CERTIFICATS : LES CHAMPS (FIELDS) D'UN CERTIFICAT X509 DCI

PRÉFACE

Nous avons vu rapidement qu'un certificat x509 était un ensemble de métadonnées. Chaque métadonnée ayant une entrée appelée champ (ou fields en anglais). De par sa structure interne, elles sont rangées par catégorie, par bloc de données. Nous allons voir chacune de ces blocs un par un.

Certificats et Chaînes ?

Dans ce chapitre, nous aurons quelques notions de chaînes de certificats, de certificat root, intermédiaire et leaf (parfois surnommés certificats *parents* ou *enfants* pour plus de clarté)

Nous verrons toutes ces notions plus en détails dans le chapitre chaînes de certificats.

Cependant, sachez juste que chaque certificat d'un appareil DCI possède une hiérarchie de certificats au-dessus de lui - plus ou moins denses suivant le constructeur - jusqu'à arriver à un certificat unique dit racine.

Si vous voulez vous visualiser cela, voyez-le comme un arbre : la racine (root), le tronc et branches (intermediates) représentant la chaine d'autorité - et les feuilles (leaf) qui réprésentent les certificats en bout de queue, ceux des appareils (player / encodeur)

Considérez également que le terme **Issuer** définit le **parent direct** d'un certificat et que le terme **Subject** définit soit le **certificat actuel** soit le certificat **enfant direct**.

A L'INTÉRIEUR D'UN CERTIFICAT X509 DCI

Revoyons l'intérieur d'un **certificat x509 DCI** et étudions maintenant chaque entrée :

```
$ openssl x509 -in certificate.pem -text
Data:
    Version: 3 (0x2)
    Serial Number: 643 (0x283)
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: O = DC2.SMPTE.DOREMILABS.COM, OU = DC.DOREMILABS.COM, CN = .DC.DMS.DC2.SMPTE, dnQualifier = "+LLvuYNO4YBJSp9Jjmlv8oippzQ="
    Validity
        Not Before: Jan  1 00:00:00 2007 GMT
        Not After : Dec 31 23:59:59 2025 GMT
    Subject: O = DC2.SMPTE.DOREMILABS.COM, OU = DC.DOREMILABS.COM, CN = CS.DMSJP2K-80119.DC.DC2.SMPTE, dnQualifier = SQFYSSqWwjefppqasMJmfdmS6lI=
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
            Public-Key: (2048 bit)
            Modulus:
                00:bc:62:a5:fb:33:4f:73:58:2a:6a:03:37:2b:52:
                62:4e:17:1a:41:6f:6e:f4:c3:98:b5:32:4c:4d:54:
                62:c4:e7:ee:e6:c1:88:39:80:05:7f:a1:66:49:fe:
                55:9d:e1:06:1b:db:5d:fe:23:7a:1a:99:48:d5:ee:
                6e:a5:3b:e4:cb:0d:42:4e:68:ae:02:9d:e9:a9:ca:
                8b:84:cf:2f:c8:f9:ed:ce:5a:08:09:33:71:7b:d0:
                08:8a:e3:a7:25:03:b5:92:12:c6:51:0d:69:80:3e:
                4c:be:97:f2:6b:fe:08:8a:a9:ea:f2:ea:51:f3:83:
                e8:2e:79:ec:10:ab:e8:c5:eb:97:6b:58:8b:ac:2c:
                83:67:29:0f:ed:86:e7:f1:4e:66:bb:37:40:aa:bf:
                58:46:e1:73:17:36:db:ab:c1:5c:af:3d:6e:3e:1c:
                8a:78:ad:22:eb:e6:50:55:d7:d3:f3:1c:b8:06:e3:
                46:41:cb:9c:8a:63:c6:a0:89:ae:fb:6c:9f:35:b1:
                1c:7c:54:1f:7c:30:39:6c:6c:d1:db:f5:c3:25:32:
                c4:6a:a3:70:7b:f5:97:67:21:a8:91:9b:48:47:39:
                85:25:81:9b:e1:9c:be:a5:81:96:20:ae:6e:9a:e8:
                63:34:51:13:b9:f3:48:e3:c9:b2:6c:d6:6d:7a:5a:
                63:23
            Exponent: 65537 (0x10001)
    X509v3 extensions:
        X509v3 Basic Constraints: critical
            CA:FALSE
        X509v3 Key Usage:
            Digital Signature, Key Encipherment, Data Encipherment
        X509v3 Subject Key Identifier:
            49:01:58:49:2A:96:C2:37:9F:A6:9A:9A:B0:C2:66:7D:D9:92:EA:52
        X509v3 Authority Key Identifier:
            keyid:F8:B2:EF:B9:83:4E:E1:80:49:4A:9F:49:8E:69:6F:F2:88:A9:A7:34
            DirName:/O=DC2.SMPTE.DOREMILABS.COM/OU=DC.DOREMILABS.COM/CN=.DMS.DC2.SMPTE/dnQualifier=RQ\/53RmuLsbzgfPXGlRYmJruwMs=
            serial:02
Signature Algorithm: sha256WithRSAEncryption
Signature Value:
    3c:1a:59:e9:39:20:f5:36:60:d8:b6:a6:2d:a3:82:3c:4e:a2:
    9e:00:6f:aa:6b:10:1d:1e:55:e1:bf:7d:b0:d5:3f:12:f7:46:
    2f:67:89:8d:91:70:e1:82:ec:c5:a1:26:3e:9a:48:18:57:4c:
    20:4f:90:24:7a:c4:0f:96:0b:68:9f:62:d5:5b:d3:6c:3d:63:
    35:fa:dc:95:6e:12:6d:ce:be:9a:54:0a:14:2e:af:38:3e:7f:
    82:8d:e1:2d:80:bc:03:9c:3d:d9:e6:bb:e6:25:fb:ed:2b:83:
    d0:30:83:d4:62:2c:e8:52:f7:10:64:ab:31:70:b9:f4:71:4a:
    e1:a8:67:22:4c:5c:53:28:55:ef:90:a7:d7:8b:a3:68:e4:29:
    88:ef:b3:07:1a:ff:55:a8:bf:3c:36:68:cb:a2:92:9b:4c:98:
    24:48:7d:ee:ae:3e:bd:06:10:95:15:92:3f:57:05:f4:88:cb:
    ba:ca:d8:05:38:d4:df:47:fb:af:28:0e:47:8b:dc:a8:bf:31:
    9c:d4:21:62:9a:c1:94:90:67:f8:a4:b7:16:a3:4a:b6:b5:28:
    1d:31:74:62:d2:e5:e0:8e:65:f7:4a:1c:b6:5f:af:b5:03:46:
    5b:1a:b8:c5:4d:f7:0e:ef:6a:5c:e0:57:04:4f:61:79:27:c6:
    dc:2b:26:57
```

Comme déjà dit, ces entrées sont appelées "Champ" ou "Field". Un certificat DCI est composé de ces 4 blocs de structures de données :

- **Métadonnées**
- **Subject Public Key Info**
- **Extensions x509v3**
- **Signature**

Ces champs ont des rôles particuliers avec des paramètres précis, étudions-les un par un.

BLOC MÉTADONNÉES - SA CARTE D'IDENTITÉ PUBLIQUE

Ce bloc représente l'essentielle des métadonnées lisible par un humain et utilisable par des systèmes pour la gestion d'un workflow cryptographique DCI :

```
Version: 3 (0x2)
Serial Number: 643 (0x283)
Signature Algorithm: sha256WithRSAEncryption
Issuer: O = DC2.SMPTE.DOREMILABS.COM, OU = DC.DOREMILABS.COM, CN = .DC.DMS.DC2.SMPTE, dnQualifier = "+LLvuYNO4YBJSp9Jjmlv8oippzQ="
Validity
    Not Before: Jan  1 00:00:00 2007 GMT
    Not After : Dec 31 23:59:59 2025 GMT
Subject: O = DC2.SMPTE.DOREMILABS.COM, OU = DC.DOREMILABS.COM, CN = CS.DMSJP2K-80119.DC.DC2.SMPTE, dnQualifier = SQFYSSqWwjefppqasMJmfdmS6lI=
```

Nom du champ x509	Paramètres
Version	x509v3
Serial Number	Numéro unique attribué par l'Issuer (celui a signé le certificat, le parent).
Signature Algorithm	Identifiant normé dans la RFC4055 indiquant l'utilisation des algorithmes **sha256WithRSAEncryption** (PKCS#1)
Issuer	Le nom unique complet de l'entité (certificat parent) qui a généré et signé ce certificat
Subject	Le nom unique complet de l'entité de ce certificat
Validity Not Before **Validity** Not After	Les dates de validité du certificat

Le champ **Subject représente l'identité du certificat** qui - d'un simple coup d'oeil - nous permet de connaître sa capacité et ses rôles, et le champ **Issuer** - qui est le **Subject de son parent**, celui qui l'a généré et signé.

Ces deux champs ont une syntaxe particulière, ce sont des attributs et nous verrons chacun plus en détail dans le chapitre suivant Identity Attributes : les attributs et leurs rôles.

Notez que les **dates de validité** ne sont qu'à titre indicatif et n'ont aucun mécanisme de blocage cryptographique : si votre workflow décide d'ignorer la date de validité, vous pourrez quand même utiliser la clef publique.

Le champ **SignatureAlgorithm** est l'algorithme utilisé par le certificat parent direct (autorité de certification / CA) afin de créer la **Signature** à la toute fin du certificat. Derrière **sha256WithRSAEncryption** se cache un identifiant - appelé OID pour ObjectID - qui permet de définir le type d'algorithme. Ici, nous aurons l'ObjectID n°1.2.840.113549.1.1.11 qui représente l'algorithme **sha256WithRSAEncryption PKCS#1 v1.5**.

Important, le champ **SignatureAlgorithm** du bloc Métadonnées doit être équivalent au champ **SignatureAlgorithm** dans le bloc **Signature**:

```
Data:
    Version: 3 (0x2)
    Serial Number: 643 (0x283)
    Signature Algorithm: sha256WithRSAEncryption   <---------------+
    (...)                                                          |
Subject Public Key Info:                                           |
    (...)                                                          |
X509v3 extensions:                                                 |
    (...)                                                          |
Signature:                                                         |
    Signature Algorithm: sha256WithRSAEncryption   <---------------+
    Signature Value:
     3c:1a:59:e9:39:20:f5:36:60:d8:b6:a6:2d:a3:82:3c:4e:a2:
    (...)
```

Le numéro de série est attribué par son parent direct (autorité de certification / CA). Il doit être unique par enfants directs :

Les certificats C, D et E du certificat parent A sont tous uniques.

Le certificat D possède le même numéro de série que le certificat F délivré par le certificat parent B mais il n'est pas l'enfant direct du certificat A, c'est donc accepté.

Si, par exemple, le certificat C pouvait signer des certificats, il aurait le droit de créer des certificats enfants avec n'importe quelle numéro, même avec le numéro 7, 19, 42 ou encore 23.

A noter que le **Serial Number** doit être un nombre positif sous 64 bits, donc 0 à 2^{64} - 1. Cela laisse de la marge :-)

BLOC SUBJECT PUBLIC KEY INFO

Ce bloc nous sera très utile lors de la création de certaines empreintes. Empreintes que nous détaillerons dans deux chapitres séparés car nécessitant de s'y attarder.

Ce bloc intègre notre clef publique (modulus) et son exposant (exponent). Ce sont eux qui vont nous servir pour les calculs cryptographiques RSA.

```
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        Public-Key: (2048 bit)
        Modulus:
            00:bc:62:a5:fb:33:4f:73:58:2a:6a:03:37:2b:52:
            62:4e:17:1a:41:6f:6e:f4:c3:98:b5:32:4c:4d:54:
            62:c4:e7:ee:e6:c1:88:39:80:05:7f:a1:66:49:fe:
            55:9d:e1:06:1b:db:5d:fe:23:7a:1a:99:48:d5:ee:
            6e:a5:3b:e4:cb:0d:42:4e:68:ae:02:9d:e9:a9:ca:
            8b:84:cf:2f:c8:f9:ed:ce:5a:08:09:33:71:7b:d0:
            08:8a:e3:a7:25:03:b5:92:12:c6:51:0d:69:80:3e:
            4c:be:97:f2:6b:fe:08:8a:a9:ea:f2:ea:51:f3:83:
            e8:2e:79:ec:10:ab:e8:c5:eb:97:6b:58:8b:ac:2c:
            83:67:29:0f:ed:86:e7:f1:4e:66:bb:37:40:aa:bf:
            58:46:e1:73:17:36:db:ab:c1:5c:af:3d:6e:3e:1c:
            8a:78:ad:22:eb:e6:50:55:d7:d3:f3:1c:b8:06:e3:
            46:41:cb:9c:8a:63:c6:a0:89:ae:fb:6c:9f:35:b1:
            1c:7c:54:1f:7c:30:39:6c:6c:d1:db:f5:c3:25:32:
            c4:6a:a3:70:7b:f5:97:67:21:a8:91:9b:48:47:39:
            85:25:81:9b:e1:9c:be:a5:81:96:20:ae:6e:9a:e8:
            63:34:51:13:b9:f3:48:e3:c9:b2:6c:d6:6d:7a:5a:
            63:23
        Exponent: 65537 (0x10001)
```

Nom du champ x509	Paramètres
Subject Public Key Info	Bloc de description pour la clef publique
Public Key Algorithm	rsaEncryption (références RFC ou OIDinfo)
Public Key (taille)	Indication seulement sur la taille de la clef publique, ici nous sommes en RSA-2048, donc la clef aura une taille de 2048 bits
Modulus	C'est **notre clef publique** d'une taille de 2048 bits
Exponent	Exposant public - toujours à 65537 pour un certificat public [1]

Public Key Algorithm identifie le type d'algorithme utilisé dans ce bloc. Ici, nous aurons du rsaEncryption. Cela indique également que nos deux nombres utiles pour le RSA modulus et exponent seront encapsulés à la suite de **Public Key Algorithm** [2].

A noter que le champ **Public Key (taille)** n'existe pas réellement dans notre certificat, ce n'est pas un champ x509 ni ASN.1, c'est seulement notre outil OpenSSL qui calcul la taille de la clef et l'affiche pour plus de visibilité. Si on regarde notre structure ASN.1, nous ne verrons pas de tag ASN.1 spécifique pour notre 2048 :

```
SEQUENCE  (2 elem)
  SEQUENCE  (2 elem)
    OBJECT IDENTIFIER  1.2.840.113549.1.1.1  rsaEncryption  (PKCS #1)
    NULL
  BIT STRING  (2160 bit) 001100001000001000000001000010100000001010000010000000010000000100000…
    SEQUENCE  (2 elem)
      INTEGER  (2048 bit) 23781446655073751128304058141265963265170186434803895824054975210494 2…
      INTEGER  65537
```

Si vous vous souvenez des quelques notions ASN.1 vues rapidement, vous remarquez simplement notre `OBJECT IDENTIFIER` (l'identifiant rsaEncryption) et notre `BIT STRING` encapsulant nos deux `INTEGERS` (modulus et exponent). Rien de plus.

Le bloc **Subject Public Key Info** va nous être très utile lors du calcul de deux empreintes cryptographiques très utilisées dans notre workflow DCI, en premier lieu notre Public Key Thumbprint - dnQualifier qui va l'utiliser directement, et en deuxième lieu notre Certificate Thumbprint qui va l'utiliser indirectement avec d'autres blocs.

BLOC X509V3 EXTENSIONS

Les extensions x509v3 ne sont pas spécifiques au DCI, elles sont normées dans les spécifications x509 et sont utilisés par tout autre système de gestion utilisant la cryptographie (PKI, VPN, etc...).

```
X509v3 extensions:
    X509v3 Basic Constraints: critical
        CA:FALSE
    X509v3 Key Usage:
        Digital Signature, Key Encipherment, Data Encipherment
    X509v3 Subject Key Identifier:
            49:01:58:49:2A:96:C2:37:9F:A6:9A:9A:B0:C2:66:7D:D9:92:EA:52
    X509v3 Authority Key Identifier:
        keyid:F8:B2:EF:B9:83:4E:E1:80:49:4A:9F:49:8E:69:6F:F2:88:A9:A7:34
        DirName:/O=DC2.SMPTE.DOREMILABS.COM/OU=DC.DOREMILABS.COM/CN=.DMS.DC2.SMPTE/dnQualifier=RQ\/53RmuLsbzgfPXGlRYmJruwMs=
        serial:02
```

Nom du champ x509	Paramètres
Basic Constraint	Indique si le certificat peut signer d'autres certificats. Indique également si ce certificat fait partie de l'autorité de certification (Certification Authority)
Key Usage	Opérations autorisées avec la clef publique
Subject Key Identifier	Une clef unique identifiant le certificat pour une chaîne de certification (Public Key Thumbprint)
Authority Key Identifier	Identifie le certificat parent (Issuer, celui qui a signé ce certificat) Vous aurez trois champs: • **keyIdentifier** (KeyId): Le champ **Subject Key Id** du certificat parent (Issuer). • **AuthorityCertIssuer** (DirName) : Le champ **Issuer** du certificat parent (Issuer) [3] • **AuthorityCertSerialNumber** (Serial) : Le numéro de série du certificat parent (Issuer)

Ces champs étant particuliers, nous allons nous attarder.

CHAMPS SUBJECT KEY IDENTIFIER & AUTHORITY KEY IDENTIFIER

Subject Key Identifier est un identifiant unique - d'une taille de 20 octets - pour ce certificat, il sera utilisé par les certificats enfants comme identifiant **Authority Key Identifier -> KeyId**. Nous verrons leurs réseaux de relations dans le chapitre sur **la chaîne de certificats**.

Cet identifiant n'est pas pris au hasard, il doit être récupéré d'un calcul sur la Subject Public Key que nous verrons également dans le chapitre suivant Public Key Thumbprint (dnQualifier), il sera la valeur SHA1 de la **Subject Public Key** et est appelé **Public Key Thumbprint** [4]

Pour le réseau de relation entre **Authority Key Identifier**, nous verrons tout ceci dans le chapitre sur **la chaîne de certificats**. La seule chose que vous devez vous rappeler est que l'ensemble des champs **Authority Key Identifier** font des liens directs avec le certificat parent, celui qui la générée.

Les relations entre **Authority Key Identifier** du **certificat enfant** et les champs **certificat parent** :

```
          Certificat Parent
         +-------------------------------------------------------------------------------------------------------------------------------
┌────────|—  Serial Number: 2 (0x2)
┌────────|—  Issuer  : dnQualifier = "RQ/53RmuLsbzgfPXGlRYmJruwMs=", CN=.DMS.DC2.SMPTE, OU=DC.DOREMILABS.COM, O=DC2.SMPTE.DOREMILABS.COM
│        |    X509v3 Subject Key Identifier :
│    ┌───|—      F8:B2:EF:B9:83:4E:E1:80:49:4A:9F:49:8E:69:6F:F2:88:A9:A7:34
│    │   +-------------------------------------------------------------------------------------------------------------------------------
│    │
│    │     Certificat Enfant
│    │   +-------------------------------------------------------------------------------------------------------------------------------
│    │   |    X509v3 Authority Key Identifier :
│    └───|—      keyid   : F8:B2:EF:B9:83:4E:E1:80:49:4A:9F:49:8E:69:6F:F2:88:A9:A7:34
└────────|—      DirName : /dnQualifier=RQ/53RmuLsbzgfPXGlRYmJruwMs=/CN=.DMS.DC2.SMPTE/OU=DC.DOREMILABS.COM/O=DC2.SMPTE.DOREMILABS.COM
         |—      serial:02
         +-------------------------------------------------------------------------------------------------------------------------------
```

CHAMP BASIC CONTRAINT & SES OPTIONS :

BasicConstraint est en lien direct avec le chapitre sur la chaîne de certificats. **BasicContraint** a deux paramètres : **CA** (boolean) et **PathLen** (unsigned integer). **BasicConstraint** permet d'indiquer clairement si ce certificat fait partie de la chaîne de certification (**Certificate Authority / CA**) et donc s'il est autorisé à signer d'autres certificats (ses enfants directs)

Avec **PathLen**, vous spécifiez la profondeur possible des certificats en dessous de lui et pouvant signer d'autres certificats. (*« certificate signing certificates »*). Par exemple, si nous avons un

PathLen à 5, nous savons qu'il est possible d'avoir une hiérarchie de 5 niveaux de certificats en dessous. Ainsi, le seul qui n'aura aucun PathLen (ou à 0) sera un certificat Leaf, utilisé comme certificat machine (celui pour un player DCI par exemple) :

```
┌──────────┐
│ PathLen=5 │   (Certificat Racine)
│  CA:TRUE  │
└──────────┘
     │
     ▼
┌──────────┐
│ PathLen=4 │   (Certificat Intermédiaire)
│  CA:TRUE  │
└──────────┘
     │
     ▼
┌──────────┐
│ PathLen=3 │   (Certificat Intermédiaire)
│  CA:TRUE  │
└──────────┘
     │
     ▼
┌──────────┐
│ PathLen=2 │   (Certificat Intermédiaire)
│  CA:TRUE  │
└──────────┘
     │
     ▼
┌──────────┐
│ PathLen=1 │   (Certificat Intermédiaire)
│  CA:TRUE  │
└──────────┘
     │
     ▼
┌──────────┐
│ PathLen=0 │   (Certificat Leaf)
│  CA:FALSE │   (Exemple: Certificat Player)
└──────────┘
```

Voici les différences entre deux types de certificats :

Exemple d'un BasicConstraint venant d'un certificat d'autorité	Exemple d'un BasicConstraint venant d'un certificat leaf, type player
`X509v3 Basic Constraints: critical` `CA:TRUE, pathlen:240`	`X509v3 Basic Constraints: critical` `CA:FALSE`

Deux exemples de paramètres dans **BasicConstraint** :

- Un certificat faisant partie de l'autorité de certification et pouvant donc signer d'autres certificats :
  - un `CA=True`
  - un `PathLen` égale ou supérieur à 0
- Un certificat leaf (playeur ou encodeur) :
  - un `CA=False`
  - un `PathLen=0` ou absent, la plupart du temps

Voici un tableau récapitulatif des paramètres par type de certificat :

	Basic Contraints		
Type de certificat	**Cert Authority**	**Path Length**	**Exemple simple**
Root	`CA = TRUE`	`PathLen >= 0`	`CA:TRUE, PathLen:2`
Intermediate	`CA = TRUE`	`PathLen >= 0`	`CA:TRUE, PathLen:1`
Leaf	`CA = FALSE`	`PathLen == 0`	`CA:FALSE, PathLen:0`

Vous remarquerez que chaque **PathLen** diminue en descendant dans la hiérarchie des certificats. Le numéro du **PathLen** suivant n'est pas forcément un nombre décrémenté par 1. On peut arriver avec un **PathLen** d'un **certificat racine** à 100 et son **certificat intermédiaire** suivant (enfant) à 50, et encore l'intermédiaire suivant à 42 ou 37 ou tout autre (à l'exception de 0).

Si un certificat parent possède un `CA=True` et `PathLen=0`, alors le certificat enfant sera un certificat leaf.

Dans un **certificat leaf**, le **PathLen** sera toujours à 0 (ou absent) car il ne peut aller plus bas (doh! :). Et surtout que son flag CA sera à False : il ne peut signer d'autres certificats en dessous de lui-même.

QUELQUES EXEMPLES CONCRETS

Un **BasicConstraint** d'un certificat faisant partie de l'autorité de certification. Ici, un certificat racine :

```
X509v3 Basic Constraints: critical
    CA:TRUE, pathlen:240
```

Un certificat faisant également partie de l'autorité de certification. Ici, un certificat intermédiaire - enfant du précédent certificat :

```
X509v3 Basic Constraints: critical
    CA:TRUE, pathlen:239
```

Et enfin, un certificat en bout de chaîne. Ici, un certificat leaf, soit un player ou soit encodeur :

```
X509v3 Basic Constraints: critical
    CA:FALSE
```

Un exemple, si nous prenons une suite de chaîne de certificats et que nous n'en sortons que leur **BasicConstraint** :

Type de certificat	CineCert	Doremi
Root	`CA:TRUE, pathlen:5`	`CA:TRUE, pathlen:240`

Intermediate	`CA:TRUE, pathlen:4`
	`CA:TRUE, pathlen:0`
		`CA:TRUE, pathlen:239`
		`CA:TRUE, pathlen:238`
		`CA:TRUE, pathlen:237`

Encodeur	`CA:FALSE`	`CA:FALSE`

Vous remarquez dans le constructeur CineCert a des certificats intermédiaires passant d'un **PathLen** de 4 à 0. Il n'y a aucune obligation à commencer à un chiffre en particulier, le constructeur Doremi a décidé de commencer sa chaîne de certificat à 240.

MARQUE CRITICAL ?

Si vous aviez remarqué, **BasicConstraints** est marqué d'un ***critical***. Quasiment l'ensemble des certificats x509 DCI possède cette marque critique. Il existe quelques certificats chez certains constructeurs n'ayant pas cette marque. Mais pourquoi ?

Dans la norme SMPTE 430-2 - Digital Certificate, il ne semble n'y avoir aucune indication d'obligation explicite à propos de cette marque critique sur le champ **BasicConstraints**. Cependant, il est indiqué dans les règles de validation que si une extension inconnue est marquée *critical*, le certificat doit être rejeté.

Pour en savoir plus, nous devons nous reporter sur le DCI - Compliance Test Plan : il est clairement indiqué que pour les certificats de l'autorité (CA:TRUE), **BasicConstraints doit être marqué critique** [5]. Pour les autres (donc les leafs), **BasicContraints** peut être marqué critique, ce n'est donc pas une obligation. Tout comme les **Key Usage** et **Authority Key Identifier** peuvent être marqués *critique*, sans aucune obligation.

CHAMP KEYUSAGE & SES OPTIONS :

KeyUsage est une collection de bits de drapeau (traduction littérale de `flag bits`) qui identifient toutes les opérations autorisées à être effectuées avec la clé publique de ce certificat (et donc par lien logique, ce qui peut être fait également avec la clef privée correspondante à cette clef publique).

Vous retrouverez toute la collection des options possibles dans KeyUsage dans la RFC 3280 - KeyUsage Cependant, nous nous attarderons seulement sur ceux utilisés et rencontrés dans notre environnement DCI :

Nom du drapeau	Description d'origine de la norme
keyCertSign (Certificate Sign)	Subject public key is used to verify signatures on certificates. *« The keyCertSign bit is asserted when the subject public key is used for verifying a signature on public key certificates. If the keyCertSign bit is asserted, then the cA bit in the b*
cRLSign (CRL Sign)	Subject public key is to verify signatures on revocation information, such as a CRL. *« The cRLSign bit is asserted when the subject public key is used for verifying a signature on certificate revocation list (e.g., a CRL, delta CRL, or an ARL). This bit MUST be asse*
digitalSignature	Certificate may be used to apply a digital signature. Digital signatures are often used for entity authentication & data origin authentication with integrity. *« The digitalSignature bit is asserted when the subject public key is used with a digital signature mechanism to support security services other than certificate signing (bit 5), or*
keyEncipherment	Certificate may be used to encrypt a symmetric key which is then transferred to the target. Target decrypts key, subsequently using it to encrypt & decrypt d *« The keyEncipherment bit is asserted when the subject public key is used for key transport. For example, when an RSA key is to be used for key management, then this bit is set*
(**dataEncipherment**)	Certificate may be used to encrypt & decrypt actual application data. *« The dataEncipherment bit is asserted when the subject public key is used for enciphering user data, other than cryptographic keys »*

Les drapeaux sont liés aux usages du certificat et sont dépendants des informations contenues dans le champ **BasicConstraint**.

Ainsi, si un certificat est considéré comme **Certificate Authority** (`CA=True`), alors les drapeaux sont activés de la sorte :

- Le drapeau **KeyCertSign** doit être à **True**
- Le drapeau **cRLSign** peut être à **True** (facultatif)

Si un certificat n'est **pas** considéré comme **Certificate Authority** (`CA=False`), alors les drapeaux sont :

- Le drapeau **KeyCertSign** doit être à **False** (ou manquant, préférable)
- Le drapeau **cRLSign** doit être à **False** (ou manquant, préférable)
- Le drapeau **DigitalSignature** doit être à **True**
- Le drapeau **KeyEncipherment** doit être à **True**.

Voici un récapitulatif des différents KeyUsages possibles par type de certificat :

Type de certificat	X509v3 Key Usage
Root Certificate	KeyCertSign, (cRLSign)
Intermediate	KeyCertSign, (cRLSign)
Leaf	DigitalSignature, KeyEncipherment, (Data Encipherment)

Et pour Data Encipherment ?

Certains certificats ont un **dataEncipherment**. Ce drapeau spécifique n'est pas indiqué dans la norme. Cette dernière indique seulement que d'autres drapeaux en dehors de ceux que nous avons déjà vus (KeyCertSign, cRLSign, Digital Signature et KeyEncipherment) peuvent être à True. Vous avez donc la liberté de l'activer ou non.

Le drapeau **dataEncipherment** est probablement utilisé par les constructeurs afin d'indiquer que ce certificat peut (dé)chiffrer des données propres aux constructeurs, comme des paquetages de mise à jour ou encore gérer des TLS. Tout ceci est en dehors du scope de notre documentation.

BLOC SIGNATURE

La Signature se trouve en toute fin du certificat. Elle doit respecter le format PKCS#1 v1.5.

```
Signature Algorithm: sha256WithRSAEncryption
Signature Value:
    3c:1a:59:e9:39:20:f5:36:60:d8:b6:a6:2d:a3:82:3c:4e:a2:
    9e:00:6f:aa:6b:10:1d:1e:55:e1:bf:7d:b0:d5:3f:12:f7:46:
    2f:67:89:8d:91:70:e1:82:ec:c5:a1:26:3e:9a:48:18:57:4c:
    20:4f:90:24:7a:c4:0f:96:0b:68:9f:62:d5:5b:d3:6c:3d:63:
    35:fa:dc:95:6e:12:6d:ce:be:9a:54:0a:14:2e:af:38:3e:7f:
    82:8d:e1:2d:80:bc:03:9c:3d:d9:e6:bb:e6:25:fb:ed:2b:83:
    d0:30:83:d4:62:2c:e8:52:f7:10:64:ab:31:70:b9:f4:71:4a:
    e1:a8:67:22:4c:5c:53:28:55:ef:90:a7:d7:8b:a3:68:e4:29:
    88:ef:b3:07:1a:ff:55:a8:bf:3c:36:68:cb:a2:92:9b:4c:98:
    24:48:7d:ee:ae:3e:bd:06:10:95:15:92:3f:57:05:f4:88:cb:
    ba:ca:d8:05:38:d4:df:47:fb:af:28:0e:47:8b:dc:a8:bf:31:
    9c:d4:21:62:9a:c1:94:90:67:f8:a4:b7:16:a3:4a:b6:b5:28:
    1d:31:74:62:d2:e5:e0:8e:65:f7:4a:1c:b6:5f:af:b5:03:46:
    5b:1a:b8:c5:4d:f7:0e:ef:6a:5c:e0:57:04:4f:61:79:27:c6:
    dc:2b:26:57
```

La **Signature** se trouve en dehors de la portion signée du certificat qui se trouve être la partie haute du certificat (vert + jaune/orange + bleu) - Cette portion est appelée **tbsCertificate** [6] (**To-Be-Signed Certificate**). En d'autres termes, elle inclue tout du certificat sauf la signature de fin - bien entendu.

Pour en savoir à propos du **tbsCertificate**, vous pouvez vous reporter au chapitre Certificate Thumbprint.

On va utiliser la partie **tbsCertificat** du certificat pour créer cette signature à l'aide d'une signature RSA et d'un hash **SHA-256**.

Nom du champ x509	Paramètres
Signature Algorithm	Identifiant normé dans la RFC4055 indiquant l'utilisation de **sha256WithRSAEncryption**
Signature Value	Empreinte SHA-256 du corps du certificat (tbsCertificate) et signé avec RSA (PKCS#1 1.5)

La seule entitée pouvant créer cette signature est celle du parent (Issuer). Pour créer cette signature, le parent va utiliser sa clef privée et signer le certificat. Pour valider la signature, il suffit d'utiliser la clef publique du parent (Issuer).

CONCLUSION

Maintenant que nous avons vu l'ensemble des champs d'un certificat x509 DCI, nous pouvons dès à présent étudier deux en particuliers - **Issuer** et **Subject** (qui ont un impact sur d'autres champs comme **DirName** et leurs utilisations dans différents endroits d'un workflow DCI - que ce soit PKL, CPL et KDM).

Vous l'avez peut-être remarqué, ils ont une syntaxe très particulière, on dirait une incantation voodo :

```
Subject : O = DC2.SMPTE.DOREMILABS.COM, OU = DC.DOREMILABS.COM, CN = CS.DMSJP2K-80119.DC.DC2.SMPTE, dnQualifier = SQFYSSqWwjefppqasMJmfdmS6lI=
Issuer  : O = DC2.SMPTE.DOREMILABS.COM, OU = DC.DOREMILABS.COM, CN = .DC.DMS.DC2.SMPTE, dnQualifier = "+LLvuYNO4YBJSp9Jjmlv8oippzQ="
```

Nous allons étudier la syntaxe de ces attributs et leurs rôles dans un worklow DCI Certificats - Identity Attributes - les attributs et leurs rôles pour démystifier tout ceci.

CHAPITRES ANNEXES

- Certificats - Les bases
- Certificats - Les champs (fields) d'un certificat x509 DCI ⬅ vous êtes ici
- Certificats - Identity Attributes - les attributs et leurs rôles
- Certificats - La chaîne de certification
- Certificats - Certificate Thumbprint - l'empreinte du certificat
- Certificats - Public Key Thumbprint (dnQualifier) - l'empreinte de la clef public
- Certificats - Création : Nos propres certificats
- RSA - La cryptographie asymétrique pour le chiffrement et les signatures.

NOTES

1. Pourquoi 65537 ? C'est un choix stratégique pour les calculs dans de la cryptographie asymétrique : c'est un nombre premier Fermat ($2^{16} + 1$); Que 65537 est un bon compromis parmi d'autres nombres premiers Fermat et que sa représentation binaire (10000000000000001) fonctionne bien avec les ordinateurs. Pour en savoir plus : ↩

   - Le chapitre RSA
   - En complément les pages suivantes:
     - Wikipedia - 65.537
     - Quora - Why is e=65537 used for most RSA encryption - Jeffrey Goldberg
     - Quora - Why is e=65537 used for most RSA encryption - Geoffrey Falk
     - John Cook - RSA Exponent
     - Cryptobook - The RSA Cryptosystem - Concepts
     - Doctrina - How RSA works with examples

2. Si ce n'est pas très clair : Selon l'ObjectID utilisé dans **Public Key Algorithm**, vous aurez un containeur différent. Pour notre cas (rsaEncryption), la RFC indique que notre structure sera de la sorte : ↩

   ```
   SEQUENCE {
     SEQUENCE {
       OBJECT IDENTIFIER
         rsaEncryption (1 2 840 113549 1 1 1)
       NULL
       }
     BIT STRING 0 unused bits, encapsulates {
         SEQUENCE {
           INTEGER
             00 E1 6A E4 03 30 97 02 3C F4 10 F3 B5 1E    \
             4D 7F 14 7B F6 F5 D0 78 E9 A4 8A F0 A3 75    |
             EC ED B6 56 96 7F 88 99 85 9A F2 3E 68 77    |
             87 EB 9E D1 9F C0 B4 17 DC AB 89 23 A4 1D    |<---- notre modulus RSA (clef publique)
             7E 16 23 4C 4F A8 4D F5 31 B8 7C AA E3 1A    |
             49 09 F4 4B 26 DB 27 67 30 82 12 01 4A E9    |
             1A B6 C1 0C 53 8B 6C FC 2F 7A 43 EC 33 36    |
             7E 32 B2 7B D5 AA CF 01 14 C6 12 EC 13 F2    |
             2D 14 7A 8B 21 58 14 13 4C 46 A3 9A F2 16    |
             95 FF 23                                     /
           INTEGER 65537        <------------------------------ notre exponent RSA
           }
         }
     }
   ```

   Juste après notre **OBJECT IDENTIFIER rsaEncryption**, nous avons un simple BIT STRING avec deux **INTEGERS** qui vont intégrer respectivement la clef publique RSA et l'exponent RSA. C'est aussi simple que cela.

   Avec d'autres algorithmes, la structure en dessous de notre **OBJECT IDENTIFIER** sera différent et dépendant des données utiles pour l'algorithme cryptographique utilisé.

3. A propos du **AuthorityCertIssuer** (DirName), la doc SMPTE n'est pas très explicite à première vue, elle indique l'issuer de l'issuer. *« They name the issuer of the issuer's certificate -- SMPTE 430-2 - Digital Certificate »*. On pense naïement qu'on demande le **Subject** du parent du parent (en gros, le grand-parent), En fait, c'est juste c'est juste le champ "**Issuer**" du parent direct. Il aurait été préférable de l'écrire "Issuer field of the issuer's certificate" pour éviter toute confusion. ↩

4. Se reporter à deux passages : ↩

   - *« For details on computing this value see [RFC3280] Section 4.2.1.2 option 1. »* -- SMPTE 430-2 - Digital Certificate,
   - *« 4.2.1.2 Subject Key Identifier : (..) for generating key identifiers from the public key are: (1) The keyIdentifier is composed of the 160-bit SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits) »* -- RFC 3280 Internet X.509 Public Key Infrastructure Certificate

5. Deux passages dans le CTP : ↩

   - *« A CA certificate that does not have a non-negative path length of zero or greater, or that does not have the basic constraints extension marked critical and containing CA:TRUE, shall be cause to fail this test. »* -- CTP
   - *« For signer certificates (certificates that have CA:TRUE), of the X.509v3 extensions listed in the certificate, "Basic Constraints" (indicated by 19) must be marked critical. "Basic Constraints" may be marked critical for leaf certificates. "Key Usage" and "Authority Key Identifier" (indicated by 17) may be marked critical. No other unrecognized X.509v3 extensions may be marked critical. »* -- CTP - 2.1.15. Unrecognized Extensions

6. Le tbsCertificate (To-Be-Signed Certificate) est l'ensemble du certificat, sauf la partie de fin : **Signature Algorithm** et **Signature Value**, tout simplement :) ↩

   En voyant sa définition normée :

```
# Définition d'un certificat x509 :
Certificate  ::=  SEQUENCE  {
    tbsCertificate          TBSCertificate,   --------------+
    signatureAlgorithm      AlgorithmIdentifier,            |
    signatureValue          BIT STRING                      |
}                                                           |
                                                            |
# Définition de TBSCertificate :                            |
TBSCertificate ::= SEQUENCE {   <---------------------------+
    version                 Version DEFAULT v1(0),
    serialNumber            CertificateSerialNumber,
    signature               AlgorithmIdentifier,
    issuer                  Name,
    validity                Validity,
    subject                 Name,
    subjectPublicKeyInfo    SubjectPublicKeyInfo,
    issuerUniqueID          IMPLICIT UniqueIdentifier OPTIONAL,
    subjectUniqueID         IMPLICIT UniqueIdentifier OPTIONAL,
    extensions              Extensions OPTIONAL
}
```