INFO002 : cryptologie
TD 5 : cryptographie asymétrique (chiffrement et signatures)

Pierre Hyvernat
Laboratoire de mathématiques de l'université de Savoie
bâtiment Chablais, bureau 17, poste : 94 22
email : Pierre.Hyvernat@univ-smb.fr
www : http://www.lama.univ-smb.fr/~hyvernat/

Exercice 1 : Cryptographie asymétrique

Question 1. Pourquoi ne peut il y avoir d'équivalent du "secret parfait" en cryptographie asymétrique ?

Indice : l'attaquant dispose de la clé publique de sa cible.

Question 2. Quelle différences y a t'il entre "indistingabilité" et "résistance aux *chosen plaintext attacks*" en cryptographie asymétrique ?

Question 3. Alice et Bob possèdent chacun la clé publique de l'autre. Décrivez une manière de faire un échange de clés de Diffie-Hellman *authentifié*.

Donnez un intérêt à faire un échange de clé de Diffie-Hellman dans cette situation.

Question 4. On considère un système de signature avec aléa : le même message sera signé avec des signatures distinctes (par exemple DSA). Expliquez comment un programme de signature malicieux peut insérer une douzaines de bits d'information dans une signature.

Question 5. En cryptographie symétrique, le chiffrement authentifié se fait en chiffrant le message, puis en authentifiant le message chiffré avec un MAC.

Que pensez-vous de ce mécanisme en cryptographie asymétrique ? Que peut faire Eve à partir du message (c, s) qu'Alice envoie à Bob, où

- c est chiffré avec la clé publique de Bob,
- s est la signature de c, obtenue avec la clé privée d'Alice ?

Quelle solution proposez vous ?

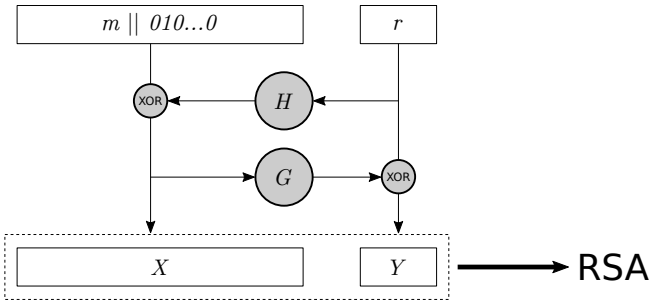
Exercice 2 : RSA

Rappel : pour chiffrer un entier m avec RSA, on calcule $m^e \bmod n$, où (e, n) est la clé publique.

Question 1. Montrez que RSA est *malléable* au sens suivant : si Eve intercepte les messages chiffrés $c = e^m \bmod n$, alors elle peut calculer le chiffrement de $(t \times m) \bmod n$ pour n'importe quel valeur de t.

Question 2. Est-ce que RSA a la propriété d'indistingabilité ? Est-ce que RSA résiste aux "chosen ciphertext attacks" ?

Question 3. Une manière standard d'utiliser RSA est RSA-OAEP ("OAEP" = "Optimal Asymmetric Encryption Padding"). Pour des clés de 2048 bits, les blocs pourraient faire 256 octets, mais on se limite en fait à des blocs plus petits car on ajoute du remplissage :



où F et G sont des fonctions de hachages cryptographique, m est un bloc de message clair et r est tiré aléatoirement pour chaque bloc.

Comment Bob doit-il déchiffrer un message codé avec RSA-OAEP ?

Question 4. Supposez que Eve intercepte un message chiffré par RSA-OAEP et parvient à inverser ("craquer") RSA, sauf sur quelques bits. Elle dispose par exemples de X et Y' où Y' et Y diffèrent sur un bit.

Peut-elle retrouver m?

Question 5. Est-ce que RSA-OAEP a la propriété d'indistingabilité ? À votre avis, est-ce que RSA-OAEP résiste aux "chosen ciphertext attacks" ?

Question 6. On suppose que la signature RSA d'un message m (si m est inférieur au module) est simplement calculée avec $m^d \bmod N$ où d est l'exposant privé, et N le module. La vérification se fait en calculant $s^e \bmod N$, qui doit être égal à m.

Expliquez comment un attaquant peut générer des paires valides (m, s). (Bien sûr, il ne peut pas choisir le message m arbitrairement.)

Exercice 3 : casser une clé RSA ?

Le fichier suivant contient une clé publique RSA factice

```
# public key
# this key will expire on the 2020-11-04 at 13:20:59
M = 57420078365315406704266135667386547401254252500627941423912019985855 4\
228007532444199892968953560241503516566449787246502271080625778138995497 2\
949158327363
e = 65537
```

qui a été générée par un petit programme Python (disponible sur ma page web) :

```
def gen_key():
    # initialize RNG with number of milliseconds since epoch (01-01-1970)
    random.seed(int(1000*datetime.datetime.now().timestamp()))
    n = 512                             # number of bits in modulus
    a = new_prime(n//2)                 # the two prime numbers
    b = new_prime(n//2)                 # ...
    e = 65537                           # public exponent
    d = inverse_mod(e, (a-1)*(b-1))     # private exponent
    # validity of one year
    expire_date = datetime.datetime.now()
    expire_date = expire_date.replace(year=expire_date.year + 1)
    save_public_key(a*b, e, expire_date)
    save_private_key(a, b, d, e, expire_date)
```

Question 1. Expliquez comment retrouver la clé privée.

Question 2. Écrivez un programme pour retrouver la clé privée. (À faire à la maison...)