AVIGNON UNIVERSITÉ

Implémentation de RSA forces et faiblesse

Mohammed KHARBOUCH

28 avril 2022

Licence informatique
Ingénierie Logicielle
UE Sécurité Informatique

Responsable
HADDAD Majed
ELAZOUZI Rachid

UFR
SCIENCES
TECHNOLOGIES
SANTÉ

AVIGNON UNIVERSITÉ
CERI

CENTRE
D'**ENSEIGNEMENT**
ET DE **RECHERCHE**
EN **INFORMATIQUE**
ceri.univ-avignon.fr

Sommaire

1 Implémentation

Pour Implémenter le cryptage à clé publique/privé RSA. On aura besoin d'effectuer plusieurs opération arithmétiques. notament le Calcul du **PGCD** pour avoir le plus grand diviseur en commun, test de **primalité** et aussi l'inversion modulaire et exponentiation modulaire

* La premiere Fonction : **pgcd(a,b)** va nous permttre de calculer le plus grand diviseur en commun entre deux entiers

```python
#Calcul du le Plus Grand Diviseur Commun' entre les 2 nombres entiers a et b
def pgcd(a, b):
    if(b == 0):
        return a
    else:
        return pgcd(b, a % b)

#print("le plus grand diviseur en commun est :", pgcd(60,48))
```

Figure 1

* La deuxième Fonction : **primalite(n)** est une fonction qui va tester si **modulo n** est premier ou pas, s'il est premier elle va nous retrouner **True** sinon elle va retrouner **False**

```python
#Test de primalité
def primalite(n):
    """ retourne True si n est premier, False dans le cas contraire """

    if n == 1 or n == 2:
        return True

    if n % 2 == 0:
        return False

    x = n ** 0.5

    if x == int(x):
        return False

    for y in range(3, int(x), 2):

        if n % y == 0:
            return False

    return True
```

Figure 2

* La Troisième Fonction : **generationNumAlea** est une fonction qui va nous générer **p, q** aléatoirement mais il vont être inférieur à $2^{k/2}, et K saisir par l'utilisateur.$

```python
#Generation des nombres premiers aleatoires
def generationNumAlea():
    k = int(input("merci de bien saisir le K: "))
    """Retourne p et q deux nombres premiers """
    p = 0
    q = 0


    x = k//2
    y = 2**x

    while primalite(p) is False:
        p = int(random.randrange(1,y))

    while primalite(q) is False:
        q = int(random.randrange(1,y))



    print("p est :",p)
    print("q est ",q)
    return p , q
```

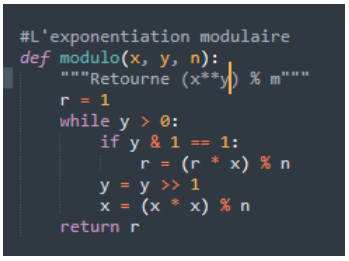
Figure 3

* La quatrième Fonction : **euclide(a,b)** est une fonction qui fasse l'algorithme d'Euclide étendu, qui permet de calculer u et v tels que a.u+b.v = pgcd(a,b). à partir de cette fonction nous pouvons calculer **d** (la clé privée) tels que e.d+k.phi(n) = pgcd(e,phi(n)).

```python
#L'algorithme d'euclide étendu
def euclide(a,b):

    r = a
    u = 1
    v = 0
    rp = b
    up = 0
    vp = 1

    while rp != 0:
        q = r // rp
        rs = r
        us = u
        vs = v

        r = rp
        u = up
        v = vp

        rp = rs - q * rp
        up = us - q * up
        vp = vs - q * vp

    return u % b
```

Figure 4

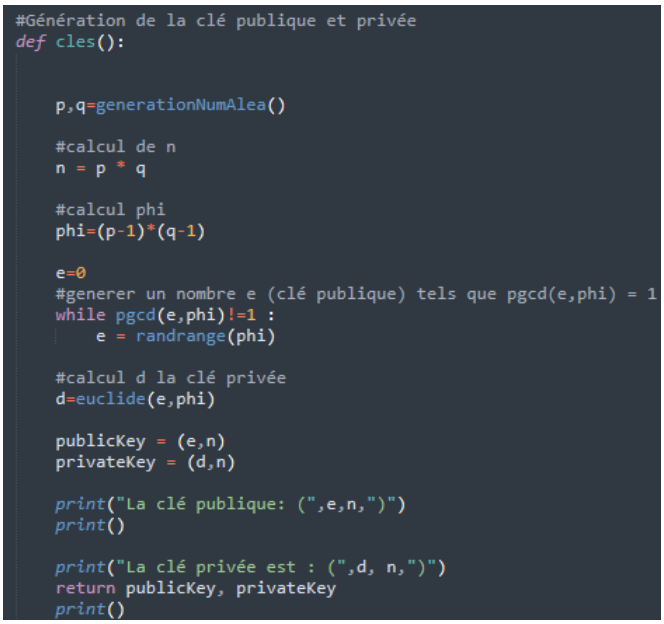
* La cinquième Fonction :**modulo(x,y,n)** , cette fonction permet de calculer l'exponentiation modulaire ($x^y_m odn) pour le chiffrement et le dechiffrement.$



```python
#L'exponentiation modulaire
def modulo(x, y, n):
    """Retourne (x**y) % m"""
    r = 1
    while y > 0:
        if y & 1 == 1:
            r = (r * x) % n
        y = y >> 1
        x = (x * x) % n
    return r
```

Figure 5

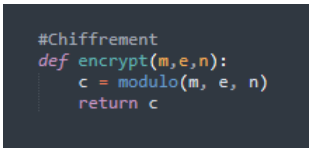
* La sixième Fonction : **cles()** cette dernière fonction nous génère les clées (privées et publique) en utilisant la fonction **generationNumAlea** pour avoir **p et q** et aussi la fonction **euclide(e,phi)** pour calculer d. La fonction **cle()** nous retourne à la fin un couple des clés publique/privée



```python
#Génération de la clé publique et privée
def cles():


    p,q=generationNumAlea()

    #calcul de n
    n = p * q

    #calcul phi
    phi=(p-1)*(q-1)

    e=0
    #generer un nombre e (clé publique) tels que pgcd(e,phi) = 1
    while pgcd(e,phi)!=1 :
        e = randrange(phi)

    #calcul d la clé privée
    d=euclide(e,phi)

    publicKey = (e,n)
    privateKey = (d,n)

    print("La clé publique: (",e,n,")")
    print()

    print("La clé privée est : (",d, n,")")
    return publicKey, privateKey
    print()
```

Figure 6

* la septième Fonction : **encrypt(m,e,n** cette fonction chargé par le chiffrement d'un message tels que c = $m^e mod n$



```python
#Chiffrement
def encrypt(m,e,n):
    c = modulo(m, e, n)
    return c
```

Figure 7

* la huitième Fonction : **decrypt(c,d,n** cette fonction chargé par le déchiffrement d'un message tels que m = c $^d mod n$

```python
#Déchiffrement
def decrypt(c,d,n):
    m = modulo(c, d, n)
    return m
    c = encrypt(message, e, n)
```

Figure 8

1.1 Test

Pour tester les fonction précedente on a créer une autre fonction **testEncryptDecrypt()** qui va nous permettre de savoir si les toutes fonctions sont bien programmée ou pas

```python
def testEncryptDecrypt():

    #generation des clés
    public, private = cles()
    message = int(input("Veillez entrer un message(Nombre) à chiffrer : "))
    print()
    e = public[0]
    n = public[1]
    d = private[0]


    while message >= n:
        message = int(input("Le message ne doit pas être suppérieur à n : "))
        print("n est :", n)
    print()

    #chiffrement
    print("---------------------------------------- chiffrement ----------------------------------------")
    c = encrypt(message, e, n)
    print("Le message chiffré est : ",c)
    print()
    print("---------------------------------------- Déchiffrement ----------------------------------------")
    #déchiffrement
    message = decrypt(c,d,n)
    print("Le message dechiffrer est : ",message)
    print()
```

Figure 9

1.2 Resultat

```
C:\Users\ACER\Desktop\cours\S6\sécurité informatique\TPs>python tp.py
---------------------------------------- Parite 1  ----------------------------------------
merci de bien saisir le K: 25
p est : 1549
q est  2657
La clé publique: ( 116939 4115693 )

La clé privée est : ( 2043491 4115693 )
Veillez entrer un message(Nombre) à chiffrer : 1523


---------------------------------------- chiffrement ----------------------------------------
Le message chiffré est :  2834006

---------------------------------------- Déchiffrement ----------------------------------------
Le message dechiffrer est :  1523
```

Figure 10

On observe que le nombre qu'on saisit c'est le même message déchifré donc les algorithmes qu'on a crée sont bien programmée

2 Test avec une clé de taille petite

déchiffrement du message : [9197, 6284, 12836, 8709, 4584, 10239, 11553, 4584, 7008, 12523, 9862, 356, 5356, 1159, 10280, 12523, 7506, 6311] avec la clé publique **12413 ; 13289**

Avant de commencer le déchiffrement, on doit créer une autre fonction **factoring(n)** qui va prendre en argument **n = p x q** et va retouner un tableau qui contient les facteurs premiers de n.

```python
def factoring(n):
    """factoring(n): décomposition d'un nombre entier n en facteurs premiers"""
    F = []
    if n == 1:
        return F
    # recherche de tous les facteurs 2 s'il y en a
    while n >= 2:
        x, r = divmod(n, 2)
        if r != 0:
            break
        F.append(2)
        n = x
    # recherche des facteurs 1er >2
    i = 3
    rn = np.sqrt(n) + 1
    while i <= n:
        if i > rn:
            F.append(n)
            break
        x, r = divmod(n, i)
        if r == 0:
            F.append(i)
            n = x
            rn = np.sqrt(n) + 1
        else:
            i += 2
    return F
```

Figure 11

Après la décomposition de n en deux nombres premiers, on peut calculer phi(n) et après d (la clé privée) qui nous a permet de déchiffrer le message.

```python
def decryptMsg1():
    print("---------------------------------------- Déchiffrement du message 1 ----------------------------------------")
    print("La clé publique : (12413 ; 13289)")
    n=13289
    e=12413


    p,q=factoring(n)

    phi=(p-1)*(q-1)

    d=euclide(e,phi)

    print("La clé privée : (",d," ; ",n,")",)

    msg=[9197, 6284, 12836, 8709, 4584, 10239, 11553, 4584, 7008, 12523, 9862, 356,5356, 1159, 10280, 12523, 7506, 6311]
    print("Le message à déchiffrer:", msg)
    messageDecrypt=[]
    #parcourir le tab de message
    for element in msg:
        #dechiffrer chaque élément
        message1=decrypt(element,d,n)
        #ajout de message déchiffré au tab messageDechiffrer
        messageDecrypt.append(message1)

    print("Le message déchiffré :",messageDecrypt)
    print()
```

Figure 12

après on obtient le résultat de déchiffrement du premier message

```
---------------------------------------- Déchiffrement du message 1 ----------------------------------------
La clé publique : (12413 ; 13289)
La clé privée : ( 3797  ;  13289 )
Le message à déchiffrer: [9197, 6284, 12836, 8709, 4584, 10239, 11553, 4584, 7008, 12523, 9862, 356, 5356, 1159, 10280,
12523, 7506, 6311]
Le message déchiffré : [5424, 6221, 2423, 1662, 1023, 1362, 2917, 1023, 2028, 6215, 2427, 6210, 2121, 6229, 1714, 6215,
1828, 1762]
```

Figure 13

déchiffrement du message : [671828605, 407505023, 288441355, 679172842, 180261802] avec la clé publique **(e=163119273 ; n=755918011)**

```python
def decryptMsg2():
    print("------------------------------ Déchiffrement du message 2 ------------------------------------------")
    print("La clé publique : (163119273;755918011)")
    n=755918011
    e=163119273

    p,q=factoring(n)
    phi=(p-1)*(q-1)


    d=euclide(e,phi)

    print("La clé privée : (", d, " ; ", n, ")", )

    msg2=[671828605, 407505023, 288441355, 679172842, 180261802]
    print("Le message à déchiffrer:", msg2)
    messageDecrypt=[]
    # parcourir le tab de message
    for element in msg2:
        # dechiffrer chaque élément
        messgae1=decrypt(element,d,n)
        # ajout de message déchiffré au tab messageDechiffrer
        messageDecrypt.append(messgae1)

    print("Le message déchiffré :", messageDecrypt)
    print()
```

Figure 14

après on obtient le résultat de déchiffrement du premier message

.

```
------------------------------ Déchiffrement du message 2 ------------------------------------------
La clé publique : (163119273;755918011)
La clé privée : ( 528303353  ;  755918011 )
Le message à déchiffrer: [671828605, 407505023, 288441355, 679172842, 180261802]
Le message déchiffré : [3924, 2329, 6251, 3649, 4438]
```

Figure 15

3 RSA et Codage

Pour chiffrer un text, il nous faut le convertir en Code et le décomposer en blocs. C'est pour cela on a crée la fonction **convertMsg(message,n)** avec un nombre de caractère qui est 40, donc on peut calculer **k** qui est le nombre de caractère par bloc pour qu'on puisse composer des blocs de tailles k. .

```
def convertmsg(message,n):
    # nombre de caractère pour code
    N = 40

    #Calcul de k (la taille du bloc)

    k = int(floor(log(n, N)))

    #k=2

    # conversion du message en codes
    codex = [str(ord(j)) for j in message]
    print(codex)

    # ajout de 0 pour avoir une longueur fixe 3 de chaque code
    for i, j in enumerate(codex):
        if len(j) < 3:

            while len(j) < 3:
                j = '0' + j

            codex[i] = j

    # compositions de blocs de taille k
    codeg = ''.join(codex)

    d, f = 0, k

    # on rajoute des 0 a la fin de  de maniere a ce que len(codeg) soit un multiple de f
    while len(codeg) % f != 0:

        codeg = codeg + '0'

    code= []

    while f <= len(codeg):
        code.append(codeg[d:f])

        d, f = f, f + k

    return code
    print(code)
```

Figure 16

Après la conversion du texte, on peut chiffrer le texte facilement à l'aide de la fonction **encryptTxt()**

.

```
def encryptTxt() :
    print("------------------------------------- Chiffrement d'un message text ----------------------------------")
    #generation des clés
    public, private= cles()
    e = public[0]
    n=public[1]
    message = input("Veillez entrer un message à chiffrer :")
    #conversion de message en code
    code=convertmsg(message,n)
    messageChiffre=[]

    #chiffrement du message
    for el in code:
        el=int(el)
        c = encrypt(el,e,n)
        messageChiffre.append(c)

    print("Le message chiffré : ",messageChiffre)

    return(messageChiffre)
```

Figure 17

3.1 Résultat

.

```
------------------------------------ Chiffrement d'un message text ----------------------------------
merci de bien saisir le K: 22
p est : 1427
q est  323
La clé publique: ( 302163 460921 )

La clé privée est : ( 181035 460921 )
Veillez entrer un message à chiffrer :Bonjours c'est mohammed
['66', '111', '110', '106', '111', '117', '114', '115', '32', '99', '39', '101', '115', '116', '32', '109', '111', '104'
, '97', '109', '109', '101', '100']
Le message chiffré :  [59101, 138973, 122368, 357999, 138973, 432965, 68742, 381331, 284742, 77816, 18621, 361011, 38133
1, 13692, 284742, 311460, 138973, 327020, 412635, 311460, 311460, 361011, 248413]
```

Figure 18