Brevet Noa
Buchy–Pétard Kenzo

Partie 1 – Cryptographie : le chiffrement RSA

Demonstration φ(n)=(p-1)(q-1) :

Soient p et q deux nombre premiers
et n = p*q
Soient A l'ensemble des entiers naturels inférieur à n,
L l'ensemble des entiers naturels inférieur a n et non premier avec n
et φ(n) le cardinal de l'ensemble des entiers naturels inférieur à n et premier avec n.
Par définition φ(n) = Card(A) - Card(L)
or Card(A) = n-1
donc φ(n) = n-1 -Card(L) = pq - 1 -Card(L)
On cherche Card(L) :
Soit k∈L, par définition k est un multiple de p ou de q et k < n
Donc k = a * q ou k = b * p avec a < p, b < q des entiers naturels
Donc Card{k∈N | k = a*q ∧ k∈L ∧ a<p } = p-1 et Card{k∈N | k = b*p ∧ k∈L ∧ a<q } = q-1
Donc Card(L) = p-1 + q-1 or φ(n) = pq - 1 - Card(L)
Donc φ(n) = pq - 1 - p + 1 -q + 1 = pq -p -q +1 or (p-1)*(q-1) = pq -p -q +1
Donc φ(n)=(p-1)(q-1)

1) Pour la fonction list_prime, nous avons créé 2 boucles, la première avec une variable "i" qui commence à 2 jusqu'à un n donné, dans cette première boucle, nous avons initialisé une variable test égal à "false". La deuxième boucle est comprise dans la première et sa variable "y" varie entre 2 et i. Si i%y est égal à 0 alors test est égal à "True" et la condition "if" s'estompe. Si à la fin de la deuxième boucle, test vaut toujours "False" alors le nombre i est ajouté à la liste "l" qui contient déjà 1.

2) Pour la fonction extended_gcd, nous avons remarqué qu'elle est la même que celle d'Euclide étendue vu en cours de math discrète donc nous l'avons importé.

3) La fonction "key_creation" doit renvoyer en sortie un nombre n, une clé publique notée pub et une clé privé notée priv. Concernant le nombre n, il est obtenu en multipliant deux nombres premiers p et q différents compris entre 2 et 1000 . Pour cela, nous avons fait appel à la fonction "choice" qui choisit un nombre au hasard dans la liste renvoyée par "list_prime(1000)". Ensuite, nous initialisons une variable phi = (p-1)*(q-1). Nous créons ensuite une variable test égale à "False". Nous bouclons ensuite tant que "test==False", dans cette nous essayons de trouver un entier e entre entre 3 et 200 qui est premier a phi, donc il faut que la fonction "pgcd(phi,e)" que nous avons créé en amont soit égal à 1. Quand il est égal à 1, la clé publique pub est égale au e choisi et la variable test devient égal à true donc la boucle s'arrête. Ensuite, pour la clé privé, nous utilisons la fonction "extended_gcd(pub,phi)", nous avons compris que nous avons besoin de la deuxième valeur de sortie de cette fonction notée "d". "Priv" est donc égale à "d%phi".

4) Pour la fonction “encryption”, nous avons dans un premier temps créer une fonction “conv_text”.

```
73  def conv_text(msg) :
74          crypt = []
75          crypt2 = ""
76          crypt3 = []
77          for i in range(0,len(msg)) :
78                  crypt.append(str(ord(msg[i])))
79                  if len(crypt[i])<3 :
80                          crypt[i] = str(0)+crypt[i]
81                  crypt2=crypt2+crypt[i]
82          for i in range(0,len(crypt2),4) :
83                  crypt3.append(crypt2[i:i+4])
84          if len(crypt3[len(crypt3)-1])<4 :
85                          for y in range (0,4-(len(crypt3[len(crypt3)-1]))) :
86                                  crypt3[len(crypt3)-1] = crypt3[len(crypt3)-1] + str(0)
87          return crypt3
88
```

Ligne 77/81 → Nous avons convertis chaque caractère en ascii, la condition “if” permet de rajouter un “0” pour les nombres de moins de 3 chiffres. Nous faisons cela pour que chaque caractère soit codé avec un nombre à exactement 3 chiffres, pour pouvoir reconnaître les caractères par la suite.
Ligne 82/83 → Nous séparons la suite de chiffres obtenu dans “crypt2” en groupe de nombres composé de 4 chiffres. Nous séparons par 4 et non par 3 pour éviter qu’il soit trop simple pour un “espion” de pirater nos messages codés et de les comprendre. Si nous avions divisé par 3, il suffit de voir quel nombre se répète le plus souvent pour comprendre que cela signifie la lettre “e”, car celle-ci est la plus utilisée dans la langue française.
Ligne 84/86 → Si crypt2 n’a pas une longueur qui est facteur à 4, alors dans le tableau “crypt3”, le dernier élément n’est pas composé d’une chaîne de caractère de longueur 4 et cela fausse ensuite la suite du programme. C’est pour cela que nous rajoutons des “0” à la fin de cette dernière chaîne de caractère.
Passons à la fonction “encryption” ci-jointe :

```
90  def encryption(n,pub,msg) :
91          msg1 = conv_text(msg)
92          chiffrer = []
93          for i in range(0,len(msg1)) :
94                  chiffrer.append(str(((int(msg1[i])**pub)%n)))
95          return chiffrer
96
```

Tout d’abord, nous utilisons la fonction “conv_text” en prenant comme argument le message qui doit être transmis. Ensuite pour les lignes 93 et 94, nous créons une boucle qui sert à prendre chaque chaîne de caractère du tableau renvoyée par “conv_text”, nous transformons chaque string en int pour pouvoir appliquer la formule qui sert à crypter les nombres, nous perdons donc les “0” qui pouvaient être présent au début de chaque string. Nous transformons ensuite chaque nombre trouvé en string et les ajoutons au tableau “chiffrer” qui est appliqué en sortie de la fonction.

5) Nous allons maintenant vous présenter notre fonction “decryption” :

```
98  def decryption(n, priv,msg) :
99          dechiffrer = []
100         dechiffrer2 = ""
101         dechiffrer3 = []
102         dechiffrer4 = ""
103         a = 0
104         for i in range(0, len(msg)) :
105                 dechiffrer.append(str((int(msg[i])**priv)%n))
106                 while len(dechiffrer[i])<4 :
107                         dechiffrer[i] = str(0) + dechiffrer[i]
108                 dechiffrer2 = dechiffrer2 + dechiffrer[i]
109         for i in range(0,len(dechiffrer2),3) :
110                 dechiffrer3.append(dechiffrer2[i:i+3])
111         if int(dechiffrer3[len(dechiffrer3)-1])==0 :
112                 dechiffrer3 = dechiffrer3 [:len(dechiffrer3)-1]
113         while a<len(dechiffrer3) :
114                 dechiffrer4 = dechiffrer4 + chr(int(dechiffrer3[a]))
115                 a = a+1
116         return dechiffrer4
```

Pour les lignes 104/105, nous voulons retrouver le tableau de nombres que nous avions dans “msg1” de la fonction “encryption”. Pour cela nous créons une boucle qui prend chaque string du résultat de “encryption”. Chaque string est converti en int, nous utilisons le calcul expliqué dans le pdf pour retrouver le nombre de base de “msg1”. Les lignes 106/107 permettent de rajouter les “0” perdus dans la fonction “encryption”. Ensuite, à la ligne 108, nous concaténons tous les nombres obtenus. Les lignes 109/110 séparent en groupe de nombres de 3 chiffres la concaténation obtenue dans “dechiffrer2”. Cela a pour but de retrouver notre message de départ en ascii. Les lignes 111/112 ont pour but de supprimer les “0” inutiles qui sont situés dans le dernier élément de “déchiffrer3”. Enfin les lignes 113/114 se chargent de transformer les nombres en ascii en caractère que nous concaténons pour retrouver le message de base envoyé par Bob !

Partie 2 – Codes correcteurs

Table de multiplication de F_2 :

(x,y)	0	1
0	0	0
1	0	1

Table d’addition de F_2 :

(x,y)	0	1
0	0	1
1	1	0

Montrer que tout vecteur de F_2^4 peut s’écrire comme une somme de vecteurs e_i.

Pour faire cette démonstration nous avons créé une fonction “f42” suivante :

```python
def f42() :
        e1 = [1,0,0,0]
        e2 = [0,1,0,0]
        e3 = [0,0,1,0]
        e4 = [0,0,0,1]
        df = []
        s = [0,1]
        for x1 in s :
                for x2 in s :
                        for x3 in s :
                                for x4 in s :
                                        d = []
                                        for i in range(0,4) :
                                                d.append((x1*e1[i])+(x2*e2[i])+(x3*e3[i])+(x4*e4[i]))
                                        df.append(d)
        return df

```

Nous avons initialisé e_1,e_2,e_3 et e_4. Ensuite nous avons créé des boucles de sorte que toutes les additions possibles entre ces 4 variables soient réalisées. Voici le résultat de cette fonction :

```
>>> f42()
[[0, 0, 0, 0], [0, 0, 0, 1], [0, 0, 1, 0], [0, 0, 1, 1], [0, 1, 0, 0], [0, 1, 0, 1], [0, 1, 1, 0], [0, 1, 1, 1], [1, 0, 0, 0], [1, 0, 0, 1], [1, 0, 1, 0], [1, 0, 1, 1],
 [1, 1, 0, 0], [1, 1, 0, 1], [1, 1, 1, 0], [1, 1, 1, 1]]
>>>
```

La fonction affiche bien les 16 combinaisons possibles.

Question 2.3 :

En se basant sur la matrice M donnée, nous en avons déduit que image(x,y,z,w)=(x+y+w,x+z+w,x,y+z+w,y,z,w). Nous avons donc créé la fonction “f72” qui s’occupe de faire ce calcul sur les 16 tableaux binaires ressortis par la fonction “f42”. Voici les résultats de “f72” :

```
>>> f72()
[[0, 0, 0, 0, 0, 0, 0], [1, 1, 0, 1, 0, 0, 1], [0, 1, 0, 1, 0, 1, 0], [1, 0, 0, 0, 0, 1, 1], [1, 0, 0, 1, 1, 0, 0], [0, 1, 0, 0, 1, 0, 1], [1, 1, 0, 0
, 1, 1, 0], [0, 0, 0, 1, 1, 1, 1], [1, 1, 1, 0, 0, 0, 0], [0, 0, 1, 1, 0, 0, 1], [1, 0, 1, 1, 0, 1, 0], [0, 1, 1, 0, 0, 1, 1], [0, 1, 1, 1, 1, 0, 0],
[1, 0, 1, 0, 1, 0, 1], [0, 0, 1, 0, 1, 1, 0], [1, 1, 1, 1, 1, 1, 1]]
>>>
```

Question 2.4 :

Dans la fonction "demopoids"(Voir la fonction dans SAE_S1_02_Corr.py). On teste pour tout u et v différents de $\in$ Im($F^4_2 \rightarrow F_2^7$) que d(u,v) >= 3. Notre fonction affiche bien True quand on la lance.

Question 2.5 :

Nous savons que d(u,v)>=3,
Or, û a maximum 1 seul bit de différence avec u et d(û,v)>=3 et d(u,û)<=1,
Donc Alice peut corriger le message bruitée parce qu’il n’y a qu’un seule et unique mot qui a une distance de 1 avec le message bruitée.

Partie 3 – Communication sécurisée

La traduction binaire intervient après la fonction “encryption”. Nous avons donc créer la fonction “tradbin” suivante :

```python
def tradbin(msg) :
    newmsg = []
    for i in range(0,len(msg)) :
        newmsg.append([])
        for y in range(0,len(msg[i])) :
            newmsg[i].append(bin(int(msg[i][y])))
    for i in range(0,len(newmsg)) :
        for y in range(0,len(newmsg[i])) :
            newmsg[i][y] = newmsg[i][y][2:]
            while len(newmsg[i][y])<4 :
                newmsg[i][y] = str(0) + newmsg[i][y]
    return newmsg

```

Les lignes 120 à 123 s'occupent de traduire en binaire chaque nombre reçu du tableau "chiffrer" de la fonction "encryption". Nous les avons codés un chiffre par un.
Cependant les chiffres étaient codés de la sorte : "0b1" pour le chiffre 1 ou "0b10" pour le chiffre 2 par exemple.
Nos lignes 124 à 128 s'occupent donc à supprimer les "0b" et à ajouter des 0 devant les nombres restants pour avoir des nombres binaires sur 4 bits.

Ensuite, nous devons transformer les nombres binaires sur 4 bits en 7 bits. Nous avons donc créé la fonction "img" pour image. Le principe de cette fonction a été démontré précédemment dans la partie 2.

Nous appliquons par la suite la fonction "noise" donnée dans le pdf. Nous devons ensuite débruiter le message. Nous avons pour cela créer la fonction "denoise" suivante :

```
155  def denoise(msg) :
156          ds = f72()
157          for i in range(0,len(msg)) :
158                  for y in range(0,len(msg[i])) :
159                          for z in range(0,len(ds)) :
160                                  if poids(msg[i][y],ds[z])==0 or poids(msg[i][y],ds[z])==1 :
161                                          msg[i][y]=ds[z]
162                                          break
163          return msg
164
```

Dans cette fonction, nous comparons les matrices message qui a subi le bruitage avec les matrices que nous donnes la fonction "f42". Nous avons démontré que si un des tableaux bruitées à un seul bit de différence avec un des tableaux de "f42" alors le tableau de bits qui a été bruité était à la base la même que celle de "f42'.
Pour cela nous avons dû créer une fonction "poids" qui nous signale combien il y a de bits qui diffèrent dans 2 tableaux binaires donnés.
Donc, dans la fonction "denoise", les lignes 157 et 158 se chargent de prendre un tableau par un tableau dans le message sortie par noise. Ensuite les lignes 159 à 162 s'occupent de trouver le tableau de base. Pour cela on a écrit que si "poids(message bruité, un tableau de f42)" était égale à 1 ou 0 alors il fallait changer le tableau par le bon dans le message bruité.

Ensuite il fallait passer le message sur 7 bits en un message sur 4 bits, pour cela nous avons remarqué que la valeur numéro 3,5,6 et 7 d'un tableau était respectivement égale au x,y,z et w de départ. C'est la fonction "antécédent" qui se charge de cela.

Enfin, la dernière fonction que nous avons utilisé est "detradbin", son but est de transformer les nombres binaires en nombres réels. Pour cela nous avons écrit dans la fonction que le premier chiffre d'un nombre binaire devait être multiplié par 8, le deuxième par 4, le troisième par 2 et le dernier par 1.