Trabalho prático 2

Grupo 5:

- Duarte Oliveira \<pg47157>
- Melânia Pereira \<pg47520>

Post-Quantum Cryptography na categoria de criptosistemas PKE-KEM

Criação de protótipo em Sagemath de uma técnica representativa da família de criptosistemas pós-quânticos BIKE (“code based”).
Pretende-se implementar um KEM, que seja IND-CPA seguro, e um PKE que seja IND-CCA seguro.
Para o desenvolvimento destas soluções foram seguidas as especificações dos documentos oficiais que podem ser encontrados aqui.

KEM-IND-CPA

In [38]:
```
import os
from hashlib import shake_256, sha3_384
```

In [80]:
```
class BIKE_KEM(object):

    def __init__(self, timeout=None):
        self.r = 12323
        self.W = 142
        self.T = 134
        self.L = 256

        self.K2 = GF(2)
        # Polynomial Ring in x over Finite Field of size 2
        F.<x> = PolynomialRing(self.K2)
        # The cyclic polynomial ring F[X]/<X^r + 1>
        R.<x> = QuotientRing(F, F.ideal(x^self.r + 1))
        self.R = R


    def geraCoef(self, w, r):
        #Gera um coeficiente aleatorio em que 1 representam os coefs
        coefs = [1]*w + [0]*(r-w-2)
        random.shuffle(coefs)
        return self.R([1]+coefs+[1])




    def h(self,seed):
        wlist = []

        s = shake_256(seed).digest(self.T)
```

```
        mask = 2**(ceil(log(len(seed),2)))-1
        print('s',s); print()
        print('mask',mask)

        for i in range(32):
            print('s',s[32*(i+1)-1:32*i])
            pos = s[32*(i+1)-1:32*i] & mask
            if (pos < len(seed)) and (pos not in wlist):
                wlist.append(pos)

        return wlist


    def l(self,e0,e1):
        h = sha3_384()
        h.update(e0)
        h.update(e1)
        return h.digest(self.L)


    def k(self,m,c):
        (c0,c1) = c
        h = sha3_384()
        h.update(m)
        h.update(c0)
        h.update(c1)
        return h.digest(self.L)



    def keyGen(self):
         #Gerar h0 e h1
        #Gerar as componentes chave privada
        h0 = self.geraCoef(self.W//2, self.r)
        h1 = self.geraCoef(self.W//2, self.r)

        h = h1 * h0**(-1)
        sigma = os.urandom(self.L)

        priv = (h0,h1,sigma)
        pub = h

        return priv, pub


    def encaps(self, pk):
        m = os.urandom(self.L)

        (e0,e1) = self.h(m)
        c = (e0+e1*pk,m^^self.l(e0,e1))

        k = self.k(m,c)

        return (k,c)



    def decaps(self,sk,c):
        _e = decoder
```

In [81]:

```
k = BIKE_KEM()

priv,pub = k.keyGen()

(h0,h1,sigma) = priv

print((h0))
print(sigma)

k.encaps(pub)
```

```
x^12322 + x^12216 + x^11835 + x^11692 + x^11559 + x^11549 + x^11349 + x^112
88 + x^10641 + x^10598 + x^10533 + x^10436 + x^10318 + x^10285 + x^10200 +
x^9926 + x^9783 + x^9621 + x^9419 + x^8643 + x^8557 + x^8329 + x^7624 + x^7
590 + x^7581 + x^7565 + x^6931 + x^6915 + x^6678 + x^6641 + x^6306 + x^6106
+ x^6004 + x^5912 + x^5774 + x^5435 + x^5335 + x^5120 + x^4999 + x^4921 + x
^4897 + x^4766 + x^4760 + x^4590 + x^4520 + x^4122 + x^4083 + x^4076 + x^39
91 + x^3689 + x^3049 + x^2975 + x^2820 + x^2704 + x^2536 + x^2504 + x^2162
+ x^2012 + x^1922 + x^1846 + x^1572 + x^1510 + x^1367 + x^452 + x^395 + x^3
71 + x^355 + x^340 + x^265 + x^119 + x^53 + x^6 + 1
b'&\xf0\xec\xc9\x19\xad\xe4\xa9\xa4\xd6\x8cTE\xc3\xbd\xf7y\xf9\xe7\x06by\x0
4\xac%\xb02\xd0\x9bD\x04\x0eGC2G\x83Ws\xa8B\x96\xd8\x9d.\xc9P\x9f\xb6km\x17
FS\x14\x99\x1a!\xf45vz^\x1d\x99\x0eG\xb7=\xf2\xdcT\xd3\xe8\x0bA\xcc\xefj\x9
c\xea%\xe6\x8b\xe1\xe1\x03\xf2\n\x83]\x94__\xb8d\x15{O#_\xdc\xd3+F\xed\xf4Z
\xa8\x0b\x05T\x9e\xbdo\x9b\x00$\x16\no\t\xecR\xb8\x94\xff;1&8\x0b\x8f-fM\x8
9\x88\x8b,\x94\xb3\xd1\x9f\x84\xba^<\xe7\x9ar9x\x9dL\xce3\xcbG\xb76\xb6\xc5
\t\xda,\xeec>\xe9\xceXM\x83\x93\xe5\xdb\xc4\xc4|\xc4\xd5\xd9\x1c\x90\xba\x8
2\xd1,\xba\xedT83\xdf&w\x00g4XP\xb31\xb3\x98\xc8\x9d\xe6\xf6\x00=\x91\xdc\x
9c7_\xde\xd8]:\t{\x9c`\x14\xac\xf2j\x15T=\xdb\xa4<r\x89=d\xf8\xa3\xe5 \x9c\
xed\xf0\xedlT\xb023\x1b\xaf@\xd3'
s b'(3\t\x8e\xd2\t\xdd\x89\x04[V9\x835\r\x9d\xd4a\xda5*6{\xd8(\x99t\xa0L\xd
0\xb3H;\xb4WB>\xc6\xa97r\xbc\xa9\xeeY\xb2=\x80Z\xee\x95\n\xd9k>\xacMcrf\x86
\xe2\xa7\xaf\x9b\xe7\x03\x87b:\xb8\xafu\xe3\x9b\x11\xfa\x91\x14\xec$)#\x0f\
x8fY\xdb\x19M\xc8\xd1Z/\xee\x8cg\x8fG\xa2\xb1t\xe4\xd9\x9dr\xd4t~eWs\xf0A.\
xbc\x88\x16\xe8\xcc\xb7$\xac\xa7;\x83\x81\xc4\xbdY\x02\x15\xe3\xf8Y'

mask 255
s b''
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/var/folders/t0/3fmfjfd52ls2vmngjg2jq99r0000gn/T/ipykernel_15386/2059062437
.py in <module>
      8 print(sigma)
      9
---> 10 k.encaps(pub)

/var/folders/t0/3fmfjfd52ls2vmngjg2jq99r0000gn/T/ipykernel_15386/2819585170
.py in encaps(self, pk)
     78         m = os.urandom(self.L)
     79
---> 80         (e0,e1) = self.h(m)
     81         c = (e0+e1*pk,m^self.l(e0,e1))
     82

/var/folders/t0/3fmfjfd52ls2vmngjg2jq99r0000gn/T/ipykernel_15386/2819585170
.py in h(self, seed)
     34         for i in range(Integer(32)):
     35             print('s',s[Integer(32)*(i+Integer(1))-Integer(1):Integ
er(32)*i])
---> 36             pos = int(s[Integer(32)*(i+Integer(1))-Integer(1):Integ
er(32)*i]) & mask
     37             if (pos < len(seed)) and (pos not in wlist):
     38                 wlist.append(pos)

ValueError: invalid literal for int() with base 10: b''
```