CRYPTOGRAPHIE : L'ALGORITHME ASYMÉTRIQUE RSA

PRÉFACE

La cryptographie RSA est une cryptographie dite asymétrie : elle nécessite deux clefs : une clef publique et une clef privée qui effectuent chacune une action particulière opposée à l'autre.

LA CLEF PRIVÉE

La clef privée servira pour le déchiffrement des données ou la signature d'un document. Elle ne doit **PAS** être communiquée, elle doit rester entre les mains du propriétaire, elle permettra de déchiffrer tout cryptogramme chiffré par la clef publique ou de créer la signature d'un document.

Dans votre playeur DCP ou encodeur DCP, cette clef privée ne vous sera jamais communiqué (officiellement... `*touss*`), elle doit être cachée dans les entrailles de la machine et ne doit pas pouvoir être extraite. Dans un playeur DCP, elle va se retrouver dans la partie **Hardware Security Module** (HSM) et la seule méthode d'avoir une interaction avec elle sera de passer par une API interne qui va se charger d'effectuer la manipulation de la clef privée à la place de vous ou du système (par exemple le programme de lecture d'un DCP). Voyez le HSM comme un proxy : vous lui donnez de la donnée et lui demandez de déchiffrer ce bloc de données. En interne, tout seul dans son coin, le HSM va lire vos données, il va utiliser la clef privée (qui se trouve de son côté et inaccessible par personne d'autre), il va chiffrer (ou déchiffrer) les données, puis il va vous renvoyer un bloc de données chiffrés (ou déchiffrés).

LA CLEF PUBLIQUE

La clef publique servira pour le chiffrement des données ou la vérification de la signature d'un document. Elle peut être donnée à n'importe qui souhaitant communiquer avec la personne possédant la clef privée ou vérifier la signature électronique d'un document signé par la clef privée.

Dans notre playeur DCP ou encodeur DCP, cette clef publique est accessible à tous. Elle n'a pas vocation à être cachée. C'est cette clef qui sera échangée avec les laboratoires afin d'avoir un KDM pour **son** playeur.

LE LIEN ENTRE CLEF PRIVÉE ET CLEF PUBLIQUE

A titre de comparaison imagée, c'est comme si vous aviez une porte avec une serrure magique : une clef peut fermer la porte mais ne jamais pouvoir l'ouvrir et l'autre clef, pouvoir ouvrir la porte mais ne jamais pouvoir la fermer. C'est le même principe avec clef privée et clef publique. Si vous voulez ouvrir et fermer cette porte, il vous faudra les deux.

LES DIFFÉRENTS TYPES DE FICHIERS RSA

Type du fichier	Entête	Description interne	Structure interne ASN.1

KeyPair `PKCS#1`	`BEGIN RSA PRIVATE KEY`	Données brutes RSA intégrant private et public
		
		Format ASN.1
		Format PKCS#1
		Wrapping Base64
			`-----BEGIN RSA PRIVATE KEY-----`
			`RSAPrivateKey ::= SEQUENCE {`
			`  version  Version,`
			`  modulus  INTEGER,  -- n`
			`  publicExponent  INTEGER,  -- e`
			`  privateExponent  INTEGER,  -- d`
			`  prime1  INTEGER,  -- p`
			`  prime2  INTEGER,  -- q`
			`  exponent1  INTEGER,  -- d mod (p-1)`
			`  exponent2  INTEGER,  -- d mod (q-1)`
			`  coefficient  INTEGER,  -- (inverse of q) mod p`
			`  otherPrimeInfos  OtherPrimeInfos OPTIONAL`
			`}`
			`-----END RSA PRIVATE KEY-----`

Public Key `PKCS#1`	`BEGIN RSA PUBLIC KEY`	Données brutes RSA.
		
		Format ASN.1
		Format PKCS#1
		Wrapping Base64
			`-----BEGIN RSA PUBLIC KEY-----`
			`RSAPublicKey ::= SEQUENCE {`
			`  modulus  INTEGER,  -- n`
			`  publicExponent  INTEGER  -- e`
			`}`
			`-----END RSA PUBLIC KEY-----`

Public Certificate	`BEGIN CERTIFICATE`	**Public Key** + Metadatas Format ASN.1 Wrapping Base64	Voir Certificates

Les fichiers PKCS#1 débuteront toujours par un entête **BEGIN RSA PRIVATE KEY** ou **BEGIN RSA PUBLIC KEY** [1] :

- **KeyPair** intègre les éléments privées **ET** les éléments publiques. À partir de ses données, nous pouvons générer une clef publique **Public Key** en extrayant simplement le **modulus** et **publicExponent**.
- **PublicKey** intègre les données publiques uniquement (modulus + publicExponent): à l'aide de la **Public Key**

Enfin, le fichier **Public Certificate** est le certificat de la machine (celui de notre playeur DCP ou de notre encodeur DCP), il va servir lors de la création de nos KDM.

Tous ces éléments vont nous servir pour nos certificats.

ASSETS

Vous pouvez retrouver des exemples concrets dans la partie assets :

- Fichier KeyPair (pub/priv) PKCS#8
- Fichier Public Key PKCS#8
- Fichier Public Certificate #PKCS1

CHAPITRES CONNEXES

- **La cryptographie** : Pour tout savoir sur la cryptographie utilisée dans le cinéma numérique.
- **Certificats** : Les certificats utilisés dans le cinéma numérique, leurs vies, leurs oeuvres.
- **Cryptographie AES** : La cryptographie symétrique utilisée pour chiffrer les MXF.
- **Cryptographie RSA** : La cryptographie asymétrique utilisée dans les KDM pour chiffrer les clefs AES - entre autres.

NOTES

1. On peut parfois voir des `BEGIN PRIVATE KEY` ou `BEGIN ENCRYPTED PRIVATE KEY`. Le premier est le même qu'un `BEGIN RSA PRIVATE KEY` au format `PKCS#1`, et le second est un `BEGIN RSA PRIVATE KEY` mais chiffré et au format `PKCS#8`. ↩

   La grande différence entre `PKCS#1` et `PKCS#8` est que le premier est pour l'algorithme RSA, le second est pour une plus grande possibilité de type de clef (pas que RSA). En interne, le `PKCS#8` avec du RSA est une sorte de containeur avec une structure `PKCS#1` RSA (en très gros résumé):

   Pour générer une keypair RSA au format `PKCS#1` :

   ```
   openssl genrsa -traditional -out root.pkcs1.key 2048
   ```

   Pour générer une keypair RSA au format `PKCS#8` :

   ```
   openssl genrsa -out root.pkcs8a.key 2048
   ```

   Pour générer la clef publique RSA au format `PKCS#1` :

   ```
   openssl rsa -in root.key -pubout -RSAPublicKey_out -out root.pkcs1.pub
   ```

   Pour générer la clef publique RSA au format `PKCS#8` :

   ```
   openssl rsa -in root.key -pubout -out root.pkcs8.pub
   ```

   La différence interne entre `PKCS#1` et `PKCS#8`, vous le verrez assez rapidement :

```
# root.pkcs1.pub

00000000: 3082 010a 0282 0101 00dd 433b ccc7 4661  0.........C;..Fa
00000010: 114d 655a 6296 3119 4a7c c27a 2acf 3a56  .MeZb.1.J|.z*.:V
00000020: f4e7 1798 5dcd 0163 f36e d16d cf7f 0736  ....]..c.n.m...6
00000030: 3cdc 7889 794b 08cd a9cd db94 921e 0ad4  <.x.yK..........
00000040: 963b b668 dc59 ecc8 a1d9 b7ed 22d4 a27b  .;.h.Y......"..{
00000050: 03fb cfc8 7133 9f78 b76f c736 a675 b021  ....q3.x.o.6.u.!
00000060: d8ad a78a cef1 599c 9be9 7d07 1c70 95bd  ......Y...}..p..
00000070: a702 3a7d d1da 6976 ce53 6995 7e5f 79bb  ..:}..iv.Si.~_y.
00000080: 2fba da7c c256 1130 d90d bd86 ecb1 ff96  /..|.V.0........
00000090: 1129 ba15 4ad6 8b6f ce71 2271 4c84 b68d  .)..J..o.q"qL...
000000a0: 3c8d 857c df56 b624 f91f 37fb 384f da2d  <..|.V.$..7.8O.-
000000b0: e7c6 a542 6227 d970 8a87 c986 48b6 e3be  ...Bb'.p....H...
000000c0: 2b24 8328 ec35 7137 582d 6b09 68cf 4272  +$.(.5q7X-k.h.Br
000000d0: 9ee8 4198 ea66 3bb0 52ca 8c44 f7f1 75e0  ..A..f;.R..D..u.
000000e0: 4123 2a0e 3674 c7c4 6d85 4df0 119a f2d2  A#*.6t..m.M.....
000000f0: a223 ad04 ea25 8d5b 0a8c 464e 3337 b469  .#...%.[..FN37.i
00000100: 24b3 0487 0d44 5a7b 5302 0301 0001       $....DZ{S.....
```

```
# root.pkcs8.pub

00000000: 3082 0122 300d 0609 2a86 4886 f70d 0101  0.."0...*.H.....
00000010: 0105 0003 8201 0f00 3082 010a 0282 0101  ........0.......
00000020: 00dd 433b ccc7 4661 114d 655a 6296 3119  ..C;..Fa.MeZb.1.
00000030: 4a7c c27a 2acf 3a56 f4e7 1798 5dcd 0163  J|.z*.:V....]..c
00000040: f36e d16d cf7f 0736 3cdc 7889 794b 08cd  .n.m...6<.x.yK..
00000050: a9cd db94 921e 0ad4 963b b668 dc59 ecc8  .........;.h.Y..
00000060: a1d9 b7ed 22d4 a27b 03fb cfc8 7133 9f78  ....".."{....q3.x
00000070: b76f c736 a675 b021 d8ad a78a cef1 599c  .o.6.u.!......Y.
00000080: 9be9 7d07 1c70 95bd a702 3a7d d1da 6976  ..}..p....:}..iv
00000090: ce53 6995 7e5f 79bb 2fba da7c c256 1130  .Si.~_y./..|.V.0
000000a0: d90d bd86 ecb1 ff96 1129 ba15 4ad6 8b6f  .........)..J..o
000000b0: ce71 2271 4c84 b68d 3c8d 857c df56 b624  .q"qL...<..|.V.$
000000c0: f91f 37fb 384f da2d e7c6 a542 6227 d970  ..7.8O.-...Bb'.p
000000d0: 8a87 c986 48b6 e3be 2b24 8328 ec35 7137  ....H...+$.(.5q7
000000e0: 582d 6b09 68cf 4272 9ee8 4198 ea66 3bb0  X-k.h.Br..A..f;.
000000f0: 52ca 8c44 f7f1 75e0 4123 2a0e 3674 c7c4  R..D..u.A#*.6t..
00000100: 6d85 4df0 119a f2d2 a223 ad04 ea25 8d5b  m.M......#...%.[
00000110: 0a8c 464e 3337 b469 24b3 0487 0d44 5a7b  ..FN37.i$....DZ{
00000120: 5302 0301 0001                           S.....
```

Et oui, pour intégrer la clef publique RSA du `PKCS#1` dans un `PKCS#8`, ils ont juste rajouté un header spécifique et le reste de la structure `PKCS#1` :)

Voyez la structure `PKCS#1` d'une clef publique RSA :

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

Et la structure `PKCS#8` d'une clef publique RSA :

```
PublicKeyInfo ::= SEQUENCE {
        algorithm       AlgorithmIdentifier,
        PublicKey       BIT STRING
}
AlgorithmIdentifier ::= SEQUENCE {
        algorithm       OBJECT IDENTIFIER,
        parameters      ANY DEFINED BY algorithm OPTIONAL
}
PublicKey ::= SEQUENCE {
        modulus           INTEGER,  -- n
        publicExponent    INTEGER   -- e
}
```

C'est comme mettre une boite dans une boite ;-)

Voici un récapitulatif :

Entête	Format	Contenu	Notes
`BEGIN RSA PRIVATE KEY`	PKCS#1	Pub + Priv	RSA seulement
`BEGIN RSA PUBLIC KEY`	PKCS#1	Modulus + Exposant public	RSA seulement
`BEGIN PRIVATE KEY`	PKCS#8	Pub + Priv	Tout type de clef
`BEGIN PUBLIC KEY`	PKCS#8	Public Key	Tout type de clef
`BEGIN ENCRYPTED PRIVATE KEY`	PKCS#8	(Public Key)	PRIVATE KEY avec chiffrement symétrique

Un excellent article est disponible à cette addresse