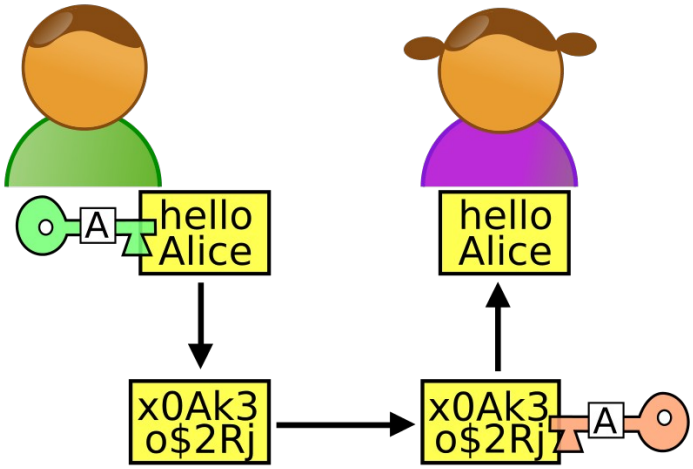
Rapport TP3 RSA



Réalisé par :

Martin BENITO-RODRIGUEZ

Imene OUAHRANI

Lien github : https://github.com/imouahrani/M1_Reseau_Securite

Nous avons écrit cette partie en C qui permet de manipuler les tableaux pour la partie 1.

Partie 1

Exercice 1.1:

```c
#define SIZE 100

typedef struct nombre nombre;
struct nombre
{
    int tab[SIZE];
};
```

On créer une constante SIZE de taille 100 qu'on utilise pour créer notre tableau et pour les boucles.

Il était dit qu'il fallait faire un tableau de caractère, mais on a préféré faire un tableau d'entier car on aura pas à mettre des apostrophes.

Exercice 1.2:

```
nombre initialiser0()
{
    nombre nbr;
    for(int i=0; i<SIZE; i++){
        nbr.tab[i]=0;
    }
    return nbr;
}
```

Rien de spécial à dire, on met chaque case du tableau à 0.

Exercice 1.3 :

- Écrire une fonction qui initialise le tableau à 1.

```
nombre initialiser1()
{
    nombre nbr;
    for(int i=0; i<SIZE; i++){
        nbr.tab[i]=1;
    }
    return nbr;
}
```

Exercice 1.4:

- Écrire une fonction de libération de la mémoire associée à un nombre, appelée libererNombre. Pour cela, on doit utiliser ‘ free()’ afin de libérer la mémoire.

```
void libererNombre(nombre nbr)
{

    for(int i=0; i<SIZE; i++){
        free(nbr.tab[i]);
    }
    free(nbr.tab);
    free(&nbr);

}
```

Exercice 1.5 :

Écrire une fonction afficher qui permet l’affichage (en binaire) du nombre.

```
void affiche(nombre nbr){
    for (int i = 0 ; i < SIZE ; i++){
        printf("%d", nbr.tab[i]);
    }
    printf("\n");
}
```

Exercice 1.6:

```
bool sontEgaux(nombre nbr1, nombre nbr2){
    for (int i = 0 ; i < SIZE ; i++){
        if(nbr1.tab[i]!=nbr2.tab[i]){
            return false;
        }
    }
    return true;
}
```

On fait une boucle sur tout le tableau, si on voit une différence sur les deux nombres, on retourne faux, sinon on retourne vrai.

Exercice 1.7 : Ecrire une fonction booléenne estPair qui détermine si un nombre n est pair ou non. Pour cela, il suffit de regarder la parité du dernier bit. C'est à dire, si le nombre est égale à 0 alors elle retourne 'true ' sinon elle retourne 'false'.

```
bool estPair(nombre nbr){
    if(nbr.tab[SIZE-1] == 0){
        return true;
    }
    else{
        return false;
    }
}
```

Exercice 1.8:

```
nombre diviserPar2(nombre nbr){

    for (int i=SIZE-1 ; 0<=i ; i--){
        nbr.tab[i]=nbr.tab[i-1];
    }

    nbr.tab[0]=0;
    return nbr;
}
```

- La boucle permet de décaler tous les bits à droite. Puis on met le bit le plus à gauche (le bit avec le plus important poids) à 0.

Exercice 1.11 :

Écrire une fonction qui prend en paramètre deux nombres a et b et retourne un résultat tel que : 1+0=1, 0+0=0, 0+1=1, 1+1=0.

```
nombre ajouter2Nombres(nombre A, nombre B){


    nombre S;


    printf("\nLe tableau Somme est:\n");
    for (int i = 0 ; i<SIZE-1 ; i++){

        S.tab[i] = A.tab[i] + B.tab[i];

        if((A.tab[i] ==0 && B.tab[i] == 1) || (B.tab[i] ==0 && A.tab[i] == 1)){
             S.tab[i] = 1;
        }else{
             S.tab[i] = 0;
        }
    }
    return S;
}
```

Exercice 12:

- Écrire une fonction qui prend en paramètre deux nombres a et b et retourne un résultat tel que : 1*0=0, 0*0=0, 0*1=0, 1*1=1.

```
nombre multiplier2Nombres(nombre A, nombre B){


    nombre M;


    printf("\nLe tableau Multiplication est:\n");
    for (int i = 0 ; i<SIZE-1 ; i++){

        M.tab[i] = A.tab[i] * B.tab[i];

        if((A.tab[i] ==0 && B.tab[i] == 1) || (B.tab[i] ==0 && A.tab[i] == 1) || (B.tab[i] ==0 && A.tab[i] == 0)){
             M.tab[i] = 0;
        }else{
             M.tab[i] = 0;
        }
    }
    return M;
}
```

Exercice 1.13 :

-Écrire une fonction qui prend en paramètre deux nombres a et b et qui calcule a^b.

```
nombre exponentiationRapideSansModulo(nombre x, nombre y){
    nombre result = initialiser1();
    printf("\nL'exponentiation Rapide Sans Modulo est:\n");
    for (int i = 0 ; i<SIZE-1 ; i++){

        while (y.tab[i]> 0)
        {
            // Si y est impair, multipliez x par le resultat
            if (y.tab[i] & 1)
                result.tab[i] =  result.tab[i] * x.tab[i];

            // y doit etre le meme maintenant
            y.tab[i] = y.tab[i]>>1; // y = y/2
            x.tab[i] = x.tab[i] * x.tab[i];  // remplacer x par x^²
        }
    }

    return result;
}
```

Exercice 1.16 :

-Écrire une fonction exponentiation rapide, qui prend en paramètre trois nombres a, b et e, et qui calcule a^b [e]. La fonction calcule le modulo à chaque étape de l'exponentiation rapide.

Principe : Étant donné trois nombres b, m et e, calculez (b ^ m)% e

Exemple :

Input: b = 2, m = 3, e = 5
Output: 3
Explanation: 2^3 % 5 = 8 % 5 = 3.

```
nombre fastExp(nombre b, nombre m, nombre e)
{

    nombre result = initialiser1();
    printf("\nL'exponentiation Rapide avec Modulo est:\n");
    for (int i = 0 ; i<SIZE-1 ; i++){

       if (1 & e.tab[i] )
         result.tab[i]  = b.tab[i] ;
       while (1) {
           if (!e.tab[i] ) break;
           e.tab[i]  >>= 1;
           b.tab[i]  = (b.tab[i]  * b.tab[i] ) % m.tab[i] ;
           if (e.tab[i]  & 1)
           result.tab[i]  = (result.tab[i]  * b.tab[i] ) % m.tab[i] ;
       }
    }

    return result;
}
```

Partie 2 – Chiffrement et déchiffrement

Exercice 2.3 :

- Écrire la fonction qui calcule m^e [n], en utilisant l'exponentiation rapide et en effectuant le calcul du modulo n à chaque étape de l'exponentiation rapide. Le déchiffrement d'un message c avec RSA se fait en calculant c^d [n]

- Pour travailler avec des nombre non binaires, J'ai décidé d'implémenter une fonction initialiserRandom() qui initilise le tableau avec rand() de la bibliothèque Math. Afin d'avoir des valeurs aléatoires.

```
nombre initialiserRandom()
{
    nombre nbr;
    for(int i=0; i<SIZE; i++){
        nbr.tab[i]= rand ();
    }
    return nbr;
}
```

```
int gcdExtended(int a, int b, int *x, int *y) {
    // initialisation
    if (a == 0) {
        *x = 0;
        *y = 1;
        return b;
    }
    int x1, y1; // Pour stocker les resultats d'un appel recursif
    int gcd = gcdExtended(b%a, a, &x1, &y1);
    // Mettre à jour x et y en utilisant les resultats de recursif
    // appel
    *x = y1 - (b/a) * x1;
    *y = x1;
    return gcd;
}
```

Résultat de l’exécution

```
ouahrani@user:~/Bureau/M1/SECURITE_RESEAUX/reseau_securite-main/RSA$ gcc main.c -o main
ouahrani@user:~/Bureau/M1/SECURITE_RESEAUX/reseau_securite-main/RSA$ ./main
Init0:00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

Init1:11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111

2div2:00111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111

retirer1:00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001

mult2:00222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222221

Le tableau Somme est:
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111110

Le tableau Multiplication est:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

L'exponentiation Rapide Sans Modulo est:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
a[n]=
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
Soutraction
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001

L'exponentiation Rapide avec Modulo est:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001

gcd(35, 15) = 5, x = 1, y = -2ouahrani@user:~/Bureau/M1/SECURITE_RESEAUX/reseau_securite-main/RSA$
```

Partie 3 – Manipulation des grands nombres (fin)

Exercice 3.1 : Le calcul de l'inverse de a modulo b se fait au moyen de l'algorithme d'Euclide étendu. Implémentez l'algorithme inverserAvecModulo (sa description se trouve sur Wikipedia par exemple).

- Ecrire une fonction qui prend 2 paramètres x et y et qui retourne leur gcd .

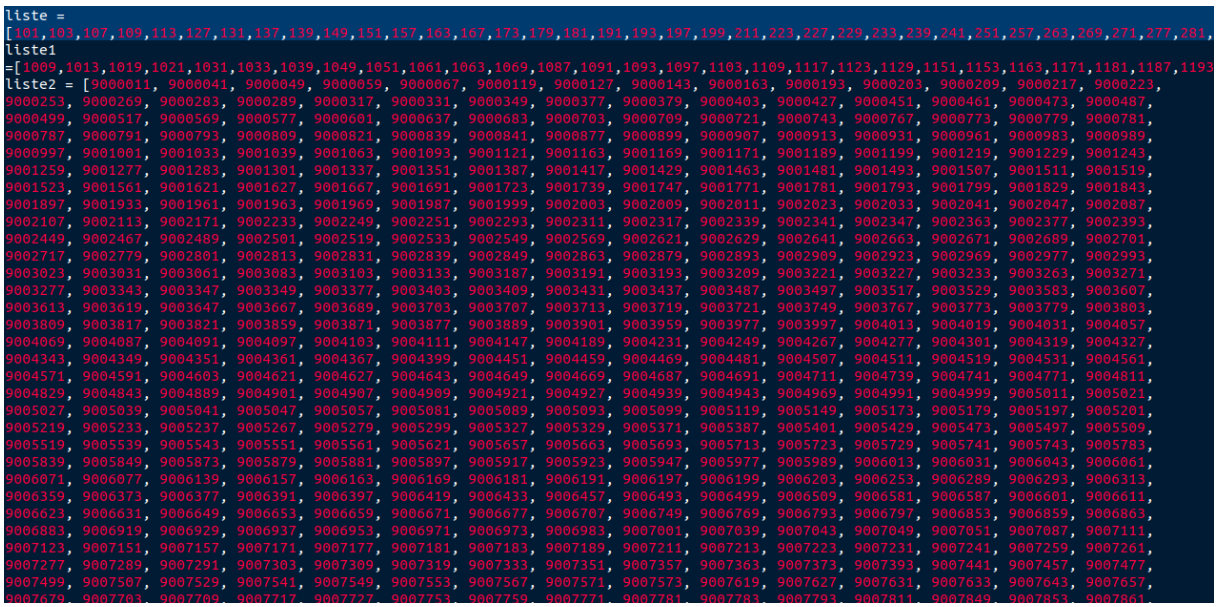
```python
def extended_gcd(x,y):
    if y == 0:
        return (x, 1, 0)
    else:
        (d, a, b) = extended_gcd(y, x % y)
        return (d, b, a - x//y * b)

print("Find a^-1 in (mod b)")
x = int(input("Entrer a *la valeur inversee*. \t"))
y = int(input("Entrer b *le modulo*. \t"))
answer = extended_gcd(x,y)[1]
if answer < 0:
    answer += y
print("L'inverse de " + str(x) + " mod  " + str(y) + " est " + str(answer) + ".")
```

```
ouahrani@user:~/Bureau/M1/SECURITE_RESEAUX/RSA/3.1$ python extended_euclid.py
Find a^-1 in (mod b)
Entrer a *la valeur inversee*.  19
Entrer b *le modulo*.   5
L'inverse de 19 mod  5 est 4.
```

Partie 4 – Création des clés

Nous avons écrit cet partie du TP en python. Cette partie travaille avec une classe liste.py qui contient une liste de nombres premiers.



Les étapes de l'algorithme RSA :

1) Préparation des clefs :

- choix de deux nombres premiers distincts p et q

- calcul de n = p * q

- calcul de phi_n = (p-1)*(q-1)

- calcul d'un exposant e tel que pgcd(e, phi_n) = 1, c'est dire qu'ils sont premiers entre eux

- calcul de d inverse de module (phi_n) par l'algorithme d'Euclide étendu : d * e ≡ 1 [phi_n]

2) Chiffrement du message :

- x = m^e [n]

3) Déchiffrement du message :

- m = x^d [n]

Exercice 4.1 : Implémentez la génération des clés.

1)- J'ai commencé par créé une classe **liste.py** qui contient une liste de nombres premiers.

2)- Créer une fonction qui prend un nombre premiers en paramètre de sorte que ce chiffre soit choisit qu'une seul fois. Afin de choisir deux nombres premiers distincts p et q par la suite.

```
from list import liste
import random

# Fonction qui choisit un nombre premier different d'un autre
def nbPremierDiff(src) :
  dest = liste[r.randrange(len(liste))]
  while src == dest:
    dest = liste[r.randrange(len(liste))]
  return dest
```

3) – Implémenter l'algorithme d'Euclide pour le pgcd afin de calculer par la suite l'exposant e tel que **pgcd(e, phi_n) = 1**, c'est dire qu'ils sont premiers entre eux .

- Tant que a mod b <> 0 on continue la division puis on récupère le diviseur b.

```
# Algorithme d'Euclide pour le pgcd
def pgcd(a,b) :
   while a%b != 0 :
      a, b = b, a%b
   return b
```

4)- Implémenter l'algorithme d'Euclide étendu afin de calculer d inverse de module (phi_n) : **d * e ≡ 1 [phi_n]**

- Prendre une valeur de e arbitraireent tel que : 1 < e < phi(n)

```
# Algorithme d'Euclide etendu
def euclide_etendu(e, phi_n) :
  d = 1
  temp = (e*d)%phi_n
  while(temp != 1):
    d = d+1
    temp = (e*d)%phi_n
  return d
```

5)- Chiffrement :

Encryption algorithm

(n, e) ← readkey(pub)

c ← (m e mod n)

return c

- Ecrire une fonction qui prend 3 paramètres le message à chiffrer qui est une chaine de caractères vide au départ, l'exposant e tel que 1 < e < phi(n), n = p * q

- Calculer le message chiffré c tel que c = m^e [n]

```
# Chiffrement du message
def chiffrer(message, e, n):
  i=0
  message_chiffre = ""
  while i != len(message):
    bloc = str(pow(ord(message[i]), e)%n)
    #print bloc
    while (len(bloc) != 6):
      bloc = "0" + bloc
    message_chiffre = message_chiffre + bloc
    i = i + 1
  return message_chiffre
```

3) Déchiffrement du message : m = x^d [n]

Decryption algorithm

(n, d) ← readkey(pri)

m ← (c d mod n)

return m

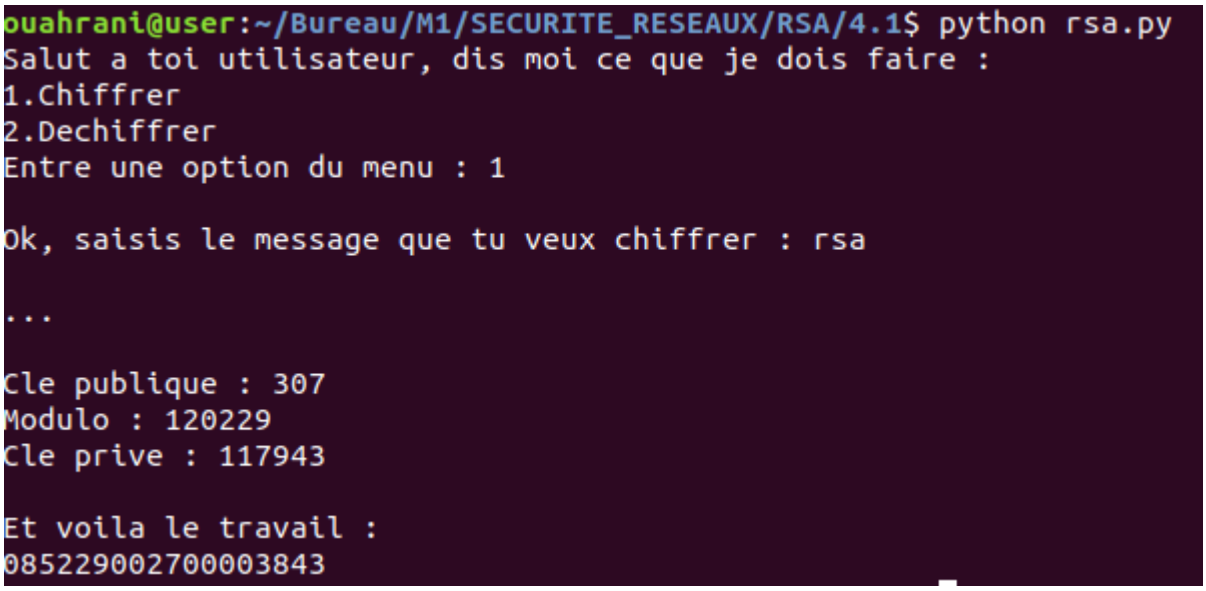
- Ecrire une fonction qui prend 3 paramètres le message à déchiffrer qui est une chaine de caractères vide au départ, d tel que d * e ≡ 1 [phi_n], n = p * q

- Calculer le message chiffré c tel que m = c^d [n]

```
# Dechiffrement du message
def dechiffrer(message_chiffre, d, n):
  i=0
  bloc=""
  message_dechiffre = ""
  while i != len(message):
    bloc = bloc + message[i]
    if (len(bloc)==6):
      #print bloc
      bloc = pow(int(bloc), d)%n
      message_dechiffre = message_dechiffre + chr(bloc)
      bloc = ""
    i = i + 1
  return message_dechiffre
```

Le résultat de l'exécution

```
######## Menu ########
choix = input('Salut a toi utilisateur, dis moi ce que je dois faire : \n1.Chiffrer\n2.Dechiffrer\nEntre une option du menu : ')
while choix != 1 and choix != 2 :
  choix = input('\nErreur, entrez soit 1 ou 2 : ')
######## Chiffrement ########
if choix==1:
  message = raw_input('\nOk, saisis le message que tu veux chiffrer : ')
  print "\n...\n"
  ######## Generation de p et q ########
  r = random.SystemRandom()
  p =liste[r.randrange(len(liste))]
  q = nbPremierDiff(p)
  n = p * q
  phi_n = (p-1)*(q-1)
  ######## Choix d'un exposant e et calcul de son inverse d ########
  e = r.choice(liste)
  d = euclide_etendu(e, phi_n)
  print "Cle publique :", e, "\nModulo :", n,"\nCle prive :", d
  # n et e = cle publique, d = cle prive
  print "\nEt voila le travail :\n", chiffrer(message, e, n)
######## Dechiffrement ########
else:
  message = raw_input('\nOk, saisis le message que tu veux dechiffrer : ')
  d = input('Super, maintenant la cle prive : ')
  n = input('Enfin le modulo : ')
  message = dechiffrer(message, d, n)
  print "\nEt voila le travail :\n", message
```


ouahrani@user:~/Bureau/M1/SECURITE_RESEAUX/RSA/4.1$ python rsa.py
Salut a toi utilisateur, dis moi ce que je dois faire :
1.Chiffrer
2.Dechiffrer
Entre une option du menu : 1
Ok, saisis le message que tu veux chiffrer : rsa
...
Cle publique : 307
Modulo : 120229
Cle prive : 117943
Et voila le travail :
085229002700003843