Sécurité Appliquée : chiffrement asymétrique, TP

Jean-François COUCHOT
couchot[arobase]femto-st[point]fr

2 mars 2022

1 RSA par la pratique

On va utiliser une bibliothèque de cryptographie pour chiffrer avec RSA.

Exercice 1.1 (Exploitation d'un code pour chiffrer un texteselon RSA).

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP

# 1
rsa_obj = RSA.generate(2048)
rsa_pub = rsa_obj.publickey()

#2
f = open('mPubKey.pem','w')
st = rsa_pub.exportKey('PEM')
f.write(st.decode('utf-8'))
f.close()

#3
fp = open('mPrivKey.pem','w')
st = rsa_obj.exportKey('PEM')
fp.write(st.decode('utf-8'))
fp.close()

#4
pubKey = RSA.importKey(open('mPubKey.pem').read())
encryptor = PKCS1_OAEP.new(pubKey)
message_to_encrypt = 'Bravo Laurent ! '
ciphertext = encryptor.encrypt(message_to_encrypt.encode('utf-8'))

#5
fc = open("cipher.txt","wb")
fc.write(ciphertext)
fc.close()

#6
fc = open("cipher.txt","rb")
ciphertext = fc.read()
fc.close()

#7
privKey = RSA.importKey(open('mPrivKey.pem').read())
decryptor = PKCS1_OAEP.new(privKey)
decrypted_message = decryptor.decrypt(ciphertext)

print(decrypted_message.decode('utf-8'))
```

1. *Eventuellement installer le module* `pycryptodome`
2. *Expliquer le programme précédent, ligne par ligne. Expliquer particulièrement quelles instructions sont effectuées par Alice (destinataire) et lesquelles par Bob (expéditeur).*
3. *Générer votre clef publique et votre clef privée.*
4. *Diffuser à un(e) voisin(e) votre clef publique et récupérer la sienne.*
5. *Chiffrer un message avec sa clef publique et transmettez lui le message chiffré.*
6. *Déchiffrer son message.*

2 Implantation de votre propre RSA

Exercice 2.1. *L'objectif de cet exercice est d'implanter les 4 étapes de l'algorithme RSA.*

1. *Développer la fonction* `pgcd(a,b)` *qui retourne le PGCD de* a *et de* b*, pour* a *et* b *entiers naturels* a *strictement positif.*
2. *Développer la fonction* `genereunnombrepremier(k)` *qui génère un nombre pseudo aléatoire sur* k *bits qui est probablement premier. Pour affirmer qu'un nombre de* k *bits est premier, on invoquera le test de Miller-Rabin avec* $k/5$ *(avec un minimum de 25) valeurs testées différentes* a*. Si toutes retournent qu'il est probablement premier, on considérera qu'il l'est.*
3. *A l'aide de l'agorithme précédent, générer deux nombres premiers* p *et* q *de 100 chiffres.*
   Calculer $n = p \times q$ *puis* `phi` *telle que* `phi`$=(p-1)\times(q-1)$.
4. *Contruire la partie* e *de la clé publique* (e, n) *comme un nombre premier de trente chiffres par exemple. Si* e *est premier avec* `phi`*, c'est une bonne clé, sinon on regénere un nombre premier de trente chiffres.*
5. *Donner le code la fonction* `bezout(a,b)` *qui retourne* x *et* y *tels que* $x.a + y.b = pgcd(a, b)$.
   Se servir de cette fonction pour générer la partie d *de la clé privée* (d, n)*. Attention, faire en sorte que* $0 \leq d \leq$`phi`.
6. *Chiffrer à l'aide de la clé publique* (e, n) *le message* $m = 3402752281514000322845$ *qui est un numéro de carte bancaire comprenant les 16 chiffres, la date de validité et le code de sécurité. Le message chiffré est* a.
7. *Déchiffrer* a *à l'aide de la clé privée. Vérifier que vous obtenez bien* m *à nouveau.*

Exercice 2.2. *Attaque par factorisation de* n*. On a vu en TD la méthode factorisation de Fermat* [1].

1. *Implanter cette méthode de factorisation.*
2. *Evaluer le temps moyen de factorisation de produits de deux nombres premiers sur* k *bits avec* k *variant de 8 à 35.*
3. *Construire une courbe qui illustre ceci.*
4. *Montrer grace à cette courbe que factoriser de très grands nombres (1024 bits) avec une telle méthode est inabordable.*

1. https://fr.wikipedia.org/wiki/M%C3%A9thode_de_factorisation_de_Fermat