

**CMLA Information**

**CMLA Technical Specification**

**Date:** 2009-11-03

**Version:** v1.3-20091103

**STATUS:** Approved

**CMLA - Contact Information**

CMLA-Services @cm-la.com

## NOTICE

CMLA TECHNICAL SPECIFICATION is the Proprietary CMLA, LLC., information that may be used for review and comment purposes only. ANY OTHER USE REQUIRES A SEPARATE LICENSE AGREEMENT BY AND BETWEEN USER AND CMLA, LLC AND WITHOUT SUCH LICENSE AGREEMENT ALL OTHER USE IS EXPRESSLY PROHIBITED.

**DISCLAIMERS:**

**THIS APPROVED CMLA TECHNICAL SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION, OR SAMPLE.**

**NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED OR INTENDED HEREBY. CMLA, INTEL, NOKIA, PANASONIC AND SAMSUNG, DISCLAIM ALL LIABILITY, INCLUDING LIABILITY FOR INFRINGEMENT OF PROPRIETARY RIGHTS, RELATING TO THE USE OF THE INFORMATION IN THIS APPROVED CMLA TECHNICAL SPECIFICATION AND TO THE IMPLEMENTATION OF INFORMATION IN THIS APPROVED CMLA TECHNICAL SPECIFICATION. CMLA, INTEL, NOKIA, MEI/PANASONIC AND SAMSUNG, DO NOT WARRANT OR REPRESENT THAT SUCH IMPLEMENTATION(S) WILL NOT INFRINGE SUCH RIGHTS.**

WITHOUT LIMITATION, CMLA, INTEL, NOKIA, PANASONIC AND SAMSUNG DISCLAIM ALL LIABILITY FOR COST OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, LOST PROFITS, LOSS OF USE, LOSS OF DATA OR ANY INCIDENTAL, CONSEQUENTIAL, DIRECT, INDIRECT, OR SPECIAL DAMAGES, WHETHER UNDER CONTRACT, TORT, WARRANTY OR OTHERWISE, ARISING IN ANY WAY OUT OF USE OR RELIANCE UPON THIS SPECIFICATION OR ANY INFORMATION HEREIN.

All product names are trademarks, registered trademarks, or service marks of their respective owners.

Copyright © 2009 CMLA, LLC.

## INDEX

<b>1.</b>	<b>INTRODUCTION.....</b>	<b>4</b>
<b>2.</b>	<b>REFERENCES.....</b>	<b>4</b>
<b>3.</b>	<b>DEFINITIONS.....</b>	<b>5</b>
<b>4.</b>	<b>ABBREVIATIONS.....</b>	<b>6</b>
<b>5.</b>	<b>OVERVIEW.....</b>	<b>8</b>
<b>6.</b>	<b>CERTIFICATES.....</b>	<b>10</b>
<b>7.</b>	<b>OCSP PROFILE.....</b>	<b>17</b>
<b>8.</b>	<b>DEVICE CRL, RIGHTS ISSUER CRLS AND CMLA ARLS.....</b>	<b>19</b>
<b>9.</b>	<b>DEVICE CRL,CMLA ARL AND DEVICE CA CERTIFICATE DELIVERY MECHANISM.....</b>	<b>22</b>
<b>10.</b>	<b>DEVICE CREDENTIALS DELIVERY.....</b>	<b>24</b>
<b>11.</b>	<b>RIGHTS ISSUER CERTIFICATION.....</b>	<b>29</b>
<b>12.</b>	<b>CMLA IP.....</b>	<b>31</b>
<b>13.</b>	<b>CMLA PARAMETERS.....</b>	<b>39</b>
<b>14.</b>	<b>DEVELOPMENT KEYS AND CERTIFICATES.....</b>	<b>39</b>
<b>15.</b>	<b>CMLA MOBILE BROADCAST.....</b>	<b>40</b>
<b>16.</b>	<b>CONTENT PROTECTION TECHNOLOGY SYSTEM RENEWABILITY MESSAGE (CPT-SRM) SUPPORT.....</b>	<b>43</b>
<b>A1.</b>	<b>CMLA DDT EXP.....</b>	<b>44</b>
<b>A2.</b>	<b>CMLA DDT PERM.....</b>	<b>46</b>
<b>B1.</b>	<b>CMLA IP TEST VECTOR.....</b>	<b>49</b>
<b>B2.</b>	<b>CMLA IP TEST VECTOR.....</b>	<b>50</b>

## 1. Introduction

OMA DRM Release 2 specifications [OMADRM-v2] define the end-to-end protocol for protected content distribution from Rights Issuers to Devices in a secure manner. These specifications rely on the existence of a Public Key Infrastructure to provide certain security services such as key and certificate provisioning mechanisms, certificate status checking, and the overall PKI hierarchy. The OMA specification builds on top of the assumption that every Device and Rights Issuer has one or more private keys, certificates and trust anchors. This specification defines the details of the CMLA deployment of OMA DRM technology.

The target audience is Client Adopters and Service Providers implementing CMLA based products and services. The reader is recommended to familiarize herself with the overall CMLA trust model that this technical specification is only one part of.

This specification supplements but does not replace OMA DRM Release 2 specifications.

## 2. References

- [18Crypt] 18Crypt Profile specified in ETSI TS 102474 V1.1.1: Digital Video Broadcasting (DVB): IP Datacast over DVB-H: Service Purchase and Protection or most recent version.
- [IEC62455] IEC 62455 First Edition 2007-06: Internet protocol (IP) and transport stream (TS) based service access or most recent version.
- [OMABCAST-SCPv1] DRM Profile specified in OMA BCAST specification OMA-TS-BCAST\_SvcCntProtection-V1\_0: Service and Content Protection for Mobile Broadcast Services.
- [OMADRM-XBS] OMA-TS-DRM\_XBS-V1\_0: OMA DRM V2.0 Extensions for Broadcast Support.
- [FIPS180-2] “FIPS 180-2 - Secure Hash Standard”, NIST, August 2002, URL: <http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf>
- [FIPS197] “FIPS 197 – Advanced Encryption Standard”, NIST, November 2001, URL: <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>
- [ISO18033-2D] “ISO/IEC 18033-2: Information technology – Security techniques - Encryption algorithms - Part 2: Asymmetric Ciphers”, Draft, Victor Shoup, January, 2004.
- [OMADRM-v2] “Digital Rights Management”, Open Mobile Alliance™, Version 2.0, OMA-DRM-DRM-v2\_0, URL: <http://www.openmobilealliance.org>
- [OMAOCSP] “Online Certificate Status Protocol Mobile Profile”, Version v1.0, OMA-WAP-OCSP-V1\_0-20040127-C, URL: <http://www.openmobilealliance.org>
- [PKCS#1] “PKCS #1 v2.1: RSA Cryptography Standard”, RSA Laboratories, June 2002
- [PKCS#8] “PKCS #8: Private-Key Information Syntax Standard”, RSA Laboratories, November 1993

- [PKCS#9] “PKCS #9 v2.0: Selected Object Classes and Attribute Types”, RSA Laboratories, February 2000
- [PKCS#10] “PKCS #10 v1.7: Certification Request Syntax Standard”, RSA Laboratories, May 2000
- [PKCS#12] “PKCS #12 v1.0: Personal Information Exchange Syntax”, RSA Laboratories, June 1999
- [RFC2560] “X.509 Internet Public Key Infrastructure – Online Certificate Status Protocol – OCSP”, M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams, June 1999. URL: <http://www.ietf.org/rfc/rfc2560.txt>
- [RFC2616] “RFC 2616 – Hypertext Transfer Protocol -- HTTP/1.1”, R. Fielding et al, June 1999. URL: <http://www.ietf.org/rfc/rfc2616.txt>
- [RFC2617] “RFC 2617 – HTTP Authentication: Basic and Digest Access Authentication”, J Franks et al, June 1999. URL: <http://www.ietf.org/rfc/rfc2617.txt>
- [RFC3279] “RFC 3279 - Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile”, R. Housley, W. Ford, W. Polk, D.Solo, April 2002. URL: <http://www.ietf.org/rfc/rfc3279.txt>
- [RFC3280] “RFC 3280 - Internet X.509 Public Key Infrastructure – Certificate and CRL Profile”, R. Housley, W. Ford, W. Polk, D.Solo, April 2002. URL: <http://www.ietf.org/rfc/rfc3280.txt>
- [RFC3369] “Cryptographic Message Syntax (CMS)”, R. Housley, August 2002. URL: <http://www.ietf.org/rfc/rfc3369.txt>
- [RFC1738] “Uniform Resource Locators”, T. Berners-Lee *et al*, December 1993. URL: <http://www.ietf.org/rfc/rfc1738>

### 3. Definitions

Client Adopter	As defined in the CMLA agreements
Client Adopter Transport Key	RSA key pair that is used to protect the confidentiality of Device Credentials DVD-R
CMLA Transport Key	RSA key pair that is used to protect the integrity and authenticity of Device Credentials Delivery
CPT-SRM	Content Protection Technology System Renewability Message generally referred to as System Renewability Message (SRM) but within this specification is defined as CPT-SRM so as not to be confused with OMA SRM terminology
Device	As defined in [OMADRM-v2]
Device CA	A CA that issues certificates for Devices

Device Credentials DVD-R	A DVD-R that contains security credentials for Client Adopters such as Device certificates and private keys
DRM Time	As defined in [OMADRM-v2]
Key Transport Algorithm	An algorithm used to transport key material to a Device, protected with the public key of the Device.
OCSP Responder	An entity producing OCSP responses
Rights Issuer	As defined in [OMADRM-v2]
Rights Issuer CA	A CA that issues certificates for Rights Issuers
Root CA	A trusted CA that is topologically on the top of the CMLA PKI hierarchy
Service Provider	As defined in the CMLA agreements
Tag Length Format	A syntax defined in the references [18Crypt], [IEC62455], and [OMABCAST-SCPv1] used to hold keyset blocks.

#### 4. Abbreviations

AES	Advanced Encryption Standard
AES-WRAP	As defined in [OMADRM-v2] section 7.1.3
ARL	Authority Revocation List
BCRO	Broadcast Rights Object
CA	Certificate Authority
CRL	Certificate Revocation List
CMLA_DDT_Exp	CMLA Data-dependent Transformation with Exponentiation
CMLA_DDT_Perm	CMLA Data-dependent Transformation with Permutation
CMLA_KDF	CMLA Key Derivation Function
CMLA_RSA_Decrypt	CMLA RSA Decryption
CMLA_RSA_Encrypt	CMLA RSA Encryption
CMLA_UNWRAP	CMLA Key Unwrap Scheme
CMLA_WRAP	CMLA Key Wrap Scheme
DP	Device Public Key
DRM	Digital Rights Management
DVD-R	Digital Versatile Disc - Recordable
IETF	Internet Engineering Task Force
KDF	Key Derivation Function
OCSP	Online Certificate Status Protocol

OMA	Open Mobile Alliance
OMA_KDF	Key Derivation Function as defined in [OMADRM-v2] section 7.1.2
MAC	Message Authentication Code
MIME	Multipurpose Internet Mail Extensions
PKCS	Public-Key Cryptography Standards
PKI	Public Key Infrastructure
REK	Rights Encryption Key
RFC	Request For Comments
ROAP	Rights Object Acquisition Protocol
ROT	Root Of Trust
RSA	Rivest, Shamir, Adleman
RSA.DECRYPT	As defined in [OMADRM-v2]
RSA.ENCRIPT	As defined in [OMADRM-v2]
SK	Session Key
TAA	Trust Authority Algorithm
TLF	Tag Length Format

## 5. Overview

### 5.1 PKI System Deliverables

The figure below illustrates the different entities producing output deliverables from the CMLA PKI system such as certificates, keys, CRLs and OCSP responses.

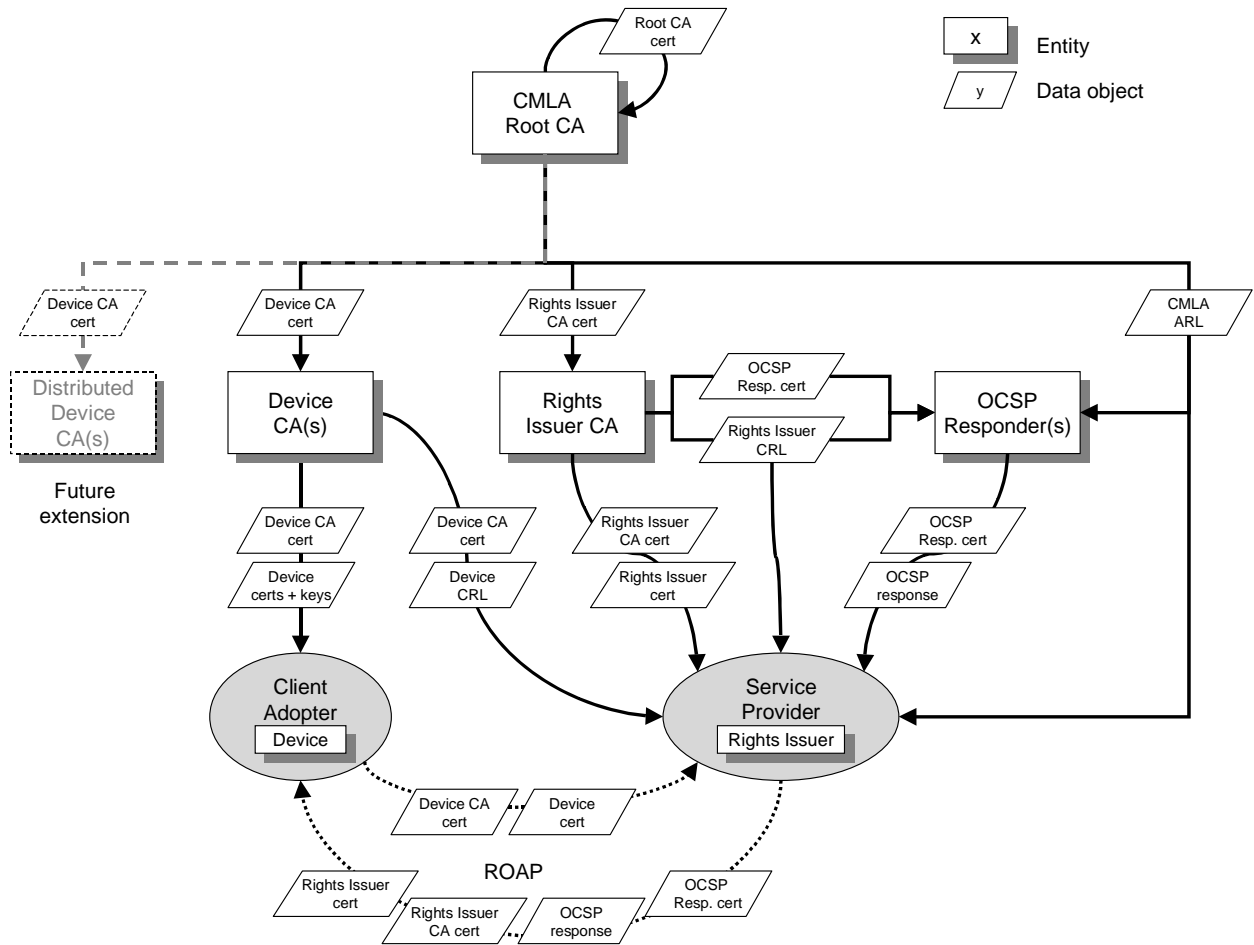


Figure 1. CMLA PKI System

Outputs from the CMLA PKI system are used by different entities according to the table below.

CMLA PKI output	Usage
Root CA certificate	<p>Root CA certificate is a self-signed CA certificate that is used by Devices and Rights Issuers as a trust anchor to validate other CMLA certificates and the CMLA ARL.</p> <p>Root CA certificate is delivered to Client Adopters and Service Providers out-of-band.</p>

Device certificate and private key	<p>Device certificates are issued by a Device CA. The certificates and corresponding private keys are delivered to the Client Adopters in a Device Credentials DVD-R. Client Adopter stores them inside the Device.</p> <p>Device certificate is sent, upon request, from the Device to the Rights Issuer as part of the Device's certificate chain.</p> <p>Device certificate is validated and used by the Rights Issuer.</p>
Device CA certificate	<p>Device CA certificate is issued by the CMLA Root CA and stored inside the Device as part of the Device's own certificate chain.</p> <p>Device CA certificate is sent, upon request, from the Device to the Rights Issuer as part of the Device's certificate chain.</p> <p>Device CA certificate is validated and used by the Rights Issuer.</p>
Rights Issuer certificate	<p>Rights Issuer certificate is issued by the Rights Issuer CA.</p> <p>Rights Issuer certificate is sent, upon request, from the Rights Issuer to the Device as a part of the Rights Issuer's certificate chain.</p>
Rights Issuer CA certificate	<p>Rights Issuer CA certificate is issued by the CMLA Root CA and stored inside the Rights Issuer as a part of Rights Issuer's certificate chain.</p> <p>Rights Issuer CA certificate is sent, upon request, from the Rights Issuer to the Device as a part of the Rights Issuer's certificate chain.</p>
OCSP response	<p>OCSP response is created and digitally signed by the OCSP Responder.</p> <p>The OCSP response is delivered to the Device through the Rights Issuer as a proof for the Rights Issuer's certificate's revocation status and for the purpose of time synchronization of the Device.</p>
OCSP Responder certificate	<p>The OCSP Responder certificate is issued by the Rights Issuer CA and stored inside the OCSP Responder.</p> <p>The OCSP Responder certificate is delivered to the Device through the Rights Issuer with the OCSP response, upon request by Device.</p>
Device CRL	<p>Device CRL is issued by the Device CA and it contains revocation status information for Device certificates issued by that Device CA. The Device CRL may, in the future, also be an indirect CRL that contains revocation status information for other Device CAs than the one issuing the Device CRL.</p> <p>The Device CRL is delivered to and used by the Rights Issuer to verify the revocation status of a Device certificate.</p>
Rights Issuer CRL	<p>Rights Issuer CRL is issued by the Rights Issuer CA and it contains revocation status information for Rights Issuer certificates issued by that CA. It is delivered to and used by the OCSP Responder to produce OCSP responses and Rights Issuers for verifying the revocation status of OCSP Responder certificates.</p>
CMLA ARL	<p>CMLA ARL is issued by the CMLA Root CA and delivered to and used by the Rights Issuer (to verify the revocation status of the Device CA).</p>

## 5.2 Future Extensions

CMLA may start to support in the future a more distributed PKI model where Client Adopters are allowed to host Distributed Device CAs by themselves. However, as CMLA needs to have full control over the Device revocation process it is expected that all CRL issuing is done centrally by CMLA. In that setting one of the existing CMLA hosted Device CAs will be issuing CRLs for distributed Device CAs.

In order to make transition to distributed model as smooth as possible Service Providers are required to support the notion of indirect CRLs.

## 6. Certificates

The structure of all CMLA certificates is based on RFC 3280. All CMLA certificates SHALL be DER encoded. All attributes of the issuer and subject fields SHALL be encoded as type UTF8String.

```

Certificate ::= SEQUENCE {
    tbsCertificate      TBSCertificate,
    signatureAlgorithm  AlgorithmIdentifier,
    signatureValue      BIT STRING }

TBSCertificate ::= SEQUENCE {
    version             [0] EXPLICIT Version DEFAULT v1,
    serialNumber        CertificateSerialNumber,
    signature           AlgorithmIdentifier,
    issuer              Name,
    validity            Validity,
    subject             Name,
    subjectPublicKeyInfo SubjectPublicKeyInfo,
    issuerUniqueID      [1] IMPLICIT UniqueIdentifier OPTIONAL,
                        -- If present, version MUST be v2 or v3
    subjectUniqueID     [2] IMPLICIT UniqueIdentifier OPTIONAL,
                        -- If present, version MUST be v2 or v3
    extensions          [3] EXPLICIT Extensions OPTIONAL
                        -- If present, version MUST be v3
}

```

### 6.1 CMLA Root CA Certificate

CMLA Root CA certificate is a self-signed CA certificate that is used by Devices and Rights Issuers as a trust anchor to validate other CMLA certificates and the CMLA ARL. Root CA certificate is delivered to Client Adopters and Service Providers out-of-band.

CMLA Root CA Certificates SHALL be based on OMA DRM specification appendix D.3 “CA Certificates” [OMADRM-v2].

Certificate field name	Value
Version	v3 (integer value 2)
Signature	<i>sha-1WithRSAEncryption</i> as defined in [RFC3279]
Issuer	Same as the <i>subject</i> field
Validity	Expires 31 <sup>st</sup> December 2034.  CMLA may in the future create a new Root CA or recertify the existing Root CA.  CMLA will decide during 2010 the key lengths and certificate validity periods that will be used after 2011.
Subject	Contains attributes: - <i>countryName</i> (e.g. “US”) - <i>organizationName</i> (e.g. “CMLA”) - <i>commonName</i> (e.g. “Root CA”)

SubjectPublicKeyInfo	During 2004-2011: carries a 2048 bit RSA public key identified with the <i>pkcs-1</i> algorithm identifier as defined in [RFC3279]. CMLA will decide during 2010 the key lengths and certificate validity periods that will be used after 2011.
IssuerUniqueId	Not used
SubjectUniqueId	Not used
BasicConstraints extension	Mandatory, critical. The boolean <i>cA</i> field is set to TRUE. The <i>pathLenConstraint</i> field is not used.
CRLDistributionPoints extension	Not used
KeyUsage extension	Mandatory, critical. Only key usage bits <i>keyCertSign</i> and <i>cRLSign</i> are set.
AuthorityKeyIdentifier extension	Not used
SubjectKeyIdentifier extension	Mandatory, non-critical. The key identifier method (1) is used as defined in [RFC3280] (160-bit SHA-1 hash of the <i>subjectPublicKey</i> field, excluding the tag, length, and number of unused bits).
CertificatePolicies extension	Not used
ExtKeyUsage extension	Not used
Id-pkix-ocsp-nocheck extension	Not used

## 6.2 Device CA Certificate

Device CA certificate is issued by the CMLA Root CA and stored inside the Device as part of the Device's own certificate chain. Device CA certificate is sent, upon request, from the Device to the Rights Issuer as part of the Device's certificate chain. Device CA certificate is validated and used by the Rights Issuer.

Device CA Certificates SHALL be based on OMA DRM specification appendix D.3 "CA Certificates" [OMADRM-v2].

Certificate field name	Value
Version	v3 (integer value 2)
Signature	<i>sha-1WithRSAEncryption</i> as defined in [RFC3279]
Issuer	Same as the corresponding CMLA Root CA <i>subject</i> field.
Validity	During 2004-2011: expires 30 years from the issuance date but not later than the certificate of the corresponding CMLA Root CA Certificate. CMLA will decide during 2010 the key lengths and certificate validity periods that will be used after 2011.

Subject	Contains attributes: - <i>countryName</i> (e.g. "US") - <i>organizationName</i> (e.g. "CMLA") - <i>commonName</i> (e.g. "Device CA 1")
SubjectPublicKeyInfo	During 2004-2011: carries a 2048 bit RSA public key identified with the <i>pkcs-1</i> algorithm identifier as defined in [RFC3279].  CMLA will decide during 2010 the key lengths and certificate validity periods that will be used after 2011.
IssuerUniqueId	Not used
SubjectUniqueId	Not used
BasicConstraints extension	Mandatory, critical.  The boolean <i>cA</i> field is set to TRUE. The <i>pathLenConstraint</i> field is set to zero.
CRLDistributionPoints extension	Not used
KeyUsage extension	Mandatory, critical.  Only key usage bits <i>keyCertSign</i> and <i>cRLSign</i> are set for CMLA hosted Device CAs.  For distributed Device CAs, CMLA may decide not to use <i>cRLSign</i>
AuthorityKeyIdentifier extension	Mandatory, non-critical.  The extension contains only the <i>keyIdentifier</i> field that has the same value as the corresponding CMLA Root CA Certificate's <i>subjectKeyIdentifier</i> extension value.
SubjectKeyIdentifier extension	Mandatory, non-critical.  The key identifier method (1) is used as defined in [RFC3280] (160-bit SHA-1 hash of the <i>subjectPublicKey</i> field, excluding the tag, length, and number of unused bits).
CertificatePolicies extension	Not used
ExtKeyUsage extension	Not used
Id-pkix-ocsp-nocheck extension	Not used

### 6.3 Rights Issuer CA Certificate

Rights Issuer CA certificate is issued by the CMLA Root CA and stored inside the Rights Issuer as a part of Rights Issuer's certificate chain. Rights Issuer CA certificate is sent, upon request, from the Rights Issuer to the Device as a part of the Rights Issuer's certificate chain.

Rights Issuer CA Certificates SHALL be based on OMA DRM specification appendix D.3 "CA Certificates" [OMADRM-v2].

Certificate field name	Value
Version	v3 (integer value 2)

Signature	<i>sha-1WithRSAEncryption</i> as defined in [RFC3279]
Issuer	Same as the corresponding CMLA Root CA <i>subject</i> field.
Validity	During 2004-2011: Expires 30 years from the issuance date but not later than the certificate of the corresponding CMLA Root CA Certificate.  CMLA will decide during 2010 the key lengths and certificate validity periods that will be used after 2011.
Subject	Contains attributes: - <i>countryName</i> (e.g. "US") - <i>organizationName</i> (e.g. "CMLA") - <i>commonName</i> (e.g. "Rights Issuer CA 1")
SubjectPublicKeyInfo	During 2004-2011: carries a 2048 bit RSA public key identified with the <i>pkcs-1</i> algorithm identifier as defined in [RFC3279].  CMLA will decide during 2010 the key lengths and certificate validity periods that will be used after 2011.
IssuerUniqueId	Not used
SubjectUniqueId	Not used
BasicConstraints extension	Mandatory, critical.  The boolean <i>cA</i> field is set to TRUE. The <i>pathLenConstraint</i> field is set to zero.
CRLDistributionPoints extension	Not used
KeyUsage extension	Mandatory, critical.  Only key usage bits <i>keyCertSign</i> and <i>cRLSign</i> are set.
AuthorityKeyIdentifier extension	Mandatory, non-critical.  The extension contains only the <i>keyIdentifier</i> field that has the same value as the corresponding CMLA Root CA Certificate's <i>subjectKeyIdentifier</i> extension value.
SubjectKeyIdentifier extension	Mandatory, non-critical.  The key identifier method (1) is used as defined in [RFC3280] (160-bit SHA-1 hash of the <i>subjectPublicKey</i> field, excluding the tag, length, and number of unused bits).
CertificatePolicies extension	Not used
ExtKeyUsage extension	Not used
Id-pkix-ocsp-nocheck extension	Not used

#### 6.4 OCSP Responder Certificate

The OCSP Responder certificate is issued by the Rights Issuer CA and stored inside the OCSP Responder. The OCSP Responder certificate is delivered to the Device through the Rights Issuer with the OCSP response, upon request by Device.

OCSP Responder Certificates SHALL be based on OMA DRM specification appendix D.4 “OCSP Responder Certificates” [OMADRM-v2].

Certificate field name	Value
Version	V3 (integer value 2)
Signature	<i>sha-1WithRSAEncryption</i> as defined in [RFC3279]
Issuer	Same as the corresponding Rights Issuer CA <i>subject</i> field.
Validity	During 2004-2011: expires 3 months from the issuance date but not later than the certificate of the corresponding Rights Issuer CA Certificate.  CMLA will decide during 2010 the key lengths and certificate validity periods that will be used after 2011.
Subject	Contains attributes: - <i>countryName</i> (e.g. “US”) - <i>organizationName</i> (e.g. “CMLA”) - <i>commonName</i> (e.g. “OCSP Responder 1”)
SubjectPublicKeyInfo	During 2004-2011: carries a 1024 bit RSA public key identified with the <i>pkcs-1</i> algorithm identifier as defined in [RFC3279].  CMLA will decide during 2010 the key lengths and certificate validity periods that will be used after 2011.
IssuerUniqueID	Not used
SubjectUniqueID	Not used
BasicConstraints extension	Not used
CRLDistributionPoints extension	Not used
KeyUsage extension	Mandatory, critical.  Only key usage bit <i>digitalSignature</i> is set.
AuthorityKeyIdentifier extension	Mandatory, non-critical.  The extension contains only the <i>keyIdentifier</i> field that has the same value as the corresponding Rights Issuer CA Certificate’s <i>subjectKeyIdentifier</i> extension value.
SubjectKeyIdentifier extension	Mandatory, non-critical  The key identifier method (1) is used as defined in [RFC3280] (160-bit SHA-1 hash of the <i>subjectPublicKey</i> field, excluding the tag, length, and number of unused bits).
CertificatePolicies extension	Not used
ExtKeyUsage extension	Mandatory, critical.  The extension contains only the <i>id-kp-OCSPSigning</i> object identifier.
Id-pkix-ocsp-nocheck extension	Mandatory, non-critical.  The extension value is NULL.

## 6.5 Device Certificate

Device certificates are issued by a Device CA. The certificates and corresponding private keys are delivered to the Client Adopters in a Device Credentials DVD-R. Client Adopter stores them inside the Device. Device certificate is sent, upon request, from the Device to the Rights Issuer as part of the Device's certificate chain. Device certificate is validated and used by the Rights Issuer.

Device certificates SHALL be based on OMA DRM specification appendix D.1 "DRM Agent Certificates" [OMADRM-v2].

Certificate field name	Value
Version	v3 (integer value 2)
SerialNumber	<p>Non-negative integers that are unique within the set of all CMLA issued Device certificates.</p> <p>The DER encoded bytes of the <i>serialNumber</i> field are structured as follows:</p> <p><i>Bytes 1-2</i>: CA Identifier. The first byte must not be 00h or have its most significant bit set.</p> <p><i>Bytes 3-6</i>: Certificate batch identifier that is unique within the specific Device CA.</p> <p><i>Bytes 7-20</i>: 14 least significant bytes of SHA-1 hash calculated over the certified public key and current date and time.</p> <p>The serial numbering scheme may be used in the Device CRLs to revoke a batch delivery of keys and certificates using the CRL mechanism defined in chapter 8.</p>
Signature	<i>sha-1WithRSAEncryption</i> as defined in [RFC3279]
Issuer	Same as the issuing Device CA Certificate's <i>subject</i> field.
Validity	<p>During 2004-2011: Expires 20 years from the issuance date but not later than the certificate of the corresponding Device CA Certificate.</p> <p>CMLA will decide during 2010 the key lengths and certificate validity periods that will be used after 2011.</p>
Subject	<p>Contains always attributes:</p> <ul style="list-style-type: none"> <li>- <i>organizationName</i> as assigned by CMLA for the specific Client Adopter</li> <li>- <i>serialNumber</i> as assigned by CMLA by calculating SHA-1 hash over the certified public key</li> </ul> <p>May contain additional attributes as defined in [OMADRM-v2]. The values of those fields are assigned by the Client Adopter.</p>
SubjectPublicKeyInfo	<p>During 2004-2011: Carries a 1024 bit RSA public key identified with the <i>pkcs-1</i> algorithm identifier as defined in [RFC3279].</p> <p>CMLA will decide during 2010 the key lengths and certificate validity periods that will be used after 2011. Current expectation is that 1536 bit Device keys will be used after 2011.</p>
IssuerUniqueID	Not used
SubjectUniqueID	Not used

BasicConstraints extension	Not used
CRLDistributionPoints extension	(a) Certificates issued by a CMLA managed Device CA: Not used. (b) Certificates issued by a distributed Device CAs (future extension): Mandatory, non-critical. The extension contains only one <i>cRLIssuer</i> field that defines the distinguished name of the Device CRL Issuer being responsible for producing the CRLs governing this certificate.
KeyUsage extension	Mandatory, critical. Only key usage bits <i>digitalSignature</i> and <i>keyEncipherment</i> are set.
AuthorityKeyIdentifier extension	Mandatory, non-critical. The extension contains only the <i>keyIdentifier</i> field that has the same value as the corresponding Device CA Certificate's <i>subjectKeyIdentifier</i> extension value.
SubjectKeyIdentifier extension	Not used
CertificatePolicies extension	Not used
ExtKeyUsage extension	Mandatory, critical. The extension contains only the <i>oma-kp-drmAgent</i> object identifier.
Id-pkix-ocsp-nocheck extension	Not used

## 6.6 Rights Issuer Certificate

Rights Issuer certificate is issued by the Rights Issuer CA. Rights Issuer certificate is sent, upon request, from the Rights Issuer to the Device as a part of the Rights Issuer's certificate chain.

Rights Issuer Certificates SHALL be based on OMA DRM specification appendix D.2 "Rights Issuer Certificates" [OMADRM-v2].

Certificate field name	Value
Version	v3 (integer value 2)
Signature	<i>sha-1WithRSAEncryption</i> as defined in [RFC3279]
Issuer	Same as the issuing Rights Issuer CA Certificate's <i>subject</i> field.
Validity	During 2004-2011: Expires 5 years from the issuance date but not later than the certificate of the corresponding Rights Issuer CA Certificate. CMLA will decide during 2010 the key lengths and certificate validity periods that will be used after 2011.
Subject	Contains always attributes: - <i>organizationName</i> (e.g. "CoolService Ltd.") as assigned by CMLA for the specific Service Provider May contain additional attributes as defined in [OMADRM-v2]. The

	values of those fields are assigned by the Service Provider.
SubjectPublicKeyInfo	During 2004-2011: Carries a 1024 bit RSA public key identified with the <i>pkcs-1</i> algorithm identifier as defined in [RFC3279].  CMLA will decide during 2010 the key lengths and certificate validity periods that will be used after 2011. Current expectation is that 1536 bit Device keys will be used after 2011.
IssuerUniqueId	Not used
SubjectUniqueId	Not used
BasicConstraints extension	Not used
CRLDistributionPoints extension	Not used
KeyUsage extension	Mandatory, critical.  Only key usage bit <i>digitalSignature</i> is set.
AuthorityKeyIdentifier extension	Mandatory, non-critical.  The extension contains only the <i>keyIdentifier</i> field that has the same value as the corresponding Rights Issuer CA Certificate's <i>subjectKeyIdentifier</i> extension value.
SubjectKeyIdentifier extension	Not used
CertificatePolicies extension	Not used
ExtKeyUsage extension	Mandatory, critical.  The extension contains only the <i>oma-kp-rights/issuer</i> object identifier.
Id-pkix-ocsp-nocheck extension	Not used

## 7. OCSP Profile

The OCSP service shall be based on and compliant with the OMA OCSP profile [OMAOCSPP]. Whilst that document takes a device-centric view, this section specifies the service that shall be provided. The OCSP transport binding to HTTP, as specified in Appendix A of [RFC2560], SHALL be used. The OCSP Responder SHALL authenticate the OCSP requestor using HTTP digest authentication method defined in [RFC2617]. The username and password are provided to the Service Provider by CMLA.

A Rights Issuer certificate chain consists of the Rights Issuer leaf certificate and the Rights Issuer CA certificate. The Device MUST check, as specified in [OMADRM-v2], the revocation status of the Rights Issuer certificate. The Device is not required to check the revocation status of the Rights Issuer CA certificate. In other words, the complete set of OCSP responses required to check the revocation status of the whole Rights Issuer certificate chain consists of only one OCSP response vouching for the revocation status of the Rights Issuer leaf certificate.

### 7.1 OCSP Requests

OCSP requests are made by Rights Issuers on behalf of the devices.

OCSP Request Field	Value
TbsRequest	Fields as defined below
Version	v1 (integer value 0)
RequestorName	Optional. May be included in request, but will be ignored by the OCSP responder.
RequestList	SHOULD contain only one <i>Request</i> ASN.1 structure
Request	SHOULD contain only one <i>ReqCert</i> ASN.1 structure
ReqCert	CertID fields as defined below
Hash-Algorithm	<i>id-sha1</i> as defined in [RFC3279].
IssuerName-Hash	Hash of Issuer's DN. The hash value SHALL NOT be truncated.
IssuerKey-Hash	Hash of Issuer's public key. The hash value SHALL NOT be truncated.
Serialnumber	Certificate serial number
SingleRequest-Extension	Not used.
RequestExtensions	Extensions that have not been defined below SHOULD NOT be used.
Nonce (id-pkix-ocsp-nonce)	Optional. Non-critical. If given, the nonce value will be repeated in the response. The value of the nonce originates from the device as defined in [OMADRM-v2].
OptionalSignature	Optional May be included in request, but will be ignored by OCSP responder.

## 7.2 OCSP Responses

OCSP Response Field	Value
ResponseStatus	The status for the response ( <i>successful</i> , <i>malformedRequest</i> , <i>internalError</i> , <i>tryLater</i> , <i>sigRequired</i> , <i>unauthorized</i> as appropriate).
ResponseBytes	<i>ResponseBytes</i> fields as defined below
ResponseType	<i>id-pkix-ocsp-basic</i>
Response	OCTET STRING consisting of <i>BasicOCSPResponse</i> as defined below
BasicOCSPResponse	
TbsResponseData	ResponseData fields as defined below
Version	v1 (integer value 0)
ResponderID	The KeyHash method shall be used (and not the Name) method. The KeyHash SHALL be calculated as specified in [RFC2560], so that it matches the subjectKeyIdentifier extension of the OCSP responder's certificate.

ProducedAt	Time of producing this OCSP Response
Responses	<i>SingleResponse</i> fields as defined below
CertID	<i>CertID</i> fields as defined below
HashAlgorithm	<i>id-sha1</i> as defined in [RFC3279]. The hash values SHALL NOT be truncated.
IssuerName-Hash	Hash of Issuer's DN
IssuerKeyHash	Hash of Issuer's public key
Serialnumber	Certificate serial number
CertStatus	Choice of <i>Good</i> , <i>revoked</i> , <i>unknown</i> ASN.1 structures as appropriate.
ThisUpdate	These will be taken from the <i>thisUpdate</i> field of the CRL on which the response is based.
NextUpdate	These will be taken from the <i>nextUpdate</i> field of the CRL on which the response is based.
SingleExtensions	Not used.
Response-Extensions	Extensions that have not been defined below SHOULD NOT be used.
Nonce (id-pkix-ocsp-nonce)	Optional. Non-critical. If given, the nonce value will be repeated in the response. The value of the nonce originates from the device as defined in [OMADRM-v2].
SignatureAlgorithm	sha1WithRSAEncryption as defined in [RFC3279].
Signature	Signature bits as calculated according to the signature algorithm.
Certs	The OCSP responder certificate will be provided with each response given by the OCSP responder to the Rights Issuer. However, this certificate may be stripped from the OCSP response before the response is passed to the device.  The RI CA and root CA certificates are not provided, since they must be present at the device already.

## 8. Device CRL, Rights Issuer CRLs and CMLA ARLs

CMLA ARL is issued by the CMLA Root CA and delivered to and used by the Rights Issuer (to verify the revocation status of the Device CA).

The structure of CMLA CRLs and ARLs is based on [RFC3280]. All CRLs and ARLs SHALL be DER encoded. All attributes of the issuer and subject fields SHALL be encoded as type UTF8String.

```

CertificateList ::= SEQUENCE {
    tbsCertList      TBSCertList,
    signatureAlgorithm AlgorithmIdentifier,
    signatureValue   BIT STRING }

TBSCertList ::= SEQUENCE {
    version          Version OPTIONAL,
                   -- if present, MUST be v2
    signature        AlgorithmIdentifier,

```

```

    issuer           Name,
    thisUpdate      Time,
    nextUpdate      Time OPTIONAL,
    revokedCertificates SEQUENCE OF SEQUENCE {
        userCertificate CertificateSerialNumber,
        revocationDate Time,
        crlEntryExtensions Extensions OPTIONAL
                                -- if present, MUST be v2
    } OPTIONAL,
    crlExtensions   [0] EXPLICIT Extensions OPTIONAL
                                -- if present, MUST be v2
    }
    
```

### 8.1 Device CRL

Device CRL is issued by the Device CA and it contains revocation status information for Device certificates issued by that Device CA or Distributed Device CA(s). It is delivered to and used by the Rights Issuer to verify the revocation status of a Device certificate.

The Device CRL SHALL be based on IETF’s [RFC3280].

CRL field name	Value
Version	v2 (integer value 1)
Signature	<i>sha-1WithRSAEncryption</i> as defined in [RFC3279]
Issuer	Same as the CMLA managed Device CA Certificate’s <i>subject</i> field.
ThisUpdate	The date of issuance
NextUpdate	180 hours. CMLA may modify this value based on operational considerations.
RevokedCertificates entries:	
UserCertificate	Revoked certificate serial number
RevocationDate	Date of revocation decision.
CertificateRange CRL entry extension	Optional, critical. Used as defined below when batch revocation is needed. Rights Issuers MUST be able to support this extension.
AuthorityKeyIdentifier extension	Mandatory, non-critical. The extension contains only the <i>keyIdentifier</i> field that has the same value as the issuing Device CA Certificate’s <i>subjectKeyIdentifier</i> extension value.
CRLNumber extension	Mandatory, non-critical. The value is a monotonically increasing sequence number.
IssuingDistributionPoint	Optional, critical This extension is used by distributed Device CAs. The boolean <i>indirectCRL</i> field is set to TRUE. Rights Issuers MUST be able to support this extension.

#### 8.1.1 CertificateRange CRL Entry Extension

The *CertificateRange* extension is a CMLA defined non-standard CRL entry extension that MAY be used in CRLs issued by CMLA. This extension can be used to revoke a batch of certificates identified with a certificate serial number range. The usefulness of this extension depends on the certificate serial numbering scheme and it is expected to be used in cases when a CMLA Device Credentials DVD-R has been compromised for some reason.

The extension defines a range of certificate serial numbers starting from the value of the *userCertificate* field of the specific CRL entry and ending in the value of the *lastRevokedCertificate* field value of the *CertificateRange* extension. All certificate serial numbers that are within these two integers (both ends inclusive) MUST be considered to be revoked.

The value of the *lastRevokedCertificate* field MUST be larger than the value of the *userCertificate* field.

The extension SHALL be marked as critical.

```
id-xx-CMLA-revokedCertificateRange OBJECT IDENTIFIER ::= { iso(1) org(3) dod(6) internet(1)
  private(4) enterprise(1) content-management-license-administrator(21888) 1 }

lastRevokedCertificate ::= CertificateSerialNumber
```

### 8.1.2 Certificate Issuer CRL Entry Extension

The Certificate Issuer CRL Entry Extension SHALL be used for CRLs that have revoked certificates not issued by the CA issuing the CRL, as defined in [RFC3280]. This will be the case when certificates issued by a Distributed Device CA have been revoked.

The extension SHALL be marked as critical. Rights Issuers MUST be able to support this extension.

## 8.2 Rights Issuer CRL

Rights Issuer CRL is issued by the Rights Issuer CA and it contains revocation status information for Rights Issuer certificates issued by that CA. It is delivered to and used by the OCSF Responder to produce OCSF responses and Rights Issuers for verifying the revocation status of OCSF Responder certificates.

The Rights Issuer CRL SHALL be based on IETF's [RFC3280].

CRL field name	Value
Version	v2 (integer value 1)
Signature	<i>sha-1WithRSAEncryption</i> as defined in [RFC3279]
Issuer	Same as the issuing Rights Issuer CA Certificate's <i>subject</i> field.
ThisUpdate	The date of issuance
NextUpdate	180 hours. CMLA may modify this value based on operational considerations.

RevokedCertificates entries:	
UserCertificate	Revoked certificate serial number
RevocationDate	Date of revocation decision.
AuthorityKeyIdentifier extension	Mandatory, non-critical. The extension contains only the <i>keyIdentifier</i> field that has the same value as the issuing Rights Issuer CA Certificate's <i>subjectKeyIdentifier</i> extension value.
CRLNumber extension	Mandatory, non-critical. The value is a monotonically increasing sequence number.

### 8.3 CMLA ARL

The CMLA ARL SHALL be based on IETF's [RFC3280].

CRL field name	Value
Version	v2 (integer value 1)
Signature	<i>sha-1WithRSAEncryption</i> as defined in [RFC3279]
Issuer	Same as the issuing CMLA Root CA Certificate's <i>subject</i> field.
ThisUpdate	The date of issuance
NextUpdate	8 months. CMLA may modify this value based on operational considerations.
RevokedCertificates entries:	
UserCertificate	Revoked certificate serial number
RevocationDate	Date of revocation decision.
AuthorityKeyIdentifier extension	Mandatory, non-critical. The extension contains only the <i>keyIdentifier</i> field that has the same value as the issuing CMLA Root CA Certificate's <i>subjectKeyIdentifier</i> extension value.
CRLNumber extension	Mandatory, non-critical. The value is a monotonically increasing sequence number.

## 9. Device CRL, CMLA ARL and Device CA Certificate delivery mechanism

Device CRLs, Rights Issuer CRLs and CMLA ARLs are used in the CMLA context for verifying the revocation status of Device certificate chains and OCSP Responder Certificates. Device CA Certificates are used to verify the signature of Device CRLs. The same delivery mechanism defined in this chapter applies for all of them.

Service Providers can fetch CRLs, ARLs and Device CA Certificates by making an HTTP GET request to URLs that can be constructed as described below.

```
URL = "http:" "/" host [ ":" port ] [ abs_path [ "?" query ] ]
```

The following rules SHALL apply for the URLs:

- The *host* name is a parameter defined by CMLA
- The *abs\_path* for a Device CRL SHALL be “/cmla/DeviceCRL/download.aspx” and the query string SHALL be “CN=*name*”, where *name* is the URL [RFC1738] encoding of the *commonName* attribute of the Device CA’s distinguished name.
  - o Example: “<http://examplehost.cm-la.com/cmla/DeviceCrl/download.aspx?CN=Device%20CA%20I>”
- The *abs\_path* for a Rights Issuer CRL SHALL be “/cmla/RightsCrl/download.aspx” and the query string SHALL be “CN=*name*”, where *name* is the URL [RFC1738] encoding of the *commonName* attribute of the Rights Issuer CA’s distinguished name. Example: “<http://examplehost.cm-la.com/cmla/RightsCrl/download.aspx?CN=Rights%20Issuer%20CA%20I>” The *abs\_path* for an ARL SHALL be “/cmla/ARL/download.aspx” and the query string shall be “CN=*name*”, where *name* is the URL [RFC1738] encoding of the *commonName* attribute of the Root CA’s distinguished name.
  - o Example: “<http://examplehost.cm-la.com/cmla/ARL/download.aspx?CN=Root%20CA>”
- The *abs\_path* for Device CA Certificate SHALL be “/cmla/DeviceCACertificate/download.aspx” and the query string SHALL be “CN=*name*”, where *name* is the URL [RFC1738] encoding of the *commonName* attribute of the Device CA’s distinguished name.
  - o Example: “<http://examplehost.cm-la.com/cmla/DeviceCACertificate/download.aspx?CN=Device%20CA%20I>”

The URLs will be populated with the most recently issued CRL/ARL or Device CA Certificate.

The CRL/ARL and Device CA Certificate SHALL be delivered to the Service Provider in the body part of the HTTP response message for the HTTP GET request made to the specified URL. The MIME-type of the CRL/ARL SHALL be “application/pkix-crl”. The MIME-type of the Device CA Certificate SHALL be “application/pkix-cert”.

As defined in the [RFC3280], the CRL Issuer will issue a new CRL/ARL and make it available from the specified URL before the NextUpdate value of the most recent CRL/ARL has been passed. In addition to the periodic updates the CRL Issuer will issue a new CRL/ARL when new certificates have been revoked. Service Providers are recommended to attempt to pull the new CRL/ARL well in advance of the NextUpdate time of the most recent CRL/ARL, even though that may result in downloading the same CRL/ARL several times. The Service Provider is recommended to use the *If-Modified-Since* request header as defined in [RFC2616] chapter 14.25 to avoid unnecessary downloads of the same CRL/ARL.

The HTTP server SHALL authenticate the requesting entity using HTTP digest authentication method defined in [RFC2617]. The username and password are provided to the Service Provider by CMLA. In case of invalid username and/or password the HTTP error code 401 “Unauthorized” will be returned.

## 10. Device Credentials Delivery

CMLA mechanism and format for the secure transfer of Client Adopter keys and certificates is based on the PKCS#12 standard. The Client Adopter keys and corresponding certificates are generated and certified by Device CA, and distributed securely from CMLA to the Client Adopter in a DVD-R.

### 10.1 Key and Certificate Delivery Procedure

The first two process steps are required to be completed before any Client Adopter key orders can be made

1. CMLA generates a CMLA Transport Key that is a 2048 bit RSA key pair and communicates the public key part to the Client Adopter during the process of agreement execution between CMLA and the Client Adopter. The CMLA Transport Key is used to protect the integrity and authenticity of the Device Credentials DVD-R. CMLA will not use the same CMLA Transport Key indefinitely and thus the Client Adopter must be able to replace the old CMLA Transport Key with a newer one, after notice from CMLA. The public parts of CMLA Transport Keys are delivered to the Client Adopter in a DVD-R that contains one file per each CMLA Transport Key. The file naming convention is “CMLA\_Transport\_Key\_##.der”, where ## is replaced with a two digit number. Each file contains the DER encoding of the `RSAPublicKey` structure defined in [PKCS#1].  
CMLA delivers to the Client Adopter, in a secure out-of-band mechanism, a list of public key file names and corresponding SHA-1 hash values calculated over the file (20 bytes in hexadecimal representation). Client Adopter must verify that each hash value matches the file on the DVD-R.
2. CMLA delivers valid CMLA Root CA certificates to Client Adopters using a similar mechanism as above.  
The same DVD-R contains one file for each CMLA Root CA certificate. The file naming convention is “CMLA\_Root\_CA\_Certificate\_##.der”, where ## is replaced with a two digit number. Each file contains the DER encoded CMLA Root CA certificate.  
CMLA delivers to the Client Adopter, in a secure out-of-band mechanism, a list of CMLA Root CA certificate file names and corresponding SHA-1 hash values calculated over the file (20 bytes in hexadecimal representation). Client Adopter must verify that each hash value matches the file on the DVD-R.
3. Client Adopter signs the Client Adopter Agreement Exhibit G that defines (a) the persons that are authorized to generate key orders, (b) one delivery address per authorized person and (c) one Client Adopter Transport Key per authorized person (the same key may be used by multiple authorized persons).  
The Client Adopter Transport Key is a 1024 bit RSA key pair and the Exhibit G includes the hexadecimal representation of the SHA-1 hash calculated over the DER encoding of the public key as stored in the file.  
The public keys are delivered to CMLA in a DVD-R that contains one file per each Client Adopter Transport Key. The file naming convention is “Client\_Adopter\_Transport\_Key\_\*\_##.der”, where \* is replaced with the

Client Adopter name and ## is replaced with a two digit number. Each file contains the DER encoding of the RSAPublicKey structure defined in [PKCS#1].

The ASN.1 definition for the RSAPublicKey structure from [PKCS#1] is given below.

```
RSAPublicKey ::= SEQUENCE {
    modulus          INTEGER,  -- n
    publicExponent  INTEGER  -- e
}
```

The subsequent steps define how the actual key order and delivery process.

4. Client Adopter sends its key order to CMLA:
  - 4.1 An authorized person of the Client Adopter submits key order to CMLA via Fax and a scanned email attachment or by signed email.
  - 4.2 CMLA will review and verify each order submitted, and if complete & accurate, generate an invoice.
  - 4.3 The Invoice is sent via email and hard copy to the Client Adopter.
  - 4.4 Upon receipt of payment on invoice, the key order is then processed and responded.
5. According to the key order, Device CA generates requested number of key pairs and certificates, packs these data and relevant authority certificates into one or more files based on PKCS#12 standard as described in the next sections. The files are burned to a DVD-R that is delivered to the Client Adopter.  
The Client Adopter validates the signature and decrypts the Device Credentials DVD-R files.
6. CMLA registers into certificate database necessary information such as the keys and certificates issued by Device CA, request data, and key and certificate delivery data.

## 10.2 Device Credentials DVD-R

The Device Credentials DVD-R contains the set of security credentials that is needed to manufacture CMLA Compliant Devices. The credentials are stored in the DVD-R in two types of files:

- Rights Issuer CA Certificates file
- Device Credentials files

The table below describes the file types in more detail.

File type	Contains	File naming convention	Comment
Rights Issuer CA Certificates	One or more Rights Issuer CA certificates.	"Rights_Issuer_CA_Certs.der"	<p>There will be only one file of this type in the DVD-R.</p> <p>Typically there will be only one Rights Issuer CA certificate in this file, but the Client Adopter must not assume that.</p> <p>Client Adopters may choose not to store the Rights Issuer CA certificates to manufactured Devices i.e. the use of this file is optional.</p>

Device Credentials	One Device CA certificate, and 1000 Device private key and certificate pairs (issued by the corresponding Device CA)	<p>"Device_Credentials_*_####.p12"</p> <p>* is replaced with the hexadecimal encoding of the CA Identifier (2 bytes) and Batch number (4 bytes)</p> <p>#### is replaced with a four digit sequence number differentiating the files from each other. First file is identified with "0001" and the number is incremented by one for subsequent files.</p> <p>Example: "Device_Credentials_0002003F270C_0019.p12"</p>	There will be one or more files of this type in the DVD-R. The maximum number of files is 1000. As each file contains 1000 Device private keys and certificates one DVD-R contains security credentials for a maximum of 1.000.000 Devices.
--------------------	--	---	---

The format of each file is defined in the next sections.

### 10.3 Rights Issuer CA Certificates file

The Rights Issuer CA certificates file contains one or more DER encoded Rights Issuer CA Certificates. Before using the Rights Issuer CA certificates on the DVD-R the Client Adopter must validate that each certificate chains back to the CMLA Root CA. Using the Rights Issuer CA certificates is optional for the Client Adopter.

### 10.4 PKCS#12 File Format

Device Credentials files are based on [PKCS#12] and [RFC3369].

Cryptographic Message Syntax (CMS) [RFC3369] stores all data using ContentInfo structures. Three ContentInfo types, Data, SignedData and EnvelopedData are used in the key and certificate delivery format.

- Data is just a wrapper for raw, unauthenticated, unencrypted arbitrary data.
- SignedData is a wrapper allowing a digital signature to be applied to arbitrary data, thereby giving authentication.
- EnvelopedData is a wrapper allowing the data to be encrypted using a symmetric message key, which in turn is encrypted under an RSA key. This gives confidentiality.

The cryptographic algorithms that are used are listed below.

- Message digest algorithm: FIPS 180-1 defines the SHA-1 message digest algorithm [FIPS180-2]. The algorithm identifier assigned by ISO is 1.3.14.3.2.26
- Asymmetric signature algorithm: PKCS#1v1.5 RSA signature with SHA1 hash has algorithm identifier = 1.2.840.113549.1.1.5 [PKCS#1]
- Asymmetric encryption algorithm: PKCS#1v1.5 RSA encryption has algorithm identifier = 1.2.840.113549.1.1.1 [PKCS#1]
- Symmetric encryption algorithm: AES with 128-bit keys, 128-bit block length, in CBC mode has algorithm identifier = 2.16.840.1.101.3.4.1.2 [FIPS197]

The top-level structure from PKCS#12 is a 'PFX'.

```
PFX ::= SEQUENCE {
    version Integer,
```

```
authSafe ContentInfo }
```

The overall PKCS #12 structure is illustrated in the figure below.

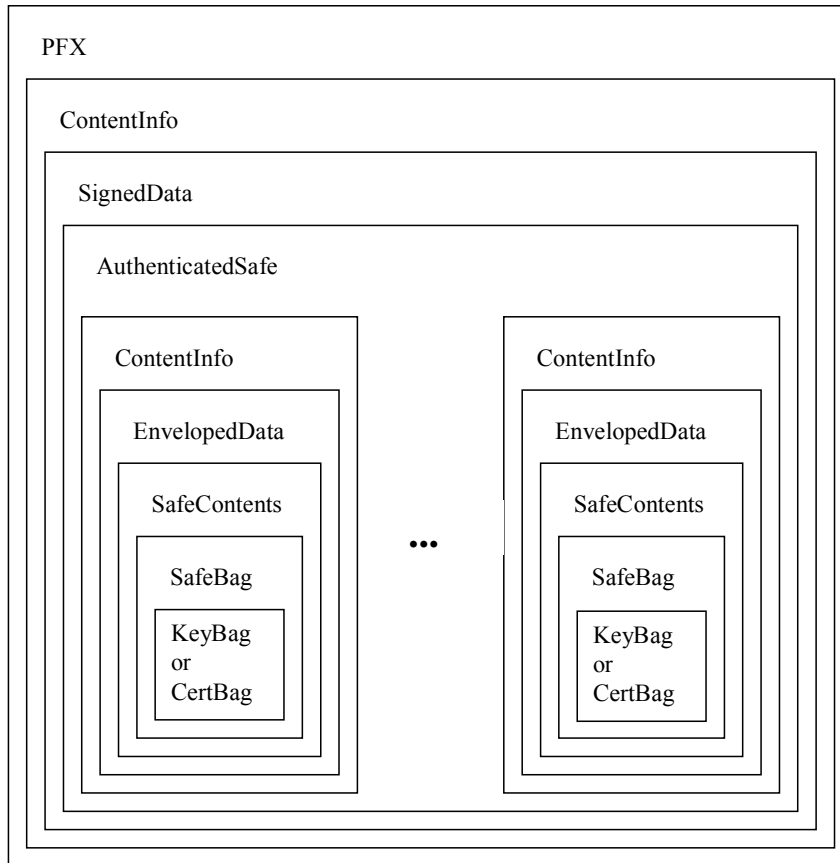


Figure 2. PKCS #12 Structure

The `ContentInfo` in the PFX comes from CMS [RFC3369], and SHALL contain a `SignedData`, which contains the encoding of an `AuthenticatedSafe` (below) as well as a digital signature and reference to the CMLA Transport Key used for a `SignerIdentifier`. `SignerIdentifier` is a `SubjectKeyIdentifier` which in turn is a SHA-1 hash over `SubjectPublicKeyInfo`. `SubjectPublicKeyInfo` shall be constructed from the CMLA Transport Key which has the `RSAPublicKey` structure. This top-level signature provides integrity and authenticity protection for all the data in the file.

The signed data is an `AuthenticatedSafe` structure:

```
AuthenticatedSafe ::= SEQUENCE OF ContentInfo
```

This SEQUENCE OF contains a number of ContentInfo structures, each containing an EnvelopedData structure. These in turn contain the encryption of a SafeContents, together with key identifier information referencing the Client Adopter Transport Key that is used for a RecipientIdentifier. RecipientIdentifier is a SubjectKeyIdentifier which in turn is a SHA-1 hash over SubjectPublicKeyInfo. SubjectPublicKeyInfo shall be constructed from the Client Adopter Transport Key which has the RSAPublicKey structure.

```
SafeContents ::= SEQUENCE OF SafeBag
```

Only one SafeBag structure will be present in each SafeContents structure.

```
SafeBag ::= SEQUENCE {
    bagId ObjectIdentifier,
    bagValue [0] EXPLICIT ANY DEFINED BY bagId,
    bagAttributes SET OF PKCS12Attribute OPTIONAL
}
```

The bagValue can contain either a KeyBag or a CertBag as identified by the bagId:

```
KeyBag ::= PrivateKeyInfo

CertBag ::= Sequence {
    certId ObjectIdentifier,
    certValue EXPLICIT ANY DEFINED BY certId }
```

where PrivateKeyInfo is as defined in [PKCS#8] and certValue is an octet string containing the DER encoding of an X.509 certificate.

For Device Credentials files: the first ContentInfo structure contains the Device CA certificate and subsequent ContentInfos contain Device private key and certificate pairs, each private key and certificate in separate ContentInfo structures.

For example, with two Device private keys and two corresponding Device certificates, there will be five ContentInfo structures, each containing an EnvelopedData, each in turn containing an encrypted SafeContents that contains a single SafeBag. These five SafeBags will contain, in sequence:

1. The Device CA Certificate issuing all Device certificates in this file
2. The first Device private key
3. The Device certificate corresponding to the first Device private key
4. The second Device private key
5. The Device certificate corresponding to the second Device private key

To assist with matching keys to certificates, the optional bagAttributes field of each SafeBag structure SHALL be used for each private key and end-entity certificate, containing identifiers unique to each key and certificate pair. The localKeyId mechanism from [PKCS#12] and [PKCS#9] shall be used, using an octet string formed from the SHA-1 hash of the public key (the computation method is irrelevant to the party receiving the file, they just need to match the localKeyId from the KeyBag to the localKeyId on the CertBag).

In general, a new session key could be used with each EnvelopedData in the PKCS#12 file. As a performance optimization, only a single session key shall be used, i.e. the same session key shall be re-used for each EnvelopedData in a single file (but shall never be re-used between two separate files). This gives a potential performance optimization, in that only one RSA decryption is required to obtain the session key, but existing tools not supporting this optimization are still able to process the PKCS#12 file without modification.

Each PKCS#12 file shall contain exactly 1000 pairs of Device private key and certificate. All key orders must therefore be made in multiples of 1000. The maximum order size shall be 1.000.000 key and certificate pairs. The same certificate batch number shall be used for all keys in the batch (and so shall be shared across all the certificates in all the files in that batch).

## 11. Rights Issuer Certification

As a prerequisite for issuing a Rights Issuer certificate for the Service Provider, it is assumed that the Service Provider has already generated a Rights Issuer RSA key pair in an environment that satisfies the security requirements stipulated by the Rights Issuer Robustness Rules, another prerequisite is for the Service Provider to have fulfilled the Exhibit G of the Service Provider Agreement and to have communicated it to CMLA.. After that has been done, the first time certification and subsequent re-certifications are done as defined below.

1. Service Provider creates self-signed DER encoded PKCS#10 certificate request [PKCS#10], burns a DVD-R containing the request, and sends the DVD-R to CMLA. This request SHALL be signed using sha-1WithRSAEncryption as defined in [RFC3279].  
The file naming convention is “Rights\_Issuer\_\*\_##\_p10”, where \* is replaced with the Service Provider’s name and ## is replaced with a two digit number.  
The Service Provider also submits the Rights Issuer Certificate ordering form to CMLA via signed email and the original via regular courier. . This ordering form will include the SHA1 hash of the public key (PKCS#1 RSAPublicKey structure in the subjectPublicKey field) inside the PKCS#10 request PKCS#10 file (20 bytes, encoded in hexadecimal).
2. CMLA will review and verify each request submitted. After verification, CMLA generates an invoice.
3. The Invoice is sent via email and hard copy to the Service Provider.
4. Service Provider makes payment on invoice.
5. Upon receipt of payment on invoice, CMLA processes order and responds. Order processing includes the verification of the hash value provided in the Rights Issuer Certificate ordering form to the hash value calculated over the to be certified public key. If none of the verifications fail CMLA creates, according to the certificate request, a Rights Issuer Certificate and delivers it with the corresponding Rights Issuer CA Certificate to the Service Provider in a DVD-R.  
The file naming convention for the Rights Issuer Certificate is “Rights\_Issuer\_\*\_##\_der”, where \* is replaced with the Service Provider’s name and ## is replaced with the two digit number. The file contains the DER encoding of the Rights Issuer Certificate

The file naming convention for the Rights Issuer CA Certificate is “Rights\_Issuer\_CA\_Certs.der”. The file contains the DER encoding of the Rights Issuer CA Certificate.

6. CMLA registers into certificate database necessary information such as the certificate issued by the RI CA, request data, and certificate delivery data.

CMLA Root CA Certificates are delivered to Service Providers in the same way as to the Client Adopters (check chapter 10.1 step 2 for details).

## 12. CMLA IP

This chapter defines seven CMLA Key Transport Algorithms that all Devices and Rights Issuers MUST support, in addition to the mandatory key transport algorithms defined in [OMADRM-v2].

Each CMLA Key Transport Algorithm contains patent pending CMLA IP technology that is owned by CMLA and licenced according to the CMLA agreements. An entity that has not acquired a licence for the CMLA IP can not use CMLA Digital Content without infringing CMLA IP patents. As a result, CMLA IP prevents such entities from participating in the CMLA ecosystem.

Each CMLA Key Transport Algorithm is identified with an algorithm identifier. The identifiers are used in the ROAP PDUs defined [OMADRM-v2] sections 5.4.2.1 Device Hello, 5.4.2.2 RI Hello and as the value of the `Algorithm` attribute of the `<xenc:EncryptionMethod>` element in a similar way as the OMA mandatory Key Transport Algorithm defined in [OMADRM-v2] section 7.2 Key Transport Mechanisms.

The CMLA Key Transport Algorithms are parameterized by the choice of an Asymmetric Encryption Scheme, a Key Derivation Function, and a Key Wrapping Scheme. For each of these either the scheme defined in [OMADRM-v2] or the scheme defined in this specification is used, according to the table below. See also section 7.2 of [OMADRM-v2].

Algorithm Identifier	Asymmetric Encryption Scheme	Key Derivation Function	Key Wrapping Scheme
<a href="http://www.cm-la.com/tech/cmlaip/cmlaip#cmlaip-1">http://www.cm-la.com/tech/cmlaip/cmlaip#cmlaip-1</a>	CMLA_RSA_Encrypt and CMLA_RSA_Decrypt	CMLA_KDF	CMLA_WRAP and CMLA_UNWRAP
<a href="http://www.cm-la.com/tech/cmlaip/cmlaip#cmlaip-2">http://www.cm-la.com/tech/cmlaip/cmlaip#cmlaip-2</a>	CMLA_RSA_Encrypt and CMLA_RSA_Decrypt	CMLA_KDF	AES-Wrap
<a href="http://www.cm-la.com/tech/cmlaip/cmlaip#cmlaip-3">http://www.cm-la.com/tech/cmlaip/cmlaip#cmlaip-3</a>	CMLA_RSA_Encrypt and CMLA_RSA_Decrypt	OMA_KDF	CMLA_WRAP and CMLA_UNWRAP
<a href="http://www.cm-la.com/tech/cmlaip/cmlaip#cmlaip-4">http://www.cm-la.com/tech/cmlaip/cmlaip#cmlaip-4</a>	CMLA_RSA_Encrypt and CMLA_RSA_Decrypt	OMA_KDF	AES-Wrap
<a href="http://www.cm-la.com/tech/cmlaip/cmlaip#cmlaip-5">http://www.cm-la.com/tech/cmlaip/cmlaip#cmlaip-5</a>	RSA.ENCRYPT and RSA.DECRYPT	CMLA_KDF	CMLA_WRAP and CMLA_UNWRAP
<a href="http://www.cm-la.com/tech/cmlaip/cmlaip#cmlaip-6">http://www.cm-la.com/tech/cmlaip/cmlaip#cmlaip-6</a>	RSA.ENCRYPT and RSA.DECRYPT	CMLA_KDF	AES-Wrap
<a href="http://www.cm-la.com/tech/cmlaip/cmlaip#cmlaip-7">http://www.cm-la.com/tech/cmlaip/cmlaip#cmlaip-7</a>	RSA.ENCRYPT and RSA.DECRYPT	OMA_KDF	CMLA_WRAP and CMLA_UNWRAP

Both the Device and RI implementations MUST support all seven algorithms defined above.

To notify RIs that Device can use CMLA IP, Device MUST send the seven CMLA IP algorithm URIs in the `<supportedAlgorithm>` parameter in ROAP-DeviceHello message. If the Device indicated support of CMLA IP algorithms, RI MUST select the following Key Transport Algorithm identifier in the `<selectedAlgorithm>` parameter in ROAP-RIHello message.

“<http://www.cm-la.com/tech/cmlaip/cmlaip#cmlaip-1>”

The usage of other algorithms is not specified in the current specification.

Rights Issuer MUST use the Key Transport Algorithm identified above to transport key materials to the Device (MAC-key, REK, Domain Key). The used algorithm must be expressed correctly in the `Algorithm` attribute of the `<xenc:EncryptionMethod>` child element of the `<encKey>` element of ROAP-ROResponse and ROAP-JoinDomainResponse messages.

In case there is a dispute, the `Algorithm` attribute of the `<xenc:EncryptionMethod>` element takes precedence over the algorithm identifier stored in the Device's RI context.

## 12.1 Overview

CMLA IP consists of the following cryptographic primitives:

- Data-dependent Transformation with Permutation (CMLA\_DDT\_Perm),
- Data-dependent Transformation with Exponentiation (CMLA\_DDT\_Exp),
- CMLA Key Derivation Function (CMLA\_KDF),
- CMLA Key Wrap Scheme (CMLA\_WRAP and CMLA\_UNWRAP) and
- CMLA RSA Encryption Scheme (CMLA\_RSA\_ENCRYPT and CMLA\_RSA\_DECRYPT).

CMLA Data-dependent Transformation with Permutation operates on a 64-bit string. Before transformation, the input is divided into 9 blocks. The first block consists of 8 bits. The remaining 8 blocks consist of 7 bits. The first block is used to select a transformation, which is then applied to the remaining part of the input. See Section 12.2.1 and 12.2.2 for details.

CMLA Data-dependent Transformation with Exponentiation operates on a 24-bit string. Before transformation, the input is divided into 2 blocks. The first block consists of 8 bits. The remaining block consists of 16 bits. The first block is used in index of exponentiation of an integer derived from the second block. The exponentiation is calculated in modulo the prime  $p=2^{16}+1=65537$ . See Section 12.2.3 and 12.2.4 for details.

CMLA Key Derivation Function operates on an octet string of length 128. The input splits in two blocks of equal length. A constant value is concatenated to the first block. Then SHA-1 is applied to the preceding result. The result and two integers derived from the second block are used in a modulus calculation to produce the Key Encrypting Key. The output shall be 16 bytes. See Section 12.3 for details.

CMLA Key Wrap Scheme wraps key data of length 32-byte using a 16-byte key encryption key. The key data splits in two blocks and CMLA\_DDT\_Perm is applied to each block. The AES Key Wrap Algorithm is applied to the preceding result. The output shall be 40 bytes. See Section 12.4 for details.

CMLA RSA Encryption Scheme produces a cipher text from a plaintext of length 128 bytes using a device public key. The input splits into two blocks and CMLA\_DDT\_Exp is applied to each block. The RSA Encryption without padding scheme is applied to the preceding result. The output shall be 128 bytes. See Section 12.5 for details.

For octet string (respectively, bit string) and integer conversion, big-endian format (known also as network byte order format) is used. See the section 5.2.5 (respectively, 5.2.4) in [ISO18033-2D].

## 12.2 CMLA Data-Dependent Transformation

In this section, we will define two CMLA\_DDTs and its inverse transformations; CMLA\_DDT\_Perm and its inverse CMLA\_DDT\_Perm\_Inv, as well as CMLA\_DDT\_Exp and its inverse CMLA\_DDT\_Exp\_Inv.

### 12.2.1 CMLA\_DDT\_Perm

CMLA\_DDT\_Perm is the CMLA Data-dependent Transformation with Permutation. It permutes last 56 bits of 64-bit input according to the first 8 bits.

#### CMLA\_DDT\_Perm ( $b$ )

**Input:**  $b$  **Bit string of length 64**

**Output:**  $B$  **Bit string of length 64**

Steps:

1. Let  $b = (b_0, b_1, \dots, b_{63})$ .
2. For  $0 \leq i \leq 7$ , set  $G_i = (b_{8+7i}, b_{9+7i}, \dots, b_{14+7i})$ , bit string of length 7.
3. The transformation works as follows for  $i=0, 1, \dots, 7$ .
  - If  $b_i = 0$ , let  $H_i = G_i$ .
  - If  $b_i = 1$ , let  $H_i = P_i(G_i)$ , where  $P_i$  is defined below.
4. Let  $B = (b_0, b_1, \dots, b_7) \parallel H_0 \parallel H_1 \parallel \dots \parallel H_7$ .
5. Output  $B$ .

The 8 permutations  $P_0, P_1, \dots, P_7$  are defined by:

Input	a	b	c	d	e	f	g
$P_0$	f	a	e	b	d	g	c
$P_1$	g	f	d	a	b	c	e
$P_2$	c	g	b	f	a	e	d
$P_3$	e	c	a	g	f	d	b
$P_4$	d	e	f	c	g	b	a
$P_5$	b	d	g	e	c	a	f
$P_6$	e	c	a	g	f	d	b
$P_7$	c	g	b	f	a	e	d

### 12.2.2 CMLA\_DDT\_Perm\_Inv

CMLA\_DDT\_Perm\_Inv is the inverse transformation of CMLA\_DDT\_Perm.

#### CMLA\_DDT\_Perm\_Inv ( $B$ )

**Input:**  $B$  **Bit string of length 64**

**Output:**  $b$  **Bit string of length 64**

Steps:

1. Let  $B = (B_0, B_1, \dots, B_{63})$ .
2. For  $0 \leq i \leq 7$ , set  $S_i = (B_{8+7i}, B_{9+7i}, \dots, B_{14+7i})$ , bit string of length 7.
3. The transformation works as follows for each  $i=0, 1, \dots, 7$ .
  - If  $B_i = 0$ , let  $T_i = S_i$ .
  - If  $B_i = 1$ , let  $T_i = P_i^{-1}(S_i)$ , where  $P_i^{-1}$  is defined below.
4. Let  $b = (B_0, B_1, \dots, B_7) \parallel T_0 \parallel T_1 \parallel \dots \parallel T_7$ .
5. Output  $b$ .

The 8 permutations  $P_0^{-1}, P_1^{-1}, \dots, P_7^{-1}$  are defined by:

Input	a	b	c	d	e	f	g
$P_0^{-1}$	b	d	g	e	c	a	f
$P_1^{-1}$	d	e	f	c	g	b	a
$P_2^{-1}$	e	c	a	g	f	d	b
$P_3^{-1}$	c	g	b	f	a	e	d
$P_4^{-1}$	g	f	d	a	b	c	e
$P_5^{-1}$	f	a	e	b	d	g	c
$P_6^{-1}$	c	g	b	f	a	e	d
$P_7^{-1}$	e	c	a	g	f	d	b

### 12.2.3 CMLA\_DDT\_Exp

CMLA\_DDT\_Exp is the CMLA Data-dependent Transformation with Exponentiation. It permutes last 16 bits of 24-bit input according to the first 8 bits.

#### CMLA\_DDT\_Exp ( $b$ )

**Input:**  $b$  **Bit string of length 24**

**Output:**  $B$  **Bit string of length 24**

Steps:

1. Let  $b = (b_0, b_1, \dots, b_{23})$ .
2. Set  $L = (b_0, b_1, \dots, b_7)$ , bit string of length 8, and set  $R = (b_8, b_9, \dots, b_{23})$ , bit string of length 16.
3. Regarding  $R$  and  $L$  as integer, compute  $Y = ((R+1)^{(2L+1)} \bmod 65537) - 1$ .
4. Let  $B = L \parallel Y$ .
5. Output  $B$ .

### 12.2.4 CMLA\_DDT\_Exp\_Inv

CMLA\_DDT\_Exp\_Inv is the inverse transformation of CMLA\_DDT\_Exp.

#### CMLA\_DDT\_Exp\_Inv ( $B$ )

**Input:**  $B$  **Bit string of length 24**

**Output:**  $b$  **Bit string of length 24**

Steps:

1. Let  $B = (B_0, B_1, \dots, B_{23})$ .
2. Set  $L = (B_0, B_1, \dots, B_7)$ , an integer, and set  $R = (B_8, B_9, \dots, B_{23})$ , an integer.
3. Compute  $e$  as the inverse of  $2L+1$  modulo  $2^{16}$ :  $e * (2L+1) = 1 \bmod 2^{16}$ .
4. Compute  $X = ((R+1)^e \bmod 65537) - 1$ .
5. Let  $b = L \parallel X$ .
6. Output  $b$ .

### 12.3 CMLA Key Derivation Function

In this section, we will define CMLA Key Derivation Function CMLA\_KDF. See the section 7.1.2 in [OMADRM-v2] for the details of OMA\_KDF.

#### 12.3.1 CMLA\_KDF

CMLA\_KDF is the CMLA Key Derivation Function.

##### CMLA\_KDF ( $x$ )

**Input:**  $X$                       **Seed value, an octet string of length 128**  
**Output:**  $KEK$                       **Key Encryption Key, an octet string of length 16**

Steps:

1. Let  $x = x_0 // x_1$ , where each  $x_i$  consists of 64 bytes octet for  $i=0,1$ .
2. Let  $C=(0x)00\ 00\ 00\ 01$  (4 bytes).
3. Compute  $Y=SHA-1(x_0 // C)$  (160 bits).
4. Let  $A$  be the first 32 bytes and  $B$  be the last 32 bytes of  $x_1$ , respectively .
5. Regarding  $Y$ ,  $A$ , and  $B$  as integers, derive key encryption key  $KEK$  taking the least significant 128 bits of  $A*Y+B \bmod p$ , where  $p=2^{192}-2^{64}-1$ .
6. Output  $KEK$ .

### 12.4 CMLA Key Wrap Scheme

In this section, we will define CMLA Key Wrap and Unwrap Schemes. CMLA Key Wrap Algorithm can be obtained by composing AES Key Wrap algorithm (AES-WRAP) and CMLA\_DDT\_Perm. CMLA Key Unwrap Algorithm can be obtained by composing AES Key Unwrap Algorithm and CMLA\_DDT\_Perm\_Inv. See the section 7.1.3 in [OMADRM-v2] for the details of AES-WRAP.

#### 12.4.1 CMLA\_WRAP

CMLA\_WRAP is the CMLA Key Wrap Algorithm.

##### CMLA\_WRAP ( $KEK, K$ )

**Input:**  $KEK$                       Key Encryption Key, an octet string of length 16  
 $K$                                       key to be wrapped, an octet string of length 32  
**Output:**  $KWrap$                       An octet string of length 40

Steps:

1. Let  $K = K_0 // K_1$ , where each  $K_i$  consists of 16 bytes octet for  $i=0,1$ .
2. Apply CMLA\_DDT\_Perm to the first 8 bytes of  $K_i$ , keeping the rest of  $K_i$  unchanged, to produce 16-byte octet string  $k_i$  for  $i=0,1$ .
3. Let  $k = k_0 // k_1$ .
4. Compute AES-WRAP ( $KEK, k$ ) to produce an octet string  $KWrap$  of 40 bytes.
5. Output  $KWrap$ .

### 12.4.2 CMLA\_UNWRAP

CMLA\_UNWRAP is the CMLA Key Unwrap Algorithm.

#### CMLA\_UNWRAP ( $KEK, KWrap$ )

<b>Input:</b>	<b><math>KEK</math></b>	<b>Key Encryption Key, an octet string of length 16</b>
	<b><math>KWrap</math></b>	<b>key to be unwrapped, an octet string of length 40</b>
<b>Output:</b>	<b><math>K</math></b>	<b>An octet string of length 32</b>

Steps:

1. Compute AES-UNWRAP ( $KEK, KWrap$ ) to produce an octet string  $k$  of length 32 octets.
2. Let  $k = k_0 // k_1$ , where each  $k_i$  consists of 16 bytes for  $i=0,1$ .
3. Apply CMLA\_DDT\_Perm\_Inv to the first 8 bytes of  $k_i$ , keeping the rest of  $k_i$  unchanged, to produce 16-byte octet string  $K_i$  for  $i=0,1$ .
4. Let  $K = K_0 // K_1$ .
5. Output  $K$ .

## 12.5 CMLA RSA Encryption Scheme

In this section, we will define RSA Encryption and Decryption Schemes. CMLA RSA Encryption Algorithm can be obtained by composing RSA Encryption and CMLA\_DDT\_Exp. CMLA RSA Decryption Algorithm can be obtained by composing RSA Decryption and CMLA\_DDT\_Exp\_Inv.

### 12.5.1 CMLA\_RSA\_ENCRYPT

CMLA\_RSA\_ENCRYPT is the CMLA RSA Encryption Algorithm.

**CMLA\_RSA\_ENCRYPT** (*pubKey*, *M*)

**Input:**            *pubKey*        Device's RSA public key  
                       *M*                Message to be encrypted, an octet string of length 128

**Output:**         *C*                Cipher text, an octet string of length 128

Steps:

1. Let  $M = M_0 // M_1$ , where each  $M_i$  consists of 64 bytes octet for  $i=0,1$ .
2. Apply CMLA\_DDT\_Exp to the first 3 bytes of  $M_i$ , keeping the rest of  $M_i$  unchanged, to produce octet string  $m_i$  for  $i=0,1$ .
3. Let  $m = m_0 // m_1$ .
4. If the integer representative of  $m$  is not strictly less than the RSA modulus, output "integer too long" and stop.
5. Encrypt  $m$  under device public key *pubKey* with RSA.ENCRYPT to produce the octet string  $c$  of length 128. That is,
 
$$c = \text{RSA.ENCRYPT}(pubKey, m).$$
6. Output  $c$ .

**12.5.2 CMLA\_RSA\_DECRYPT**

CMLA\_RSA\_DECRYPT is the CMLA RSA Decryption Algorithm.

**CMLA\_RSA\_DECRYPT** (*pKey*, *c*)

**Input:**            *pKey*            Device's RSA private key  
                       *c*                Message to be decrypted, an octet string of length 128

**Output:**         *M*                Plain text, an octet string of length 128

Steps:

1. Decrypt  $c$  under device private key *pKey* with RSA.DECRYPT to produce the octet string  $m$  of length 128. That is,
2.  $m = \text{RSA.DECRYPT}(pKey, c)$ .
3. Let  $m = m_0 // m_1$ , where each  $m_i$  consists of 64 bytes octet for  $i=0,1$ .
4. Apply CMLA\_DDT\_Exp\_Inv to the first 3 bytes of  $m_i$ , keeping the rest of  $m_i$  unchanged, to produce octet string  $M_i$  for  $i=0,1$ .
5. Let  $M = M_0 // M_1$ .
6. Output  $M$ .

### 13. CMLA Parameters

In addition to what has been specified in this document the following parameters will be communicated to adopters separately:

- CMLA Root CA certificate(s). (for both Client Adopters and Service Providers)
- Host names and username/password for fetching the Device CA certificate(s), the most recent Device CRL(s) and CMLA ARL (for Service Providers)
- URL(s) and username/password for OCSP Responder service (for Service Providers)
- Certificate Practice Statement

### 14. Development Keys and Certificates

In order to facilitate the R&D efforts of Client Adopters and Service Providers, CMLA will provide a separate development PKI and associated services that can be used for R&D purposes. The development PKI is completely separated from the production PKI and thus the liabilities for production keys, as set forth in the CMLA agreements, do not apply for the development keys.

In order to differentiate between development and production credential all file names used for development credentials MUST be prefixed with “**DEV\_**”, as listed below.

- “DEV\_CMLA\_Transport\_Key\_##.der”
- “DEV\_CMLA\_Root\_CA\_Certificate\_##.der”
- “DEV\_Client\_Adopter\_Transport\_Key\_\*\_##.der”
- “DEV\_Rights\_Issuer\_CA\_Certs.der”
- “DEV\_Device\_Credentials\_\*\_####.p12”
- “DEV\_Rights\_Issuer\_\*\_##.p10”
- “DEV\_Rights\_Issuer\_\*\_##.der”

## 15. CMLA Mobile Broadcast

**USE AND IMPLEMENTATION OF THE FUNCTIONALITY IN THIS CHAPTER REQUIRES A SEPARATE ADDENDUM TO THE CMLA TECHNOLOGY LICENSE AGREEMENT. CMLA Technology Licensees wishing to implement the functionality specified in this chapter may only do so after signing the relevant Mobile Broadcast Addendum to their CMLA Technology License. ALL OTHER USE IS STRICTLY PROHIBITED AND NO LICENSE IS HEREBY GRANTED BY ESTOPPEL OR OTHERWISE EXCEPT AS SET FORTH IN THE MOBILE BROADCAST ADDENDUM.**

### 15.1 Overview

The keys used in Broadcast Mode are delivered to the device in `device_registration_response` or `domain_registration_response` messages, in a data structure called keyset. By default, the keyset is protected with a session key (SK) during delivery, and the SK and the encrypted keyset are further protected by encrypting with the public key of the device (DP).

The steps that differ from the default keyset protection at the service provider when a CMLA algorithm is used are the following:

- additionally prepend a descriptor (`ROT_algorithm_descriptor` or `TAA_descriptor`, as specified in section 15.2 to the keyset to indicate that the keyset has been protected by a CMLA algorithm, and to identify the particular CMLA algorithm
- before using the default protection method, use the CMLA specific protection method specified in section 15.3

At the receiving end, after `AES_UNWRAP` decryption with SK, if the device detects a `TAA_descriptor` / `ROT_algorithm_descriptor`, it checks whether the keyset has been further protected with a CMLA algorithm, and if it is, the device decrypts it with the indicated algorithm.

## 15.2 ROT ALGORITHM DESCRIPTOR / TRUST AUTHORITY ALGORITHM DESCRIPTOR

### 15.2.1 Generic Syntax

The syntax of the keyset relies on so-called Tag-Length Format (TLF), which is defined to identify the `keyset_items` in the keyset. A `keyset_item` is identified by following syntax:

<tag> [optional <clarifier>] <length> <keyset\_item>

The syntax of the `ROT_algorithm_descriptor` or `TAA_descriptor` TLF fields is specified in following Table.

Field	Length	Value	Type
<code>ROT_algorithm_descriptor()</code> / <code>TAA_descriptor {</code>			
Tag	4	1110 <sub>b</sub>	Bslbf
Clarifier	10		Uimsbf
Length	3	111 <sub>b</sub>	Bslbf
ROT_ID / TAA_TA_ID	10		Uimsbf

ROT_parameter	8*clarifier - 10 - n		Bslbf
Padding	n	0 <sub>b</sub> (n times)	Bslbf
}			

**tag** – This is the tag for ROT\_algorithm\_descriptor, previously “reserved for future use”.

**clarifier** – This is the clarifier for ROT\_algorithm\_descriptor. It shall represent the length of the keyset\_item in bytes (excluding the tag, clarifier and length fields).

**length** – This has the nil (111) value, as the actual length is specified by the clarifier

**ROT\_ID / TAA\_TA\_ID** – This field contains, in binary format, the Identifier of the Root of Trust that defined the algorithms for extra encryption of the keyset block in which this descriptor is located.

**ROT\_parameter** – This field contains the Root of Trust specific parameter(s). Note that in IEC 62455 TAA\_descriptor, this field has a predefined structure consisting of TAA\_report\_value (56 bits) and TAA\_algorithm (8 bits), followed by TAA\_parameter.

**padding** – This field is an n-bit zero-valued padding field.

### 15.2.2 CMLA-specific Values

Clarifier: 26 (this results in 8\*26-10-6 = 192 bits as the size of ROT\_parameter)

ROT\_ID: the ROT\_ID value assigned by DVB to CMLA, 0x0002.

ROT\_parameter:

Field	Length	Value	Type
TAA_report_value	56	0	Bslbf
CMLA_algorithm_id	8	5, 6 or 7	Uimsbf
CMLA_key_material	128		Bslbf

TAA\_report\_value: This field is not used by CMLA, but it needs to be there for [IEC 62455]. Therefore, it is filled with zeros, regardless of whether the service is based on [18Crypt], [IEC 62455] or [OMABCAST-SCPv1] ([OMADRM-XBS] is required to implement) specifications.

CMLA\_algorithm\_id: This is a 8-bit field that identifies the CMLA algorithm used to protect that particular keyset, the values of which correspond to the number following “cmlaip-“ in the algorithm identifier that is specified for <xenc:EncryptionMethod> fields in the Interactive Mode. (Note that CMLA\_RSA is not used.)

CMLA_algorithm_id	Key Derivation Function	Key Wrapping / Unwrapping
5	CMLA_KDF	CMLA_WRAP and CMLA_UNWRAP

6	CMLA_KDF	AES-Wrap
7	OMA_KDF	CMLA_WRAP and CMLA_UNWRAP

CMLA\_key\_material: randomly generated number

Padding: 000000 (6 zero-valued bits)

### 15.3 KEYSSET Protection Details

The CMLA ROT\_algorithm (Trust Authority Algorithm) that is used for an additional encryption layer (before the standardized encryption steps, and after the standardized decryption steps) depends on the value of CMLA\_algorithm\_id as follows:

#### Encryption:

If CMLA\_algorithm\_id is 5, KEK = CMLA\_KDF(CMLA\_key\_material).

If CMLA\_algorithm\_id is 6, KEK = CMLA\_KDF(CMLA\_key\_material).

If CMLA\_algorithm\_id is 7, KEK = OMA\_KDF(CMLA\_key\_material).

The concatenated keyset SHALL be padded with one bit with the value '1' and, after this 1-valued bit, 0 to 63 bits with the value '0', such that the length of the padded keyset is a multiple of 64 bits, see Appendix A of [NIST 800-38A]. Note that if the non-padded keyset was already a multiple of 64 bits in length, it is padded with 64 bits.

If CMLA\_algorithm\_id is 5, use CMLA\_WRAP to encrypt the keyset with KEK, starting at the last bit of ROT\_algorithm\_descriptor (or TAA\_descriptor).

If CMLA\_algorithm\_id is 6, use AES\_WRAP to encrypt the keyset with KEK starting at the last bit of ROT\_algorithm\_descriptor (or TAA\_descriptor).

If CMLA\_algorithm\_id is 7, use CMLA\_WRAP to encrypt the keyset with KEK starting at the last bit of ROT\_algorithm\_descriptor (or TAA\_descriptor).

#### Decryption:

If CMLA\_algorithm\_id is 5, KEK = CMLA\_KDF(CMLA\_key\_material).

If CMLA\_algorithm\_id is 6, KEK = CMLA\_KDF(CMLA\_key\_material).

If CMLA\_algorithm\_id is 7, KEK = OMA\_KDF(CMLA\_key\_material).

If CMLA\_algorithm\_id is 5, use CMLA\_UNWRAP to decrypt the keyset with KEK, starting at the last bit of ROT\_algorithm\_descriptor (or TAA\_descriptor).

If CMLA\_algorithm\_id is 6, use AES\_UNWRAP to decrypt the keyset with KEK starting at the last bit of ROT\_algorithm\_descriptor (or TAA\_descriptor).

If CMLA\_algorithm\_id is 7, use CMLA\_UNWRAP to decrypt the keyset with KEK starting at the last bit of ROT\_algorithm\_descriptor (or TAA\_descriptor).

Remove padding.

## 16. Content Protection Technology System Renewability Message (CPT-SRM) Support

This chapter describes CMLA provisions for the carriage of CPT-SRMs in the OMA DRM Content Format. Currently CMLA supports HDCP and DTCP content protection technologies. CPT-SRM support is Optional and is provided to support various business requirements.

The Mutable DRM information Box section 5.2.4 of [OMA-DCF] provides for the modification, extension, truncation, deletion or addition of the MutableDRMInformation box. So for delivery of DTCP and HDCP system renewability messages into either a DCF or a PDCF the following new CPT-SRM boxes are defined:

- DTCP CPT-SRM Delivery Box, carries the DTCP SRM as defined in Chapter 7 of [DTCPV1]

```
Aligned(8) class DTCPsrm extends FullBox ('dsrm', 0, 0) { byte Data[ ]; // DTCP SRM
}
```

- HDCP CPT-SRM Delivery Box carries the HDCP SRM as defined in section 5 of [HDCP-V1.3]

```
Aligned(8) class HDCPsrm extends FullBox ('hsrm', 0, 0) { byte Data[ ]; // HDCP SRM
}
```

Where:

- The SRM data is to be treated as binary data.
- Devices MUST NOT add or delete CPT-SRM delivery boxes to the MutableDRMInformation box.

## Appendix A. CMLA IP Source Code

### A1. CMLA DDT Exp

#### CMLA\_DDT\_EXP.H

```

/*****
*****
Copyright (c) 2009 CMLA, LLC. All rights reserved.

This material is protected by copyright controlled by CMLA LLC. All rights are reserved.
Copying, reproducing, storing, adapting or translating of any or all of this material requires the prior
written consent of CMLA, LLC.

*****
*****/
#ifndef CMLA_DDT_EXP_H
#define CMLA_DDT_EXP_H

#ifdef __cplusplus
extern "C" {
#endif/**
 * Calculates the Data-Dependent Transformation given by exponentiation.
 *
 * If the %parameter poutput is NULL then only the size of the output
 * is returned (in loutput), otherwise the result of the DDT is written
 * at the location pointed to by poutput.
 *
 * @param pinput pointer to data to transform
 * @param linput length of data
 * @param poutput pointer to hold the transformed data
 * @param loutput pointer to hold the length of the transformed data
 * @note The parameters pinput and poutput must be disjoint
 */

void CMLA_DDT_exp(const unsigned char *pinput, long linput, unsigned char *poutput, long *loutput);

/**
 * Calculates the inverse of Data-Dependent Transformation given by exponentiation.
 *
 * If the %parameter poutput is NULL then only the size of the output
 * is returned (in loutput), otherwise the result of the DDT is written
 * at the location pointed to by poutput.
 *
 * @param pinput pointer to data to transform
 * @param linput length of data
 * @param poutput pointer to hold the transformed data
 * @param loutput pointer to hold the length of the transformed data
 * @note The parameters pinput and poutput must be disjoint
 */

void CMLA_DDT_exp_inv(const unsigned char *pinput, long linput, unsigned char *poutput, long *loutput);
#ifdef __cplusplus
}
#endif
#endif

```

#### CMLA\_DDT\_EXP.C

```

/*****
*****
Copyright (c) 2009 CMLA, LLC. All rights reserved.

This material is protected by copyright controlled by CMLA LLC. All rights are reserved.
Copying, reproducing, storing, adapting or translating of any or all of this material requires the prior
written consent of CMLA, LLC.

*****
*****/

#include "cmla_ddt_exp.h"
// function for reducing an a, 0 < a <= 2^32 modulo 2^16+1
static unsigned int modp(unsigned int a)
{
    unsigned int x, y;

```

```

// since a > 0, this must be a = 2^32
if ( a == 0 )
    return 1;

// write a = x + 2^16y = y(2^16+1) + x - y = x - y (modulo 2^16+1)
x = a & 65535;
y = a >> 16;

return ( x > y ) ? (x-y) : ( 65537 - (y-x) );
}

// function for computing a^e mod 2^16+1
static unsigned int expmodp(unsigned int a, unsigned int e)
{
    unsigned int count = 1, res = 1;

    while ( count < e ) count <<= 1;

    count >>= 1;
    // do a square and multiply
    while ( count > 1 ) {
        if ( count & e ){
            res *= a;
            res = modp(res);
        }
        res *= res;
        res = modp(res);
        count >>= 1;
    }
    res *= a;
    return modp(res);
}

void CMLA_DDT_exp(const unsigned char *pinput, long linput, unsigned char *poutput, long *loutput)
{
    unsigned int a, e, res;

    *loutput = 3;

    // a = B2||B3 + 1 = A+1
    a = ( pinput[1] << 8 ) + pinput[2]+1;
    // e = 2*B1 + 1
    e = ( pinput[0] << 1 ) + 1;
    // res = (a^e mod 2^16+1) - 1
    res = expmodp( a, e )-1;

    // copy result to output
    poutput[2] = res % (1 << 8);
    poutput[1] = (res >> 8) % (1 << 8);
    poutput[0] = pinput[0];
}

// function to invert an odd integer < 2^32 modulo 2^16, using the Extended Euclidean Algorithm
static unsigned int inv( unsigned int a)
{
    unsigned int w0 = 0, w1 = 1, r0 = ( 1 << 16 ), r1 = a, r2, w2, q;
    int sgn_w2 = 1;

    while ( r1 > 0 ) {
        r2 = r0%r1;
        q = r0/r1;
        w2 = q*w1+w0;
        w2 = w2 & 65535;
        sgn_w2 = -sgn_w2;

        r0 = r1;
        r1 = r2;
        w0 = w1;
        w1 = w2;
    }

    if (sgn_w2 == 1)
        return ( 1 << 16 ) - w0;
    else
        return w0;
}

void CMLA_DDT_exp_inv(const unsigned char *pinput, long linput,
                    unsigned char *poutput, long *loutput){
    unsigned int a, e, res;

```

```

// a = B2||B3 + 1 = A+1
a = ( pinput[1] << 8 ) + pinput[2]+1;
// e = 2*B1 + 1
e = ( pinput[0] << 1 ) + 1;
// e = (2*B1 + 1)^-1 modulo 2^16+1
e = inv(e);
// res = (a^e mod 2^16+1) - 1
res = expmodp( a, e )-1;

// copy result to output
poutput[2] = res % (1 << 8);
poutput[1] = (res >> 8) % (1 << 8);
poutput[0] = pinput[0];
}

```

## A2. CMLA DDT Perm

### CMLA\_DDT\_PERM.H

```

/*****
*****
Copyright (c) 2009 CMLA, LLC. All rights reserved.

This material is protected by copyright controlled by CMLA LLC. All rights are reserved.
Copying, reproducing, storing, adapting or translating of any or all of this material requires the prior
written consent of CMLA, LLC.

*****/
#define CMLA_DDT_PERM_H
#define CMLA_DDT_PERM_H

#ifdef __cplusplus
extern "C" {
#endif /**
 * Calculates the Data-Dependent Transformation given by permutation
 *
 * If the %parameter poutput is NULL then only the size of the output
 * is returned (in loutput), otherwise the result of the DDT is written
 * at the location pointed to by poutput.
 *
 * @param pinput pointer to data to transform
 * @param linput length of data
 * @param poutput pointer to hold the transformed data
 * @param loutput pointer to hold the length of the transformed data
 * @note The parameters pinput and poutput must be disjoint
 */
void CMLA_DDT_perm(const unsigned char *pinput, long linput, unsigned char *poutput, long *loutput);

/**
 * Calculates the inverse of the Data-Dependent Transformation given by permutation
 *
 * If the %parameter poutput is NULL then only the size of the output
 * is returned (in loutput), otherwise the result of the DDT is written
 * at the location pointed to by poutput.
 *
 * @param pinput pointer to data to transform
 * @param linput length of data
 * @param poutput pointer to hold the transformed data
 * @param loutput pointer to hold the length of the transformed data
 * @note The parameters pinput and poutput must be disjoint
 */
void CMLA_DDT_perm_inv(const unsigned char *pinput, long linput, unsigned char *poutput, long *loutput);

#ifdef __cplusplus
}
#endif
#endif

```

### CMLA\_DDT\_PERM.C

```

/*****
*****
Copyright (c) 2009 CMLA, LLC. All rights reserved.

This material is protected by copyright controlled by CMLA LLC. All rights are reserved.
Copying, reproducing, storing, adapting or translating of any or all of this material requires the prior
written consent of CMLA, LLC.

*****/

#include <string.h>
#include "cmla_ddt_perm.h"
/* function for doing the actual permutation */
static void dopermute( const unsigned char in, unsigned char *out, int i )
{
    *out = 0x00;
    switch(i) {
    case 0:
        *out |= ( in & 0x10 ) ? 0x01 : 0x00;
        *out |= ( in & 0x01 ) ? 0x02 : 0x00;
        *out |= ( in & 0x08 ) ? 0x04 : 0x00;
        *out |= ( in & 0x20 ) ? 0x08 : 0x00;
        *out |= ( in & 0x04 ) ? 0x10 : 0x00;
        *out |= ( in & 0x40 ) ? 0x20 : 0x00;
        *out |= ( in & 0x02 ) ? 0x40 : 0x00;
        break;
    case 1:
        *out |= ( in & 0x04 ) ? 0x01 : 0x00;
        *out |= ( in & 0x10 ) ? 0x02 : 0x00;
        *out |= ( in & 0x20 ) ? 0x04 : 0x00;
        *out |= ( in & 0x40 ) ? 0x08 : 0x00;
        *out |= ( in & 0x08 ) ? 0x10 : 0x00;
        *out |= ( in & 0x02 ) ? 0x20 : 0x00;
        *out |= ( in & 0x01 ) ? 0x40 : 0x00;
        break;
    case 2:
    case 7:
        *out |= ( in & 0x08 ) ? 0x01 : 0x00;
        *out |= ( in & 0x04 ) ? 0x02 : 0x00;
        *out |= ( in & 0x40 ) ? 0x04 : 0x00;
        *out |= ( in & 0x02 ) ? 0x08 : 0x00;
        *out |= ( in & 0x20 ) ? 0x10 : 0x00;
        *out |= ( in & 0x01 ) ? 0x20 : 0x00;
        *out |= ( in & 0x10 ) ? 0x40 : 0x00;
        break;
    case 3:
    case 6:
        *out |= ( in & 0x20 ) ? 0x01 : 0x00;
        *out |= ( in & 0x08 ) ? 0x02 : 0x00;
        *out |= ( in & 0x02 ) ? 0x04 : 0x00;
        *out |= ( in & 0x01 ) ? 0x08 : 0x00;
        *out |= ( in & 0x40 ) ? 0x10 : 0x00;
        *out |= ( in & 0x10 ) ? 0x20 : 0x00;
        *out |= ( in & 0x04 ) ? 0x40 : 0x00;
        break;
    case 4:
        *out |= ( in & 0x40 ) ? 0x01 : 0x00;
        *out |= ( in & 0x20 ) ? 0x02 : 0x00;
        *out |= ( in & 0x01 ) ? 0x04 : 0x00;
        *out |= ( in & 0x10 ) ? 0x08 : 0x00;
        *out |= ( in & 0x02 ) ? 0x10 : 0x00;
        *out |= ( in & 0x04 ) ? 0x20 : 0x00;
        *out |= ( in & 0x08 ) ? 0x40 : 0x00;
        break;
    case 5:
        *out |= ( in & 0x02 ) ? 0x01 : 0x00;
        *out |= ( in & 0x40 ) ? 0x02 : 0x00;
        *out |= ( in & 0x10 ) ? 0x04 : 0x00;
        *out |= ( in & 0x04 ) ? 0x08 : 0x00;
        *out |= ( in & 0x01 ) ? 0x10 : 0x00;
        *out |= ( in & 0x08 ) ? 0x20 : 0x00;
        *out |= ( in & 0x20 ) ? 0x40 : 0x00;
        break;
    }
}

/* function for permuting the i'th block */
static void permute( const unsigned char *pinput, unsigned char *poutput, int i ) {
    unsigned char temp, res;

    // temp = i'th group of 7 bits with 0 zero in front

```

```

temp = ( (( unsigned char ) ( pinput[i] << (8-i) )) >> 1 ) | ( pinput[i + 1] >> ( i + 1 ) );

// if b_i = 1 do the permutation, otherwise just copy
if ( pinput[0] & ( 1 << (7-i) ))
    dopermute( temp, &res, i );
else
    res = temp;

// copy the result to the right place in the output
poutput[i] |= res >> (7-i);
if ( i < 7 )
    poutput[i+1] = (unsigned char) ( res << ( i + 1 ) );
}

/* function for doing the inverse permutation of the i'th block */
static void permute_inv( const unsigned char *pinput, unsigned char *poutput, int i ) {
    unsigned char temp, res;

    // temp = i'th group of 7 bits with 0 zero in front
    temp = ( (( unsigned char ) ( pinput[i] << (8-i) )) >> 1 ) | ( pinput[i + 1] >> ( i + 1 ) );

    // if b_i = 1 do the inverse permutation, otherwise just copy
    if ( pinput[0] & ( 1 << (7-i) ))
        dopermute( temp, &res, ( i < 6 ) ? ( 5 - i ) : ( i - 4 ) );
    else
        res = temp;

    // copy the result to the right place in the output
    poutput[i] |= res >> (7-i);
    if ( i < 7 )
        poutput[i+1] = (unsigned char) ( res << ( i + 1 ) );
}

void CMLA_DDT_perm(const unsigned char *pinput, long linput,
                  unsigned char *poutput, long *loutput) {

    int i;

    *loutput = 8;
    memset(poutput, 0, 8);

    // run through the 8 blocks, and do the translation
    for ( i = 0; i < 8; i++ ) {
        permute( pinput, poutput, i );
    }

    // copy the b_i's
    poutput[0] = pinput[0];
}

void CMLA_DDT_perm_inv(const unsigned char *pinput, long linput,
                      unsigned char *poutput, long *loutput) {

    int i;

    *loutput = 8;
    memset(poutput, 0, 8);

    // run through the 8 blocks, and do the inverse translation
    for ( i = 0; i < 8; i++ ) {
        permute_inv( pinput, poutput, i );
    }

    // copy the b_i's
    poutput[0] = pinput[0];
}

```

## Appendix B. CMLA IP Test Vectors

### B1. CMLA IP Test Vector

```

# -----
# Components of the RSA Key Pair
# -----

# RSA modulus n:
a2 ba 40 ee 07 e3 b2 bd 2f 02 ce 22 7f 36 a1 95
02 44 86 e4 9c 19 cb 41 bb bd fb ba 98 b2 2b 0e
57 7c 2e ea ff a2 0d 88 3a 76 e6 5e 39 4c 69 d4
b3 c0 5a 1e 8f ad da 27 ed b2 a4 2b c0 00 fe 88
8b 9b 32 c2 2d 15 ad d0 cd 76 b3 e7 93 6e 19 95
5b 22 0d d1 7d 4e a9 04 b1 ec 10 2b 2e 4d e7 75
12 22 aa 99 15 10 24 c7 cb 41 cc 5e a2 1d 00 ee
b4 1f 7c 80 08 34 d2 c6 e0 6b ce 3b ce 7e a9 a5

# RSA public exponent e:
01 00 01

# Prime p:
d1 7f 65 5b f2 7c 8b 16 d3 54 62 c9 05 cc 04 a2
6f 37 e2 a6 7f a9 c0 ce 0d ce d4 72 39 4a 0d f7
43 fe 7f 92 9e 37 8e fd b3 68 ed df f4 53 cf 00
7a f6 d9 48 e0 ad e7 57 37 1f 8a 71 1e 27 8f 6b

# Prime q:
c6 d9 2b 6f ee 74 14 d1 35 8c e1 54 6f b6 29 87
53 0b 90 bd 15 e0 f1 49 63 a5 e2 63 5a db 69 34
7e c0 c0 1b 2a b1 76 3f d8 ac 1a 59 2f b2 27 57
46 3a 98 24 25 bb 97 a3 a4 37 c5 bf 86 d0 3f 2f

# p's CRT exponent dP:
9d 0d bf 83 e5 ce 9e 4b 17 54 dc d5 cd 05 bc b7
b5 5f 15 08 33 0e a4 9f 14 d4 e8 89 55 0f 82 56
cb 5f 80 6d ff 34 b1 7a da 44 20 88 53 57 7d 08
e4 26 28 90 ac f7 52 46 1c ea 05 54 76 01 bc 4f

# q's CRT exponent dQ:
12 91 a5 24 c6 b7 c0 59 e9 0e 46 dc 83 b2 17 1e
b3 fa 98 81 8f d1 79 b6 c8 bf 6c ec aa 47 63 03
ab f2 83 fe 05 76 9c fc 49 57 88 fe 5b 1d df de
9e 88 4a 3c d5 e9 36 b7 e9 55 eb f9 7e b5 63 b1

# CRT coefficient qInv:
a6 3f 1d a3 8b 95 0c 9a d1 c6 7c e0 d6 77 ec 29
14 cd 7d 40 06 2d f4 2a 67 eb 19 8a 17 6f 97 42
aa c7 c5 fe a1 4f 22 97 66 2b 84 81 2c 4d ef c4
9a 80 25 ab 43 82 28 6b e4 c0 37 88 dd 01 d6 9f

# -----
# Setting
# -----

# seed Z:
7c 09 a0 0a c8 05 88 6e 80 42 7a f1 0f e9 ef 21
d9 74 4c c4 85 51 90 3c ec a8 d9 24 45 ff 7d 6c
31 04 67 dc 31 aa a5 68 fe 45 e9 cb 47 e7 36 fb
53 5f ef 87 81 e2 d2 fd 09 13 d8 e1 03 d5 ac 82
18 6b 16 2b 9c 25 ff 38 50 d6 90 8a 9c 78 c9 88
d3 59 ad 5d 7b f7 66 8f de 64 20 a8 35 5f 45 6d
b1 c6 b2 9c 1a 18 94 d3 83 2f 21 86 0a 92 eb 9d
3e 69 6c 2e 59 24 f9 23 d1 32 d2 f1 73 3b ff b4

# MacKey :
1d 55 bd 0c 59 d7 90 0c 9c 16 3a 15 0d d3 09 f7

# REK :
95 27 57 09 93 c7 d9 77 02 8d ae b2 2c 34 5c 66

# -----
# CMLA_RSA_ENCRYPT
# -----

# CMLA_DDT_Exp (Z) = m:
7c 69 78 0a c8 05 88 6e 80 42 7a f1 0f e9 ef 21
d9 74 4c c4 85 51 90 3c ec a8 d9 24 45 ff 7d 6c
31 04 67 dc 31 aa a5 68 fe 45 e9 cb 47 e7 36 fb
53 5f ef 87 81 e2 d2 fd 09 13 d8 e1 03 d5 ac 82
18 ee 42 2b 9c 25 ff 38 50 d6 90 8a 9c 78 c9 88
d3 59 ad 5d 7b f7 66 8f de 64 20 a8 35 5f 45 6d
b1 c6 b2 9c 1a 18 94 d3 83 2f 21 86 0a 92 eb 9d
3e 69 6c 2e 59 24 f9 23 d1 32 d2 f1 73 3b ff b4

# RSA_Encrypt (m) = C1:
83 d7 e0 e0 fd 69 c7 d9 df c7 83 84 88 ce f1 ab
b7 96 ca 84 65 14 01 72 16 37 41 90 c5 2d 88 17
dd 85 39 38 ef 15 28 5f 7a 0d dc e5 ed c7 4d 10
66 61 e7 e9 f1 d7 c4 7c 21 02 c4 29 45 bb 8f 07
da 57 65 21 da 83 a3 e6 a3 f3 92 ca 9d eb e3 e1
cd 1d f3 e1 05 4c 64 ff 4c 93 18 dc 56 1c b5 4e
e8 d2 5a 5f 41 33 d8 07 81 0a 82 4c ca 5f 7d aa
ca 9b cb 9e 83 3f aa 79 c8 ee c6 9d b1 19 22 c1

# -----
# CMLA_KDF
# -----

# CMLA_KDF (Z) = KEK :
74 98 52 e9 dd 93 9a 34 5c 80 57 81 f7 2c 7e de

# -----
# CMLA_WRAP
# -----

#CMLA_DDT_Perm (MacKey) = k0 :
1d 55 bd 0d 8e 2b d0 03 9c 16 3a 15 0d d3 09 f7

#CMLA_DDT_Perm (REK) = k1 :
95 87 57 0a a3 cf 59 7e 02 8d ae b2 2c 34 5c 66

# AES_WRAP (KEK, k0|k1) = C2 :
96 5d e9 95 93 61 c2 2f d9 93 d8 48 22 f4 27 1a
41 3c 87 99 84 7f 75 20 49 e2 8d f4 a0 71 ce da
a8 64 0d 9e 12 3f 7f a4

# -----
# CMLAIP-1
# -----

# C1|C2 = cipherValue :
83 d7 e0 e0 fd 69 c7 d9 df c7 83 84 88 ce f1 ab
b7 96 ca 84 65 14 01 72 16 37 41 90 c5 2d 88 17
dd 85 39 38 ef 15 28 5f 7a 0d dc e5 ed c7 4d 10
66 61 e7 e9 f1 d7 c4 7c 21 02 c4 29 45 bb 8f 07
da 57 65 21 da 83 a3 e6 a3 f3 92 ca 9d eb e3 e1
cd 1d f3 e1 05 4c 64 ff 4c 93 18 dc 56 1c b5 4e
e8 d2 5a 5f 41 33 d8 07 81 0a 82 4c ca 5f 7d aa
ca 9b cb 9e 83 3f aa 79 c8 ee c6 9d b1 19 22 c1
96 5d e9 95 93 61 c2 2f d9 93 d8 48 22 f4 27 1a
41 3c 87 99 84 7f 75 20 49 e2 8d f4 a0 71 ce da
a8 64 0d 9e 12 3f 7f a4

```

## B2. CMLA IP Test Vector

```
# -----
# Components of the RSA Key Pair
# -----

# RSA modulus n:
bb 6f 15 cb dc 0f 48 88 86 d6 3d 5b d0 35 81 ca
ca 66 e0 86 07 a1 cd ef ae d1 6a 3e 67 e9 7f 12
28 86 43 40 e8 96 d1 d3 f1 01 e7 4e f2 33 e6 60
26 13 d6 cc 22 d2 f0 70 3d c2 6e 34 24 36 85 5b
af b0 94 c1 07 d7 2d 18 5d 77 e1 33 2c 85 84 81
e6 64 0c 6d 9f 39 9b 44 67 d7 ac 77 b1 71 51 aa
2f a4 7d a0 72 7c cb 9a 35 08 70 98 6e 4a f6 c4
f8 61 7f 15 25 ce 4e da 04 2a 44 1e 54 73 72 33

# RSA public exponent e:
01 00 01

# Prime p:
dc 60 76 8c fa c1 d1 55 96 76 7f a0 e4 13 ac fe
db cb 25 9b f8 75 ec 27 44 ea 9f 50 b1 8d 0c e7
19 9d 03 0d e2 ba 3c 76 8e ec f2 49 c0 4e 11 2f
d2 aa 9f 9d ef d4 7c c9 2b 78 05 28 5c 11 0d bb

# Prime q:
d9 bb 64 e1 73 81 34 07 51 9c 68 5a 00 90 00 57
6a 19 46 9f 09 b6 c3 aa 60 54 5e 55 fe 44 0e 7c
a4 ef 64 12 c5 e5 06 92 15 ec e7 81 70 22 e4 f7
2f 39 f3 9a 75 01 a7 08 0a 7b 60 2c 40 01 29 e9

# p's CRT exponent dP:
b6 d2 ab e6 ff 2c 85 9e ac 69 78 2e 20 a8 96 0f
04 86 7e 97 eb 42 e0 fc 1e f7 49 dd e6 be 2e 63
16 63 a4 a2 03 63 1f 3f f8 08 68 64 78 ce 0b 02
12 92 0c 43 39 30 9d a9 42 8f 9a 2f c6 59 3b 7b

# q's CRT exponent dQ:
83 de 26 ff 14 81 90 0d 4e b4 37 cf 2a c0 0c 34
e5 21 61 d0 38 85 e4 83 6f cc 29 46 53 b4 cc 41
59 73 53 5c bb 56 36 60 8b be eb 87 4d 6f 14 d5
50 58 fc dc f3 38 88 fd 29 bc 07 47 45 8c 6e e1

# CRT coefficient qInv:
32 81 1c be 78 ff 22 ff 25 b7 0d 53 bc 4c 36 e3
4b 99 5d dc fd ba 1a b9 6a ae 90 cc a0 8a ca b0
31 07 6f ca 45 95 82 3c a2 5d ba 86 7c 00 28 98
54 d8 ab 2d f5 d5 ac a9 b7 a6 a7 72 d1 ed 3b 47

# -----
# Setting
# -----

# seed Z:
18 5e 03 e2 ee 6b f4 07 9f cb 1e 62 a5 f3 b2 f6
51 15 eb c1 1f 93 81 4d f3 eb e9 34 bd 2e 09 d0
51 5b 6c ca 4c 6a da 45 bc c3 f3 b9 58 05 70 a1
46 f0 e2 7b 66 e2 83 43 8f c2 7b 30 09 66 48 08
1c b7 26 0d 0c 28 1e be 81 69 b8 11 ca d3 90 a3
81 2e 24 15 12 63 6a be f3 03 67 0b 2f 05 d6 df
ce f0 c6 90 46 dd b7 b6 a2 b5 5b 1b eb 25 94 9e
7f ee 87 15 58 f0 b4 31 fd ac 6c 6c a6 01 dc 10

# MacKey :
4c 52 93 00 58 37 21 a5 ba 72 35 41 5f 9b a9 8d

# REK :
83 81 10 bd 01 63 2f b7 2e 5a a0 43 55 43 2d c9
```

```
# -----
# CMLA_RSA_ENCRYPT
# -----

# CMLA_DDT_Exp (Z) = m:
18 9c 2b e2 ee 6b f4 07 9f cb 1e 62 a5 f3 b2 f6
51 15 eb c1 1f 93 81 4d f3 eb e9 34 bd 2e 09 d0
51 5b 6c ca 4c 6a da 45 bc c3 f3 b9 58 05 70 a1
46 f0 e2 7b 66 e2 83 43 8f c2 7b 30 09 66 48 08
1c 7a 0a 0d 0c 28 1e be 81 69 b8 11 ca d3 90 a3
81 2e 24 15 12 63 6a be f3 03 67 0b 2f 05 d6 df
ce f0 c6 90 46 dd b7 b6 a2 b5 5b 1b eb 25 94 9e
7f ee 87 15 58 f0 b4 31 fd ac 6c 6c a6 01 dc 10

# RSA_Encrypt (m) = C1:
5d 98 4f e5 29 15 cd e5 39 e5 9f 61 96 da 04 e4
07 08 dd ab 3d b8 3c c3 a5 98 73 dd da 4a 92 08
5e a8 9f b3 ad 35 0c e9 3f 25 3d 77 fb 52 d7 b0
2c d0 70 c5 4f c3 89 80 3c 4f 8f 7e 74 df f6 d9
4f 1c b6 69 5f fb ad da d0 30 42 4e 3f 12 d3 42
43 3f 19 7b aa 57 03 2b 20 e0 dc 00 7e 89 43 ba
e3 a0 01 99 12 e1 ce 90 22 00 c6 19 b0 2f a9 79
86 2d fa e3 55 ed d5 4a 13 62 ee 5a 30 8b 48 1c

# -----
# CMLA_KDF
# -----

# CMLA_KDF (Z) = KEK :
23 f2 70 fd 8a 25 80 5a a4 cc 15 77 1e 29 f8 fc

# -----
# CMLA_WRAP
# -----

#CMLA_DDT_Perm (MacKey) = k0 :
4c 52 17 00 50 ab a1 a5 ba 72 35 41 5f 9b a9 8d

#CMLA_DDT_Perm (REK) = k1 :
83 41 10 bd 01 63 3f 7a 2e 5a a0 43 55 43 2d c9

# AES_WRAP (KEK, k0|k1) = C2 :
f0 a3 f8 df 0c bd c9 ce 58 8b 21 05 a6 2b 86 88
bc 7c 6d c0 3b 7c 88 d7 c9 1a d9 65 f6 5a 3a 4c
75 2f ca 37 8a b8 82 7d

# -----
# CMLAIP-1
# -----

# C1|C2 = cipherValue :
5d 98 4f e5 29 15 cd e5 39 e5 9f 61 96 da 04 e4
07 08 dd ab 3d b8 3c c3 a5 98 73 dd da 4a 92 08
5e a8 9f b3 ad 35 0c e9 3f 25 3d 77 fb 52 d7 b0
2c d0 70 c5 4f c3 89 80 3c 4f 8f 7e 74 df f6 d9
4f 1c b6 69 5f fb ad da d0 30 42 4e 3f 12 d3 42
43 3f 19 7b aa 57 03 2b 20 e0 dc 00 7e 89 43 ba
e3 a0 01 99 12 e1 ce 90 22 00 c6 19 b0 2f a9 79
86 2d fa e3 55 ed d5 4a 13 62 ee 5a 30 8b 48 1c
f0 a3 f8 df 0c bd c9 ce 58 8b 21 05 a6 2b 86 88
bc 7c 6d c0 3b 7c 88 d7 c9 1a d9 65 f6 5a 3a 4c
75 2f ca 37 8a b8 82 7d
```