \rightarrow delivers(d1, c, cft) \wedge allows(cft, c, d2) \quad (4.2)$$

meaning if d1 allows to share purchased c with d2 where both d1 and d2 have certificates from ca1 and ca2, respectively (that is, d1 and d2 belong to different DRM systems) then d1 delivers c to cft and instructs cft to deliver c to d2.

$$delivers(d1, c, cft) \wedge allows(cft, c, d2) \wedge certifies(ca1, d1) \wedge certifies(ca2, d2) \wedge certifies(ca1, cft) \wedge certifies(ca2, cft) \wedge certifies(ca1, ls1) \wedge certifies(ca2, ls2) \rightarrow authenticates(ls1, cft) \wedge authenticates(ls2, cft) \wedge authenticates(ls1, d1) \wedge authenticates(ls2, d2) \quad (4.3)$$

meaning if d1 delivers c to cft and cft gets instruction to deliver c to d2 where cft has device certificates from both DRM systems then cft can perform mutual authentication with ls1 and ls2. Moreover, ls1 and ls2 can perform mutual authentication with d1 and d2, respectively.

$$authenticates(ls1, cft) \wedge issuelicensecount(c) > 0 \rightarrow delivers(ls1, l1, cft) \quad (4.4)$$

meaning if ls1 performs mutual authentication with cft and c can be played at least once then ls1 delivers l1 to cft.

$$delivers(ls1, l1, cft) \wedge authenticates(ls2, cft) \rightarrow delivers(ls2, l2, cft) \wedge storeslicense(c) := TRUE \quad (4.5)$$

meaning if ls1 delivers l1 to cft and ls2 authenticates with cft then ls2 delivers l2 to cft and l2 is stored by ls2.

$$delivers(ls1, l1, cft) \wedge delivers(ls2, l2, cft) \wedge delivers(d1, c, cft) \rightarrow transcrypts(cft, c, l1, l2) \quad (4.6)$$

meaning if ls1 and ls2 delivers l1 and l2, respectively to cft and d1 delivers c to cft then cft decrypts c with l1 and re-encrypts with l2.

$$allows(cft, c, d2) \wedge transcrypts(cft, c, l1, l2) \rightarrow delivers(cft, c, d2) \quad (4.7)$$

meaning if cft is instructed to deliver c to d2 and it has transcrypted c then it delivers c to d2.

$$delivers(cft, c, d2) \wedge c.purchased = TRUE \wedge requests(d2, l, ls2) \wedge authenticates(ls2, d2) \wedge storeslicense(c) = TRUE \rightarrow delivers(ls2, l_r, ls1) \wedge issuelicensecount(c) := issuelicensecount(c) - 1 \wedge storeslicense(c) := FALSE \wedge delivers(ls2, l, d2) \quad (4.8)$$

meaning if cft delivers purchased c to d2 and d2 requests to ls2 for l and d2 mutually authenticates with ls2 and ls2 stores l2 for c then ls2 first acquires l_r from ls1, ls1 decrements play count by one, ls2 deletes l2, and finally ls2 delivers l to d2.

$$delivers(*, c, d2) \wedge delivers(ls2, l, d2) \wedge islicensevalid(l) = TRUE \rightarrow plays(d2, c) \quad (4.9)$$

meaning if d2 gets c and ls2 delivers l to d2 and l is valid then d2 can play c.

$$requests(d1, l0, ls1) \wedge authenticates(ls1, d1) \wedge delivers(cds, c, d1) \wedge c.purchased = TRUE \wedge playscount(c) > 0 \rightarrow delivers(ls1, l0, d1) \wedge issuelicensecount(c) := issuelicensecount(c) - 1 \quad (4.10)$$

meaning if d1 requests and mutually authenticates with ls1 to acquire l0 for purchased c having play count 1 or more then ls1 delivers l0 to d1 and ls1 decrements play count by one.

Applying Rules to Share Content between Devices

To show how the proposed solution works, consider the following derivation. It applies above mentioned rules to share the content between different devices.

It is assumed that $d1$ pays $n\$$ for c, requests cds to get c, and desires to share c with $d2$. Formally, it can be expressed as:

$$pays(d1, c, n\$) \quad (4.11)$$

$$requests(d1, c, cds) \quad (4.12)$$

$$allows(d1, c, d2) \tag{4.13}$$

First of all, combining (4.11) and (4.12) using rule (4.1) yields:

$$delivers(cds, c, d1) \wedge c.purchased = TRUE \wedge issuelicensecount(c) > 0 \tag{4.14}$$

By definition of $d1$ and $d2$:

$$certifies(ca1, d1) \wedge certifies(ca2, d2) \tag{4.15}$$

combining (4.13), (4.14), and (4.15) using rule (4.2) yields:

$$delivers(d1, c, cft) \wedge allows(cft, c, d2) \tag{4.16}$$

By definition of cft, $ls1$, and $ls2$:

$$certifies(ca1, cft) \wedge certifies(ca2, cft) \wedge certifies(ca1, ls1) \wedge certifies(ca2, ls2) \tag{4.17}$$

combining (4.16), (4.15), and (4.17) using rule (4.3) yields:

$$authenticates(ls1, cft) \wedge authenticates(ls2, cft) \wedge authenticates(ls1, d1) \wedge authenticates(ls2, d2) \tag{4.18}$$

combining (4.18) and (4.14) using rule (4.4) yields:

$$delivers(ls1, l1, cft) \tag{4.19}$$

combining (4.19) and (4.18) using rule (4.5) yields:

$$delivers(ls2, l2, cft) \wedge storeslicense(c) = TRUE \tag{4.20}$$

combining (4.19), (4.20), and (4.16) using rule (4.6) yields:

$$transcrypts(cft, c, l1, l2) \tag{4.21}$$

combining (4.16) and (4.21) using rule (4.7) yields:

$$delivers(cft, c, d2) \tag{4.22}$$

It is also assumed that $d2$ requests to $ls2$ for l which is valid at the time of play:

$$requests(d2, l, ls2) \tag{4.23}$$

$$islicensevalid(l) = TRUE \tag{4.24}$$

combining (4.22), (4.14), (4.23), (4.18), and (4.20) using rule (4.8) yields:

$$delivers(ls2, l_r, ls1) \wedge issuelicensecount(c) := issuelicensecount(c) - 1 \wedge storeslicense(c) = FALSE \wedge delivers(ls2, l, d2) \tag{4.25}$$

Finally, combining (4.22), (4.25), and (4.24) using rule (4.9) yields:

$$plays(d2, c) \tag{4.26}$$

This is how the proposed solution works.

4.5.4 Possible Optimizations in the Proposed Solution

There are some cases where the proposed interoperability solution can be optimized. They are described as follows:

Stateless License

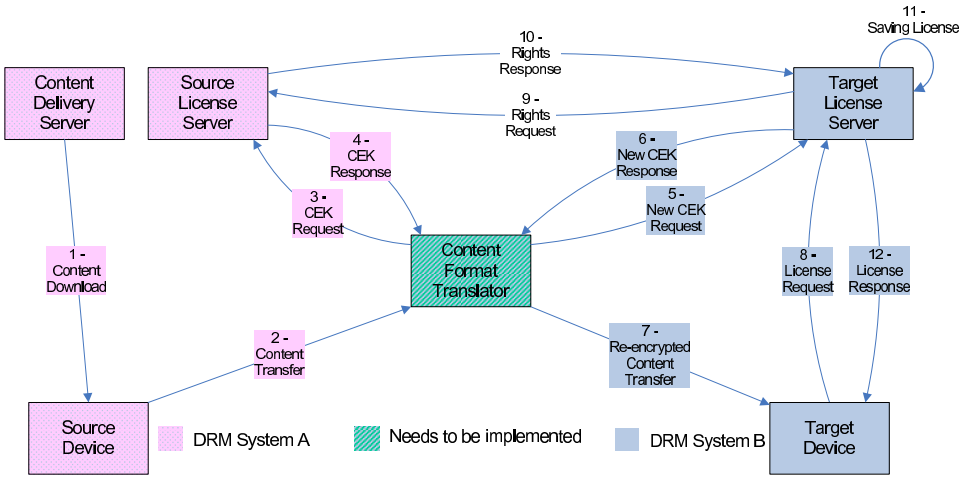

Figure 4.4: Stateless license

If a license is stateless then a target license server does not need to acquire the rights every time from the source license server. That is, a target license server can save that license locally for the subsequent requests as shown in Figure 4.4.

Same Algorithm for Content Encryption

If both DRM systems use the same algorithm, the same mode of operation and applies the algorithm in the same way to encrypt the content then a CFT does not need to interact with the source license server to acquire the CEK (as shown in Figure 4.5). That is, there is no need to transcrypt the content. However, this case would still require a target license server to acquire the rights as well as the corresponding CEK from the source license server.

Content with Embedded License

If the devices desire to share the content with embedded license then both (the source and the target) license servers do not need to interact with each other to acquire the rights (as shown in Figure 4.6) because they are already in the content file. However, it requires a CFT to implement the

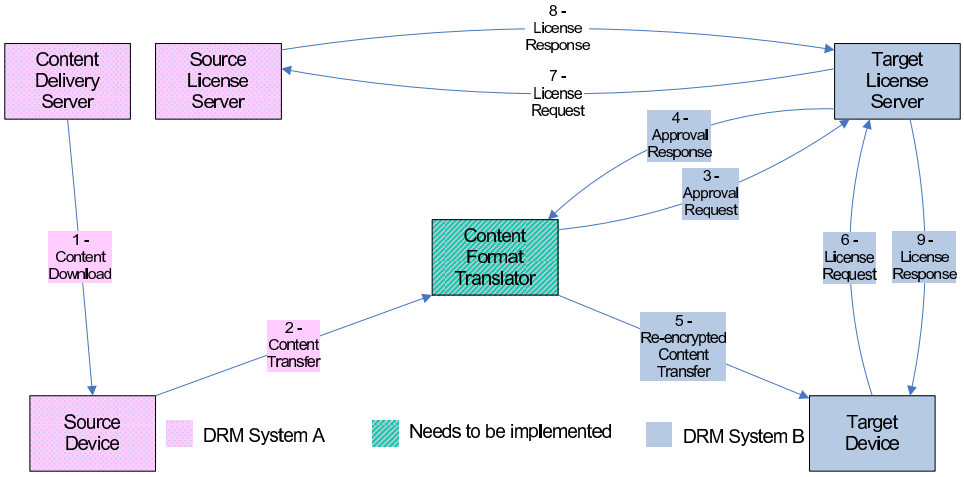

Figure 4.5: Same algorithm for content encryption

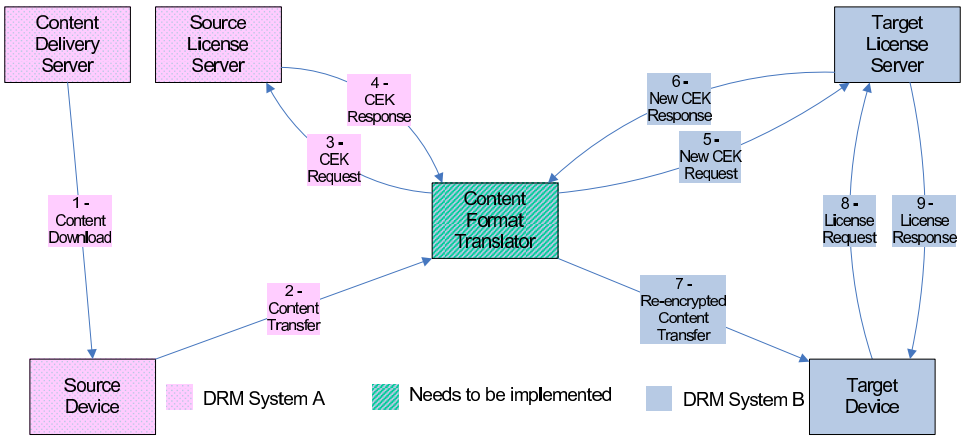

Figure 4.6: Content with embedded license

rights translators to translate the rights from the source to the target DRM system.

4.5.5 Streaming Case

It is an extension to the generic interoperability solution. It considers *Basic Stream Transfer* use case (discussed in Subsection 2.5.3) to share the streaming content between different devices. The idea is same as in Figure 4.3 except message 2. Before sharing the streaming content, a source device sends a bookmark (that is, message 2a) to a CFT which forwards it (that is, message 2b) to the target device as shown in Figure 4.7. Next, a target device requests for the streaming (that is, message 2c) to the CFT. A CFT forwards that streaming request (that is, message 2d) to the content delivery

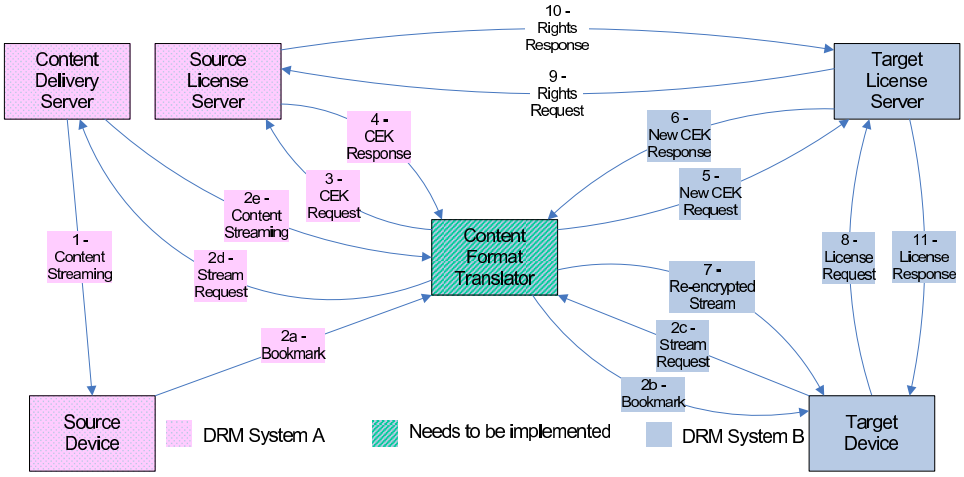

Figure 4.7: Streaming case of Generic Interoperability Solution

server which replies with content streaming (that is, message 2e). In this communication (that is, from message 2a to message 2e), a CFT works as a bridge between a source and a target device.

4.6 Evaluation of the Proposal

This section focuses on security aspects and evaluates the proposed interoperability solution based on the requirements listed in Section 4.2.

4.6.1 Security Aspects

The proposed interoperability solution introduces a new entity, that is, a CFT. Both DRM systems issue device certificates to a CFT and it has to obey the compliance and robustness rules of both DRM systems. A CFT does not compromise the security of the existing entities. The security considerations for a CFT are as follows:

Threat Analysis

Obtaining a License without Content Sharing or Purchasing If the content is neither shared nor purchased then it is not possible to obtain a license. According to rule (4.8) and rule (4.10), a license is delivered either the content is shared or purchased, respectively.

Playing the Content without a License The proposed interoperability solution does not allow to play the content without a license, see rule (4.9). Actually, the content is encrypted all the times (before and after the transcryption). Both DRM systems do not allow to play the

content without a license. More precisely, it is same as to play the content without a license within a DRM system.

Playing the Content More than Allowed The content is purchased having the play count one or more, see rule (4.1). Once a license is delivered to a device, the play count is decremented by one as mentioned in rules (4.8) and (4.10). If the play count is zero then it is not possible to acquire the corresponding license as per rule (4.4) and (4.10).

Sharing the Content without Device Consent The content can only be shared if a device allows it, see rule (4.2). Otherwise, the content sharing is not possible.

Circumventing the Rights in Target DRM System The proposed interoperability solution does not allow the target device to circumvent the rights because of the compliance rules enforced by a target DRM system, see assumption 5.

CFT Exposing the Content A CFT cannot expose the content because it has a device certificate from both (the source and the target) DRM systems and a device is robust, see assumption 4. Moreover, a CFT cannot expose the content during transcryption that is performed under the trusted environment, see assumption 1.

CFT Exposing the CEK A CFT cannot expose the CEK because it implements the robustness rules defined by a source DRM system, see assumption 4.

CFT Exposing the new CEK A CFT cannot expose the CEK because it implements the robustness rules defined by a target DRM system, see assumption 4.

CFT Impersonating Target License Server Before describing this threat, it is important to know that a CFT stores the location of a target license server and implements a mechanism to update that. The threat is to update a CFT with the location of any other license server instead of the real one. In such a case, a target device will try to interact with that license server. However, it is not possible to acquire the license for a target device because the mutual authentication (between the device and the license server) would fail, see rule (4.3).

Compromisation of Proxy Server In the proxy server approach, a target license server implements a source DRM client. In this case, it may be possible to leak or compromise the target license server through the source DRM client. The proposed interoperability solution prevents it by implementing the source DRM client with the restricted functionality.

Denial of Service (DoS) Attack A CFT is vulnerable to DoS attack because an attacker can send dummy requests to it. If CFT is a device in a home environment then it is not a potential threat. However, if a CFT is a Web service then DoS attack is really a potential threat. In such case, a CFT should receive a request containing license acquisition information from a source device to transcrypt the content. Next, it should verify from a source license server if the license is really purchased. If verification is successful then it should start fetching the content from the source device for transcryption. Otherwise, it MUST terminate the protocol. In this way, the severity of DoS attack can be diminished but cannot be eliminated completely.

Misbehave Identification

A DRM system can use existing mechanism (as used to identify the misbehaves of a device) to identify the misbehaves of a CFT because it has device certificates from both DRM systems.

Revocation

Similarly, a DRM system can use existing mechanism such as a CRL or an OCSP to revoke CFT certificate. If the certificate is revoked, it would not be possible for a CFT to function anymore because the mutual authentication between the license server and CFT would fail, see rule (4.3).

4.6.2 User Acceptance

Ease of use for the end user because it does not require a user to install any additional software on the device.

4.6.3 Content Provider Acceptance

A content provider may accept the proposed interoperability solution because of the following reasons:

Device End Changes

The proposed interoperability solution does not require any implementation change at the device end.

Backward Compatibility

The proposed interoperability solution does not cause any trouble for the existing functionality. Moreover, it supports the exiting business models.

Integration

The proposed interoperability solution introduces a CFT which is easy to integrate with the existing architecture.

Deployment

The proposed interoperability solution is easy to deploy because it requires to setup only a single entity, that is a CFT. Moreover, it requires some changes to both license servers to interact with each other and with a CFT.

Maintenance and Troubleshooting

The proposed interoperability solution seems easy to maintain and troubleshoot but this should be verified by implementing it, which is subject to future work.

4.6.4 Supported Use Cases

The proposed interoperability solution supports both content downloading and streaming use cases.

4.6.5 Cost

The proposed interoperability solution is a low cost solution. It does not require any additional cost such as contract agreement with any third party. A CFT is cheaper device because it only has device certificates and manages additional resources to transcrypt the content. However, it may be a bit expensive than a source or a target device.

For content streaming use case, it may get expensive because streaming requires additional resources to perform real-time transcryption.

4.6.6 Extendibility

The proposed interoperability solution is extendible. It may be applicable to a number of DRM systems.

4.6.7 Performance Scalability

The performance scalability covers:

Distributed

The interoperability solution is inherently distributed as CFTs are in the home environments.

Response Time

The response time can be determined after implementing the proposed interoperability solution, that is future work.

Availability

The availability can be calculated after implementing the proposed interoperability solution, that is future work.

4.6.8 Completeness

The proposed interoperability solution is complete and independent. It does not depend on any other system such as on ecosystem.

4.6.9 Implementation Complexity

The proposed solution does not generate intermediate rights[9] instead it performs direct translations of the rights from one DRM system to another. If there are N number of DRM systems then the required number of translators to translate the rights from a source DRM system to a target DRM system are:

$$2 * \begin{pmatrix} N \\ 2 \end{pmatrix} = N(N-1)$$

Therefore, the implementation complexity of the proposed solution is reasonably high. However, the number of (interoperable) DRM systems are always very limited.

4.7 Interoperability between OMA DRM v2.1 and Marlin DRM

This section applies the proposed solution to resolve the interoperability between OMA DRM v2.1 and Marlin DRM. It is assumed that a CFT is a device in the home environment. First of all, a Marlin device downloads the encrypted content as shown in Figure 4.8. After that it pays for the corresponding rights and obtains an acknowledgement. Next, it transfers the encrypted content to the CFT. Alternatively, a Marlin device can acquire the license and can transfer the encrypted content along with a license to the CFT[10] as mentioned in Subsection *Move and Copy Action* of [Com06a].

A CFT request a Marlin license server to obtain the action token. The action token specifies a sequence of actions required to acquire the license

[9] In Coral, the rights tokens are intermediate rights.

[10] This case does not require a CFT to interact with Marlin license server to acquire the CEK.

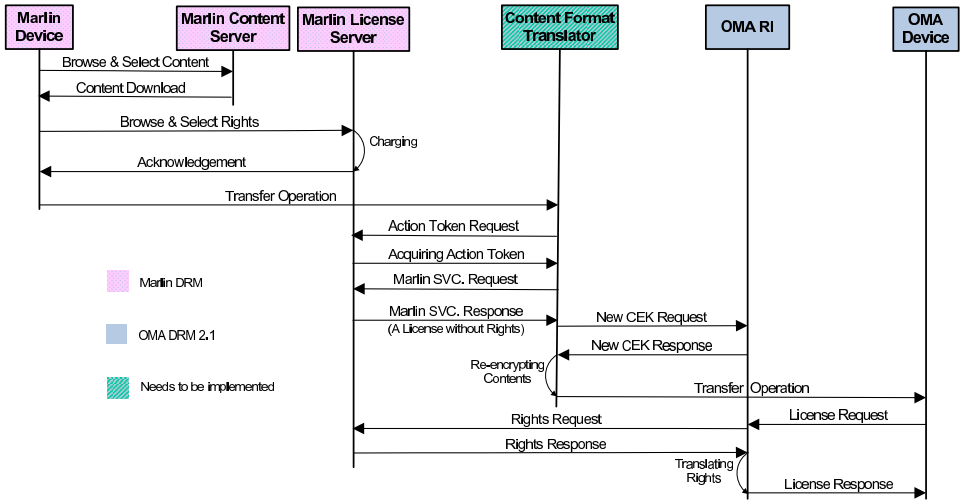

Figure 4.8: Interoperability between OMA DRM v2.1 and Marlin DRM

(to know further about action token, see [Com07]). After acquiring action token, a CFT requests Marlin license server to obtain the CEK. Next, it requests OMA RI to generate a new CEK in advance. A CFT decrypts the encrypted content with the CEK and re-encrypts with the new CEK. After transcrypting the content, it transfers the content to an OMA device.

After obtaining the transcrypted content, OMA device requests for the license. Once OMA RI receives the license request, it interacts with Marlin license server to obtain the rights. After obtaining the rights, OMA RI translates them. Next, it puts the translated rights and the new CEK in an RO. This RO is encrypted with OMA device public key and sent to OMA device. Finally, OMA device obtains that RO and play the content.

4.8 Synopsis

This chapter has proposed a generic interoperability solution to share the content between different devices. It has been seen that there are problems in the existing interoperability solutions. The proposed solution fulfills all of the requirements to be expected from an interoperability solution. Table 4.1 summarizes this chapter by comparing OMArlin, Coral, and the proposed solution based on the requirements (listed in Section 4.2).

Table 4.1: Comparison of interoperability solutions

Requirements	**OMArlin**	**Coral**	**Proposed Solution**
Security Aspects			
–*Misbehave Identification*	Yes	Yes	Yes
–*Revocation*	Yes	Yes	Yes
–*DoS Attack*	Yes	Yes	Yes
User Acceptance	Average	Low	High
–*Required Action*	Registration	Client Installation	No
Content Provider Acceptance			
–*Device End Changes*	No	Yes	No
–*Backward Compatibility*	Yes	Yes	Yes
–*Integration*	Average	Difficult	Easy
–*Deployment*	Average	Difficult	Easy
–*Maintenance and Troubleshooting*	Average	Difficult	Easy
Supported Use Cases	Download	Download	Download Streaming
Cost	Low	High	Low
–*Required Certificates*	1 Server	1 Server	N Device
–*No. of New Entities*	Single	Multiple	Single
Extendability	No	Yes	Yes
Performance Scalability			
–*Distributed*	Yes	Yes	Yes
–*Response Time*	-	-	-
–*Availability*	-	-	-
Completeness	Yes	No	Yes
Implementation Complexity	Low	Average	High
–*No. of Translators*	0	$2 * N$	$N * (N - 1)$

where N is a number of DRM systems
'-' is for future work

Chapter 5

Conclusions

5.1 Introduction

This chapter concludes thesis work and highlights the important facts found during the research. First of all, Section 5.2 presents a brief summary of the work. Section 5.3 gives some recommendations. Finally, Section 5.4 identifies the possibilities to extend this work for future research.

5.2 Summary

This work has investigated different approaches and suggested a solution for DRM interoperability. First of all, Chapter 2 has provided information about a DRM system and discussed different DRM systems. Chapter 3 has discussed cross-certification, super CA, and proxy server approaches to resolve interoperability in PKI. Moreover, it has contributed by applying those approaches for DRM interoperability between similar DRM systems each having different trust anchor. Chapter 4 has presented the main contributions in this thesis by proposing a solution to resolve DRM interoperability problem. It also has pointed out what the drawbacks in existing approaches are and evaluated the proposed solution based on the requirements on an interoperability solution. Moreover, it has shown how the system works and validated the security requirements by making a formal model.

The proposed solution has resolved DRM interoperability by introducing a new content translation entity called CFT. The devices share the content through a CFT. A CFT has device certificates from both DRM systems. Therefore, a CFT can communicate with license servers in both DRM system to acquire keys for the content transcryption. Furthermore, a CFT can also interact with both devices using super-distribution, embedding license, or by registering on a domain. After obtaining the transcrypted content, a device

requests for a license to its license server in the target DRM system. That license server acquires the rights from a license server of the source DRM system. The license servers in both DRM systems uses interoperability in PKI approaches to trust on each other. A rights translator needs to be implemented by a license server to translate the rights from one DRM system to another.

From security point of view, the proposed solution withstands different threats which include: to obtain a license without content sharing or purchasing, to play the content without a license, to play the content more than allowed, to share the content without device consent, to circumvent the rights in the target DRM system, and to expose the content or the keys.

5.3 Conclusions and Recommendations

On the basis of evaluation and comparison of different interoperability solutions, it is concluded that the proposed design would be accepted by users and content providers as it does not require to do any change at the device end. The proposed solution has advantage over existing solutions because it supports both content downloading and streaming use cases while they only support content downloading scenario. Moreover, it requires lower cost for the initial setup as compared to other solutions.

Following are some of the key recommendations which could be helpful to achieve DRM interoperability goal:

- A DRM system should use existing mechanisms (as used to identify the misbehaves of a device) to identify the misbehaves and revoke the license of a CFT.
- It is recommended to put CFT in the home environment to get better response time and availability.
- A target license server should acquire the rights from a source license server through proxy server approach. That is, a target license server implements the client of source DRM system with the restricted functionality to avoid information leakage or compromisation of the license server. Moreover, a target license server should implement the rights translator.
- In case if CFT is a Web service, there should be a number of CFTs in the distributed environment so that a single point of failure could be avoided. Furthermore, it should verify the purchased license information before fetching the content from the requesting device.

5.4 Future Research

Following are some possibilities which can facilitate future work in this area:

- Currently used DRM systems express rights using different RELs. A research can be conducted to develop translators for the rights translation from one REL to others.
- Marlin expresses the rights using a control program which is totally different than a REL. It can be investigated how to translate a program into a REL and vice versa.
- In case a CFT is a Web service then it can withstand a DoS attack by black listing the devices which are originating these requests. It can be worked out how to implement such a mechanism.
- In case a CFT is a Web service then it can be scrutinized how to deploy a number of CFTs in the distributed environment so that a single point of failure could be avoided.

Bibliography

[AFKM05] C. Adams, S. Farrell, T. Kause, and T. Mononen. Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP). RFC 4210 (Proposed Standard), September 2005. Available: `http://www.ietf.org/rfc/rfc4210.txt`.

[Alla] Open Mobile Alliance. Mobile Broadcast Services v1.0. Available: `http://www.openmobilealliance.org/technical/release_program/bcast_v1_0.aspx` [April 14, 2009].

[Allb] Open Mobile Alliance. OMA Digital Rights Management v1.0. Available: `http://www.openmobilealliance.org/technical/release_program/drm_v1_0.aspx` [March 30, 2009].

[Allc] Open Mobile Alliance. OMA Digital Rights Management v2.0. Available: `http://www.openmobilealliance.org/technical/release_program/drm_v2_0.aspx` [March 30, 2009].

[Alld] Open Mobile Alliance. OMA Digital Rights Management v2.1. Accessed 31 March 2009.

[Alle] Open Mobile Alliance. OMA Secure Content Exchange v1.0. Available: `http://www.openmobilealliance.org/technical/release_program/srm_v1_0.aspx` [March 31, 2009].

[All06] Open Mobile Alliance. DRM Specification v2.1. Technical report, May 2006.

[All08a] Open Mobile Alliance. DRM Architecture v2.1, 2008. Available: `http://www.openmobilealliance.org/technical/release_program/drm_v2_1.aspx` [March 31, 2009].

[All08b] Open Mobile Alliance. DRM Content Format v2.1. Technical report, October 2008.

[All09] Open Mobile Alliance. OMArlin Specification v1.0.3. Technical report, June 2009.

[Ano07] Anonymous. How FairPlay Works: Apple's iTunes DRM Dilemma, February 2007. Available: http://www.roughlydrafted.com/RD/RDM.Tech.Q1.07/2A351C60-A4E5-4764-A083-FF8610E66A46.html.

[Ars07] Arstechnica. Musicload: 75% of customer service problems caused by DRM, March 2007. Available: http://arstechnica.com/tech-policy/news/2007/03/75-percent-customer-problems-caused-by-drm.ars.

[AV99] B. Aboba and J. Vollbrecht. Proxy Chaining and Policy Implementation in Roaming. RFC 2607 (Informational), June 1999. Available: http://www.ietf.org/rfc/rfc2607.txt.

[Bar06] Chris Barlas. Digital Rights Express Languages (DRELs). Technical report, July 2006. JISC Technology and Standards Watch.

[BBGR03] Eberhard Becker, Willms Buhse, Dirk Gnnewig, and Niels Rump. *Digital Rights Management: Technological, Economic, Legal and Political Aspects*. Springer, Berlin Heidelberg, Germany, 2003.

[BHR99] S. Boeyen, T. Howes, and P. Richard. Internet X.509 Public Key Infrastructure LDAPv2 Schema. RFC 2587 (Proposed Standard), June 1999. Available: http://www.ietf.org/rfc/rfc2587.txt.

[BNP] William E. Burr, Noel A. Nazario, and W. Timothy Polk. A Proposed Federal PKI using X.509 v3 Certificates. Available: http://csrc.nist.gov/nissc/1996/papers/NISSC96/paper042/pkipap1.pdf.

[CD01] I. Cooper and J. Dilley. Known HTTP Proxy/Caching Problems. RFC 3143 (Informational), June 2001. Available: http://www.ietf.org/rfc/rfc3143.txt.

[CDH+05] M. Cooper, Y. Dzambasow, P. Hesse, S. Joseph, and R. Nicholas. Internet X.509 Public Key Infrastructure: Certification Path Building. RFC 4158 (Informational), September 2005. Available: http://www.ietf.org/rfc/rfc4158.txt.

[Clo] Cloakware. DRM Compliance and Robustness Rules. Available: http://security.cloakware.com/products/robustness-rules.php.

[CM05] Brenton Cooper and Paul Montague. Translation of Rights Expression, 2005. Available: `http://crpit.com/confpapers/CRPITV44Cooper.pdf`.

[CML05] CMLA. CMLA Technical Specification v1.0. Technical report, December 2005.

[Com06a] Marlin Developer Community. Marlin - Core System Specification v1.3, September 2006. Available: `http://www.marlin-community.com/develop/downloads/specifications` [March 31, 2009].

[Com06b] Marlin Developer Community. NEMO Trust Management Bindings v1.1. Technical report, September 2006. [February 18, 2009].

[Com06c] Marlin Developer Community. Octopus Objects v1.0.1. Technical report, September 2006.

[Com06d] Marlin Developer Community. The Role of NEMO in Marlin. Technical report, 2006. Available: `http://www.marlin-community.com/public/RoleofNEMOinMarlin.pdf` [February 10, 2009].

[Com06e] Marlin Developer Community. The Role of Octopus in Marlin. Technical report, 2006. Available: `http://www.marlin-community.com/public/RoleofOctopusinMarlin.pdf` [February 10, 2009].

[Com07] Marlin Developer Community. Marlin Broadband Architecture Overview. Technical report, November 2007. For Marlin Adopters.

[Com08] Marlin Developer Community. Marlin - Broadband Network Service Profile Specification v1.0, 2008. Available: `http://www.marlin-community.com/develop/downloads/specifications` [March 31, 2009].

[Con] Coral Consortium. Creative Content Online: Coral response to the EU Commission Consultation. Available: `http://www.coral-interop.org/main/20080228_Coral_Response_to_EU.pdf`.

[Con06] Coral Consortium. Coral Consortium Whitepaper. Technical report, February 2006. Available: `http:///www.coral-interop.org`.

[Con07] Coral Consortium. Forming a Coral Ecosystem. Technical report, October 2007. Available: http:///www.coral-interop.org.

[Con08a] Coral Consortium. Coral Core Architecture Specification 4.1. Technical report, March 2008. Available: http:///www.coral-interop.org.

[Con08b] Coral Consortium. Coral Domain Architecture Specification v4.1. Technical report, March 2008. Available: http:///www.coral-interop.org.

[Cur97] Ian Curry. Trusted Public-Key Infrastructures v1.1. Technical report, December 1997. Available: http://www.entrust.com.

[Gro] Trusted Computing Group. Trusted Computing Group. Available: https://www.trustedcomputinggroup.org/specs/TPM [March 30, 2009].

[Gua02] Content Guard. XrML 2.0 Technical Overview v1.0. Technical report, March 2002. Available: http://www.xrml.org/.

[Har04] Frank Hartung. DRM Components and Rights Expression Languages, 2004. Slides on Multimedia Content Protection.

[Ian01] Renato Iannella. Open Digital Rights Language (ODRL), 2001. Available: http://odrl.net/1.0/ODRL-10-HTML/ODRL-10.html [March 26, 2009].

[IND05] INDICARE. Consumer Survey on Digital Music and DRM, May 2005. Available: http://www.indicare.org/survey [March 5, 2009].

[ITU00] ITU. ITU-T X.509. Technical report, March 2000. International Standard 9594-8.

[JHMO06] Pramod A. Jamkhedkar, Gregory L. Heileman, and Ivan Martinez-Ortiz. The Problem with Rights Expression Languages, October 2006. Available: http://www.ece.unm.edu/~drake/products/ACM_DRM_06_paper.pdf.

[KLMM03] Rob H. Koenen, Jack Lacy, Michael MacKay, and Steve Mitchell. The Long March to Interoperable Digital Rights Management, September 2003. Available: http://www.intertrust.com/main/research/whitepapers/Interoperable_DRM.pdf [April 7, 2009].

[KM05] David W. Kravitz and Thomas S. Messerges. Achieving media portability through local content translation and end-to-end rights management. In *DRM '05: Proceedings of the 5th ACM workshop on Digital rights management*, pages 27–36, New York, NY, USA, 2005. ACM.

[Mica] Microsoft. Compliance and Robustness Rules for Windows Media DRM. Available: `https://wmlicense.smdisp.net/wmdrmcompliance/default.asp`.

[Micb] Microsoft. Digital Rights Management (DRM). Available: `http://www.microsoft.com/windows/windowsmedia/forpros/drm/default.mspx` [[April 11, 2009]].

[Mic04] Microsoft. Architecture of Windows Media Rights Manager, May 2004. Available: `http://www.microsoft.com/windows/windowsmedia/howto/articles/drmarchitecture.aspx` [March 31, 2009].

[Mic08] Microsoft. Microsoft PlayReady Content Access Technology, July 2008. Available: Available: `http://www.microsoft.com/PlayReady/Overview.mspx` [March 31, 2009].

[OMA] OMA. Open Mobile Alliance (OMA). Available: `http://www.openmobilealliance.org/` [February 12, 2009].

[OW07] Andre Osterhues and Marko Wolf. Digital Rights Management Demonstrator: Requirements, Analysis, and Design, March 2007. Available: `http://www.emscb.de/download/Turaya.FairDRM_req_design_070313.pdf` [May 8, 2009].

[PJ07] Milan Petkovic and William Jonker. *Security, Privacy, and Trust in Modern Data Management*. Springer, Berlin Heidelberg, Germany, 2007.

[STDD06] Carlos Serrao, Victor Torres, Jaime Delgado, and Miguel Dias. Interoperability Mechanisms for Registration and Authentication on Different Open DRM Platforms. *IJCSNS International Journal of Computer Science and Network Security*, December 2006. Available: `http://paper.ijcsns.org/07_book/200612/200612B18.pdf`.

[TCG06] Gelareh Taban, Alvaro A. Cardenas, and Virgil D. Gligor. Towards a Secure and Interoperable DRM Architecture, October 2006. Available: `http://www.eecs.berkeley.edu/~cardenas/Papers/DRM2006.pdf` [March 20, 2009].

[Tur00] Jim Turnbull. Cross-Certification and PKI Policy Networking v1.0. Technical report, August 2000. Available: `http://www.entrust.com/resources/pdf/cross_certification.pdf`.

[Wan05] Xin Wang. On DRM Interoperability and Compatibility, 2005. Slides.

[ZYL06] Wenjun Zeng, Heather Yu, and Ching-Yung Lin. *Multimedia security technologies for digital rights management.* Elsevier Press, Burlington, United States of America, 2006.

Appendix A

Rights Expression in ODRL

A.1 ODRL Permission and Constraint Sets

Every REL has some permission and constraint sets to express the rights. To get an overview, ODRL permission and constraint sets are depicted in Figure A.1 and Figure A.2 respectively.

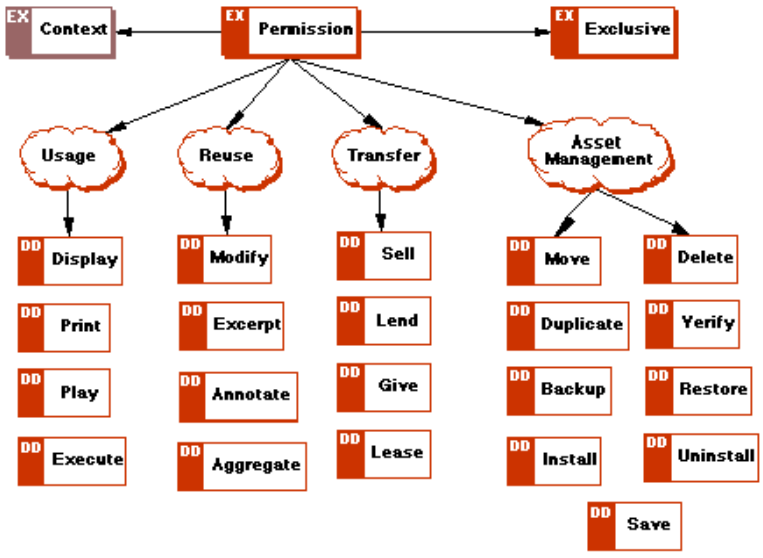

Figure A.1: ODRL permission terms [Ian01]

To know more about ODRL, see [Ian01].

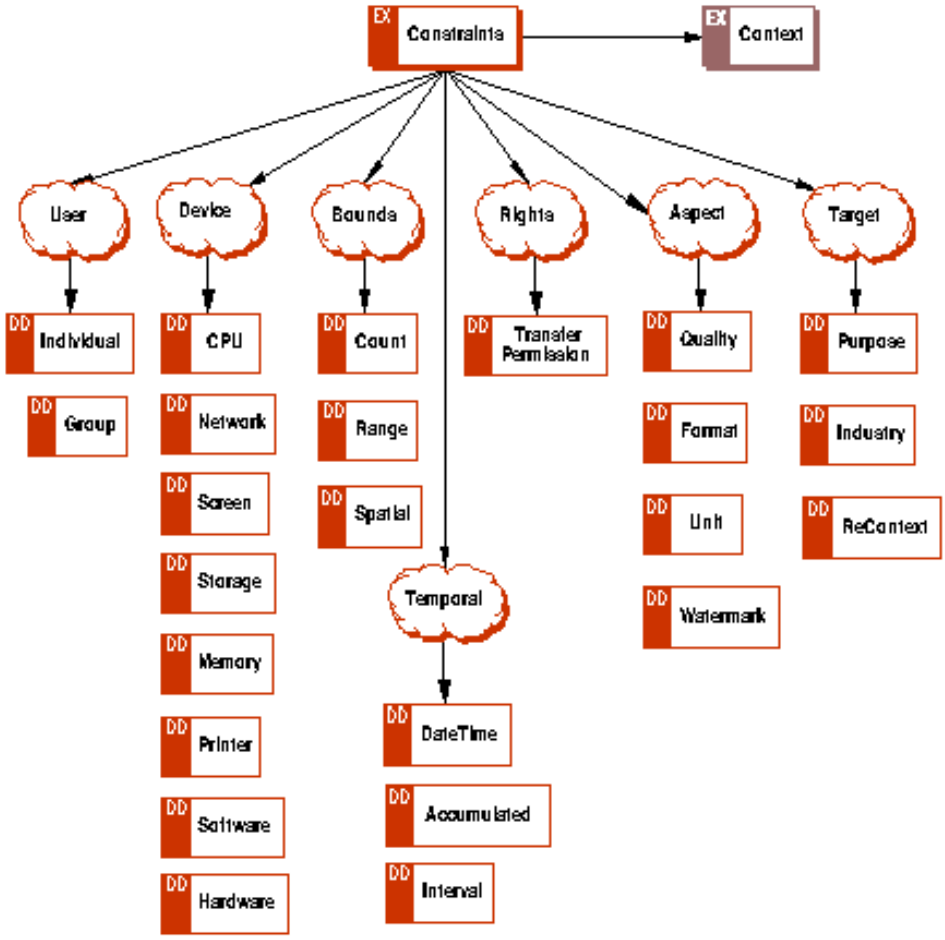

Figure A.2: ODRL constraint terms [Ian01]

A.2 ODRL Example

Alice can print *"Why Cats Sleep and We don't"* book at most thrice. Above rights can be expressed in ODRL as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<o-ex:rights xmlns:o-ex="http://odrl.net/1.1/ODRL-EX"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:o-dd="http://odrl.net/1.1/ODRL-DD"
          xsi:schemaLocation="http://odrl.net/1.1/ODRL-EX ../schemas/ODRL-EX-11.xsd
          http://odrl.net/1.1/ODRL-DD ../schemas/ODRL-DD-11.xsd">

   <o-ex:asset>
      <o-ex:context>
         <o-dd:uid>urn:ebook.world/999999/ebook/rossi-000001</o-dd:uid>
         <o-dd:name>Why Cats Sleep and We don't</o-dd:name>
      </o-ex:context>
   </o-ex:asset>

   <o-ex:permission>
      <o-dd:print>
         <o-ex:constraint>
            <o-dd:count>3</o-dd:count>
         </o-ex:constraint>
      </o-dd:print>
   </o-ex:permission>

   <o-ex:party>
      <o-ex:context>
         <o-dd:name>Alice</o-dd:name>
      </o-ex:context>
   </o-ex:party>
</o-ex:rights>
```

Object

Permission
Constraint

Subject

Figure A.3: Rights expression in ODRL [Har04]

Appendix B

Rights Object

A license is the most important part of any DRM system. An RO is a license format in OMA DRM system. In OMA DRM system, an RO is protected and known as *Protected RO*. The protected RO is a sequence of RO and MAC, where MAC is hash of the RO encrypted using a symmetric key algorithm with a key, K_{MAC}. The MAC element is used to provide the integrity to the RO element. The protected RO can be sent separately, included in a ROAP RO response, or a DCF. The protected RO is shown in Figure B.1.

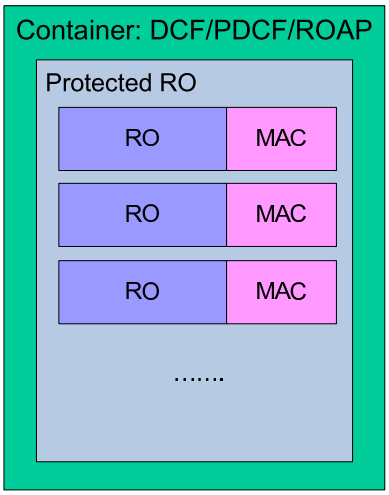

Figure B.1: OMA protected RO

According to [All06], an RO is an object consisting of:

- A sequence of:
  - *RI ID:* It is an identifier which identifies the RI.

  - *Rights:* This element contains the usage rights.
  - *Signature:* This element must be present when the RO is a domain RO.
  - *Time Stamp:* Its value is given in *Universal Coordinated Time (UTC)*. This element provides replay protection.
  - *EncKey:* It consists wrapped concatenation[1] of a CEK (that is, K_{CEK}) to encrypt the content and a MAC key (that is, K_{MAC}) to provide the integrity to the RO element.
- *Version:* It indicates the version of the ROPayload type.
- *ID:* This element identifies an RO.
- *IsStateful:* A boolean flag to indicate whether an RO contains stateful rights (that is, needs replay protection) or not.
- *IsDomainRO:* A boolean flag to indicate whether an RO is for a domain or not.
- *RI URL:* It contains a URL that a device can use to acquire the RO.

For further information about RO, see [All06].

[1]The details of wrapping operations have been suppressed in this report. For further information about wrapping, see *Key Management* section of [All06].

Appendix C

The 4-Pass Registration Protocol

The 4-pass registration protocol is performed for the mutual authentication between the device and the RI. After the successful completion of this protocol, the device establishes an *RI Context* containing RI-specific security related information such as agreed protocol parameters, protocol version, and certificate preferences. Figure C.1 shows the 4-pass registration protocol.

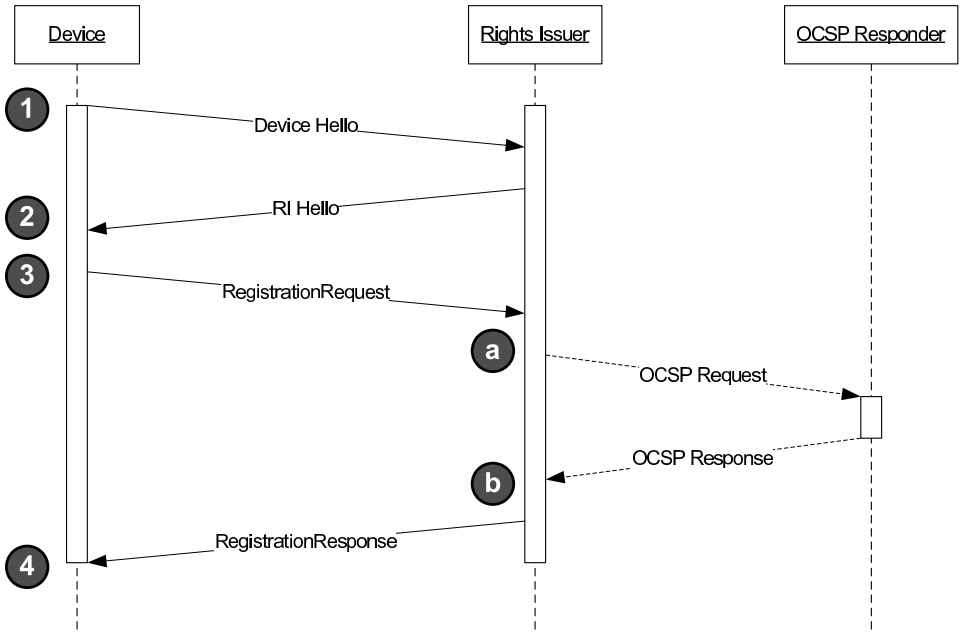

Figure C.1: The 4-pass registration protocol [All06]

The detail of each message in this protocol is described below:

C.1 Device Hello

Device Hello is the first message of the 4-pass registration protocol. It is sent from the device to the RI to initiate the protocol. It expresses the device information and preferences containing the following parameters:

- *Version:* The highest ROAP version number supported by the device.
- *Device ID:* This parameter identifies the device to the RI.
- *Supported Algorithms:* This parameter identifies the cryptographic algorithms such as hash, MAC, and signature, supported by the device.
- *Extensions:* One defined extension for this message is the *Certificate Caching* which indicates to the RI that the device has a capability to store information in the RI context.

C.2 RI Hello

This is the second message of the registration protocol. It is sent from the RI to the device in response to the first message. It expresses the RI preferences based on the values supplied in device hello message. It contains the following parameters:

- *Status:* It indicates the status (success or error) of the device hello message.
- *Session ID:* This is an identifier which is set by the RI. This allows concurrent sessions between the device and the RI.
- *Selected Version:* This is the selected ROAP version. This is the minimum of: suggested version by the device and the highest version supported by the RI.
- *RI ID:* This parameter identifies the RI to the device.
- *Selected Algorithms:* This parameter identifies the cryptographic algorithms such as hash, MAC, and signature, supported by the RI.
- *RI Nonce:* This is a random nonce generated by the RI.
- *Trusted Device Authorities:* This a list of device trust anchors recognized by the RI.
- *Server Info:* It contains server-specific information that the device must return this parameter unmodified in a subsequent message (that is, the *Registration Request* message).

- *Extensions:* There are following extensions:
  - *Peer Key Identifier:* An identifier to indicate to the device that the RI has already stored the device public key.
  - *Certificate Caching:* This extension indicates to the RI that the device has a capability to store information in the RI context.
  - *Device Details:* This extension requests to the device to return device-specific information such as manufacturer and model, in a subsequent message (that is, registration request message).

C.3 Registration Request

This is the third message of the registration protocol. It is sent from the device to the RI. It contains the following parameters:

- *Session ID:* This is identical to the *Session ID* parameter of the preceding message (that is, RI hello message). If this is not identical then the RI terminates the registration protocol.
- *Device Nonce:* This is a random nonce generated by the device.
- *Request Time:* It is the current *DRM Time* as measured by the device.
- *Certificate Chain:* A certificate chain including the device certificate.
- *Trusted RI Authorities:* A list of RI trust anchors recognized by the device.
- *Server Info:* If a server info parameter is present in the RI hello message then the device should set it with that parameter value. That is, received in the RI hello.
- *Extensions:* There are following extensions:
  - *Peer Key Identifier:* An identifier to indicate to the RI that the device has already stored the RI public key.
  - *No OCSP Response:* It indicates to the RI that there is no need to send the OCSP response.
  - *OCSP Responder Key Identifier:* It identifies a trusted OCSP responder key stored in the device. This extension is used to save the bandwidth.
  - *Device Details:* This extension sends device-specific information such as manufacturer, model, and version.
- *Signature:* The signature is made using the device private key on two previous messages (that is, device hello message and RI hello message) and the current message.

C.4 Registration Response

This is the fourth and the last message of the registration protocol. It is sent from the RI to the device. On success, it enables the device to establish an RI Context. It contains the following parameters:

- *Status:* It indicates the status (success or error) of the registration request message.
- *Session ID:* This is identical to the session ID parameter of the preceding message (that is, registration request message). If this is not identical then the device MUST terminate the registration protocol.
- *RI URL:* This parameter indicates the ROAP URL that is stored in the RI Context.
- *Certificate Chain:* A certificate chain including the RI certificate.
- *OCSP Response:* A complete set of the valid OCSP responses.
- *Extensions:* One defined extension is *Domain Name Whitelist* which allows an RI to specify a list of *Fully Qualified Domain Name (FQDN).*
- *Signature:* The signature is made using the RI private key on the previous message (that is, registration request message) and the current message.

For further information about the registration protocol, see [All06].

Appendix D

RO Acquisition Protocol

The RO acquisition protocol is run between the device and the RI to let the device to acquire a license from the RI. After the successful completion of this protocol, the device obtains an RO encrypted with the device public key while signed by the RI signing key. Figure D.1 shows the RO acquisition protocol.

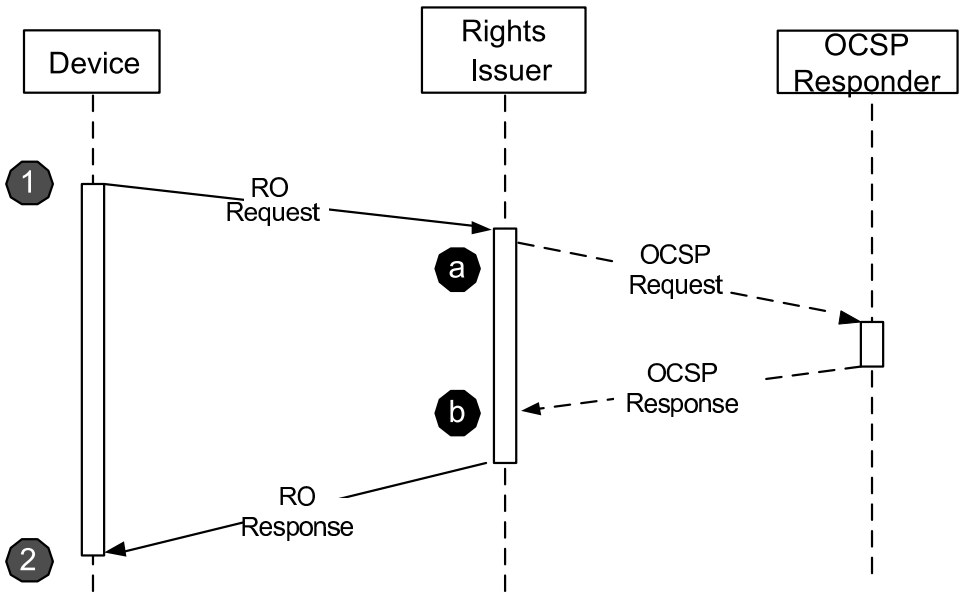

Figure D.1: The 2-pass RO acquisition protocol [All06]

The detail of each message in this protocol is described below:

D.1 RO Request

This is the first message of the RO acquisition protocol. It is sent from the device to the RI. It contains the following parameters:

- *Device ID:* This parameter identifies the requesting device.
- *Domain ID:* It identifies the domain.
- *RI ID:* It identifies the authorizing RI.
- *Device Nonce:* This is a random nonce generated by the device.
- *Request Time:* It is the current *DRM Time* as measured by the device.
- *RO Info:* It identifies the requested RO.
- *Certificate Chain:* A certificate chain including the device certificate. This parameter is sent unless it is indicated in *Peer Key Identifier* that the RI context has stored the necessary device certificate information.
- Extensions: There are following extensions:
  - *Peer Key Identifier:* An identifier to indicate to the RI that the device has already stored the RI public key.
  - *No OCSP Response:* It indicates to the RI that there is no need to send the OCSP response.
  - *OCSP Responder Key Identifier:* It identifies a trusted OCSP responder key stored in the device.
  - *Transaction Identifier:* This identifier allows a device to provide the RI with information for tracking of transactions.
- *Signature:* This is a signature on the current message and is computed using the device private key.

D.2 RO Response

This is the second message of the RO acquisition protocol. It is sent from the RI to the device. It contains the following parameters:

- *Status:* It indicates the status (success or error) of *RO Request* message.
- *Device ID:* This value is equal to the *Device ID* parameter sent by the device in the preceding message (that is, RO request message).
- *RI ID:* This value is equal to the *RI ID* sent by the device in the preceding message (that is, RO Request message).
- *Device Nonce:* This value is equal to the RI ID sent by the device in the preceding message (that is, RO Request message).

- *Protected RO:* This contains sensitive information (such as CEK) which is encrypted with the device public key.
- *Certificate Chain:* A certificate chain including the RI certificate. This parameter is sent unless a preceding message (that is, RO request message) contains the peer key identifier.
- *OCSP Response:* A complete set of the valid OCSP responses.
- *Extensions:* One defined extension is *Transaction Identifier* which allows an RI to provide the device with the information for tracking of transactions.
- *Signature:* This is a signature on the current message and is computed using the RI private key.

For further information about RO acquisition protocol, see [All06].

List of Figures

2.1 Abstract level view of a DRM system 6
2.2 The fundamental principle of a DRM system 7
2.3 A generic DRM architecture 9
2.4 DRM container format 10
2.5 REL landscape 12
2.6 Basic license acquisition 16
2.7 Basic content transfer 17
2.8 Basic stream transfer 18
2.9 OMA DRM v2.1 architecture 20
2.10 BNS overview 21

3.1 Cross-certification relationship between CA1 and CA2 26
3.2 Path length constraint 27
3.3 Super CA signing CA1 and CA2 28
3.4 Proxy server approach 30
3.5 CMLA PKI system 32
3.6 Cross-certification hierarchy 34
3.7 Registration protocol between D1 and RI2 35
3.8 Super CA hierarchy 36
3.9 RI1 working as a proxy server 39

4.1 OMArlin architecture 45
4.2 Coral interoperability framework architecture 46
4.3 Generic Interoperability Solution 49
4.4 Stateless license 65
4.5 Same algorithm for content encryption 66
4.6 Content with embedded license 66
4.7 Streaming case of Generic Interoperability Solution 67
4.8 Interoperability between OMA DRM v2.1 and Marlin DRM . 72

A.1 ODRL permission terms 85
A.2 ODRL constraint terms 86
A.3 Rights expression in ODRL 87

B.1 OMA protected RO 89

C.1 The 4-pass registration protocol 91

D.1 The 2-pass RO acquisition protocol 95

List of Tables

2.1 The trust models of different DRM systems 15

4.1 Comparison of interoperability solutions 73

Glossary

AES	Advanced Encryption Standard
ARL	Authority Revocation List
BNS	Broadband Network Service
CA	Certification Authority
CE	Consumer Electronics
CEK	Content Encryption Key
CFT	Content Format Translator
CI	Content Issuer
CMLA	Content Management License Administrator
CRL	Certificate Revocation List
DCF	DRM Content Format
DECE	Digital Entertainment Content Ecosystem
DES	Data Encryption Standard
DN	Distinguished Name
DoS	Denial of Service
DPRL	Digital Property Rights Language
DREL	Digital Rights Expression Language
DRM	Digital Rights Management
FQDN	Fully Qualified Domain Name
HTTP	Hyper Text Transfer Protocol
IPTV	Internet Protocol Television
IPTV-ES	IPTV Endpoint System
IV	Initial Vector
MAC	Message Authentication Code
MBB	Marlin Broadband
MDC	Marlin Development Community

MIME	Multipurpose Internet Mail Extensions
MPEG	Moving Picture Experts Group
MTMO	Marlin Trust Management Organization
NEMO	Networked Environment for Media Orchestration
NSN	Nokia Siemens Networks
OCSP	Online Certificate Status Protocol
ODRL	Open Digital Rights Language
OMA	Open Mobile Alliance
PC	Personal Computer
PKC	Public Key Certificate
PKI	Public Key Infrastructure
REL	Rights Expression Language
RI	Rights Issuer
RO	Rights Object
ROAP	Rights Object Acquisition Protocol
RT	Rights-Token
SCE	Secure Content Exchange
SRM	Secure Removable Media
URL	Uniform Resource Locator
UTC	Universal Coordinated Time
VM	Virtual Machine
WMDRM	Windows Media DRM
WMF	Windows Media Format
XACL	eXtensible Access Control Language
XACML	eXtensible Access Control Markup Language
XBS	Extension for Broadcast Support
XMCL	eXtensible Media Commerce Language
XML	eXtensible Markup Language
XML-AC	XML Access Control
XrML	eXtensible Rights Markup Language