Niveau : 3 **Vert**

Forme de la biclé RSA générée par OpenSSL

1 Introduction, notations

Nous allons examiner les informations concernant la biclé RSA qui figurent dans le fichier généré par OpenSSL lorsqu'on demande la création d'une telle biclé. Nous remarquerons que les données qui sont stockées correspondent à la méthode d'utilisation décrite par RSALab dans sa norme **PKCS#1**.

2 Obtention du fichier

On construit une biclé RSA de module 1024 bits qui va se présenter dans un fichier au format PEM (Privacy Enhanced Mail), sans la protéger, c'est à dire sans la chiffrer.

```
$ openssl genrsa -out essai.pem 1024
```

On extrait les informations de ce fichier (essai.pem). Ces informations vont se retrouver dans le fichier texte essai.txt.

```
$ openssl rsa -text -in essai.pem -out essai.txt -noout
```

Voici le contenu de ce fichier texte :

```
Private-Key: (1024 bit)
modulus:
    00:ed:2c:b7:58:6a:89:04:91:f8:95:14:c5:17:3b:
    78:f5:83:78:f0:c4:8d:3d:3b:55:d6:92:a7:3f:30:
    a6:88:c7:1d:fe:30:43:d5:49:51:2a:cd:c3:db:d9:
    59:46:f6:a8:30:89:a2:e4:21:e2:53:3c:40:63:92:
    70:a2:46:0e:3c:b9:0c:eb:5b:f1:12:39:6e:5a:b8:
    e9:b1:c1:85:aa:3f:9e:3e:52:0e:81:0c:7d:4a:8a:
    d2:38:09:39:b1:57:11:4a:f5:14:a5:90:75:b5:ee:
    91:cc:b8:64:bd:ff:f9:ce:4e:76:fd:64:28:59:63:
    17:88:95:43:a9:a3:95:41:3d
publicExponent: 65537 (0x10001)
privateExponent:
    00:d0:a2:bc:6a:7f:d5:26:7b:0e:14:7e:df:dd:cf:
    08:59:d9:1c:a3:98:13:b2:e0:a6:63:0a:41:2b:9a:
    2d:75:36:cb:d0:96:3e:a5:ba:48:37:82:28:a0:16:
    33:ad:f4:3f:01:eb:a1:aa:53:90:57:ae:31:ea:25:
    a1:05:6e:e1:07:a2:26:db:14:1d:d5:69:3d:99:ee:
    33:73:bf:b6:51:52:05:a4:95:1b:b5:ba:09:c0:68:
```

```
    af:8a:92:11:fe:f6:c5:72:b6:21:98:01:14:8c:7f:
    bd:50:10:81:9a:3b:22:41:ba:77:c5:ab:f2:98:14:
    60:55:f9:c2:36:bf:e3:b9:61
prime1:
    00:f9:96:63:5d:60:00:d1:71:9d:5b:1f:6c:79:d8:
    34:ec:0d:d8:42:57:8b:0a:9a:63:93:f4:b4:55:1c:
    a8:0a:9d:9e:2b:f3:c4:ff:f7:c8:af:a7:5a:0d:a6:
    df:c2:10:04:fe:be:ef:b6:53:0a:9a:1c:11:b6:18:
    5b:9e:51:ef:b5
prime2:
    00:f3:44:af:79:a1:a5:82:a2:7d:6e:c8:42:61:2d:
    dc:2a:cc:e1:55:e7:92:6e:13:ca:4e:c8:81:58:ed:
    ee:63:e1:f2:d0:7d:4c:07:dd:1e:b0:89:3a:6b:77:
    41:9a:9b:54:fe:4f:0e:90:5e:87:42:79:0d:c5:f5:
    a3:8f:e8:30:69
exponent1:
    7e:e1:f2:4f:d4:ef:75:8a:81:c9:82:57:1e:36:48:
    e9:3c:3c:95:b5:75:8d:05:61:dc:24:c9:cc:7f:0d:
    fa:9f:98:7a:95:a9:af:cb:22:ee:11:70:d9:81:dd:
    3d:05:f1:d4:23:f3:2f:48:56:1a:74:6c:98:9b:17:
    70:8c:0d:05
exponent2:
    18:72:5c:98:02:90:99:72:2e:dc:c5:2f:36:88:df:
    49:45:d1:97:4a:70:42:b3:a6:6f:08:63:47:46:91:
    e0:63:c2:7d:05:3a:70:c7:dd:df:ab:ca:bd:25:fc:
    e7:c6:98:61:fe:1b:de:92:41:51:82:cc:c9:8a:07:
    e7:dc:53:91
coefficient:
    3b:d9:ad:6e:32:a4:29:d1:22:3e:f0:69:c9:32:e7:
    34:53:0f:35:b8:75:24:21:df:1a:14:a4:f6:66:80:
    68:9e:cc:bd:7e:8e:3d:1e:f3:cd:b5:bc:08:c1:1c:
    23:f0:f1:8b:3a:f7:
```

3 Analyse du fichier

La lecture de ce fichier nous apprend qu'on a construit un RSA à partir d'un module de 1024 bits. On trouve ensuite dans l'ordre sous forme hexadécimale sauf pour l'exposant public e :

1. le module n ($n = p \times q$),
2. l'exposant public e (forme décimale),
3. l'exposant privé d,
4. le premier facteur premier p de n,
5. le deuxième facteur premier q de n,
6. le premier exposant $d_1 = d \mod p - 1$,
7. le deuxieme exposant $d_2 = d \mod q - 1$,
8. le coefficient $h = q^{-1} \mod p$.

Remarque 3.1 *En réalité lorsqu'on crée une biclé, on a intérêt à protéger son contenu par un chiffrement symétrique (aes-cbc par exemple) dont la clé secrète est calculée à partir d'une passphrase choisie et retenue par le propriétaire de la biclé. Ceci se fait par la commande :*

```
$ openssl genrsa -aes128 -out essai.pem 1024
```

Et même le plus souvent nous seulement on crée la biclé, mais aussi en même temps une requête de signature par une autorité de certification.

4 Quelques remarques en forme de rappel de la théorie

Ainsi, nous récupérons dans le fichier les nombres n (le module), p (premier nombre premier), q (second nombre premier) : nous savons que $n = p \times q$. L'exposant de chiffrement e est le 4^{e} nombre de Fermat $2^{16} + 1$. L'exposant de déchiffrement (ou de signature) est le nombre d tel que $0 < d < (p-1)(q-1)$ qui vérifie :

$$ed \equiv 1 \quad \big((p-1)(q-1)\big).$$

Il se calcule avec l'algorithme d'Euclide étendu. Enfin, le fichier donne dans l'ordre les nombres $d_1 = d \mod p-1$, $d_2 \mod q-1$ et $h = q^{-1} \mod p$.
Comment utiliser ces données ? Le propriétaire de la clé, qui veut soit déchiffrer soit signer doit calculer une expression du type :

$$S = c^d \mod n.$$

Cela peut se faire directement par l'algorithme « square and multiply », ou par l'algorithme combiné « Montgomery, square and multiply ». Mais on peut améliorer un peu le calcul en procédant de la façon suivante :

1. on calcule $S_1 = c^d \mod p$,
2. on calcule $S_1 = c^d \mod q$,
3. on reconstitue S par le théorème des restes chinois.

Cette méthode est connue sous le nom de **CRT** (comme **C**hinese **R**emainder **T**heorem). Entrons dans les détails. Les deux calculs $S_1 = c^d \mod p$ et $S_1 = c^d \mod q$ se font par un algorithme square and multiply éventuellement amélioré par une méthode du type Montgomery, en tenant compte du fait que :

$$S_1 = c^d \mod p = c^{d_1} \mod p$$
$$S_2 = c^d \mod p = c^{d_2} \mod q.$$

C'est en effet une conséquence directe du théorème de Fermat que j'exprime en disant que si on travaille modulo un nombre premier p au rez-de-chaussée, on travaille modulo $p-1$ au premier étage. Ceci nous permet d'utiliser les exposants d_1 et d_2 au lieu de d, exposants dont la taille est à peu près la moitié de celle de d.
Pour la reconstitution de S à partir de S_1 et S_2, nous partons du théorème de Bezout qui nous permet d'écrire :

$$hq - kp = 1$$

avec $0 < k < q$ et $0 < h < p$. Donc ici, h est l'inverse de q modulo p.
On vérifie tout de suite que :

$$S = hS_1q - kS_2p \mod n.$$

Donc regardons plus en détail l'expression :

$$T = hS_1q - kS_2p.$$

Comme :

$$kp = hq - 1,$$

on peut écrire :

$$T = hS_1q - S_2(hq - 1) = S_2 + h(S_1 - S_2)q.$$

Posons alors :

$$H = h(S_1 - S_2) \mod p.$$

Avec cette notation,

$$S \equiv S_2 + Hq \quad (n).$$

Mais comme $0 \leq H < p$ on a :

$$0 \leq qH \leq q(p - 1).$$

Par ailleurs $0 \leq S_2 < q$. Donc :

$$0 \leq S_2 + Hq < n,$$

ce qui prouve que :

$$S = S_2 + qH.$$

En conclusion :

1. on calcule $S_1 = c^{d_1} \mod p$ (d_1 et p sont dans le fichier).
2. on calcule $S_2 = c^{d_2} \mod q$ (d_2 et q sont dans le fichier).
3. on calcule $H = h(S_1 - S_2) \mod p$ (h et p sont dans le fichier).
4. la signature (ou le texte clair) est $S = S_2 + qH$.

Remarquons aussi l'emploi de la valeur particulière $e = 65537 = 2^{16} + 1$ qui permet un calcul rapide de $m^e \mod n$, c'est à dire d'un chiffré ou d'une vérification de signature.

Cette méthode utilisant la reconstitution du message par le théorème des restes chinois est sensible à des attaques par faute (voir la fichecrypto209). On ne doit pas l'employer telle quelle sur une carte à puce.