# Challenge cryptographique - casser une clé RSA !!!

Contexte :

Supposons que vous travailliez au service de criminologie informatique à la Direction de la Surveillance du Cyber-territoire (DSC). Les services secrets ont intercepté un message crypté (*cipher text*) envoyé par une personne (sous surveillance) à son ami. Le message a été chiffré avec l'algorithme de chiffrement symétrique **AES-256-CBC.** La clé avec laquelle ce message a été chiffré dérive d'un mot de passe envoyé chiffré par l'algorithme de chiffrement asymétrique **RSA-2048 bits**.

Informations importantes :

Vous disposez de 2h pour casser la clé et retrouver le message initial (*plain text*). Selon les premières informations dont vous disposez, l'outil qui a été utilisé pour générer la paire des clés asymétriques était mal configuré. Par conséquent, la qualité des nombres premiers choisis était mauvaise. Le modulo de chiffrement était calculé par la multiplication des deux nombres premiers (un grand *p* et un *q* relativement petit) choisis au hasard à condition que leur produit soit de la taille souhaitée (**2048 bits** dans cet exemple).

Remarque :

Si cette clé RSA avait été correctement générée, il aurait été très difficile de la casser avant des années !

Clé AES chiffrée par la clé publique : en base64

```
EZ9pt97WjJuHCV+2IXHVQohPZVIKNOvSAnb0T/HtG+olfyA/T39deKMNRWqA+z2Y
fctL6Brwn2ky/0yJ5DaFIIS4JCRRkv+ZmiiAlGj/fMLYwN/De4iVYqjicwVdXVt1
2bkpWWOju9FTgjoiwL1jZyCpyN77osPz2TJZjpF/Q/aUY/cKYBBMSNVtncLWoguv
VZwC5hO3Otl+DIzen/hx7Dh+PI2nzNQA9gMmrnUrxDGHXgf1TUb6XGbAZXqfRtte
APRDaxROII3bc/6M7JE49y3hqwCiY9I+atoRZwICKawQPAn5KvooLTzcmWWQ80s6
FmBHXU0DbmvF1BBAhp6U41i+1w==
```

Message chiffré par la clé symétrique (en base64)

```
Q2V0dGUgbnVpdCwgb24gYXR0YXF1ZSBsYSBiYW5xdWUg4CAyM2gwMC4uLg==
```

Clé publique du destinataire (2048 bits)

```
-----BEGIN PUBLIC KEY-----
MIIBJTANBgkqhkiG9w0BAQEFAAOCARIAMIIBDQKCAQQAjaVpGbUm1FIlrO1L5kUi
zvBKY5ELn2/+prESVUEBO+RdSLb7JnG3VA5qTgtV46nkxFqNX1SgaZxlMtShKH+s
sAixxW411gHcKp4uZlGJ6qNdIte+olLB7PJwMatlfVs16CzecPglnS4U6YbzYuPo
bnvY5IEqUvLozC9puLDJWXeP2yQKjhfLlXJFcBLYO2xycpDlC459oo/r36v1I9oD
uUuUMir0IfnKAq3mBNqrks2cKCRENvEV/b7XbSyFAHrKf85JiTr4DnlVYy7HuJ9W
+rAYduD+iCmaNzQNQ5yy4Bs8B+YMiEdCG/4EnVmVQGsm9KCLINFJs8YarqNTHWL4
+NHihwIDAQAB
-----END PUBLIC KEY-----
```

Travail à réaliser :

1. Ecrire un programme en JAVA ou autre langage pour retrouver les nombres premiers (*p* et *q*) afin de reconstituer la clé privée.

2. Ecrire la clé privée sous le format DER qui encode une structure ASN.1. ASN.1 est une norme permettant la représentation de données. La structure ASN.1 d'une clef privée RSA est la suivante :

```
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
}
```

La version vaut toujours 0 et le *publicExponent* est fixé à 65537 (valeur par défaut) pour toutes les clefs générées avec l'outil OpenSSL. Créer un fichier *Private.txt* qui contient :

```
asn1=SEQUENCE:rsa_key
[rsa_key]
version=INTEGER:0
modulus=INTEGER:VALEUR DU MODULO EN INTEGER
pubExp=INTEGER: VALEUR DU e EN INTEGER
privExp=INTEGER: VALEUR DU d EN INTEGER
p=INTEGER: VALEUR DU p EN INTEGER
q=INTEGER: VALEUR DU q EN INTEGER
e1=INTEGER: VALEUR DU d mod (p-1)  EN INTEGER
e2=INTEGER: VALEUR DU d mod (q-1)  EN INTEGER
coeff=INTEGER: VALEUR(inverse of q) mod p EN INTEGER
```

3. Déchiffrer le fichier cipher.txt avec la clé privée et y récupérer le mot de passe.
4. Déchiffrer le Messagecipher.txt avec AES-256-CBC (sans *salt*).

## Annexe :

La commande **asn1parse** est un utilitaire de diagnostic permettant d'interpréter des structures ASN.1. Il peut aussi être utilisé pour extraire des informations de données au format ASN.1.

**OPTIONS**

**-inform** DER|PEM
: Le format d'entrée. **DER** est le format binaire et **PEM** (par défaut) est chiffré base64

**-in nomfichier**
: Le fichier d'entrée, par défaut l'entrée standard (stdin)

**-out nomfichier**
: Le fichier de sortie où les données chiffrées DER sont stockées. Sans cette option, aucune donnée ne sera générée. Ceci est en particulier utile en combinaison avec l'option **-strparse**.

**-noout**
: Ne pas afficher la version interprétée du fichier d'entrée.

**-offset nombre**
: Décalage initial avant le début de l'interprétation, par défaut on commence au début du fichier.

**-length nombre**
: Nombre d'octets à interpréter, par défaut jusqu'à la fin de fichier.

**-i**
: Indentation du fichier de sortie en fonction de la profondeur ``depth'' de la structure

**-oid nomfichier**
: Fichier contenant des identifiants d'objets (OIDs) supplémentaires. Le format utilisé pour ce fichier est décrit dans la section NOTES ci-dessous.

**-strparse décalage**
: Interprète le contenu de l'objet ASN.1 à partir du **décalage**-ième octet. On peut faire appel à cette option plusieurs fois afin de descendre dans une structure imbriquée.