# TP1 RS40 2021

**PROBLEMATIQUE**

Bob souhaite envoyer un secret à Alice donc il doit utiliser le système de chiffrement RSA et Alice doit aussi vérifier que le message provient bien de Bob. Bob doit également faire le processus plus efficace et plus sûr.

**Professeur :** Abdeljalil ABBAS-TURKI
**Réalisé par :** Yuan Cao

**Sommaire**

## INTRODUCTION

Ce TP considère un message envoyé de Bob vers Alice, où chacun dispose d'une paire de clés (publique, privée) pour le chiffrement RSA. Bob chiffre le message avec la clé publique d'Alice. Il procède aussi à la signature de l'empreinte numérique du message avec sa clé privée. Alice reçoit le message le déchiffre et vérifie la signature de Bob (Ce n'est pas sécurisé).

Le programme a besoin de quelques améliorations:

1. Améliorer le processus de vérification de la signature de Bob (Utilisation d'une fonction de hachage approriée de la librairie hashlib : par exemple sha256).
2. Augmenter la taille des messages échangés.
3. Réduire le temps de calcul à l'aide du théorème du reste chinois.
4. Utiliser le CBC pour le chiffrement des blocs.

## 1.Présentation du projet

**Description**

1. Algorithme d'Euclide généralisé

L'algorithme d'Euclide généralisé est un algorithme qui permet de calculer $d = \text{pgcd}(a, b)$, ainsi que deux entiers $u$ et $v$ tels que $au + bv = d$, c'est-à-dire une relation de Bézout, de façon simultanée. On suppose $a \geq b > 0$.

On calcule une suite $r_0; r_1; \ldots r_k; \ldots$ de restes obtenus par divisions euclidiennes successives à partir de $r_0 = a$, $r_1 = b$. En effet, l'algorithme d'Euclide montre qu'il existe un rang $n$ tel que $r_n \neq 0$ et $r_{n+1} = 0$ :

$$\begin{cases} r_0 = r_1 q_1 + r_2 \\ r_1 = r_2 q_2 + r_3 \\ \vdots \\ r_{n-2} = r_{n-1} \times q_{n-1} + r_n \\ r_{n-1} = r_n \times q_n + 0 \end{cases}$$

Ainsi, pour calculer $u$ et $v$, on utilise deux autres suites d'entiers $u_k$ et $v_k$ définies de la façon suivante : on pose $u_0 = 1$, $v_0 = 0$, $u_1 = 0$, $v_1 = 1$ et pour $k \geq 1$

$$u_k = u_{k-2} - q_{k-1} u_{k-1}$$
$$v_k = v_{k-2} - q_{k-1} v_{k-1}$$

On a alors :

$$\boxed{d = \text{pgcd}(a, b) = r_n \quad \text{et} \quad au_n + bv_n = d}$$

2. Signature RSA

Tout comme la signature manuscrite que l'on appose sur un document, la signature électronique (ou numérique) permet de valider la conformité d'un document. Elle ne doit pas être confondue avec la signature numérisée faite par exemple avec un stylet sur une tablette même si cette dernière est aussi reconnue légalement depuis 2010. Un système de signature numérique requiert la mise en place de sécurités informatiques faisant appel à la cryptographie.

La signature électronique permet à quelqu'un de prouver qu'il est celui qu'il prétend être ou qu'il a le droit d'accéder à un service. C'est la fonction réalisée par exemple lors d'une connexion à un ordinateur à l'aide d'un mot de passe où lorsque l'on tape le code secret de sa carte bancaire.

3. Sha-256

Le Sha-256 est une fonction de l'algorithme Sha-2 (au même titre que les versions 384, 512, et plus récemment 224), qui est similaire au Sha-1, lui-même tiré du Sha-0. Cet algorithme de "hashage" a été crée par la NSA pour répondre au problème de sécurité posé par le Sha-1, depuis la découverte théorique de collisions à 2^63 opérations. L'algorithme accepte en entrée un message de longueur maximum 2^64 bits et produit un hash, ou condensé, de 256 bits. Le sha256 semble de plus en plus courant en remplacement du md5, notamment car il propose un bon équilibre entre espace de stockage en ligne et sécurité. Comme les autres fonctions cryptographiques de sa famille, Sha-256 est unilatéral et on ne peut retrouver le message originel avec le seul hash sha256.

4. Théorème du reste chinois

Le théorème des restes chinois permet d'accélérer les calculs de déchiffrement de la méthode RSA, une méthode cryptographique asymétrique, où clef de chiffrement et clef de déchiffrement sont distinctes. Elle est très utilisée sur Internet pour l'authentification et l'échange de clefs

symétriques ainsi que dans les cartes bancaires. Son principe repose sur les nombres premiers et l'arithmétique modulaire.

5. Le mode de chiffrement avec chaînage de blocs

Le mode de chiffrement avec chaînage de blocs, en anglais "Cipher Block Chaining" (CBC), est un des modes les plus populaires.

Il offre une solution à la plupart des problèmes du mode ECB. Le chaînage utilise une méthode de rétroaction comme le résultat du chiffrement du bloc précédent est réutilisé pour le chiffrement du bloc courant. Plus précisément, l'opérateur binaire XOR est appliqué entre le bloc actuel de texte en clair et le bloc précédent de texte chiffré et on applique ensuite l'algorithme de chiffrement au résultat de cette opération.

## 2. Conception

1. Exponentiation modulaire rapide

On utilise strictement ce pseudo-code :

**Algorithme 1** Calcul de $y = x^p \mod (n)$

**Entrées:** $n \geq 2, x > 0, p \geq 2$
**Sortie:** $y = x^p \mod (n)$

**Début**
$p = (d_{k-1}; d_{k-2}; \cdots; d_1; d_0)$ % Écriture de $p$ en base 2
$R_1 \leftarrow 1$
$R_2 \leftarrow x$

**Traitement**
**Pour** $i = 0; \cdots; k-1$ **Faire**
  **Si** $d_i == 1$ **Alors**
    $R_1 \leftarrow R_1 \times R_2 \mod (n)$ % Calcul de la colonne 4 du tableau si le bit est 1
  **Fin Si**
  $R_2 \leftarrow R_2^2 \mod (n)$ % carré modulo $n$ de la colonne 3 du tableau
**Fin Pour**

```
def home_mod_expnoent(x,y,n): #exponentiation modulaire
    num_b = str(bin(y))
    tab_l = list(num_b) #ici inclure '0x'
    tab = tab_l[2:] #commencer par l'élément après '0x'
    tab.reverse()
    length = len(tab) #longeur de tab

    r1 = 1
    r2 = x

    for i in range(0, length):
        if tab[i] == '1':
            r1 = r1 * r2 % n
        r2 = r2 * r2 % n
    return r1
```

Mais on peut réduire le temps de calcul à l'aide du théorème du reste chinois.

On utilise strictement ce pseudo-code :

Algorithme de calcul $m=cd\%n$ en utilisant CRT

Calcul préalable :

1- Avec $n=xixj$ prendre $q=xi$ et $p=xj$ tel que $xi<xj$

2- Calculer $q-1$ dans $\mathbb{Z}p$

3- Calculer $dq=d\%(q\text{-}1)$ et $dp=d\%(p\text{-}1)$

Ces calculs sont réalisés qu'une seule fois et les valeurs de $q-1$, $dq$ et $dp$ sont gardés.

A la réception d'un message $c$, effectuer les opérations suivantes :

1- Calculer $mq=cdq\%q$ et $mp=cdp\%p$

2- Calculer $h=((mp-mq)q-1)\%p$

3- Calculer $m=(mq+h\times q)\%n$

```
def CR_Theorem(c,d,n1,n2): #Chinese remainder theorem
      if(n1 <= n2):
          xi = n1
          xj = n2
      else:
          xi = n2
          xj = n1
      q = xi
      p = xj
      q_inverse = home_ext_euclide(p, q)
      dq = d % (q-1)
      dp = d % (p-1)

      mq = home_mod_expnoent(c, dq, q)
      mp = home_mod_expnoent(c, dp, p)

      h = ((mp - mq)*q_inverse) % p
      m = (mq + h * q) % (n1*n2)
      return m
```

2. Algorithme d'Euclide généralisé

```
# def home_ext_euclide(y,b): #algorithme d'euclide étendu pour la recherche de l'exposant secret
#     n = 1
#     q = [0]; u = [0]*3
#     left=y; right=b
#     while right % left != 0:
#         if n == 1: #lorsqu'il n'y pas d'élément dans la liste
#             q[0] =right//left
#             n+=1
#             mid = left
#             left = right % left
#             right = mid
#         else:
#             q.append(right//left)
#             mid = left
#             left = right % left
#             right = mid
#     q.append(right//left)
#     u[0]=1;u[1]=0
#     n = len(q)
#     for i in range(2, n+1):
#         u[i] = u[i-2] - q[i-2] * u[i-1]  #ici c'est q[i-2] pas i-1
#         u.append(0)
#     u.pop()
#     return u[i] % y
```

Cette fonction qui est écrite par moi marche très bien mais dans ce TP on utilise la fonction donnée par le professeur :

```python
def home_ext_euclide(a,b): #algorithme d'euclide étendu pour la recherche de l'exposant secret
    (r, nouvr, t, nouvt) = (a, b, 0, 1)

    while(nouvr > 1) :
        q = (r//nouvr)
        (r, nouvr, t, nouvt) = (nouvr, r%nouvr, nouvt, t-(q*nouvt))

    return nouvt % a
```

3. Signature RSA et Sha-256

On suivre le processus ci-dessous:

Données: Alice (nA,eA,dA). Bob (nB ,eB ,dB ).

Alice souhaite envoyer un message M à Bob d'une manière secrète mais elle veut aussi que Bob soit sûr que le message vient bien d'elle. Pour ceci, elle effectue les opérations suivantes :

1. Elle génère d'abord la signature du message par sa clef secrète comme :

S = home_mod_expnoent(M, dA, nA)

On aura remarqué qu'elle est la seule à pouvoir générer cette signature étant la seule à posséder dA.

2. Elle chiffre ensuite S et M par la clef publique de Bob comme :

Cs = home_mod_expnoent(S, eB, nB)

C = home_mod_expnoent(M, eB, nB)

3. Le document signé est alors le couple (C,Cs ) et est envoyé à Bob.

Lorsque Bob reçoit le document signé (C,Cs ), il déchiffre d'abord Cs par sa clef secrète dB pour trouver la signature S du message:

S = home_mod_expnoent(Cs, dB, nB)

Ensuite, il vérifie l'authenticité de la signature par récupération du message M par la clef publique d'Alice (nA,eA) et sa clef secrète dB comme :

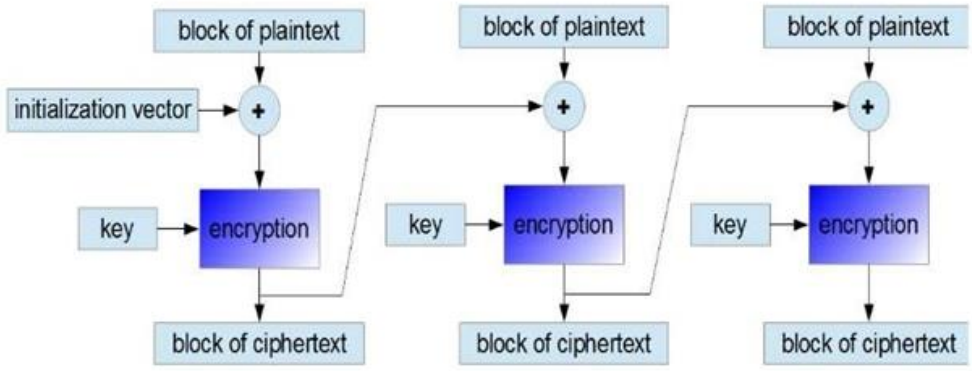
home_mod_expnoent(C, dB, nB) = M = home_mod_expnoent(S, eA, nA)

Si, au contraire, home  mod  expnoent(C, dB, nB) != home  mod  expnoent(S, eA, nA) alors la

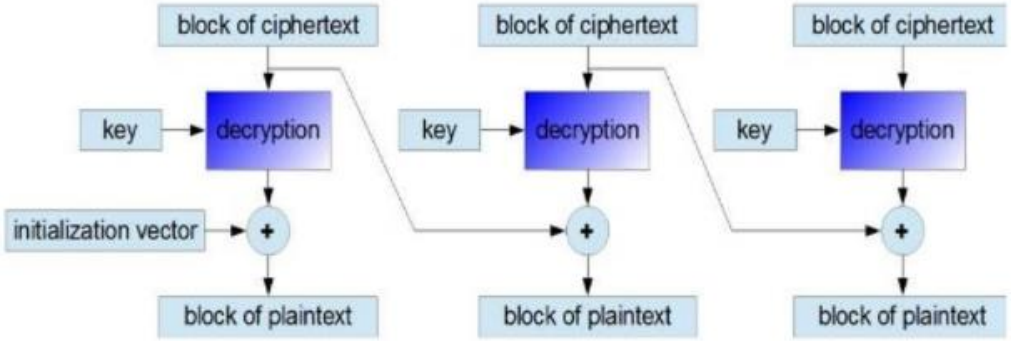
```python
#utiliser signature RSA

#sha256 du message
print("voici le hash en nombre décimal ")
Bhachis0=hashlib.sha256(secret.encode(encoding='UTF-8',errors='strict')).digest()
Bhachis1=binascii.b2a_uu(Bhachis0)
Bhachis2=Bhachis1.decode() #en string
Bhachis3=home_string_to_int(Bhachis2)

print("**********************************************************************")
print("On utilise la signature RSA pour obtenir la signature")
print("voici la signature avec la clé privée de Bob")

S = CR_Theorem(Bhachis3, db, x1b, x2b)
Cs = CR_Theorem(S, ea, x1a, x2a)
C =  CR_Theorem(Bhachis3, ea, x1a, x2a)
```

```python
designe = CR_Theorem(Cs, da, x1a, x2a)
print(designe)

if (CR_Theorem(C, da, x1a, x2a) == CR_Theorem(S, eb, x1b, x2b) == Bhachis3):
    print("La signature est valide\n\nTout se passe très bien\nFélicitation!!!")
else:
    print("La signature n'est pas valide")
```

4. Le mode de chiffrement avec chaînage de blocs




*Cryptage en mode CBC*




*Décryptage en mode CBC*

Donc on augmente la taille des messages échangés par principalement utiliser cette méthode CBC et aussi agrandir la taille de key.

Attention :
On doit retourner les blocs chiffrés au lieu des nombres chiffrés dans la fonction **CBC_chiffrement** parce que la taille des blocs non chiffrés (qui est 100 caractères) et la taille des blocs chiffrés (qui est plus grand que 100 caractères) sont totalement différentes. Donc il va provoquer un problème quand on coupe les messages chiffrés avec chaque bloc de taille 100 dans le déchiffrement. C'est pourquoi on donne directement les blocs chiffrés au processus de déchiffrement.

On suivre les étapes suivantes :

Obtenir le vecteur d'initialisation

```
def get_vec_ini():
    vec_ini = [1]#creer le tableau avec le premier nombre 1, parce que si le premier est 0, on va le perdre dans la transformatin après
    i = 1
    while(i<100):
        i = i + 1
        vec_ini.append(random.randint(0, 1))#mettre le nombre aléatoir dans le tableau l'un après l'autre
    vec_int = int(''.join(str(i) for i in vec_ini))#transformer le tableau en un int(par exemple [1,0,1,0]=>1010)
    return vec_int
```

Couper le message long et obtenir les blocs avec la même taille

```
def getBloc_chif(num_sec):#ici pour avoir la même taille de chaque bloc on met tous 0 pour le reste

    num_sec=str(num_sec)
    list_b=list(num_sec)
    list_len = len(list_b)#obtenir le longeur de message(par exemple 4232424=>7 elements)

    num_bloc=0
    num_in_bloc=0
    if(list_len%100==0):                                        #par exemple, si length=100, on a besoin 1 bloc
        bloc = [['0' for i in range(100)] for j in range(list_len//100)]#mais list_len//100+1=2, donc il y a un bloc inutile
    else:
        bloc = [['0' for i in range(100)] for j in range(list_len//100+1)]#initialisation de bloc avec tous '0', on obtient
                                                                  #par exemple [['0','0',...],['0','0',...],['0','0',...]]
    i = 0
    while(i<list_len):
        if(num_in_bloc < 100):
            bloc[num_bloc][num_in_bloc] = list_b[i]#mettre les éléments dans chaque bloc
            num_in_bloc = num_in_bloc + 1
            i = i + 1
        else:
            num_in_bloc = 0#nouveau bloc si ce bloc est plein(100 éléments)
            num_bloc = num_bloc + 1
    bloc[num_bloc] = bloc[num_bloc][0:num_in_bloc]#supprimer les element "0" dans le dernier bloc, parce que
                                                  #dans le chiffrement 1000 et 1 est totalement différent
    for i in range(num_bloc+1):                    #Dans chaque grand bloc, transformer des sous-blocs avec les éléments string en un int
        bloc[i] = int(''.join(i1 for i1 in bloc[i]))#par exemple [['3','9',...],['6','3',...],['1','2',...]] => [39..., 63..., 12...]

    return bloc
```

Faire le chiffrement/déchiffremen

```
def CBC_chiffrement(vec_ini, num_sec, e, n1, n2):
    bloc_secret = []

    bloc = getBloc_chif(num_sec)#obtenir les blocs avec la même taille
    for i in range(len(bloc)):
        if(i==0):
            bloc_secret.append(CR_Theorem(vec_ini^bloc[0], e, n1, n2))
        else:
            bloc_secret.append(CR_Theorem(bloc_secret[i-1]^bloc[i], e, n1, n2))


    chif_num = int(''.join(str(i1) for i1 in bloc_secret))#transformer des blocs avec les éléments int en un int
                                                          #par exemple [42...,53...,12...] => 42...53...12...
    print("voici le message chiffré avec la publique d'Alice : ")
    print(chif_num)
    return bloc_secret#retourner les blocs de secret pour que l'on puisse les utiliser directement dans le déchiffrement
```

```
def CBC_dechiffrement(vec_ini, bloc_secret, d, n1, n2):
    bloc_claire = []

    for i in range(len(bloc_secret)):
        if(i==0):
            bloc_claire.append(vec_ini^CR_Theorem(bloc_secret[0], d, n1, n2))
        else:
            bloc_claire.append(CR_Theorem(bloc_secret[i], d, n1, n2)^bloc_secret[i-1])

    dechif_num = int(''.join(str(i1) for i1 in bloc_claire))#transformer des blocs avec les éléments int en un int
                                                            #par exemple [42...,53...,12...] => 42...53...12...
    return dechif_num
```

### 3. Exécution

1. Entrer un petit secret :

```
In [2]: runfile('C:/Users/18019/Desktop/rs40/tp/rsa_correct/RSA_B_A(avec CBC).py', wdir='C:/Users/18019/Desktop/rs40/tp/rsa_correct')
Vous êtes Bob, vous souhaitez envoyer un secret à Alice
voici votre clé publique que tout le monde a le droit de consulter
n = 9533618719394850296974626436771807520057128045304157550004315845554227656385060604729126639017031664279273690558284164 9
exposant : 23
voici votre précieux secret
d = 2590657260705122363308322401296686826102480447093521073370720666847858771619456727899068826383831860656494512327043 27
************************************************************
Voici aussi la clé publique d'Alice que tout le monde peut conslter
n = 4399846286521073166957155075986701390416232209164342907865545304288430924740091279139414325435107602652250702455529 1327
exposent : 17
************************************************************
il est temps de lui envoyer votre secret
************************************************************

appuyer sur entrer

donner un secret : J'aime RS40.
************************************************************
voici la version en nombre décimal de  J'aime RS40.  :
142945859911780841321283684 58
voici le message chiffré avec la publique d'Alice :
269283563174624037316772771054737003840062199494693651221115560419733579277761718574983061195809268787545255213339350 11
voici le hash en nombre décimal
************************************************************
On utilise la signature RSA pour obtenir la signature
voici la signature avec la clé privée de Bob
249032654186126281741626755903753100006962049488318474186151299053150035788652873457433534134589963082688279212740843 11
************************************************************

appuyer sur entrer
************************************************************
************************************************************
Alice déchiffre le message chiffré ce qui donne
J'aime RS40.
************************************************************
************************************************************
Alice déchiffre la signature de Bob
 249032654186126281741626755903753100006962049488318474186151299053150035788652873457433534134589963082688279212740843 11
 ce qui donne  en décimal
249032654186126281741626755903753100006962049488318474186151299053150035788652873457433534134589963082688279212740843 11
La signature est valide

Tout se passe très bien
Félicitation!!!
```

2. Entrer un grand secret :

```
In [5]: runfile('C:/Users/18019/Desktop/rs40/tp/rsa_correct/RSA_B_A(avec CBC).py', wdir='C:/Users/18019/Desktop/rs40/tp/
rsa_correct')
Vous êtes Bob, vous souhaitez envoyer un secret à Alice
voici votre clé publique que tout le monde a le droit de consulter
n =
95336187193948502969746264367718075200571280453041575500043158455542276563850606047291266390170316642792736905582841649
exposant : 23
voici votre précieux secret
d = 25906572607051223633083224012966868261024804470935210733707206668478587716194567278990688263838318606564945123270432 7
**********************************************************************
Voici aussi la clé publique d'Alice que tout le monde peut conslter
n =
43998462865210731669571550759867013904162322091643429078655453042884309247400912791394143254351076026522507024555291327
exposent : 17
**********************************************************************
il est temps de lui envoyer votre secret
**********************************************************************

appuyer sur entrer

donner un secret : Le système RSA est un système utilisé dans plusieurs applications comme les télécommunications, les
transactions sécurisée sur Internet, la finance et les cartes a puce.
**********************************************************************
voici la version en nombre décimal de  Le système RSA est un système utilisé dans plusieurs applications comme les
télécommunications, les transactions sécurisée sur Internet, la finance et les cartes a puce.  :
178153832557611691616499169431107529749275061837131717795987322286087701453515716897210958443516388486214134518817398845706703891350661843479453204902034859251149567399484268875389301668781946198361721932328992661706583125108421318098247983913710541082575201532858765881593779409558509909166317802887204395423236257133988374798977126284088217231629547049077321430358991430504257225794587594092077944647447 80
voici le message chiffré avec la publique d'Alice :
266586655113982603899252195177077902066117562984881766914698275020307242202200917214962708156701285531718956797176440453325435795376424423018238820087818552205796836497720818274077944416503589285476052216636159167142577935331384343631517047117684774181132168114615466312094806042770870609866037878809633362321933580381705521270056923049583966281442738394002433606611757754008600706793946555135674229712769929913193549467865003263224357096793610497630708514443246620453531935720783725830537567647287267467095945424996626621516103232581578892181323300800516274178942708186165661493973756058714258
voici le hash en nombre décimal
**********************************************************************
On utilise la signature RSA pour obtenir la signature
voici la signature avec la clé privée de Bob
30188107349607957240496616309950987651424249258495362269585768300411593835577025486621727459900408226567451430329014829
**********************************************************************

appuyer sur entrer
**********************************************************************
**********************************************************************
Alice déchiffre le message chiffré ce qui donne
Le système RSA est un système utilisé dans plusieurs applications comme les télécommunications, les transactions sécurisée
sur Internet, la finance et les cartes a puce.
**********************************************************************
**********************************************************************
Alice déchiffre la signature de Bob
 30188107349607957240496616309950987651424249258495362269585768300411593835577025486621727459900408226567451430329014829
 ce qui donne  en décimal
30188107349607957240496616309950987651424249258495362269585768300411593835577025486621727459900408226567451430329014829
La signature est valide

Tout se passe très bien
Félicitation!!!
```

## 4. Conclusion

Ce projet me permet de mettre en pratique les connaissances acquises en cours de l'UV RS40. Grâce à celui, je me familiarise avec la notion de sécurité numérique et cryptographie.
Mais il y a encore un problème dans le code, c'est quand on entre un très grand secret, l'application nous donne un problème d'affichage des caractères(mojibake) probablement à cause du maximum caractère la fonction int_to_string/string_to_int peut transformer. J'ai essayé les fonctions bytes_to_long/long_to_bytes dans le package Pycryptodome/Crypto.Util.number et beaucoup d'autres façons pour le remplacer mais ça ne marche pas donc je me permets de l'améliorer par couper deux fois le message par exemple.