AVIGNON UNIVERSITÉ

Rendu TP1 : Implémentation de RSA

Groupe TD4, B
RHADI Meryam

31 mars 2021

Ingénierie Logicielle
UE Sécurité informatique

Responsable
Majed haddad

UFR
SCIENCES
TECHNOLOGIES
SANTÉ

AVIGNON UNIVERSITÉ
CERI

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

Sommaire

1 Introduction générale

Pour l'implémentation du tp1, j'ai choisi de travailler avec langage de programmation **Python**, qui est le plus facile pour implémenter.

2 Implémentation

Pour l'implémentation de l'algorithme de **RSA**, on a besoin d'implémenter plusieurs fonctions et d'opération arithmétique, certain pour faire des tests avant de chiffrer ou déchiffrer, et d'autre pour les utiliser pour le (dé)chiffrement, qui sont comme suit :

- **La fonction de PGCD :** Cette fonction est utilisé pour calculer le plus grand commun diviseur entre deux nombres entiers donné en paramètre, et qui est implémente pour l'utiliser dans le test que le PGCD de phi et e est égale à 1.

```
#La fonction de PGCD
def pgcd(a,b):
    while b!=0 :
        a,b = b, a % b
    return a
```

Figure 1. La fonction de PGCD

- **La fonction de test de prima-lité :** Cette fonction nous permet de tester si un nombre donné en paramètre est première ou non, il nous permet particulièrement d'appliquer les condition d'RSA, qui consiste sur la primalité de p et q, et pour que la clé soit très difficile à inverser.

```
#La fonction de test de primalité
def primaliter(n):
    d=2
    r=n%d
    if r==0:
     return False
    d=3
    while d<=sqrt(n):
        r=n%d
        if r==0:
            return False
        d=d+2
    return True
```

Figure 2. La fonction de test primalité

- **La fonction pour l'inversion modulaire :** Cette fonction permet de calculer l'inverse modulaire, il prend deux paramètres, et retourne l'inverse modulaire quand il le trouve.

```
#La fonction pour inversion modulaire
def modulo(a,b):
    for n in range(b):
        if(a*n)%b==1:
            return n
            break
        elif n==b-1:
            return "Null"
        else:
            continue
```

Figure 3. La fonction de l'inversion modulaire

- **La fonction pour l'exponentiation modulaire :** Cette fonction est un outil de calcul de puissance modulaire, il prend trois paramètres et retourne le calcul de abˆ modulo n.

```
#La fonction exponentiation modulaire
def expo(a,b,n):
    resultat=1
    while b>0:
        if b&1>0:
            resultat=(resultat*a)%n
        b >>= 1
        a= (a*a)%n
    return resultat
```

Figure 4. La fonction pour l'exponentiation modulaire

- **La fonction pour calculer d :** Cette fonction sert à calculer la clé privé d à partir de la clé public e et phi_n.

```
#La fonction d'Euclide etendu pour calculer d
def TrouverD(e, phi) :
  d=1
  temp=(e*d)%phi
  while(temp!=1):
    d=d+1
    temp=(e*d)%phi
  return d
```

Figure 5. La fonction pour calcul de d

- **La fonction pour chiffrer :** Cette fonction permet de chiffrer un message, qui est sous forme d'un nombre passé en paramètre, avec l'utilisation de la clé public et la valeur de n.

```
def chiff(m,e,n):
    nbr_chiffre = pow(m,e,n)
    return nbr_chiffre
```

Figure 6. La fonction pour le chiffrement

- **La fonction pour déchiffrer :** Cette fonction nous permet de déchiffrer le message déjà chiffré, en l'envoyant comme paramètre de la fonction avec la clé privé d et n.

```
def dichff(c,d,n):
    resultat=pow(c,d,n)
    return resultat
```

Figure 7. La fonction pour le déchiffrement

3 Test local

Pour le test RSA, j'ai fais un autre fichier que j'ai appelé **test.py**, qui importe les fonctions de RSA déjà implémenter, et j'ai demandé à l'utilisateur d'entrer deux nombre P et Q, on test que ces deux nombres sont première avant de commencer l'algorithme en appelant la fonction de primalité, et si ce n'est pas le cas on demande à l'utilisateur de saisir à nouveau p et q, on calcul phi, et on demande de nouveau à l'utilisateur de saisir la valeur e la clé public e, on fais des tests une autre fois sur la primalité et le que le PGCD entre e et n est égale à 1, après on calcule phi, et la clé privé d, et on demande de saisir le nombre à chiffrer, on retourne résultat et on le déchiffre en utilisant le message chiffré, la clé privé d, et la valeur de n qui est la multiplication de p et q.

```
import rsa_fonction


def main():
    print("******************Bonjour dans le programme de RSA********************
    p=input("Merci d'entrer un nombre premiere P: ")
    while rsa_fonction.primaliter(p)== False:
        p=input("P n'est pas première, Merci d'entrer un autre nombre premiere
    print("p=",p)
    q=input("Merci d'entrer un autre nombre premiere Q :")
    while rsa_fonction.primaliter(q)== False:
        q=input(" Q n'est pas première, Merci d'entrer un autre nombre premier
    print("q=",q)
    n=int(p)*int(q)
    print("n=p*q = ",n)
    e=input("Merci d'entrer la clé public e : ")
    while rsa_fonction.primaliter(e)== False:
        if rsa_fonction.pgcd(e, n)!=1:
            e=input("Entrer e : ")
    phi=(int(p)-1)*(int(q)-1)
    print("phi=",phi)
    d=rsa_fonction.TrouverD(e, phi)
    print("Clé publique : ", e, ", Clé privée : " , d)
    nombre=input("Merci d'entrer un nombre pour le chiffrer : ")
    chiffrer=rsa_fonction.chiff(nombre,e,n)
    print("Le nombre entrer avec le chiffremet est : ", chiffrer)
    dechiffrer=rsa_fonction.dichff(chiffrer,d,n)
    print("Le nombre après le dechiffremet est : ", dechiffrer)
```

Figure 8. Code du test

J'ai appliqué l'algorithme de RSA, sur l'exemple étudié dans le td, il a donné le même résultat trouvé dans l'exercice.

```
**********Bonjour dans le programme de RSA**********
Merci d'entrer un nombre premiere P: 11
p= 11
Merci d'entrer un autre nombre premiere Q :23
q= 23
n=p*q =  253
Merci d'entrer la clé public e : 3
phi= 220
Clé publique :  3 , Clé privée :  147
Merci d'entrer un nombre pour le chiffrer : 165
Le nombre entrer avec le chiffremet est :  110
Le nombre après le dechiffremet est :  165
>>>
```

Figure 9. Le résultat du test effectué

4 Test distant à 2

Pour le test à distant, mon collègue ma envoyé sa clé publique dont j'ai chiffré un message avec sa clé, et lui envoyé se message qui a déchiffré avec sa clé privé, et il a obtenu le même message que j'ai chiffré en premier.

5 Taille petite :

Pour cette partie, il est demandé de déchiffrer un message donner dans le tp, dont on ne connais que la clé public et la valeur de n, donc il faut premièrement savoir la valeur de p et q, pour quand puisse calculer phi, qui va nous permettre de calculer la clé privé pour qu'en puisse décrypter le message donner .
Pour cela, j'ai ajouter une fonction qui permettre de trouver p et q à partir de n, et retourne la valeur de phi.

```
def TrouverPetQ(n):
    N=2
    while N:
        while n%N!=0:
            N=N+1
        if n/N==1 :
            global p
            p = N
            print("p= ",p)
            break
        global q
        q=N
        print("q= ",q)
        n=n/N;
    phi=(p-1)*(q-1)
    return phi
```

Figure 10. La fonction qui permettre de trouver p et q

Dans le main du programme, j'ai entrer la valeur de e et n et la liste des message à déchiffrer en dur, et pour l'affichage des résultat j'ai fais une boucle qui appelle à chaque fois la fonction qui décrypte et affiche le résultat.

```
def main():
      e= 12413
      n= 13289
      M=[9197,6284,12836,8709,4584,10239,11553,4584,7008,12523,9862,356,5356,
      phi=TrouverPetQ(n)
      print("phi= ",phi)
      d=rsa_fonction.TrouverD(e, phi)
      for m in M:
         dechiffrer = rsa_fonction.dichff(m,d, n)
         print("%d -> %d" % (m, dechiffrer))
```

Figure 11. La fonction qui test et fais le déchiffrage de la liste

```
q=  97
p=  137
phi=  13056
9197 -> 5424
6284 -> 6221
12836 -> 2423
8709 -> 1662
4584 -> 1023
10239 -> 1362
11553 -> 2917
4584 -> 1023
7008 -> 2028
12523 -> 6215
9862 -> 2427
356 -> 6210
5356 -> 2121
1159 -> 6229
10280 -> 1714
12523 -> 6215
7506 -> 1828
6311 -> 1762
>>>
```

Figure 12. Le résultat

6 Chiffrement d'un texte

Pour cette partie, il est un peu différente du chiffrement d'un nombre, parce que dans le chiffrement d'un message que se soit un mot ou une phrase, on vas diviser notre message à des blocs.
Dans notre cas, après avoir demandé à l'utilisateur d'entrer un message, on vas calculer sa taille, et initialiser une variable à 0, pour l'utiliser dans la boucle.
Donc tant que la taille du message est plus grand de 0, il faut qu'on divise le message à des blocs, en utilisant la fonction ordi qui génère un code asci pour la première lettre du message entrer et le stocke dans une variable, et après il vas faire la même chose pour deuxième lettre du message.

Maintenant on vas obtenir des blocs qui contient deux lettre, et après on vas faire la concaténation de ces deux lettres et le considérer comme un bloc, et chiffrer chaque bloc trouvé, on vas stocker le résultat crypter de chaque bloc dans une variable, à chaque fois on incrémente i, qui vas permettre d'arrêter la boucle quand i sera plus grand que la taille de notre message entrer par l'utilisateur.

```
>>> exo5()
*****************Bonjour dans le programme de RSA*************************
Merci d'entrer un nombre premiere P: 11
p= 11
Merci d'entrer un autre nombre premiere Q :13
q= 13
n=p*q =  143
Merci d'entrer la clé public e : 7
phi= 120
Clé publique :  7 , Clé privée :  103
Merci d'entrer le message pour le chiffrer : meryam
bloc 8678
bloc 9198
bloc 7486
phrase crypt =  [32, 84, 41]
```

Figure 13. Le résultat

7 Tester les failles

Pour les failles de l'algorithme de RSA, après son implémentation et son test, j'ai observé que c'est facile de déduire la clé privé d, en factorisant le nombre n dans le cas où on a utiliser une petite valeur pour p et q, donc c'est pour sa il est forcément recommandé d'utiliser un grande nombre pour la clé public, et aussi pour la valeurs de p et q qui doit être premier et de grande taille.