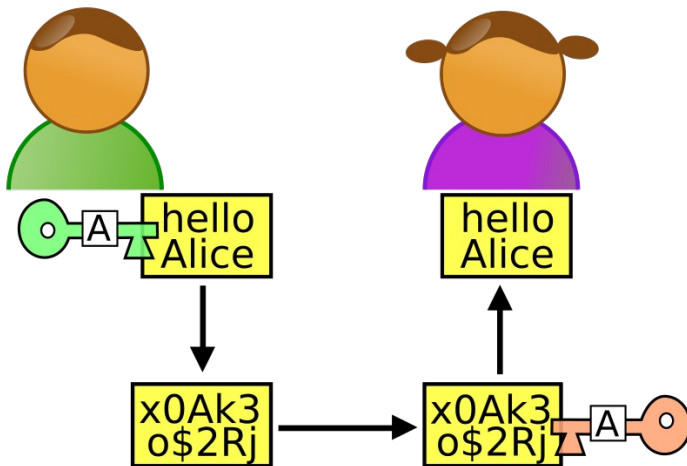


Rapport TP3 RSA



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:

```
#define SIZE 100

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

On crée 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/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 cette partie du TP en python. Cette partie travaille avec une classe `liste.py` qui contient une liste de nombres premiers.

```
liste =
[101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281,
liste1
=[1009, 1013, 1019, 1021, 1031, 1033, 1039, 1049, 1051, 1061, 1063, 1069, 1087, 1091, 1093, 1097, 1103, 1109, 1117, 1123, 1129, 1151, 1153, 1163, 1171, 1181, 1187, 1193,
liste2 = [9000011, 9000041, 9000049, 9000059, 9000067, 9000119, 9000127, 9000143, 9000163, 9000193, 9000203, 9000209, 9000217, 9000223,
9000253, 9000269, 9000283, 9000289, 9000317, 9000331, 9000349, 9000377, 9000379, 9000403, 9000427, 9000451, 9000461, 9000473, 9000487,
9000499, 9000517, 9000569, 9000577, 9000601, 9000637, 9000683, 9000703, 9000709, 9000721, 9000743, 9000767, 9000773, 9000779, 9000781,
9000787, 9000791, 9000793, 9000809, 9000821, 9000839, 9000841, 9000877, 9000899, 9000907, 9000913, 9000931, 9000961, 9000983, 9000989,
9000997, 9001001, 9001033, 9001039, 9001063, 9001093, 9001121, 9001163, 9001169, 9001171, 9001189, 9001199, 9001219, 9001229, 9001243,
9001259, 9001277, 9001283, 9001301, 9001337, 9001351, 9001387, 9001417, 9001429, 9001463, 9001481, 9001493, 9001507, 9001511, 9001519,
9001523, 9001561, 9001621, 9001627, 9001667, 9001691, 9001723, 9001739, 9001747, 9001771, 9001781, 9001793, 9001799, 9001829, 9001843,
9001897, 9001933, 9001961, 9001963, 9001969, 9001987, 9001999, 9002003, 9002009, 9002011, 9002023, 9002033, 9002041, 9002047, 9002087,
9002107, 9002113, 9002171, 9002233, 9002249, 9002251, 9002293, 9002311, 9002317, 9002339, 9002341, 9002347, 9002363, 9002377, 9002393,
9002449, 9002467, 9002489, 9002501, 9002519, 9002533, 9002549, 9002569, 9002621, 9002629, 9002641, 9002663, 9002671, 9002689, 9002701,
9002717, 9002779, 9002801, 9002813, 9002831, 9002839, 9002849, 9002863, 9002879, 9002893, 9002909, 9002923, 9002969, 9002977, 9002993,
9003023, 9003031, 9003061, 9003083, 9003103, 9003133, 9003187, 9003191, 9003193, 9003209, 9003221, 9003227, 9003233, 9003263, 9003271,
9003327, 9003343, 9003347, 9003349, 9003377, 9003403, 9003409, 9003431, 9003437, 9003487, 9003497, 9003517, 9003529, 9003583, 9003607,
9003613, 9003619, 9003647, 9003667, 9003689, 9003703, 9003707, 9003713, 9003719, 9003721, 9003749, 9003767, 9003773, 9003779, 9003803,
9003809, 9003817, 9003821, 9003859, 9003871, 9003877, 9003889, 9003901, 9003959, 9003977, 9003997, 9004013, 9004019, 9004031, 9004057,
9004069, 9004087, 9004091, 9004097, 9004103, 9004111, 9004147, 9004189, 9004231, 9004249, 9004267, 9004277, 9004301, 9004319, 9004327,
9004343, 9004349, 9004351, 9004361, 9004367, 9004399, 9004451, 9004459, 9004469, 9004481, 9004507, 9004511, 9004519, 9004531, 9004561,
9004571, 9004591, 9004603, 9004621, 9004627, 9004643, 9004649, 9004669, 9004687, 9004691, 9004711, 9004739, 9004741, 9004771, 9004811,
9004829, 9004843, 9004889, 9004901, 9004907, 9004909, 9004921, 9004927, 9004939, 9004943, 9004969, 9004991, 9004999, 9005011, 9005021,
9005027, 9005039, 9005041, 9005047, 9005057, 9005081, 9005089, 9005093, 9005099, 9005119, 9005149, 9005173, 9005179, 9005197, 9005201,
9005219, 9005233, 9005237, 9005267, 9005279, 9005299, 9005327, 9005329, 9005371, 9005387, 9005401, 9005429, 9005473, 9005497, 9005509,
9005519, 9005539, 9005543, 9005551, 9005561, 9005621, 9005657, 9005663, 9005693, 9005713, 9005723, 9005729, 9005741, 9005743, 9005783,
9005839, 9005849, 9005873, 9005879, 9005881, 9005897, 9005917, 9005923, 9005947, 9005977, 9005989, 9006013, 9006031, 9006043, 9006061,
9006071, 9006077, 9006139, 9006157, 9006163, 9006169, 9006181, 9006191, 9006197, 9006199, 9006203, 9006253, 9006289, 9006293, 9006313,
9006359, 9006373, 9006377, 9006391, 9006397, 9006419, 9006433, 9006457, 9006493, 9006499, 9006509, 9006581, 9006587, 9006601, 9006611,
9006623, 9006631, 9006649, 9006653, 9006659, 9006671, 9006677, 9006707, 9006749, 9006769, 9006793, 9006797, 9006853, 9006859, 9006863,
9006883, 9006919, 9006929, 9006937, 9006953, 9006971, 9006973, 9006983, 9007001, 9007039, 9007043, 9007049, 9007051, 9007087, 9007111,
9007123, 9007151, 9007157, 9007171, 9007177, 9007181, 9007183, 9007189, 9007211, 9007213, 9007223, 9007231, 9007241, 9007259, 9007261,
9007277, 9007289, 9007291, 9007303, 9007309, 9007319, 9007333, 9007351, 9007357, 9007363, 9007373, 9007393, 9007441, 9007457, 9007477,
9007499, 9007507, 9007529, 9007541, 9007549, 9007553, 9007567, 9007571, 9007573, 9007619, 9007627, 9007631, 9007633, 9007643, 9007657,
9007679, 9007703, 9007709, 9007717, 9007727, 9007753, 9007759, 9007771, 9007781, 9007783, 9007793, 9007811, 9007849, 9007853, 9007861]
```

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 $\text{pgcd}(e, \phi_n) = 1$, c'est dire qu'ils sont premiers entre eux
- calcul de d inverse de module (ϕ_n) par l'algorithme d'Euclide étendu : $d * e \equiv 1 [\phi_n]$

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 créer 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 choisi qu'une seule 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 $a \bmod b \neq 0$ 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) : **$d * e \equiv 1 \pmod{\phi_n}$**

- Prendre une valeur de e arbitraire 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

$(n, e) \leftarrow \text{readkey}(\text{pub})$

$c \leftarrow (m^e \bmod n)$

return c

- Ecrire une fonction qui prend 3 paramètres le message à chiffrer qui est une chaîne de caractères vide au départ, l'exposant e tel que $1 < e < \phi(n)$, $n = p * q$
- Calculer le message chiffré c tel que $c = m^e \bmod n$

```
# 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

$(n, d) \leftarrow \text{readkey}(\text{pri})$

$m \leftarrow (c^d \bmod n)$

return m

- Ecrire une fonction qui prend 3 paramètres le message à déchiffrer qui est une chaîne de caractères vide au départ, d tel que $d * e \equiv 1 [\phi_n]$, $n = p * q$
- Calculer le message chiffré c tel que $m = c^d [n]$

```
# Déchiffrement du message
def dechiffrer(message_chiffre, d, n):
    i=0
    bloc=""
    message_dechiffre = ""
    while i != len(message_chiffre):
        bloc = bloc + message_chiffre[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)
##### Déchiffrement #####
else:
    message = raw_input('\nok, saisis le message que tu veux déchiffrer : ')
    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
```