# Code RSA

par Léo Peyronnet

Décembre 2022

## 1 Réponses Exercices

**Exercice 1**
$N = 391$, $E = 151$ et $D = 7$

1. Message reçu et crypté : $C = 17$
   Soit $M$ le message tel qu'envoyé (non crypté), alors :
   $M = C^D[N] = 17^7[391] = 204$.
2. On sait que $N = p \times q$ avec $p, q$ deux nombres premiers. On a donc :
   $391 = p \times q = 17 \times 23$ *(résultat obtenu avec le programme cf* **??***)*
   Nous pouvons donc déduire $\varphi(N)$ :
   $\varphi(N) = (p-1)(q-1) = 16 \times 22 = 352$
3. Nous connaissons la relation suivante : $E.D \equiv 1[\varphi(N)]$.
   Cette relation peut être vérifiée dans notre cas :
   $151 \times 7 \equiv 1[352] \leftrightarrow 151 \times 7[352] = 1$ *(vérifié avec le programme cf* **??***)*

**Exercice 2**

1. $N = 221$, $E = 11$ et $D = 35$
   (a) Soit $M = 112$ le message et $C$ le message crypté, alors :
   $C = M^E[N] = 112^{11}[221] = 122$
   (b) Soit $C = 78$ le message reçu et $m$ le message originel, alors :
   $M = C^D[N] = 78^{35}[221] = 65$
2. $p = 53$, $q = 71$
   (a) $N = 53 \times 71 = 3763$
   $\varphi(N) = 52 \times 70 = 3640$
   (b) $E = 307$ : $E < \varphi(N) \wedge pgcd(\varphi(N), E) = 1$
   — 307<3640
   — $pgcd(\varphi(N), E) = 3640 \times (-7) + 307 \times 83 = 1$
   $E$ est donc acceptable.
   $D = E^{-1}[\varphi(N)] = 83$
   (c) — Clé publique : $E = 307$ et $N = 3763$
   — Clé privée : $D = 83$ et $N$ *(déjà connu avec la clé publique)*
   (d) Les éléments restants sont $p$ et $q$. Étant les générateurs de $N$, ils doivent être dissimulés car ils sont de fait les détenteurs de l'asymétrie du code RSA. Pour rappel, afin de pouvoir décoder et lire RSA, il faut posséder $D$ qui est l'inverse modulaire de $E$ modulo $(p-1)(q-1)$.

Le processus pour déterminer $p$ et $q$ à partir de $N$ est gourmand en ressources (cf. **??**). Ainsi, plus $p$ et $q$ seront grands, plus le décodage par *"brute force"* demandera de ressources temporelles ou spatiales.

**Exercice 3**

$E = 257$, $N = 1073$, $D = 353$.

1. Chiffrer "METHODE" :
   Correspond à $12; 04; 19; 07; 14; 03; 04$.
   Soit regroupé par paquet de 3 : $120; 419; 071; 403; 04$.
   — $120^{257}[1073] = 589$
   — $419^{257}[1073] = 673$
   — $71^{257}[1073] = 238$
   — $403^{257}[1073] = 308$
   — $4^{257}[1073] = 1024$
   Nous avons donc : $589; 673; 238; 308; 1024$.
2. Déchiffrer $263; 115; 613; 10$ :
   — $263^{353}[1073] = 21$
   — $115^{353}[1073] = 724$
   — $613^{353}[1073] = 151$
   — $10^{353}[1073] = 914$
   Nous avons donc : $21; 724; 151; 914$.
   Soit regroupé par paquet de 2 : $21; 07; 24; 15; 19; 14$.
   Le message est : "CRYPTO".
3. Chiffrer "AVEZVOUSBIENREUSSI" :
   Correspond à $00; 21; 04; 25; 21; 14; 20; 18; 01; 08; 04; 13; 17; 04; 20; 18; 18; 08$.
   Soit regroupé par paquet de 3 : $002; 104; 252; 114; 201; 801; 080; 413; 170; 420; 181; 808$.
   — $2^{257}[1073] = 32$
   — $104^{257}[1073] = 916$
   — $252^{257}[1073] = 546$
   — $114^{257}[1073] = 983$
   — $201^{257}[1073] = 403$
   — $801^{257}[1073] = 1001$
   — $80^{257}[1073] = 709$
   — $413^{257}[1073] = 857$
   — $170^{257}[1073] = 716$
   — $420^{257}[1073] = 1034$
   — $181^{257}[1073] = 567$
   — $808^{257}[1073] = 919$
   Nous avons donc : $32; 916; 546; 983; 403; 1001; 709; 857; 716; 1034; 567; 919$.
4. Déchiffrer $1019; 35; 567; 36; 384; 703; 99; 59$ :
   — $1019^{353}[1073] = 180$
   — $35^{353}[1073] = 13$
   — $567^{353}[1073] = 181$
   — $36^{353}[1073] = 517$
   — $384^{353}[1073] = 140$
   — $703^{353}[1073] = 111$
   — $99^{353}[1073] = 41$
   — $59^{353}[1073] = 204$

Nous avons donc : $180; 13; 181; 517; 140; 111; 41; 204$.
Soit regroupé par paquet de 2 : $18; 00; 13; 18; 15; 17; 14; 01; 11; 04; 12; 04$.
Le message est : "SANSPROBLEME".

5. Déchiffrer $553; 813$ :
   — $553^{353}[1073] = 36$
   — $813^{353}[1073] = 813$

   Nous avons donc : $36; 813$.
   Soit regroupé par paquet de 2 : $03; 06; 08; 13$.
   Le message est : "DGIN".

# 2 Présentation des programmes

## 2.1 Algorithmes relatifs à l'exercice 1 et 2

### 2.1.1 erathosthene()

Fonction relative au Crible d'Eratosthène. Renvoie une liste $r$ contenant l'ensemble des entiers premiers $P \subset [0, N[$.

```
import math
def erathosthene(n):
    t=[] # liste booléenne de taille n
    r=[] # liste des entiers premiers dans l'intervalle
        [0,n[
    t+=[False] #0 n'est pas premier
    t+=[False] #1 n'est pas premier
    for i in range(2,n): #2 <= i < n
        t+=[True] #initialisation des n-2 autres entiers
            (2,3,4,...,n-1)

    for i in range(2,int(math.sqrt(n))): #2<= i < sqrt(n)
        j=2*i #j appartient à l'ensemble des multiples de
             i de 0 à n-1
        while j<len(t):
            t[j]=False # j n'est pas premier
            j=j+i # prochain multiple de i

    for i in range(2,n): #2 <= i < n
        if t[i]: #si i est premier
            r+=[i] #i appartient à r
    return r #renvoie r
```

### 2.1.2 scan()

Fonction de "brute force". Permet de déduire $p$ et $q$ à partir de $N$. Pour ce faire, elle génère une matrice triangulaire supérieure $A = (i \times j)_{j \geqslant i; i,j \in P}$. Le choix de la triangulaire se justifie par le fait que la multiplication est commutative. Cela permet donc d'éviter les doublons $(i \times j)$ et $(j \times i)$.

```
def scan(n):
    tab=erathosthene(n) #import des nombres premiers
        compris entre 0 et n exclu
    s=len(tab)
    for i in range(s): # 0 <= i < len(tab)
        for j in range(i,s): # pyr <= y < len(tab)
            if tab[i]*tab[j]==n: # si le produit des deux
                nombres entiers d'indices i et y est égal
                à n
                return [tab[i],tab[j]] # renvoie p et q
    return False
```

### 2.1.3 euclideEtt()

Fonction relative à l'algorithme d'Euclide Étendu. Renvoie l'identité de Bézout $(a.u + b.v = pgcd(a, b))$ sous forme d'une liste de trois entiers. r1 : pgcd ; u1 : u et v1 : v.

```
def euclideEtt(a,b):
    r1=b
    r2=a
    u1=0
    v1=1
    u2=1
    v2=0
    while r2!=0:
        q=r1//r2

        r3=r1
        u3=u1
        v3=v1

        r1=r2
        u1=u2
        v1=v2

        r2=r3-q*r2
        u2=u3-q*u2
        v2=v3-q*v2
    return [r1,u1,v1] # retourne l'identité de bézout
```

### 2.2 Algorithmes relatifs à l'exercice 3

*(Je me suis rendu compte trop tard que les algorithmes ci-dessous n'étaient absolument pas demandés. Cependant, je me suis permis de quand même les laisser dans ce rapport car leur présentation peut rester intéressante.)*

#### 2.2.1 rsa()

Fonction de chiffrage RSA d'une liste de paquets. Soit $x$ un paquet ; $rsa(x) = x^{\text{cle}}[N]; \text{cle} \in \{E, D\}$, $E$ pour chiffrement, $D$ pour déchiffrement.

```
def rsa(message,cle,n):
    for i in range(len(message)):
        message[i]=int(message[i])**cle%n
```

#### 2.2.2 alphatonum()

Fonction retournant une liste de paquets correspondant à un mot par rapport à un alphabet.

```
def alphatonum(mot,alpha):
    li=[] #liste de chiffres par bloc de 3
    temp="" #mémoire temporaire
    c=0 #état
    for i in range(len(mot)): #0 <= i < len(mot)
        y=0 #position de la i-ème lettre du mot dans
            alpha (alphabet)
        while mot[i]!=alpha[y]: #tant que la i-ème lettre
             ne correspond pas à une lettre de l'alphabet
            y+=1 #y est incrémenté

        y=str(y)
        if len(y)==1: # si y est un chiffre (0,1,2,...,9)
            y="0"+y

        if c==0: #état 0 (mémoire temp vide)
            temp=y
            c+=1 #passage à l'état 1
        elif c==1: #état 1 (mémoire temp: 2 chiffres)
            li+=[temp+y[0]]
            temp=y[1]
            c+=1 #passage à l'état 2
        elif c==2: #état 2 (mémoire temp: 1 chiffre)
            li+=[temp+y]
            temp=""
            c=0 #retour à l'état 0
    if temp!="": #si la mémoire temp n'est pas vide
        li+=[temp] #flush
    return li #renvoie li
```

### 2.2.3 alphatonum()

Fonction réciproque à alphatonum(). Peut se schématiser par la vérification de trois paramètres :

1. la taille du paquet reçu $\in \{1, 2, 3\}$.
2. l'état de la mémoire temporaire $\in \{0, 1\}$
3. la cohérence de l'indice calculé : indice calculé $\geqslant$ taille de l'alphabet ?

*Cette fonction ayant une syntaxe lourde, il est préférable de pouvoir la lire sur un format adéquat. Ainsi, j'ai joint à ce rapport le lien du dépôt distant github de ce projet :* ***`https://github.com/LaiPe/crypto-rsa`***. *La fonction se trouve dans le sous-répertoire "programmes", dans le fichier "exo3.py"*