TP de Cryptologie

Tony Chouteau - Info 2

Avril 2020

Fait via le Jupyter Notebook de Google Colab

Contents

Introduction

Imports

```
In [0]: import random as rand
        import sys
        import sympy as sy
        import time
        import copy
        #import math
```

Définition

J'utilise pour ce projet:

1. La bibliothèque *sys* pour augmenter le nombre maximum de récursion (utile pour la génération de très grands nombres premiers).
2. La bibliothèque *sympy* pour l'ensemble de ses fonctions permettant de trouver, vérifier des nombres premiers, et donc tester mes algorithmes.
3. La bibliothèque *time* pour mesurer la complexité temporelle réelle de mess algorithmes. J'en profite pour définir une fonction *millis*() qui renvoie le nombre de millisecondes depuis *epoch*, soit le *1er janvier 1970, 00:00:00 (UTC)*.

```
In [183]: print("La version de python utilisé est :", sys.version)
          print("")

          limit = 10**7
          print("Nombre de récursions maximum définie à :",limit)
          sys.setrecursionlimit(limit)
          print("")

          millis = lambda: int(round(time.time() * 10**3))
          print("Milliseconde actuelle :",millis())

          micro = lambda: int(round(time.time() * 10**6))
          print("MicroSeconde actuelle :",micro())

          def displayPeriod(t):
            if (t<10**3):
              return (str(t)+"µs")
            if (t<10**6):
              return (str(t/10**3)+"ms")
            else:
              return (str(t/10**6)+"s")
```

```
La version de python utilisé est : 3.6.9 (default, Nov  7 2019, 10:44:02)
[GCC 8.3.0]

Nombre de récursions maximum définie à : 10000000

Milliseconde actuelle : 1587312760032
MicroSeconde actuelle : 1587312760032500
```

Partie 1 - Arithmétique dans Z et Z/nZ avec Python

2.1 Question 1

2.1.1 Euclide Simple

Définition

```
In [0]: def euclideSimple(a, b):
          assert (a > 0 and b > 0), "a et b doivent être positifs"

          if a < b:
            a, b = b, a

          a0, b0 = a, b

          while b:
            a, b = b, a % b

          return a, a0//a, b0//a
```

Test

```
In [185]: a = 60
          b = 45

          r, a0, b0 = euclideSimple(a, b)

          print("Le PGCD de", a, "et de", b, "est :",r)
          print("")
          print("Ainsi : ")
          print(a0, "*", r, "=", a)
          print(b0, "*", r, "=", b)
```

```
Le PGCD de 60 et de 45 est : 15

Ainsi :
4 * 15 = 60
3 * 15 = 45
```

2.1.2 Euclide Étendu

Définition

```
In [0]: def euclideEtendu(a, b):
          assert type(a)==type(b), "Les deux paramêtres doivent avoir le même type"

          r, u, v, r0, u0, v0 = a, 1, 0, b, 0, 1

          while r0:
            q = r//r0
```

```
            r, u, v, r0, u0, v0 = r0, u0, v0, r-q*r0, u-q*u0, v-q*v0

        return r, u, v
```

Test

```
In [187]: a = 45
          b = 60

          r, u, v = euclideEtendu(a, b)

          print("Le PGCD de", a, "et de", b, "est :",r)
          print("")
          print("De plus : r=au+bv avec u =",u," et v =",v)
          print("")
          print(r, "=", a, "*", u, "+", b, "*", v, "=", a*u, "+", b*v)
```

```
Le PGCD de 45 et de 60 est : 15

De plus : r=au+bv avec u = -1  et v = 1

15 = 45 * -1 + 60 * 1 = -45 + 60
```

2.2 Question 2

2.2.1 Inverse Modulaire

Définition

```
In [0]: def inverseMod(x, n):
          r, u, _ = euclideEtendu(x, n)

          assert r==1, "x et n doivent être premiers entre eux"

          return u%n
```

Test

```
In [189]: a = 23
          n = 29

          inv = inverseMod(a, n)

          print("L'inverse de",a,"modulo",n,"est :",inv)
```

```
L'inverse de 23 modulo 29 est : 24
```

2.3 Question 3

2.3.1 Exponentiation Rapide

Définition

```
In [0]: def expoRapide(x, k, n=None):
          if (k == -1 and n != None): #Inverse modulaire
            return inverseMod(x,n)
          if (k == 1):
            if (n!=None):
              return x%n
            else:
              return x
          if (not k%2): #Paire
            if (n!=None):
              return expoRapide(x*x%n, k//2, n)%n
            else:
              return expoRapide(x*x, k//2, n)
          if (k%2): #Impaire
            if (n!=None):
              return x*expoRapide(x*x%n, (k-1)//2, n)%n
            else:
              return x*expoRapide(x*x, (k-1)//2, n)
```

Test

```
In [191]: x = 4
          k = 87

          r = expoRapide(x, k)

          print(x,"à la puissance",k,"donne :",r)
```

```
4 à la puissance 87 donne : 2394524282602951341184917229922358099404279878411878 4
```

2.4 Question 4

2.4.1 IntegerModRing

Utilisant Jupyter je n'ai pas accès à l'objet *IntegerModRing* de SAGE, je ne peux donc pas proposer d'études temporelle. Je l'ai tout de même re-créé en utilisant une nouvelle classe et de nouvelles méthodes.

Définition

```
In [0]: class IntegerModRing: #Z/nZ

          def __init__(self, n): # Constructeur : IntegerModRing(n)
            self.n = n
            self.__call__.n = self.n

          class __call__: # Rendre l'objet "Callable" : A(x)
              def __init__(self, x):
                self.value = x % self.n
```

```
            def __add__(self, a): # Redéfinition de l'addition
              return (self.value+a) % self.n
            def __mul__(self, multi): # Redéfinition de la multiplication
              return (self.value * multi) % self.n
            def __pow__(self, power): # Redéfinition de la puissance
              return expoRapide(self.value, power, self.n)
```

Test

```
In [193]: x = 11
          k = 2

          n = 5

          print("A = IntegerModRing(",n,")")
          A = IntegerModRing(n)
          print("")

          r = A(x)+k
          print("Addition A(x)+k :\n",x,"+",k,"%",n,"=",r)
          print("")

          r = A(x)*k
          print("Multiplication A(x)**k :\n",x,"*",k,"%",n,"=",r)
          print("")

          r = A(x)**k
          print("Puissance A(x)*k :\n",x,"^",k,"%",n,"=",r)
          print("")

          r = A(x)**(-1)
          print("Inverse A(x)**(-1) :\n",x,"^ -1 %",n,"=",r)
```

```
A = IntegerModRing( 5 )

Addition A(x)+k :
 11 + 2 % 5 = 3

Multiplication A(x)**k :
 11 * 2 % 5 = 2

Puissance A(x)*k :
 11 ^ 2 % 5 = 1

Inverse A(x)**(-1) :
 11 ^ -1 % 5 = 1
```

Partie 2 - Codage et Décodage

3.1 Question 1

3.1.1 Alphabet et Dictionnaires

Définition

```
In [0]: abc = "$\n abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890,.;:!?'()&%-+*/=@"

        cDico = {abc[key]:key for key in range(len(abc))}
        dDico = {key:abc[key] for key in range(len(abc))}
```

Test

```
In [195]: print("Début \"",abc[0], "-", abc[1],"\" Fin")
          print("Tailles :", len(abc), len(cDico), len(dDico))
          print("")

          print("Alphabet :",abc)
          print("")
          print("Dico de codage :",cDico)
          print("Dico de décodage :",dDico)
```

```
Début " $ -
 " Fin
Tailles : 82 82 82

Alphabet : $
 abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890,.;:!?'()&%-+*/=@

Dico de codage : {'$': 0, '\n': 1, ' ': 2, 'a': 3, 'b': 4, 'c': 5, 'd': 6, 'e': 7, 'f': 8, 'g': 9,
                  ......
                  ')': 73, '&': 74, '%': 75, '-': 76, '+': 77, '*': 78, '/': 79, '=': 80, '@': 81}

Dico de décodage : {0: '$', 1: '\n', 2: ' ', 3: 'a', 4: 'b', 5: 'c', 6: 'd', 7: 'e', 8: 'f', 9: 'g',
                    ......
                    73: ')', 74: '&', 75: '%', 76: '-', 77: '+', 78: '*', 79: '/', 80: '=', 81: '@'}
```

3.2 Question 2

3.2.1 Encodage d'un bloc

Définition

Remarque : Dans le cas d'un caractère inconnu, il est remplacé par le dernier caractère de l'alphabet, rajouté pour l'occasion : "@".

```
In [0]: def encodageBloc(bloc):

          result = []
```

```
          for c in bloc:
            try:
              result.append(cDico[c])
            except KeyError:
              result.append(len(abc)-1)

          number = 0
          for i in range(len(result)):
            number += result[i]*len(abc)**i

          return number
```

Test

```
In [197]: msg = "Hello World"
          print("Le message est : \""+msg+"\"")

          print("")
          cMsg = encodageBloc(msg)
          print("L'encodage du message donne :", cMsg)
```

```
Le message est : "Hello World"

L'encodage du message donne : 84856814622316789146
```

3.3 Question 3

Pour une taille de bloc P, et un alphabet de taille N, le code numérique maximal d'un bloc est de N^{P-1}

3.3.1 Décodage d'un bloc

Définition

Remarque : Une taille de bloc est nécessaire en paramètre pour connaitre la puissance maximale de l'encodage du dit-bloc.

```
In [0]: def decodageBloc(bloc, maxChar=20):
          assert maxChar<len(abc), "Taille de bloc trop grande"
          assert bloc<(len(abc)**(maxChar)), "Bloc non valide, valeur trop grande"

          result = []

          reste = bloc
          for i in range(maxChar-1, -1, -1):
            if(reste >= len(abc)**i):
              result.append(reste//(len(abc)**i))
              reste = reste%(len(abc)**i)

          msg = ""
          for n in result:
            try:
              msg = dDico[n] + msg
            except KeyError:
              msg = "@" + msg

          return msg
```

Test

Ajout d'un caractère avant le décodage : ajout d'un $len(dDico)$ (Afin d'être toujours en dehors de l'alphabet).

```
In [199]: dMsg = decodageBloc(cMsg)
          print("Le decodage du message donne : \""+dMsg+"\"")
          print("                Vérification : \""+msg+"\"")
```

```
Le decodage du message donne : "Hello World"
                Vérification : "Hello World"
```

3.4 Question 4

3.4.1 Encodage d'un message entier

Définition

```
In [0]: def encodage(msg, tailleBloc=20):

          encodedMsg = []

          for i in range(0, len(msg), tailleBloc):
            encodedMsg.append(encodageBloc(msg[i:i+tailleBloc]))

          return encodedMsg
```

Test

```
In [201]: msg = "à Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque egestas eget orci
                 sit amet ullamcorper. Curabitur venenatis non nulla eu ullamcorper. Sed dignissim,
                 ......
                 tristique at. Morbi augue purus, elementum non posuere ac, pulvinar sit amet odio.
                 Nulla vitae. "
          print("Le message est : \""+msg+"\"")

          print("")
          cMsg = encodage(msg)
          print("L'encodage du message complet donne :", cMsg)
```

```
Le message est : "à Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque egestas eget
orci sit amet ullamcorper. Curabitur venenatis non nulla eu ullamcorper. Sed dignissim, libero sed
......
tristique at. Morbi augue purus, elementum non posuere ac, pulvinar sit amet odio. Nulla vitae. "

L'encodage du message complet donne : [5175605931744531058174221399946258109,
536102571529470541201406318087641543 23, 10375429670758679101586952198308662 6148,
......
484449109008567046176926222695446230 71, 469694101339807492386039096009015 6871, 853019927550]
```

3.4.2 Décodage d'un message codé entier (soit une liste de blocs encodés)

Définition

```
In [0]: def decodage(blocs, tailleBloc=20):

          decodedMsg = ""

          for bloc in blocs:
            decodedMsg += decodageBloc(bloc, tailleBloc)

          return decodedMsg
```

Test

Remarque : Le caractère special "à" à été remplacé par un "@" comme prévu

```
In [203]: dMsg = decodage(cMsg)
          print("Le decodage du message complet donne : \""+dMsg+"\"")
          print("                        Vérification : \""+msg+"\"")
```

```
Le decodage du message complet donne : "@ Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Quisque egestas eget orci sit amet ullamcorper. Curabitur venenatis non nulla eu ullamcorper
......
tristique at. Morbi augue purus, elementum non posuere ac, pulvinar sit amet odio. Nulla vitae. "

                        Vérification : "à Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Quisque egestas eget orci sit amet ullamcorper. Curabitur venenatis non nulla eu ullamcorper
......
tristique at. Morbi augue purus, elementum non posuere ac, pulvinar sit amet odio. Nulla vitae. "
```

Partie 3 - RSA

4.1 Question 1

4.1.1 Production des valeurs p, q et a

Définition

```
In [0]: def bobPrive(min=10**19, max=10**20):

          p, q = 0, 0
          while q==p:
            p, q = sy.randprime(min, max), sy.randprime(min, max)

          assert (min>0 and max > min), "Le paramêtre min doit être supérieur à 0 et max doit être
                                        supérieur à min"

          a, r = p, 0
          while r!=1:
            a = rand.randint(min, (p-1)*(q-1)-1)
            r, _, _ = euclideSimple(a, (p-1)*(q-1))

          return p, q, a
```

Test

Partie Privé

Si *power* > 14, la fonction factor va devenir très chronophage.

```
In [205]: power = 19

          q, p, a = bobPrive(10**power, 10**(power+1))
          print("Bob produit 2 nombres premiers :")
          print("p =",p,"et q =",q)
          print("")

          print("Bob produit un nombre aléatoire premier avec phi(n)=(p-1)(q-1) :")
          print("a =",a)
```

```
Bob produit 2 nombres premiers :
p = 70839248108601181661 et q = 55527490103495060851

Bob produit un nombre aléatoire premier avec phi(n)=(p-1)(q-1) :
a = 820408104386500733550364758094963808171
```

Partie Public

```
In [206]: n = p*q
          print("Bob publie n=pq :")
          print("n =",n)
```

```
          print("")

          b = inverseMod(a, (p-1)*(q-1))
          print("Bob publie l'inverse de a modulo phi(n) :")
          print("b =",b)
```

```
Bob publie n=pq :
n = 39333343823194900991176421424957002535 11

Bob publie l'inverse de a modulo phi(n) :
b = 72445839558856047732133526800573386731
```

Factor()

La fonction utilisé ici est la fonction $primefactors(n)$ de la bibliothèque Sympy, le calcul est très chronophage pour des valeurs supérieures à 10^{15} (cf : Partie Privé/Test)

On remarque qu'il y a l'air d'y avoir que 2 facteurs premiers.

```
In [207]: power = 14
          q, p, a = bobPrive(10**power, 10**(power+1))
          n = p*q
          print(p,q)

          if (power <= 15):
            l = sy.primefactors(n)
            print(l)
          else:
            print("La valeur power est trop haute, le calcul durerait trop longtemps")
```

```
934263701871971 808037973730219
[808037973730219, 934263701871971]
```

4.2 Question 2

4.2.1 Obtention la taille du bloc

Définition

```
In [0]: def tailleBlocMax(n):

          tailleBloc = 0
          while len(abc)**(tailleBloc+1)<n:
            tailleBloc+=1

          return tailleBloc
```

Test

```
In [209]: print("Pour n =",n,", la taille du bloc est de maximum :",tailleBlocMax(n))
```

```
Pour n = 39333343823194900991176421424957002535 11 , la taille du bloc est de maximum : 20
```

4.2.2 Chiffrement RSA d'un simple bloc

Définition

```
In [0]: def chiffrementRSA(x, n, b):
          assert x<n, "x doit être inférieur à n"

          return expoRapide(x, b, n)
```

Test

```
In [211]: msg = "HelloWorld"
          cMsg = encodageBloc(msg)
          rsa = chiffrementRSA(cMsg, n, b)
          print("Le message \""+msg+"\" à été chiffré :",rsa)
```

```
Le message "HelloWorld" à été chiffré : 3068046575213518710256046111929775664104
```

4.2.3 Chiffrement RSA d'un message entier

Définition

```
In [0]: def messageToRSA(msg, n, b):

          taille = tailleBlocMax(n)
          l = encodage(msg, tailleBloc=taille)

          result = []
          for elt in l:
            result.append(chiffrementRSA(elt, n, b))

          return result
```

Test

```
In [213]: msg = "à Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque egestas eget orci
                 sit amet ullamcorper. Curabitur venenatis non nulla eu ullamcorper. Sed dignissim,
                 ......
                 tristique at. Morbi augue purus, elementum non posuere ac, pulvinar sit amet odio.
                 Nulla vitae. "

          msgRSA = messageToRSA(msg, n, b)

          print("Le message \""+msg+"\"")
          print("à été chiffré :",msgRSA)
```

```
Le message "à Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque egestas eget orci sit
amet ullamcorper. Curabitur venenatis non nulla eu ullamcorper. Sed dignissim, libero sed sodales
......
tristique at. Morbi augue purus, elementum non posuere ac, pulvinar sit amet odio. Nulla vitae. "

à été chiffré : [29849973066596168854032340446915071178 92, 52423916820063530209673040729407778624 1,
64142755627242053458963935326834076598 6, 24510474005905718468808307783739995563 99,
......
```

```
33278359866577162321822506531264123082 45, 19396313574610293175678473194386487582 74,
88006252140530666706210124620806428922 6, 95545106572552414151140463742821602720 4]
```

4.3 Question 3

4.3.1 Déchiffrement RSA d'un message

Définition

```
In [0]: def rsaToMessage(rsa, n, a):

          result = []
          for elt in rsa:
            result.append(chiffrementRSA(elt, n, a))

          taille = tailleBlocMax(n)
          l = decodage(result, tailleBloc=taille)

          return l
```

Test

Le message est donc bien déchiffré, exepté, encore une fois, pour le caractère spécial "à" qui a été remplacé par un "@".

```
In [215]: print("Le message à déchiffrer est :",msgRSA)
          print("")

          msgAfter = rsaToMessage(msgRSA, n, a)
          print("Message déchiffré :",msgAfter)
          print("     Vérification :",msg)
```

```
Le message à déchiffrer est :
[29849973066596168854032340446915071178 92, 52423916820063530209673040729407778624 1,
64142755627242053458963935326834076598 6, 24510474005905718468808307783739995563 99,
......
33278359866577162321822506531264123082 45, 19396313574610293175678473194386487582 74,
88006252140530666706210124620806428922 6, 95545106572552414151140463742821602720 4]

Message déchiffré : @ Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque egestas eget
......
tristique at. Morbi augue purus, elementum non posuere ac, pulvinar sit amet odio. Nulla vitae.

     Vérification : @ Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque egestas eget
......
tristique at. Morbi augue purus, elementum non posuere ac, pulvinar sit amet odio. Nulla vitae.
```

4.4 Question 4

4.4.1 Déchiffrement frauduleux

1) x doit être premier avec n :

Le principe du chiffrement RSA est fondé sur la complexité de la décomposition d'un grand nombre en facteurs premiers, en d'autres mots, de retrouver p et q à partir de n.

Si x n'est pas premier avec n, alors x vaut soit q, soit p (x étant inférieur à n), ainsi si x n'est pas premier, il est possible de retrouver les facteurs premiers de n qui sont donc x et $\frac{n}{x}$.

Définition

```
In [0]: def probXPrimeWithN(n):

          xMax = tailleBlocMax(n)
          return "2/"+str(len(abc)**xMax)
```

Test

```
In [217]: print("La probabilité que le message encodé ne soit pas premier avec n est de :",
                probXPrimeWithN(n))
```

```
La probabilité que le message encodé ne soit pas premier avec n est de :
2/188919613181312032574569023867244773376
```

Partie 4 - Génération de nombres premiers

5.1 Question 1

5.1.1 Resolution de l'équation $(n-1) = 2^s \times d$

Définition

```
In [0]: def resolveMillerFormula(n):
          assert n%2, "n doit être impair"

          s = 0

          nMinus1Bin = bin(n-1)[2:]
          while (s<len(nMinus1Bin) and nMinus1Bin[len(nMinus1Bin)-1-s]=='0'):
            s+=1

          d = int(nMinus1Bin[:len(nMinus1Bin)-s], 2)

          return s, d
```

Test

```
In [219]: n = 565

          s, d = resolveMillerFormula(n)
          print(n-1, "= 2 ^", s, "*", d)
```

```
564 = 2 ^ 2 * 141
```

5.1.2 Test de Miller en base définie

Définition

```
In [0]: def testMillerBase(n, a):
          assert n>3, "n doit être supérieur à 3"
          assert a<n, "a doit être inférieur à n"

          s, d = resolveMillerFormula(n)

          x = expoRapide(a, d, n)

          if x==1 or x==n-1:
            return False

          for _ in range(s-1):
            x = expoRapide(x, x, n)
            if x==n-1:
              return False

          return True
```

Test

```
In [221]: n = 23
          a = 6
          r = testMillerBase(n, a)
          print(n,"est premier :",not r)
          print("Comparaison avec la bibliothèque Sympy :", sy.isprime(n))
          print("")

          n = 561
          a = 50
          r = testMillerBase(n, a)
          print(n,"est premier :",not r)
          print("Comparaison avec la bibliothèque Sympy :", sy.isprime(n))
          print("Le témoin 50 est un menteur pour l'entier 561.")
```

```
23 est premier : True
Comparaison avec la bibliothèque Sympy : True

561 est premier : True
Comparaison avec la bibliothèque Sympy : False
Le témoin 50 est un menteur pour l'entier 561.
```

5.2 Question 2

5.2.1 Test de Miller

Définition

```
In [0]: def testMiller(n, m=20):
          if n<=1:
            return False
          if n<=3:
            return True
          if not n%2:
            return False

          for _ in range(m):
            a = rand.randint(2,n-2)
            if testMillerBase(n, a):
              return False
          return True
```

Test

```
In [223]: n = 23
          r = testMiller(n)
          print(n,"est premier :",r)
          print("Comparaison avec la bibliothèque Sympy :", sy.isprime(n))
          print("")

          n = 221
          r = testMiller(n)
          print(n,"est premier :",r)
          print("Comparaison avec la bibliothèque Sympy :", sy.isprime(n))
```

```
          print("L'utilisation de multiples témoins, permet de mettre en défaut les témoins
                menteurs'")
```

```
23 est premier : True
Comparaison avec la bibliothèque Sympy : True

221 est premier : False
Comparaison avec la bibliothèque Sympy : False
L'utilisation de multiples témoins, permet de mettre en défaut les témoins menteurs'
```

5.3 Question 3

5.3.1 Générateur de nombre premier

Définition

```
In [0]: def generateurPremier(k):
          assert k>=2, "k doit être supérieur strictement à 1"

          x = 1
          while (not testMiller(x, 100)):
            x = rand.randint(2**(k-1), 2**(k)-1) | 1 # Permet d'avoir un nombre impair

          return x
```

Test

```
In [225]: power = [5, 10, 100, 250, 500]#, 1000, 1000, 2000, 3000, 4000, 5000]

          for k in power:
            t = micro()
            x = generateurPremier(k)
            print(k, "-", x, "est bien premier :",sy.isprime(x))
            print("#Nombre premier trouvé en",displayPeriod(micro()-t))
            print("")
```

```
5 - 23 est bien premier : True
#Nombre premier trouvé en 1.473ms

10 - 983 est bien premier : True
#Nombre premier trouvé en 2.412ms

100 - 1115478283892045139697472002619 est bien premier : True
#Nombre premier trouvé en 32.81ms

250 - 1505919279241641237811087301853396062518 24 ..... 76530187998579707 est bien premier : True
#Nombre premier trouvé en 154.745ms

500 - 2975503490251944515827818804459268639262 93 ..... 06617779917107783 est bien premier : True
#Nombre premier trouvé en 759.682ms
```

Partie 5 - ElGamal

6.1 Construction de p

6.1.1 Question 1

Condition de complexité

L'algoritme ElGamal repose sur la complexité du problème de logarithme discret (déchiffrement par force brut), contrairement à l'exponentiation.

Le problème de logarithme discret requiert une clé p, qui doit être très grande et nombre premier, et $p-1$ avec un très grand facteur premier, d'où $p = q \times 2 - 1$.

6.1.2 Question 2

Génération d'une nombre premier ElGamal

Définition

```
In [0]: def premierElGamal(k):

          q = 0
          p = q*2+1
          while not testMiller(p):
            q = generateurPremier(k)
            p = q*2+1

          return p, q
```

Test

```
In [227]: power = [2, 5, 10, 100]

          for k in power[1:]:
            t = micro()
            p, q = premierElGamal(k)
            print(k, "donne", p,"=", q,"* 2 + 1")
            print(p,"est premier :",sy.isprime(x))
            print("#Nombre premier de ElGamal trouvé en",displayPeriod(micro()-t))
            print("")
```

```
5 donne 47 = 23 * 2 + 1
47 est premier : True
#Nombre premier de ElGamal trouvé en 5.809ms

10 donne 1319 = 659 * 2 + 1
1319 est premier : True
#Nombre premier de ElGamal trouvé en 19.484ms

100 donne 1309176998925936328789683684599 = 654588499462968164394841842299 * 2 + 1
1309176998925936328789683684599 est premier : True
#Nombre premier de ElGamal trouvé en 169.448ms
```

6.2 Construction d'une clé (p,a)

6.2.1 Question 1

Clé Privé ElGamal

Définition

```
In [0]: def clePriveElGamal(k):

            p, _ = premierElGamal(k)
            a = rand.randint(0, p-2)

            return p, a
```

Test

```
In [229]: k = 100
          p, a = clePriveElGamal(k)
          print("Génération d'une clé privé ElGamal (p,a) :", (p, a))
```

```
Génération d'une clé privé ElGamal (p,a) : (20247192057812491783673342903 03,
                                            25836483961090725876869744 0970)
```

Clé Public ElGamal

Définition

```
In [0]: def clePublicElGamal(p, a):

            m = rand.randint(0, p-1)
            n = expoRapide(m, a, p)

            return p, m, n
```

Test

```
In [231]: _, m, n = clePublicElGamal(p, a)
          print("Génération d'une clé public ElGamal (p,m,n) :", (p, m, n))
```

```
Génération d'une clé public ElGamal (p,m,n) : (20247192057812491783673342903 03,
                                               13649582182302708620593528905 33,
                                               33671689056064370901997413472 8)
```

6.3 Implémentation de ElGamal

6.3.1 Chiffrement

Définition

```
In [0]: def chiffrementElGamal(x, public):

            p, m, n = public
            assert x<p, "x doit être inférieur à p"

            k = rand.randint(0, p-1)

            y1 = expoRapide(m, k, p)
            y2 = x*expoRapide(n, k, p)

            return y1, y2
```

Test

```
In [233]: msg = "Hello World !"
          print("Le message est : \""+msg+"\"")
          print("")

          print("Un bloc est de taille maximum :",tailleBlocMax(p))
          print("")

          cMsg = encodageBloc(msg)

          elGamal = chiffrementElGamal(cMsg, (p, m, n))
          print("Le bloc chiffré donne :", elGamal)
```

```
Le message est : "Hello World !"

Un bloc est de taille maximum : 15

Le bloc chiffré donne : (197486957419771004372090929440 6,
669771393878983878845940816007446857242952539309627 0496)
```

6.3.2 Déchiffrement

Définition

```
In [0]: def dechiffrementElGamal(y, prive):

            y1, y2 = y
            p, a = prive
            x = (expoRapide(y1, p-1-a, p) * y2) %p

            return x
```

Test

```
In [235]: print("Le message à déchiffré est :", elGamal)
          print("")

          dMsg = dechiffrementElGamal(elGamal, (p, a))
          print("Le bloc déchiffré donne :", dMsg)
          print("")

          newMsg = decodageBloc(dMsg)
          print("Le message déchiffré et décodé donne :",newMsg)
          print("                      Message initial :",msg)

          cMsg = encodageBloc(msg)
```

```
Le message à déchiffré est : (197486957419771004372090929440 6,
669771393878983878845940816007446857242952539309627049 6)

Le bloc déchiffré donne : 6379322887179225045360026

Le message déchiffré et décodé donne : Hello World !
                    Message initial : Hello World !
```

6.3.3 Chiffrement ElGamal d'un message entier

Définition

```
In [0]: def messageToElGamal(msg, public):

          p, _, _ = public

          taille = tailleBlocMax(p)
          l = encodage(msg, tailleBloc=taille)

          result = []
          for elt in l:
            result.append(chiffrementElGamal(elt, public))

          return result
```

Test

```
In [237]: msg = "à Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque egestas eget orci
                 sit amet ullamcorper. Curabitur venenatis non nulla eu ullamcorper. Sed dignissim,
                 ......
                 tristique at. Morbi augue purus, elementum non posuere ac, pulvinar sit amet odio.
                 Nulla vitae. "

          msgElGamal = messageToElGamal(msg, (p, m, n))
          print("Le message \""+msg+"\"")
          print("à été chiffré :",msgElGamal)
```

```
Le message "à Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque egestas eget orci sit
            ......
            Morbi augue purus, elementum non posuere ac, pulvinar sit amet odio. Nulla vitae. "

à été chiffré : [(109434280020782806597252569263,
934309489404227287951670803618982807173118985947282268019),
(145979244213053550246020534229 5, 190341826296741114782942134599622832023185501363877877685 3),
......
(863087472162997549675753508325, 976857556035481490247804987927615071380108815839369400033),
(141258520103501720957368748 7619, 42349057854697552331875014964757 0)]
```

6.3.4 Déchiffrement ElGamal d'un message

Définition

```
In [0]: def elGamalToMessage(elGamal, prive):

          result = []
          for elt in elGamal:
            result.append(dechiffrementElGamal(elt, prive))

          p, _ = prive
          taille = tailleBlocMax(p)
          l = decodage(result, tailleBloc=taille)

          return l
```

Test

Le message est donc bien déchiffré, excepté, encore une fois, pour le caractère spécial "à" qui a été remplacé par un "@".

```
In [239]: print("Le message à déchiffrer est :",msgElGamal)
          print("")

          msgAfter = elGamalToMessage(msgElGamal, (p, a))
          print("Message déchiffré :",msgAfter)
          print("     Vérification :",msg)
```

```
Le message à déchiffrer est : [(109434280020782806597252569263,
934309489404227287951670803618982807173118985947282268019),
(145979244213053550246020534229 5, 190341826296741114782942134599622832023185501363877877685 3),
......
(863087472162997549675753508325, 976857556035481490247804987927615071380108815839369400033),
(141258520103501720957368748 7619, 42349057854697552331875014964757 0)]

Message déchiffré : @ Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque egestas eget
                    ......
                    Morbi augue purus, elementum non posuere ac, pulvinar sit amet odio. Nulla vitae.

     Vérification : à Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque egestas eget
                    ......
                    Morbi augue purus, elementum non posuere ac, pulvinar sit amet odio. Nulla vitae.
```

Partie 6 - Arithmétique dans K[X] et application à AES

7.1 Question 1

7.1.1 Implémentation de K[X] dans un corps

Implémentation de la fonction $inverseMod(x,n)$ pour $K[X]$ est demmandé Question 2 mais est requis pour l'implémentation de la divison de *PolynomeModRing*

Définition

Cette implémentation concerne les polynômes à une variable, par exemple : $2x^3 + 4x^2 + 5x + 10$. Les polynomes sont à coéfficients dans $F_n[X]$. Ces coefficients sond donc des entiers a_k tel que $0 \leq a_k <$ n.

```
In [288]: class PolynomeModRing:

            #Constructor
            def __init__(self, p, n):
              assert isinstance(p, list), "p doit être sous forme de liste de coefficients"
              assert isinstance(n, int), "n doit être un entier"

              self.p = [p[i]%n for i in range(len(p))]
              self.n = n


            ......
            ...... VOIR DANS LE FICHIER "PolyModRing.py" pour le code complet
            ...... (cela surchargerai le compte rendu de le mettre en entier ici)
            ......

            def __str__(self):
              return self.toString()

            def toString(self):
              #return str(self.p[len(p)-1])

              if self.powerMax() == 0:
                return "("+str(self.p[self.size()-1])+")"

              m = []
              for k in range(self.size()):
                if (self.p[k] != 0):
                  m.append((self.p[k],self.size()-1-k))

              return "("+" + ".join([str(m[i][0])+"x^"+str(m[i][1]) for i in range(len(m))])+")"
```

Test

```
In [289]: n = 29
          p = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 15]
          q = [-1, 0, 15]

          P = PolynomeModRing(p, n)
          Q = PolynomeModRing(q, n)
```

```
R = P+Q
print("Addition de deux polynômes P+Q :", P,"+", Q,"\n=", R)
print("")

k = 2
R = P*k
print("Produit d'un polynôme et d'un entier P*2 :",P,"*",k,"\n=",R)
print("")

R = P*Q
print("Produit de deux polynômes P*Q :", P, "*", Q, "\n=",R)
print("")

R = P//Q
print("Division euclidienne de deux polynômes P//Q=R :", P, "//", Q, "\n=",R)
```

```
Addition de deux polynômes P+Q : (1x^10 + 15x^0) + (28x^2 + 15x^0)
= (1x^10 + 28x^2 + 1x^0)

Produit d'un polynôme et d'un entier P*2 : (1x^10 + 15x^0) * 2
= (2x^10 + 1x^0)

Produit de deux polynômes P*Q : (1x^10 + 15x^0) * (28x^2 + 15x^0)
= (28x^12 + 15x^10 + 14x^2 + 22x^0)

Division euclidienne de deux polynômes P//Q=R : (1x^10 + 15x^0) // (28x^2 + 15x^0)
= (28x^8 + 14x^6 + 7x^4 + 18x^2 + 9x^0)
```

7.2 Question 2

7.2.1 Euclide Etendu dans $F_n[X]$

Re-Définition

```
In [0]: def euclideEtendu(a, b):

          r, r0 = a, b
          u, v, u0, v0 = None, None, None, None

          if isinstance(a, int):
            u, v, u0, v0 = 1, 0, 0, 1
          elif isinstance(a, PolynomeModRing):
            u, v, u0, v0 = PolynomeModRing([1], a.n), PolynomeModRing([0], a.n),
                           PolynomeModRing([0], a.n), PolynomeModRing([1], a.n)
          else:
            raise Exception("TypeError","Cette fonction a été définie pour des paramètres de type
                                        entiers ou polynomes")

          while (isinstance(r0, int) and r0) or (isinstance(r0, PolynomeModRing) and
                (r0.powerMax()!=0 or r0.p[r0.size()-1])):
            q = r//r0
            r, u, v, r0, u0, v0 = r0, u0, v0, r-(q*r0), u-(q*u0), v-(q*v0)

          return r, u, v
```

Test

```
In [292]: R, U, V = euclideEtendu(P, Q)

          print("Le PGCD de", P, "et de", Q, "est :",R)
          print("")
          print("De plus : R=P*U+Q*V avec U =",U," et V =",V)
          print("")
          print("Verification :",P,"*",U,"+",Q,"*",V,"\n=",P*U+Q*V)
```

```
[22, 0, 18]
Le PGCD de (1x^10 + 15x^0) et de (28x^2 + 15x^0) est : (25)

De plus : R=P*U+Q*V avec U = (1)  et V = (1x^8 + 15x^6 + 22x^4 + 11x^2 + 20x^0)

Verification : (1x^10 + 15x^0) * (1) + (28x^2 + 15x^0) * (1x^8 + 15x^6 + 22x^4 + 11x^2 + 20x^0)
= (25)
```

7.2.2 Inverse Modulaire dans $F_n[X]$

Re-Définition

```
In [0]: def inverseMod(x, n):
          r, u, _ = euclideEtendu(x, n)

          if isinstance(r, int):
            assert r==1, "x et n doivent être premiers entre eux"

          if isinstance(r, PolynomeModRing):
            assert r.powerMax() and r.p[r.size()-1]==[1], "x et n doivent être premiers entre eux"

          return u%n
```

Test

```
In [0]: #NE FONCTIONNE PAS

        #N = PolynomeModRing([n], n+1)

        #R = inverseMod(P, N)
        #print("L'inverse de",P,"modulo",n,"est :",R)
```

7.3 Question 2

7.3.1 Utilisation de cette implémentation dans la routine SubBytes d'AES

Cette partie du chiffrement AES consiste à effectuer des clalculs sur des bits sous forme de polynôme, il faudra donc utiliser pour chaque octet $PolynomeModRing([a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0], 2)$. Ensuite il sera donc possible d'effectuer les opérations sur ces polynômes afin d'obtenir la matrice de transformation afin de l'application sur chaque octet de l'entrée.