MXF : CODES & FICHIERS

JS-MXF

Prototype d'analyseur MXF sans besoin d'upload du fichier : tout est analysé en local sur votre machine.

Vous pouvez le tester avec ce fichier MXF.

DRAG YOUR MXF HERE

0 elements found
0 %

RÉCUPÉRATION DE L'IDENTIFIANT D'UN MXF (ASDCPLIB)

```
$ asdcp-info -i "picture.mxf" | awk '/AssetUUID/ { print $2 }'
3bd3d849-117b-46b0-bc45-3d3228c987c6
```

RÉCUPÉRATION DE L'IDENTIFIANT D'UN MXF (PYTHON)

Le programme prototype est disponible ici mxf-get-assetuid.py

```
$ mxf-get-assetuid.py "jp2k_video.mxf"
3bd3d849-117b-46b0-bc45-3d3228c987c6
```

RÉCUPÉRATION DE LA CRYPTOGRAPHIC-KEYID D'UN MXF (ASDCPLIB)

```
$ asdcp-info -i "picture.mxf" | awk '/CryptographicKeyID/ { print $2 }'
cf2ab7c6-c00f-4d52-aae2-3c3396a89b93
```

RÉCUPÉRATION DE LA CRYPTOGRAPHIC-KEYID D'UN MXF (PYTHON)

À faire

AFFICHER UN SMPTE UNIVERSAL LABEL - 16 OCTETS (PYTHON)

Au format **4x4 octets** :

```
bytes = b'\x06\x0e\x2b\x34\x02\x05\x01\x01\x0d\x01\x02\x01\x01\x02\x01\x00'
print("{:02x}{:02x}{:02x}{:02x}.{:02x}{:02x}{:02x}{:02x}.{:02x}{:02x}{:02x}{:02x}.{:02x}{:02x}{:02x}{:02x}".format(*
060e2b34.02050101.0d010201.01020100
```

Au format **UUID** :

```
bytes = b'\x06\x0e\x2b\x34\x02\x05\x01\x01\x0d\x01\x02\x01\x01\x02\x01\x00'
print("{:02x}{:02x}{:02x}{:02x}-{:02x}{:02x}-{:02x}{:02x}-{:02x}{:02x}-{:02x}{:02x}{:02x}{:02x}{:02x}{:02x}".format(*
060e2b34-0205-0101-0d01-020101020100
```

AFFICHER UN SMPTE UNIVERSAL LABEL - 12 OCTETS (PYTHON)

Au format **3x4 octets** :

```
bytes = b'\x06\x0e\x2b\x34\x02\x05\x01\x01\x0d\x01\x02\x01'
print("{:02x}{:02x}{:02x}{:02x}.{:02x}{:02x}{:02x}{:02x}.{:02x}{:02x}{:02x}{:02x}".format(*bytes))
060e2b34.02050101.0d010201
```

BER : DÉTERMINER SI BER EST SHORT FORM OU LONG FORM (C/C++)

```
bool ber_get_form(uint8_t bercode) {
    return (bool)(bercode & 0x80);
}

// Exemple d'utilisation
ber_get_form(0x83);  /* long form */
>>> true

ber_get_form(0x10);  /* short form */
>>> false
```

BER : RETOURNER LA TAILLE DU BER SELON SON CODE D'ENTÊTE (C/C++)

```
uint8_t ber_get_size(uint8_t bercode) {
    if (bercode >= 0x80)
        return (uint8_t)(bercode ^ 0x80);
    return 0;
}

// Exemple d'utilisation - (0x83 == BER à 3)
ber_get_size(0x83);
>>> 3
```

BER : CALCULER LA TAILLE (C/C++)

Réecrire cette fonction

```
unsigned long int ber_get_datasize(char *berdata, unsigned int bersize) {
    unsigned long int size = 0;
    unsigned short int index, shift, i;
    for(i = 0; i < bersize; i++) {
        index = 1-(i-(bersize-1))-1;   // swap-index
        shift = 8 * i;
        size |= ((unsigned long int)berdata[index] & 0xFF) << shift;
    }
    return size;
}

// Exemple d'utilisation - (0x000170 == 368)
ber_get_datasize("\x00\x01\x70", 3);
>>> 368   /* octets */
```

BER : CALCULER LA TAILLE VENANT D'UN BER 0X83 (EN PYTHON)

Réécrire cette fonction pour gérer l'ensemble des BER

```
def ber(bercode):
    return int.from_bytes(bercode, byteorder='big') ^ 0x83000000
```

IS SMTPE UNIVERSAL LABEL (C/C++)

Vérifie si le MXF ou le KLV est un SMPTE Universal Label (SMPTE UL) :

```
bool is_smpte_ul(char *buffer) {
    return (bool)(strncmp("\x06\x0e\x2b\x34", buffer, 4) == 0);
}
```

AFFICHER UN SMPTE UNIVERSAL LABEL - 16 OCTETS (C/C++)

Change 4x4o => uuid format

```
int smpte_uuid(char *src, char *dst) {
    return snprintf(dst,
        36,
        "%02x%02x%02x%02x.%02x%02x%02x%02x.%02x%02x%02x%02x.%02x%02x%02x%02x",
        src[0],  src[1],  src[2],  src[3],
        src[4],  src[5],  src[6],  src[7],
        src[8],  src[9],  src[10], src[11],
        src[12], src[13], src[14], src[15]
    );
}

// Exemple d'utilisation
char uuid_src[16] = "\x06\x0e\x2b\x34\x01\x02\x03\x04\x05\x06\x07\x08\x09\x1A\x1B\x1C";
char uuid_dst[36];
smpte_uuid(uuid_src, uuid_dst);
>>> uuid_dst = "060e2b34.01020304.05060708.091a1b1c"
```

LIRE UN MXF ET AFFICHER UNIVERSAL LABEL, TAILLE ET PARTIE DES DONNÉES (C/C++)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

/* this functions are defined in another place in documentation */
extern int smpte_uuid(char *, char *);
extern bool is_smpte_ul(char *);
extern uint8_t ber_get_datasize(char *, unsigned int);
extern uint8_t ber_get_size(uint8_t);
extern uint8_t ber_get_form(uint8_t);

/*
    Ce code n'est ni optimisé ni dans les règles de l'art du C.
    Ce code sert d'apprentissage : Ainsi les déclarations et autres mallocs
    sont placés par exemple dans le while alors qu'ils ne devraient pas.
    Certaines parties ont été fusionnées pour éviter des allers-retours inutiles.
*/

int main(int argc, char **argv) {

    if (argc < 2) {
        printf("Usage: %s <mxf>\n", argv[0]);
```

```
        return 1;
    }

    FILE *fh = fopen(argv[1], "r");

    while(!feof(fh)) {

        // Récupération de la position dans le fichier
        // (optionnel: pour l'affichage)
        unsigned long long int offset;
        offset = ftell(fh);
        printf("%12llu | ", offset);

        // -------------------------
        // ---------- Key ----------
        // -------------------------

        // Récupération du SMPTE UUID
        char key[16];
        fread(key, 16, 1, fh);

        // Affichage de la clef au format UUID
        char *uuid = (char *)malloc(sizeof(char *) * 36);
        smpte_uuid(key, uuid);
        printf("%s | ", uuid);
        free(uuid);

        // Arrêt si la clef n'est pas un UUID SMPTE (optionnel)
        if( !is_smpte_ul(key) ) {
            printf("\n**SMPTE UL not found**\n");
            break;
        }

        // -------------------------
        // -------- Length ---------
        // -------------------------

        // Récupération du 1er octet du BER (ex. 0x83)
        uint8_t ber_code;
        fread(&ber_code, 1, 1, fh);

        // Calcul de la véritable taille des données
        unsigned long long int data_size;
        if ( ber_get_form(ber_code) == 1 ) {      // Long form (>= 0x80)
            // get size of length
            unsigned int ber_size;
            ber_size = ber_get_size(ber_code);
            // read full length data
            char ber[ber_size];
            fread(ber, ber_size, 1, fh);
            // convert length data to size of data
            data_size = ber_get_datasize(ber, ber_size);
        } else {                                  // short BER (< 0x80)
            // get size of data
            data_size = ber_code;
        }
        printf("%10llu | ", data_size);

        // -------------------------
        // --------- Value ---------
        // -------------------------

        char *data = (char *)malloc(sizeof(char *) * data_size);
        fread(data, data_size, 1, fh);

        // Affichage des 16 premiers octets
        for(int i=0; i<16; i++) {
            printf("%02x", data[i] & 0xFF);
        }
        printf("\n");

        free(data);
    }

    fclose(fh);
```

```
    return 0;
}

// Exemple d'utilisation :
$ gcc mxf-reader.c -o mxf-reader

$ ./mxf-reader
$   Usage: ./mxf-reader <mxf>

$ ./mxf-reader video.mxf
           0 | 060e2b34.02050101.0d010201.01020400 |        104 | 0001000300000200000000000000000
         124 | 060e2b34.01010102.03010210.01000000 |        368 | 0000000000000000000000000000000
         512 | 060e2b34.02050101.0d010201.01050100 |       1772 | 0000006200000123c0a060e2b340101
        2303 | 060e2b34.01010102.03010210.01000000 |        237 | 0000000000000000000000000000000
        2560 | 060e2b34.02530101.0d010101.01012f00 |        154 | 3c0a0010adab44242f254dc792ff29bd
        2732 | 060e2b34.02530101.0d010101.01013000 |        168 | 3c0a0010adab44242f254dc792ff29bd
        2918 | 060e2b34.02530101.0d010101.01011800 |         92 | 3c0a0010adab44242f254dc792ff29bd
        3027 | 060e2b34.02530101.0d010101.01013600 |        124 | 3c0a0010adab44242f254dc792ff29bd
         ... |                  ...                |        ... |              ...
    62565888 | 060e2b34.02050101.0d010201.01040400 |        104 | 0001000300002000000000003baae00
    62566012 | 060e2b34.01010102.03010210.01000000 |        368 | 0000000000000000000000000000000
    62566400 | 060e2b34.02530101.0d010201.01100100 |        271 | 3c0a0010adab44242f254dc792ff29bd
    62566691 | 060e2b34.01010102.03010210.01000000 |        201 | 0000000000000000000000000000000
    62566912 | 060e2b34.02050101.0d010201.01110100 |        256 | 0000000000000000000000000000001
    62567187 | 060e2b34.02050101.0d010201.01110100 |        256 | 0000000000000000000000000000001
```

Bug d'affichage sur la dernière ligne - à corriger

CRYPTOGRAPHIE

XOR

```
def xor(v1, v2):
    return bytes(a ^ b for a, b in zip(v1, v2))

print(xor(
    b'\x01\x23\x45\x67\x89\xAB\xCD\xEF',
    b'\x45\xA9\x1E\x06\xB4\xCD\x87\x1A'
).hex().upper())
```

CHIFFREMENT AES-128-CBC

```
from cryptography.hazmat.primitives.ciphers import ( Cipher, algorithms, modes )
from cryptography.hazmat.backends import default_backend

# La clef de chiffrement : 'UNE CLEF SECRETE'
key       = b'\x55\x4E\x45\x20\x43\x4C\x45\x46\x20\x53\x45\x43\x52\x45\x54\x45'

# IV pas trop aléatoire ;-)
iv        = b'\x01\x02\x03\x04\x05\x06\x07\x08\x09\x10\x11\x12\x13\x14\x15\x16'

# Le contenu en clair : 'BONJOUR LE MONDE'
plaintext = b'\x42\x4F\x4E\x4A\x4F\x55\x52\x20\x4C\x45\x20\x4D\x4F\x4E\x44\x45'

# On définit le moteur cryptographique
cipher = Cipher(
    algorithms.AES(key),
    modes.CBC(iv),
    backend=default_backend()
)
encryptor = cipher.encryptor()

# On chiffre notre message en clair
ciphertext = encryptor.update(plaintext)

# Donne :
#   0xEB 0x36 0xD4 0x79
#   0x25 0x6C 0x9E 0x6D
#   0x1B 0xB4 0x11 0x52
#   0x6f 0x00 0x9C 0x1B
print(ciphertext, ciphertext.hex())
```

DÉCHIFFREMENT AES-128-CBC

```
from cryptography.hazmat.primitives.ciphers import ( Cipher, algorithms, modes )
from cryptography.hazmat.backends import default_backend

# La clef de chiffrement : 'UNE CLEF SECRETE'
key       = b'\x55\x4E\x45\x20\x43\x4C\x45\x46\x20\x53\x45\x43\x52\x45\x54\x45'

# IV pas trop aléatoire ;-)
iv        = b'\x01\x02\x03\x04\x05\x06\x07\x08\x09\x10\x11\x12\x13\x14\x15\x16'

# Le contenu chiffré
ciphertext = b'\xeb\x36\xd4\x79\x25\x6c\x9e\x6d\x1b\xb4\x11\x52\x6f\x00\x9c\x1b'

# On définit le moteur cryptographique
cipher = Cipher(
    algorithms.AES(key),
    modes.CBC(iv),
    backend=default_backend()
)
decryptor = cipher.decryptor()

# On déchiffre notre message chiffré
plaintext = decryptor.update(ciphertext)

# Donne 'BONJOUR LE MONDE'
print(plaintext, plaintext.hex())
```

PROCESSUS DE DÉCHIFFREMENT DE LA VALUE DU KLV ENCRYPTED ESSENCE CONTAINER

Ce code est un exemple pour déchiffrer les 16 premiers octets d'un contenu chiffré utilisant la norme "SMPTE-429-6 - Essence Encryption" (KLV JPEG2000, KLV WAV PCM, KLV Dolby Atmos, KLV Subtitles, etc...)

```
#!/usr/bin/env python3

from cryptography.hazmat.primitives.ciphers import ( Cipher, algorithms, modes )
from cryptography.hazmat.backends import default_backend

# Clef de déchiffrement utilisé pour déchiffrer le MXF
key = b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'

# Initialization Vector (IV) ou vecteur d'initialisation
# Bloc de 16 octets utilisé pour initier le moteur cryptographique
iv  = b'\xc7\x51\x65\x82\xd2\x74\x77\x9e\x09\xf8\x44\x01\x77\x0a\x61\x60'

# "CheckValue" sont les premiers 16 octets qui intègrent une valeur définie
# Déchiffré, cela donne '43:48:55:4B:43:48:55:4B:43:48:55:4B:43:48:55:4B' ou 'CHUKCHUKCHUKCHUK'
checkvalue = b'\x44\x8d\xf3\xbc\x12\x60\x6b\xb2\xe3\xcc\x4a\xed\xec\x31\xbe\x78'

# Les 16 premiers bits de "Value" de notre KLV (ceci est le Header JPEG2000 chiffré)
data = b'\xbb\x07\x4c\x4f\x11\x37\x30\x6a\x64\x45\xad\x3d\xaf\x0c\xc3\x05'

# Initialisation du moteur cryptographique
cipher = Cipher(
    algorithms.AES(key),
    modes.CBC(iv),
    backend=default_backend()
)
decryptor = cipher.decryptor()

# On intègre les premiers 16 octets qui sont "CheckValue" chiffré
decryptor.update(checkvalue)

# On intègre les 16 octets de notre exemple
plaintext = decryptor.update(data)

# Les données déchiffrées devraient donner "ff4fff51002f00030000080000000438"
# Ces données sont les headers JPEG2000
print(plaintext.hex())
```

Pour avoir la version complète (simplifiée) du programme permettant de lire un MXF chiffré et d'en extraire les essences déchiffrées : mxf-encrypted-decryption.py

```
$ mxf-encrypted-decryption.py
Usage: ./parse-klv-encrypted-essence.py <mxf> <aeskey>

$ mxf-encrypted-decryption.py encrypted-key-00000000000000000000000000000000.mxf 00000000000000000000000000000000
060e2b34020401010d010301027e0100 -  40300 - 8300001067bec4fc40de4996aac7fa42...
CryptographicContextLink Length         : 83000010
CryptographicContextLink Value          : 67bec4fc40de4996aac7fa42a6b0ed5e
PlaintextOffset Length                  : 83000008
PlaintextOffset Value                   : 0000000000000000 (0 bytes)
SourceKey Length                        : 83000010
SourceKey Value                         : 060e2b34010201010d01030115010801
SourceLength Length                     : 83000008
SourceLength Value                      : 0000000000009cc8 (40136 bytes)
Encrypted Source Length                 : 83009cf0 (40176 bytes)
Encrypted Source Value - IV             : 765a067b36dfd2e89da94a9c6af0902f
Encrypted Source Value - CheckValue     : 7d95b3c594116732ed0b2d9b13ac5283
Encrypted Source Value - Plaintext Data :
Encrypted Source Value - Encrypted Data : 9c52432ad90a1bba64fd0ac5c604a1c9...
TrackFile ID Length                     : 83000010
TrackFile ID Value                      : 89af85f04a1545ec8a769008829b2029
Sequence Number Length                  : 83000008
Sequence Number Value                   : 0000000000000001
Message Integrity Code (MIC) Length     : 83000014
Message Integrity Code (MIC) Value      : 5b594d66d09cf6ddfda8f6e691e4291ea7097bc8
Plaintext Source Value                  : 40144 bytes
Padding                                 : 8 bytes
```

Notez que ce programme est orienté extraction des frames JPEG2000 mais marche avec les autres types d'essences (à quelques exceptions, voir les différentes sections pour chaque type d'essence).

Des samples de MXF chiffrés : Voir Annexes

CRÉATION D'UN MXF CHIFFRÉ (ASDCPLIB)

```
asdcp-test \
    -L \
    -j deadbeefdeadbeefdeadbeefdeadbeef \
    -k 00000000000000000000000000000000 \
    -c output.mxf \
    ./
```

Explication:

```
-L : genère un MXF SMPTE (pas Interop)
-j <keyId> : défini manuellement la "Cryptographic Key Id"
-k <aeskey> : la clef AES pour le chiffrement
-c <outputfile>
```

CRÉATION D'UN MXF CHIFFRÉ AVEC PLAINTEXT OFFSET (ASDCPLIB)

```
asdcp-test \
    -L \
    -E \
    -j deadbeefdeadbeefdeadbeefdeadbeef \
    -k 00000000000000000000000000000000 \
    -c output.mxf \
    ./
```

Explication :

```
-L : genère un MXF SMPTE (pas Interop)
-E : ne chiffre pas les headers (36 premiers octets)
-j <keyId> : défini manuellement la "Cryptographic Key Id"
-k <aeskey> : la clef AES pour le chiffrement
-c <outputfile>
```

EXTRACTION DES ESSENCES D'UN MXF CHIFFRÉ (ASDCPLIB) :

```
asdcp-unwrap \
    -v \
    -k 00000000000000000000000000000000 \
    encrypted.mxf \
    extract/
```

Explication :

```
-v : Verbose
-k <aeskey> : la clef AES pour le déchiffrement
```

EXTRACTION DES ESSENCES D'UN MXF CHIFFRÉ (MXF-ANALYZER)

Disponible ici : assets/MXF/mxf-analyzer/

```
mxf-analyzer.py \
    -f encrypted.mxf \
    -x extract_directory/ \
    -k 00000000000000000000000000000000
```

CRÉATION D'UN MXF CHIFFRÉ AVEC ASDCPLIB (C++)

Ce code est un proof-of-concept, il créera un MXF qu'avec une seule frame chiffrée :

```
#include <AS_DCP.h>
#include <KM_prng.h>  /* FortunaRNG */
#include <Metadata.h> /* MXF:: */

using namespace ASDCP;

int main(void) {

    WriterInfo Info;
    JP2K::MXFWriter Writer;
    JP2K::FrameBuffer FrameBuffer(1024 * 1024);
    JP2K::PictureDescriptor PDesc;
    JP2K::SequenceParser Parser;
    // AES
    AESEncContext* Context = 0;
    const byte_t aes_key[16] = {
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
    };
    Kumu::FortunaRNG RNG;
    byte_t IV_buf[CBC_BLOCK_SIZE];
    // HMAC
    HMACContext* HMAC = 0;

    // Set Header
    Info.LabelSetType = LS_MXF_SMPTE;
    Kumu::GenRandomUUID(Info.ContextID);
    // AssetUUID == TrackFile ID (MIC)
    Kumu::GenRandomUUID(Info.AssetUUID);

    // Set Cryptographic
    Kumu::GenRandomUUID(Info.CryptographicKeyID);
    Info.EncryptedEssence = true;
    Context = new AESEncContext;
    Context->InitKey(aes_key);
    Context->SetIVec(RNG.FillRandom(IV_buf, CBC_BLOCK_SIZE));

    // Set HMAC
    Info.UsesHMAC = true;
    HMAC = new HMACContext;
    HMAC->InitKey(aes_key, Info.LabelSetType);

    // Set Parser from files
    Parser.OpenRead("essences/");
    Parser.FillPictureDescriptor(PDesc);

    // Go to the first file
    Parser.Reset();

    // Open MXF
    Writer.OpenWrite("dump.mxf", Info, PDesc);

    // --- foreach frame -------------------------------------
    // Read each frame (only one here)
    // Each call of ReadFrame() shift to the next frame
    // ReadFrame() returns a zero if no new frame
    Parser.ReadFrame(FrameBuffer);
    FrameBuffer.PlaintextOffset(0);  // force no plaintext
    // Write each frame into MXF (only one here)
    Writer.WriteFrame(FrameBuffer, Context, HMAC);
    // -------------------------------------------------------

    // Close MXF
    Writer.Finalize();

    // Show 256 bytes from JPEG2000
    FrameBuffer.Dump(stderr, 256);
    // Show all metadatas from JPEG2000
    JP2K::PictureDescriptorDump(PDesc);

    return 0;
}
```

Pour compiler :

```
# Vous devrez d'abord compiler asdcplib
# afin d'avoir les librairies libasdcp et libkumu
ASDCPLIB="/chemin/asdcplib/"

g++ -g -O2 \
    -lpthread \
    -Wl,-bind_at_load \
    -DHAVE_OPENSSL=1 \
    -I$(ASDCPLIB)/src/ \
    $(ASDCPLIB)/src/.libs/libasdcp.so \    # Linux
    $(ASDCPLIB)/src/.libs/libkumu.so \     # Linux
    $(ASDCPLIB)/src/.libs/libasdcp.dylib \ # MacOS
    $(ASDCPLIB)/src/.libs/libkumu.dylib \  # MacOS
    `pkg-config openssl --cflags` \
    `pkg-config openssl --libs` \
    asdcplib-create-encrypted-mxf.cpp \
    -o asdcplib-create-encrypted-mxf
```

Pour démarrer :

```
# Vous devrez d'abord compiler asdcplib
# afin d'avoir les librairies libasdcp et libkumu
ASDCPLIB="/chemin/asdcplib/"

# Linux
LD_LIBRARY_PATH="$(ASDCPLIB)/src/.libs:${LD_LIBRARY_PATH}" ./asdcplib-create-encrypted-mxf

# MacOS
DYLD_LIBRARY_PATH="$(ASDCPLIB)/src/.libs:${DYLD_LIBRARY_PATH}" ./asdcplib-create-encrypted-mxf
```

Vous retrouverez le code source ici : asdcplib-create-encrypted-mxf.cpp + Makefile

FICHIERS

- **Outils** :
  - encrypted-decryption.py Permet de lire, déchiffrer et extraire les données d'un MXF chiffré en Python
  - encrypted-hmac.py Calcul du Message Integrity Code (MIC) au format HMAC-SHA1-128 en Python
  - get-assetuid.py Donne l'AssetUUID d'un MXF en Python
  - proto-hmac.c Prototype d'implémentation de l'algorithme HMAC-SHA1-128 en C
  - proto-hmac.py Prototype d'implémentation de l'algorithme HMAC-SHA1-128 en Python
- **Librairie asdcplib** :
  - hmac.cpp + patch Makefile Génération d'une clef de dérivation (Derivation Key) et d'un Message Integrity Code (MIC) au format HMAC-SHA1-128 en C++, en utilisant la librairie ascplib.
  - asdcplib-create-encrypted-mxf.cpp Création d'un MXF chiffré en C++, en utilisant la librairie asdcplib.
  - asdcplib.plaintextoffset.patch Patch pour pouvoir définir le Plaintext Offset via variable d'environnement PLAINTEXT_OFFSET quand on utilise les outils asdcplib. (quick & dirty patch)
  - create-samples.sh Exemples d'utilisation asdcp-test pour créer des MXF.
  - asdcplib-fips-lite.cpp Fonction unitaire de dérivation de clef (Derivation Key) respectant la norme FIPS 186-2, "General Purpose Random Number Generation", en C++.
- **Assets MXF** :
  - **MXF chiffrés** :
    - MXF encrypted.mxf (clef AES: `00000000000000000000000000000000`)
    - MXF encrypted-plaintextoffset.mxf (clef AES: `00000000000000000000000000000000`)
    - MXF `DCP-INSIDE-CRYPTE` jp2k_video.mxf (clef AES: `6e256ec2308835ea1d46d8a359296f38`)
    - MXF `DCP-INSIDE-CRYPTE` wav_audio.mxf (clef AES: `f5a3d36ab03412984de4aa313199437a`)

    - Layers **KLV** provenant de MXF chiffrés :
        - KLV encrypted (dir)
        - KLV encrypted-plaintextoffset (dir)
- **MXF non-chiffrés** :
    - MXF `DCP-INSIDE` (non-chiffré) jp2k_video.mxf & wav_audio.mxf
    - MXF audio.smpte.mxf - 2 channels, 48 kHz, 24 bits
    - MXF 2D.mxf - 1 frame 4K JPEG2000 XYZ - 24/1 fps
- **Assets sources** :
    - Frame JPEG2000 4K XYZ (générée par DVS Clipster 5.10)

    - Son WAV PCM 2 channels 48 kHz 32 bits - Tambour (~182 ms) Remplacer par une version 24 bits

---