# CIPHERVALUE DECRYPTION

## RETRIEVING OF THE RAW RSA DATA (256 OCTETS)

```
echo -ne 'UQ5FHBSnCRM15dvq2vXGTc1DnAIUFv869qh0jRJv3rc5lBZaESmw
CxA54Y8aRS732CoP/MOEbB/xynTXbaV1daeODKPBpdpX4stjS4D5oO1D8P7/e4iTit19YyEJ7sph
mJMVNbJCmKnuEpWkGzZYkX901MIDixYsOGC5VHZMwRQg3BcxZuj8XoF/VGGhb9uOkW3visbi/KpF
5LRcfjbs5FbDQipbVnu6TcbR8i0M+vHS+Kf84Wv0M0ocTS+SUTOhUzxi/3cAqxt3Gkqnav2HhtUI
gQhKJkmZyx11KUkXFCcZe5VI5WwYoqoiBR2ueN5AkomgQKNB0TWRQyuL8xwnVw==' \
    | openssl base64 -d \
    | xxd     # only for the output display
```

Résultat :

```
 offset            données au format hexadécimal              format string
--------------------------------------------------------------------------------
00000000  51 0e 45 1c 14 a7 09 13  35 e5 db ea da f5 c6 4d  |Q.E..@..5@@@@@@M|
00000010  cd 43 9c 02 14 16 ff 3a  f6 a8 74 8d 12 6f de b7  |@C....@:@t..o□|
00000020  39 94 16 5a 11 29 b0 0b  10 39 e1 8f 1a 45 2e f7  |9..Z.)@..9@..E.@|
00000030  d8 2a 0f fc c3 84 6c 1f  f1 ca 74 d7 6d a5 75 75  |@*.@@.l.@@t@m@uu|
00000040  a7 8e 0c a3 c1 a5 da 57  e2 cb 63 4b 80 f9 a0 ed  |@..@@@@W@@cK.@@|
00000050  43 f0 fe ff 7b 88 93 8a  dd 7d 63 21 09 ee ca 61  |C@@@{...@}c!.@@a|
00000060  98 93 15 35 b2 42 98 a9  ee 12 95 a4 1b 36 58 91  |...5@B.@@..@.6X.|
00000070  7f 74 d4 c2 03 8b 16 2c  38 60 b9 54 76 4c c1 14  |.t@@...,8`@TvL@.|
00000080  20 dc 17 31 66 e8 fc 5e  81 7f 54 61 a1 6f db 8e  | @.1f@@^..Ta@o@.|
00000090  91 6d ef 8a c6 e2 fc aa  45 e4 b4 5c 7e 36 ec e4  |.m@.@@@E@\~6@@|
000000a0  56 c3 42 2a 5b 56 7b ba  4d c6 d1 f2 2d 0c fa f1  |V@B*[V{@M@@@-.@@|
000000b0  d2 f8 a7 fc e1 6b f4 33  4a 1c 4d 2f 92 51 33 a1  |@@@@k@3J.M/.Q3@|
000000c0  53 3c 62 ff 77 00 ab 1b  77 1a 4a a7 6a fd 87 86  |S<b@w.@.w.J@j@..|
000000d0  d5 08 81 08 4a 26 49 99  cb 1d 75 29 49 17 14 27  |@...J&I.@.u)I..'|
000000e0  19 7b 95 48 e5 6c 18 a2  aa 22 05 1d ae 78 de 40  |.{.H@l.@@"..@x@@|
000000f0  92 89 a0 40 a3 41 d1 35  91 43 2b 8b f3 1c 27 57  |..@@A@5.C+.@.'W|
```

Here is our cryptographic content **raw RSA binary** 256 octets long output (related to the RSA-2048 bits (256 octets) certificates)

# CIPHERVALUE DECRYPTION

## COMPLETE DECRYPTION (RSA+OAEP+MGF1+SHA1)

This script needs the RSA **private key** of the player (Recipient)

Using `openssl pkeyutl -decrypt` and specific paraemters to use OAEP, MGF1 and SHA1 :

```
$ echo -ne 'UQ5FHBSnCRM15dvq2vXGTc1DnAIUFv869qh0jRJv3rc5lBZaESmw
CxA54Y8aRS732CoP/MOEbB/xynTXbaV1daeODKPBpdpX4stjS4D5oO1D8P7/e4iTit19YyEJ7sph
mJMVNbJCmKnuEpWkGzZYkX901MIDixYsOGC5VHZMwRQg3BcxZuj8XoF/VGGhb9uOkW3visbi/KpF
5LRcfjbs5FbDQipbVnu6TcbR8i0M+vHS+Kf84Wv0M0ocTS+SUTOhUzxi/3cAqxt3Gkqnav2HhtUI
gQhKJkmZyx11KUkXFCcZe5VI5WwYoqoiBR2ueN5AkomgQKNB0TWRQyuL8xwnVw==' \
    | openssl base64 -d \
    | openssl pkeyutl \
        -decrypt \
        -inkey private_key.pem \
        -pkeyopt rsa_padding_mode:oaep \
        -pkeyopt rsa_oaep_md:sha1 \
        -pkeyopt rsa_mgf1_md:sha1 \
    | xxd     # only for the output display
```

Résultat :

```
 offset        data in hexadecimal format           string string
------------------------------------------------------------------
00000000: f1dc 1244 6016 9a0e 85bc 3006 42f8 66ab  ...D`.....0.B.f.
00000010: 09d5 3df0 13c0 7fa4 341f ded0 eb57 cf7a  ..=.....4....W.z
00000020: 807a 687d 1ce4 61e5 548f 4b91 a70a 60b7  .zh}..a.T.K...`.
00000030: bc07 7fb5 4d44 454b 205e f3c1 e260 4b13  ....MDEK ^...`K.
00000040: 90d9 b961 9825 37e5 3230 3136 2d30 372d  ...a.%7.2016-07-
00000050: 3239 5432 333a 3539 3a30 302b 3032 3a30  29T23:59:00+02:0
00000060: 3032 3032 322d 3037 2d30 3154 3030 3a30  02022-07-01T00:0
00000070: 303a 3030 2b30 323a 3030 b16f 5fd3 860c  0:00+02:00.o_...
00000080: 11ad 24cd eaaf bafc 7b90                 ..$.....{.
```

Data structure :

| Name of the field | Position | Size | Format | Value |
|---|---|---|---|---|
| Structure ID | 0 | 16 octets | UUID | `f1 dc 12 44 60 16 9a 0e 85 bc 30 06 42 f8 66 ab` |
| **Certificate ThumbPrint** | 16 | 20 octets | Hash | `09 d5 3d f0 13 c0 7f a4 34 1f de d0 eb 57 cf 7a 80 7a 68 7d` |
| CPL Id | 36 | 16 octets | UUID | `1c e4 61 e5 54 8f 4b 91 a7 0a 60 b7 bc 07 7f b5` |
| Key Type | 52 | 4 octets | 4 chars | `4d 44 45 4b` (**MDEK**) |
| Key Id | 56 | 16 octets | UUID | `20 5e f3 c1 e2 60 4b 13 90 d9 b9 61 98 25 37 e5` |
| Date Not Valid Before | 72 | 25 octets | Datetime | **`2016-07-29T23:59:00+02:00`** |
| Date Not Valid After | 97 | 25 octets | Datetime | **`2022-07-01T00:00:00+02:00`** |
| AES key | 122 | 16 octets | AES | `b1 6f 5f d3 86 0c 11 ad 24 cd ea af ba fc 7b 90` |
| **Total** | | **138 octets** | | |

## OPEN ALL CIPHERVALUE WITHIN A KDM :

> This script needs the RSA **private key** of the player (Recipient)

This simple script reads a KDM, analyzes each **CipherValue**, decrypts them - using the RSA private key - and provides an output with metadata and AES keys :

```
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives import hashes
import base64
from lxml import etree

PRIVATE_KEY_FILENAME = "private_key.pem"
PRIVATE_KEY_PASSWORD = None
KDM_FILENAME = "kdm.xml"

# open private key (RSA PRIVATE KEY)
with open(PRIVATE_KEY_FILENAME, "rb") as file:
    private_key_content = file.read()

# load private key
private_key = serialization.load_pem_private_key(
    data = private_key_content,
    password = PRIVATE_KEY_PASSWORD
)

# create padding with OAEP and SHA-1
pad = padding.OAEP(
    mgf = padding.MGF1(hashes.SHA1()),
    algorithm = hashes.SHA1(),
    label = None
)

# open and read kdm file
with open(KDM_FILENAME, "rb") as xml:
    tree = etree.fromstring(
        text = xml.read()
    )

# show all KeyId from KeyIdList
# AuthenticatedPublic > RequiredExtensions > KDMRequiredExtensions > KeyIdList > TypedKeyId[…] > KeyId
keys = tree.xpath("//*[local-name()='TypedKeyId']")

# read each public KeyId and KeyType
# this part is not encrypted
for key in keys:
    keyType = key.xpath("./*[local-name()='KeyType']/text()")[0]
    keyId = key.xpath("./*[local-name()='KeyId']/text()")[0]
    print("KeyId %s - KeyType %s" % (keyId, keyType))

# find all cipher values, each contains lot of data, including AES key
# AuthenticatedPrivate > enc:EncryptedKey[…] > enc:CipherData
cipher_values = tree.xpath("//*[local-name()='CipherValue']")

# read each private cipherValue
for cipher_value in cipher_values:

    # base64 decryption
    encrypted_value = base64.b64decode(cipher_value.text)

    # RSA decryption
    plaintext = private_key.decrypt(
        ciphertext = encrypted_value,
        padding = pad
    )

    # parse all plaintext datas
    # data is a static structure
    print("* Cipher Base64          : %s" % cipher_value.text)
    print("* Cipher Text            : %s" % encrypted_value.hex())
    print("* Plaintext              : %s" % plaintext.hex())
    print("* Structure ID           : %s" % plaintext[0:0+16].hex())
    print("* Certificate ThumbPrint : %s" % plaintext[16:16+20].hex())
    print("* CPL Id                 : %s" % plaintext[36:36+16].hex())
    print("* Key Type               : %s" % plaintext[52:52+4].decode('utf-8'))
    print("* Key Id                 : %s" % plaintext[56:56+16].hex())
    print("* Date Not Valid Before  : %s" % plaintext[72:72+25].decode('utf-8'))
    print("* Date Not Valid After   : %s" % plaintext[97:97+25].decode('utf-8'))
    print("* AES Key                : %s" % plaintext[122:122+16].hex())
    print("")
```

## CREATE A CIPHERVALUE (WITHOUT STRUCTURE)

This script needs the RSA **public certificate** of the player (Recipient)

Warning, this example don't generate the real **CipherValue** structure defined in the KDM SMPTE documents. You must complete the PLAINTEXT to integrate the different mandatory missing elements and defined in the SMPTE standard.

This script creates a **CipherValue** following the different algorithms required by the KDM SMPTE standard (without its structure). This way, we will have our sequence OAEP + MGF1 + SHA1 + Base64 :

```
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.backends import default_backend
from cryptography.x509 import load_pem_x509_certificate
import base64

PUBLIC_CERTIFICATE = "public_certificate.pem"
PLAINTEXT = b"\xab\xad\xca\xfe\xab\xad\xca\xfe\xab\xad\xca\xfe\xab\xad\xca\xfe"

# read public certificate
with open(PUBLIC_CERTIFICATE, "rb") as file:
    public_certificate_content = file.read()

# load public certificate and extract public key
certificate = load_pem_x509_certificate(
    data = public_certificate_content,
    backend = default_backend()
)
public_key = certificate.public_key();

# create padding OAEP and SHA1
pad = padding.OAEP(
    mgf = padding.MGF1(algorithm=hashes.SHA1()),
    algorithm = hashes.SHA1(),
    label = None
)

# RSA encryption
ciphertext = public_key.encrypt(
    plaintext = PLAINTEXT,
    padding = pad
)

# Base64 encryption
ciphertext64 = base64.b64encode(ciphertext)

print("Ciphertext - RSA =", ciphertext.hex())
print("Ciphertext - Base64 =", ciphertext64)
```

## CREATE A CIPHERVALUE (VIA OPENSSL, WITHOUT STRUCTURE)

```
printf "HELLOWORLD" \
    | openssl pkeyutl \
        -encrypt \
        -pubin \
        -inkey public_key.pem \
        -pkeyopt rsa_padding_mode:oaep \
        -pkeyopt rsa_oaep_md:sha1 \
        -pkeyopt rsa_mgf1_md:sha1 \
    | openssl base64 -e
```

## XMLSEC: THE BASIS OF THE DIGITAL SIGNATURE OF THE KDM

Here is an example of a valid simplified XML (non SMPTE/DCI) and following these standard:

- XML Signature (xmldsig)
- XML Encryption (xmlcore-sec)
- Canonical XML (C14N)

Here is the XML template before the signature :

```
<?xml version="1.0" encoding="UTF-8"?>
<Test>

    <Foo Id="Bar">
        <!-- metadatas -->
    </Foo>

    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments"/>
            <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
            <ds:Reference URI="#Bar">
                <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                <ds:DigestValue/>
            </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue/>
    </ds:Signature>

</Test>
```

We will create the **DigestValue** for **Foo** and create the **SignatureValue** in the same time using xmlsec :

```
# xmlsec with our private key :
xmlsec1 \
    --sign \
    --id-attr:Id Foo \
    --privkey-pem private-key.pem \
    xmlsec.sample.xml
```

Don't forget to have your RSA private key (or to generate it).

Here is the output from xmlsec with complete **DigestValue** and **SignatureValue** :

```
<?xml version="1.0" encoding="UTF-8"?>
<Test>

    <Foo Id="Bar">
        <!-- metadatas -->
    </Foo>

    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments"/>
            <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
            <ds:Reference URI="#Bar">
                <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                <ds:DigestValue>p35xH4WZ5LEYS6r20H/ewlUjtQnOp8Ce9k7khaP2qfM=</ds:DigestValue>
            </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>QoFMZCMwwav9hFP3Fczdq30aGb4Zn/pKirzBAF+Aoj2fhLv89pul/pu/maRcuNRX
+t2WrxF0BjHMkR4dx9j/+GaQ0FRkqmY5gg9QzxBlG7NZSHiEBSZgYtY+P+0vjuYR
/hpulS3EHAvg4OaKkN8eAhQgNVaSsvFTmC8NDAY+k7hWukWl1NrIumvCt/bXWiBV
60kiZ+sCg1wt/KugPbmfOLiyAakWl+kx67G8LD8BwllG2ySSTY4BAMp8Il9QSKB9
PFI/ADjKr0mrDbch6N4OspqCREIC0SGdMnN1ZPw4S8v+MGuIiVZYxdA8IioerE4Q
WpHqx9rw6VLo01shPpwP+w==</ds:SignatureValue>
    </ds:Signature>

</Test>
```

From there, the same can be done following SMPTE/DCI standards to generate or verify KDM signatures.

## GENERATE SIGNATURES (DIGESTVALUE + SIGNATUREVALUE) OF A KDM

> The private key used here in a DCI workflow is the from an KDM encoder, not player.

A KDM template is required, with the **Reference** and **Signature** tags properly conditioned and prepared for processing by tools such as xmlsec1.

An XML example "ready-to-use". The **Authenticated** sections are empty for a better understanding :

```
<?xml version="1.0" encoding="UTF-8"?>
<DCinemaSecurityMessage xmlns="http://www.smpte-ra.org/schemas/430-3/2006/ETM">

  <AuthenticatedPublic Id="ID_AuthenticatedPublic">
        <!-- all metadatas -->
  </AuthenticatedPublic>

  <AuthenticatedPrivate Id="ID_AuthenticatedPrivate" xmlns:enc="http://www.w3.org/2001/04/xmlenc#">
        <!-- all encrypted Keys -->
  </AuthenticatedPrivate>

    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments"/>
            <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>

            <!-- for the block AuthenticatedPublic -->
            <ds:Reference URI="#ID_AuthenticatedPublic">
                <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                <ds:DigestValue/>   <!-- empty -->
            </ds:Reference>

            <!-- for the block AuthenticatedPrivate -->
            <ds:Reference URI="#ID_AuthenticatedPrivate">
                <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                <ds:DigestValue/>   <!-- empty -->
            </ds:Reference>

        </ds:SignedInfo>

        <!-- Signature (all DigestValue) -->
        <ds:SignatureValue />   <!-- empty -->

    </ds:Signature>

</DCinemaSecurityMessage>
```

You can find a complete template "ready-to-use" here: KDM.template-xmlsec.xml

Now, we pass this KDM, without any **DigestValue** or **SignatureValue** to the signature step.

```
# Signature KDM - with its private key
xmlsec1 sign \
    --id-attr:Id AuthenticatedPublic \
    --id-attr:Id AuthenticatedPrivate \
    --privkey-pem encoder_private_key.pem \
    KDM.xml
```

Don't forget to have your RSA private key (or to generate it).

Result (only the **Signature** part, the rest remains unchanged) :

```
(...)
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments"/>
        <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
        <ds:Reference URI="#ID_AuthenticatedPublic">
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>qN4dvJpemd94ppazl6ii6nmo9JflBczdpT9yXb3ltow=</ds:DigestValue>
        </ds:Reference>
        <ds:Reference URI="#ID_AuthenticatedPrivate">
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>9bowz05/W7f4qTJfO4K1VXTYEI14uQgJDYr6Z1uP/Ho=</ds:DigestValue>
        </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>FylhfvacbWQ8mcxLiGI3B6HL0EczuISQBYd+Ebkrll4oWs5RUaKnq4GA6o6+LE2m
    yNf2OdNcJ/7IKPvrDw8NFXR7KVrDWcJa9CGaXd87uxFpsUiBBj3u9Q/EIM4gaBH/
    RaaRsy0tKmEenguo6JWMVBBLE20bfLdOrBirpIyTbaIDUCyiUaI4qLrxR0uhuHvJ
    gTejDcNbznGPn4esFjcZHTO/C6EDW1U/N3t+AGOcCCjYBf80dIoAOluhVNyglWtV
    DNeW02sMtYuWc7m1swzjYqiBk+INkHnPrUvRsxZgzWoo3XGfgbXr15e2TY/IFN2C
    7bdJ5r6vXpB4dPfHThNxmg==</ds:SignatureValue>
</ds:Signature>
(...)
```

You have a signed KDM :-)

## VERIFY SIGNATURES (DIGESTVALUE + SIGNATUREVALUE) OF A KDM

> The public key used here in a DCI worklow is from the KDM encoder, not player

```
# Verify KDM - with public key
xmlsec1 verify \
    --id-attr:Id AuthenticatedPublic \
    --id-attr:Id AuthenticatedPrivate \
    --pubkey-cert-pem public-certificate.pem \
    KDM.xml
```

## VERIFY SIGNATURES (VIA OPENSSL)

```
openssl base64 -d -in signature.txt -out signature.txt.sha256
openssl dgst -sha256 -verify public-key.pem -signature signature.txt.sha256 filename
```

```
openssl dgst -sha256 -sign private-key.pem -out filename.sha256 filename
openssl base64 -in filename.sha256 -out signature.sha256
```

## CREATE A DIGESTVALUE (VIA OPENSSL)

```
printf '<Foo Id="Bar"></Foo>' | openssl sha256 -binary | openssl base64
r4kvmLusMibh32vkz03cxPDWcX/PUj5g1OJ146Z8OLM=
```

Verification, using xmlsec1 :

```
<?xml version="1.0" encoding="UTF-8"?>
<Test>

    <Foo Id="Bar"></Foo>

    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments"/>
            <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
            <ds:Reference URI="#Bar">
                <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                <ds:DigestValue/>
            </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue/>
    </ds:Signature>

</Test>
```

Output :

```
<?xml version="1.0" encoding="UTF-8"?>
<Test>

    <Foo Id="Bar"/>

    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments"/>
            <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
            <ds:Reference URI="#Bar">
                <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                <ds:DigestValue>r4kvmLusMibh32vkz03cxPDWcX/PUj5g1OJ146Z8OLM=</ds:DigestValue>
            </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>MXDe3khop+BGrSsQBe4A3+F+mYfrskMAn15G2ObHg7oCZhzzr+Yx5UZrm3LDPduS
6UUkUfBymoE7mcXhSpGrzeYca4xG3cKJvKEBCOTDeQefk9XGTt8fEa/Z2mCs/ZLp
QpnyVz9Ufq21M9Eo4oeRkV69dBY7u0Y1Op+i02ImVpyWKktim8wEvCAVp/sDaLIo
l46zdjpiynhg2pLNJdZhsC+lilIUcD9nDe/6BLF23d4UsjjRTOn4su1pT+VQqoCJ
6xPw4Sm0RkEpKKA4pupwk8t6pXbEs7HbvphqHsxqgkDR2sJg3Dk8d4HMlbnmp70q
3AtBAGruDVjRswxwnX7acQ==</ds:SignatureValue>
    </ds:Signature>

</Test>
```

Note that the encoding uses the long form ( `<Foo Id="Bar"></Foo>` ), which will be used during the sha256+base64 process to generate `r4kvmLusMibh32vkz03cxPDWcX/PUj5g1OJ146Z8OLM=` , whereas the XML output from xmlsec1 will be `<Foo Id="Bar"/>`

## CREATE A DIGESTVALUE (VIA OPENSSL)

```
openssl dgst -sha256 \
    -binary KDM-DigestValue-Foo.xml \
    | openssl base64

"r4kvmLusMibh32vkz03cxPDWcX/PUj5g1OJ146Z8OLM="
```

File KDM-DigestValue-Foo.xml

```
<Foo Id="Bar"></Foo>
```

## CREATE A SIGNATUREVALUE (VIA OPENSSL)

```
openssl dgst -sha256 \
    -sign sample_private_key_non-smpte.pem \
    -binary KDM-SignatureValue-SignedInfo.xml \
    | openssl base64

"NaAMJplu6vSiLFsHZ05ORr9adFJrhTZ33hnT5XS/elIAQqttOH30izLbNoFTYK/K
vbyxZfxpTsMoTXr1gz7v/01dNryZuxe19APRvOBMPCdoZlwzuXTb8X6udDlbAkFk
OGSwmhLuOvNgg3tpdPqH8nT/bxAPgx2gEJ0o/4r7VX3JDlZMQiL5+Yt+PJIMtbZ1
OHZGEZvAWLfkIeYV5IonFaRijWCqNl8P7x3UzlNsGKNfd7N71RZJ5HLI8bI57Cj0
9uRzmDh7z3ndPXRRUcE6B5x2efS46/llUzk13Xzgjsbq1Qg9hi/NzgbTOwpZc6oq
8GjNuWstqwXVC4fM8SkjkA=="
```

File KDM-SignatureValue-SignedInfo.xml

```
<ds:SignedInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments"></ds:CanonicalizationMethod>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"></ds:SignatureMethod>
    <ds:Reference URI="#Bar">
        <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"></ds:DigestMethod>
        <ds:DigestValue>r4kvmLusMibh32vkz03cxPDWcX/PUj5g1OJ146Z8OLM=</ds:DigestValue>
    </ds:Reference>
</ds:SignedInfo>
```

(careful with conversion `\s` -> `\t` and the CRLF)

## CANONICALIZE XML C14N (PYTHON)

```
#!/usr/bin/env python3
from lxml import etree

#
#   SignedInfo need Comments on canonicalized output
#   All <!-- --> are conserved
#

"""
Snippet : C14N from file to file (withComments)
    with open("c14n_output.xml", mode='w', encoding='utf-8') as out_file:
        etree.canonicalize(
            from_file = "inputfile.xml",
            out = out_file,
            with_comments = True
        )
"""

# Quick XML opening
with open("KDM-SignatureValue-SignedInfo-withComments.xml", "rb") as xml:
    tree = etree.fromstring(
        text = xml.read()
    )

# Read SignedInfo block
ap = tree.xpath("//*[local-name()='SignedInfo']")

# Go to canonicalization with Comments
ap_c4n = etree.canonicalize(etree.tostring(ap[0]).decode('utf-8'), with_comments=True)  # with_comments !

# Display output
print(ap_c4n)

# Write output
print("writing c14n.xml")
with open("c14n.xml", "w") as file:
    file.write(ap_c4n)
```

References Python :

- **withComments**: `lxml.tostring()` : https://lxml.de/api/lxml.etree-module.html#tostring
- **withComments**: `lxml.write_c14n()` : https://lxml.de/api/lxml.etree.ElementTree-class.html#write_c14n

## SUBCHAPTERS

- **KDM - Authenticated Public** : the non-encrypted part and readable by anyone

- **KDM - Authenticated Private** : the encrypted part and readable only by the receiver
- **KDM - Signature** : our part in authenticating and validating the KDM
- **KDM - Codes** : Source code and technical tips on each part of the KDM

## RELATED CHAPTERS

- **Cryptography** : To understanding the cryptography used in digital cinema.
- **Certificate** : The certificates used in digital cinema.
- **AES Cryptography** : The symmetric cryptography used to encrypt the MXF.
- **RSA Cryptography** : The asymmetric cryptography used in KDM to encrypt AES keys (among others things).