TP2-Problema3 - KYBER

May 2, 2022

1 TRABALHO PRÁTICO 2 - GRUPO 14

1.1 Problema 3 - KYBER

Este problema consistia em implementar o algoritmo KEM que seja IND-CPA seguro e num algoritmo PKE que seja IND-CCA seguro para a técnica pós-quântica baseada em reticulados, **KYBER**. Contudo, a implementação do KYBER apresentada no paper KYBER corresponde às etapas necessárias para implementar o PKE-IND-CPA e um KEM-IND-CCA. Para conseguir implementar um KEM-IND-CPA irá se transformar o PKE-IND-CPA num KEM tal como está apresentado na secção "Resolução do problema - KEM (IND-CPA)". Para se transformar um PKE-IND-CPA num PKE-IND-CCA aplica-se as transformações de Fujisaki-Okamoto tal como está nos apontamentos

IMPORTS

[175]:
```
import random
import numpy as np
from cryptography.hazmat.primitives import hashes
from pickle import dumps, loads
```

CLASSES AUXILIARES **NTT** Nesta classe fornecidas pelos docentes são implementados as operações designadas por *number-theoretic transform* (NTT) que permite realizar operações de multiplicação no anel Rq de forma eficiente. Por simplicidade estes esta classe é inicializada com os valores n=256 e q=7681.

[176]:
```
#Classe que implementa as multiplicações em R - number-theoretic transform
↪(NTT)
class NTT:

    def __init__(self, n=128, q=None):

        if not  n in [32,64,128,256,512,1024,2048]:
            raise ValueError("improper argument ",n)
        self.n = n
        if not q:
            self.q = 1 + 2*n
            while True:
                if (self.q).is_prime():
```

```
                    break
                self.q += 2*n
        else:
            if q % (2*n) != 1:
                raise ValueError("Valor de 'q' não verifica a condição NTT")
            self.q = q

        self.F = GF(self.q) ;  self.R = PolynomialRing(self.F, name="w")
        w = (self.R).gen()

        g = (w^n + 1)
        xi = g.roots(multiplicities=False)[-1]
        self.xi = xi
        rs = [xi^(2*i+1)  for i in range(n)]
        self.base = crt_basis([(w - r) for r in rs])

    def ntt(self,f):

        def _expand_(f):
            u = f.list()
            return u + [0]*(self.n-len(u))

        def _ntt_(xi,N,f):
            if N==1:
                return f
            N_ = N/2 ; xi2 =  xi^2
            f0 = [f[2*i]   for i in range(N_)] ; f1 = [f[2*i+1] for i in␣
↪range(N_)]
            ff0 = _ntt_(xi2,N_,f0) ; ff1 = _ntt_(xi2,N_,f1)

            s  = xi ; ff = [self.F(0) for i in range(N)]
            for i in range(N_):
                a = ff0[i] ; b = s*ff1[i]
                ff[i] = a + b ; ff[i + N_] = a - b
                s = s * xi2
            return ff

        return _ntt_(self.xi,self.n,_expand_(f))

    def invNtt(self,ff):
        return sum([ff[i]*self.base[i] for i in range(self.n)])
```

MyMatrix Nesta classe são implementados as operações sobre matrizes e vetores. É de notar que ao longo da explicação da implementação do KYBER todos os cálculos entre elementos de matrizes e vetores serão realizadas com recurso a esta classe.

[177]:
```
#Classe que implementa as operações sobre matrizes e vetores
class MyMatrix:

    #Soma entre matrizes
    def sumMatrix(self,e1,e2,n) :

        for i in range(len(e1)):
            e1[i] = self.sumValue(e1[i], e2[i],n)

        return e1

    #Subtração entre matrizes
    def subMatrix(self,e1,e2,n) :

        for i in range(len(e1)):
            e1[i] = self.subValue(e1[i], e2[i],n)

        return e1

    #Multiplicação entre matrizes
    def multMatrix(self,vec1, vec2, n):

        for i in range(len(vec1)):
            vec1[i] = self.multValue(vec1[i], vec2[i],n)

        tmp = [0] * n
        for i in range(len(vec1)):
            tmp = self.sumValue(tmp, vec1[i],n)

        return tmp

    #Multiplicação entre matriz e vector
    def multMatrixVector(self,M,v,k,n) :
        for i in range(len(M)):
            for j in range(len(M[i])):
                M[i][j] = self.multValue(M[i][j], v[j],n)

        tmp = [[0] * n] * k
        for i in range(len(M)):
            for j in range(len(M[i])):
                tmp[i] = self.sumValue(tmp[i], M[i][j],n)

        return tmp

    #Soma elementos entre dois vetores
    def sumValue(self,ff1, ff2,n):
```

```
        res = []

        for i in range(n):
            res.append((ff1[i] + ff2[i]))

        return res

    #Multiplica elementos entre dois vetores
    def multValue(self,ff1, ff2,n):

        res = []

        for i in range(n):
            res.append((ff1[i] * ff2[i]))

        return res

    #Subtrai elementos entre dois vetores
    def subValue(self,ff1, ff2,n):

        res = []

        for i in range(n):
            res.append((ff1[i] - ff2[i]))

        return res
```

1.1.1 Resolução do Problema

1.1.2 Implementação PKE (IND-CPA)

Na classe abaixo é implementado o algoritmo PKE-IND-CPA, segundo a especificação que se encontra no paper, que irá cifrar uma informação aleatória gerada a partir do anel Rq dado por $F2[X]/(X^r+1)$. Esta classe irá ser utilizada, posteriormente, para conseguir implementar um KEM-IND-CPA e um PKE-IND-CCA, tal como é pedido no enunciado. Assim sendo, será necessário implementar 3 funcionalidades principais:

Geração do par de chaves: Esta secção foi implementada segundo o algoritmo 4 apresentado no paper que vai gerar uma chave privada e outra pública para serem usadas no *encrypt* e *decrypt*, através da função `keyGen`. Este algoritmo começa por determinar a matriz A Rq no dominio NTT e os vetores e, s Rq. Para isso utilizou-se as funções `Parse`, `CBD` que foram implementadas com recurso ao algoritmos 1 e 2, respetivamente. Para além desta funções também foram implementadas as funções de hash `XOF`, `G`, `PRF` que utilizam, respetivamente, Shake-128, Sha3-356 e Shake-256 tal como está previsto na documentação. As variáveis geradas são depois utilizadas para calcular a chave pública que será dada por $pk = (A * s + e, \rho)$, com $\rho = G(d)[32:]$ e d = valor pseudo-aleatório. A chave privada será dada por $sk = s$.

Cifra: A função `encrypt` foi implementada com recurso ao algoritmo 5 do paper que permite gerar um ciphertext de uma mensagem m gerada a partir do anel Rq, utilizando para isso a chave pública e um valor *coins* gerado aleatóriamente. Esta função começa por gerar novamente a matriz A^T Rq no dominio NTT e os vetores $e1, r$ Rq para além do vetor $e2$ Rq que serão utilizadas para calcular as duas componetes do ciphertext, *c1* e *c2*. Para calcular *c1* utilizou-se a função `Compress` que irá comprimir o vetor $u = A^T * r + e1$ de forma a emilinar bits de *lower-order*. Para gerar a componente *c2* utilizou-se novamente a função *Compress* que irá comprimir o vetor $v = t^T * r + e2 + Decompress(m, 1)$, sendo t = $pk[0]$ e m = mensagem a enviar. Desta forma, geramos o ciphertext $c = (Compress(u, du), Compress(v, dv))$

Decifra: A função `decrypt` foi implementada com recurso ao algoritmo 6 do paper e permite decifrar o *ciphertext* que recebe como pârametro utilizando a chave privada. Este algoritmo começa por recalcular os vetores u e v através da função `Decompress` que serão utilizadas para gerar a mensagem original calculada apartir da expressão $m = Compress(v - s^T * u, 1)$, com a s = chave privada.

[178]:
```
#Classe que implementa o KYBER_PKE
class KYBER_PKE:

    #Função de inicialização das variaveis a usar nos métodos
    def __init__(self):
        self.n, self.q, self.T, self.k, self.n1, self.n2, self.du, self.dv,␣
↪self.Rq = self.setup()

    #Parâmetros da técnica KYBER 512
    def setup(self):

        n = 256
        q = 7681
        k = 2
        n1 = 3
        n2 = 2
        du = 10
        dv = 4

        Zq.<w> = GF(q)[]
        fi = w^n + 1
        Rq.<w> = QuotientRing(Zq ,Zq.ideal(fi))

        T = NTT(n,q)

        return n,q,T,k,n1,n2,du,dv,Rq

    #Função auxiliar que transforma bytes em bits
    def bytesToBits(self, bytearr):
        bitarr = []

        for elem in bytearr:
```

```
            bitElemArr=[]

            for i in range(0,8):

                bitElemArr.append(mod(elem//2**(mod(i,8)),2))

                for i in range(0,len(bitElemArr)):

                    bitarr.append(bitElemArr[i])

        return bitarr

    #Função Hash que devolve um par
    def G(self, h):

        digest = hashes.Hash(hashes.SHA3_512())
        digest.update(bytes(h))
        g = digest.finalize()

        return g[:32],g[32:]

    #Função extendable output function (XOF)
    def XOF(self,b,b1,b2):

        digest = hashes.Hash(hashes.SHAKE128(int(self.q)))
        digest.update(b)
        digest.update(bytes(b1))
        digest.update(bytes(b2))
        m = digest.finalize()
        return m

    #Função pseudorandom function (PRF)
    def PRF(self,b,b1):

        digest = hashes.Hash(hashes.SHAKE256(int(self.q)))
        digest.update(b)
        digest.update(bytes(b1))
        return digest.finalize()

    #Função que faz sampling dos elementos em Rq - Algoritmo 1
    def Parse(self,b):

        coefs=[]

        i = 0
        j = 0
```

```
        while j < self.n:
            d1 = b[i] + 256 * mod(b[i+1],16)
            d2 = b[i+1]//16 + 16 * b[i+2]

            if d1 < self.q :
                coefs.append(d1)
                j = j+1
            if d2 < self.q and j<self.n:
                coefs.append(d2)
                j = j+1
            i = i+3

        return self.Rq(coefs)


    #Função que faz sampling dos elementos de uma distribuição binomial - Algoritmo 2
    def CBD(self,arrayB,nn):

        f=[0]*self.n

        bitArray = self.bytesToBits(arrayB)

        for i in range(256):

            a = 0
            b = 0

            for j in range(nn):

                a += bitArray[2*i*nn + j]
                b += bitArray[2*i*nn + nn + j]

            f[i] = a-b

        return self.Rq(f)

    #Função que apartir de bytes gera um polinómio f pertencente a Rq
    def Decode(self,arrayB,l):

        f = []

        bitArray = self.bytesToBits(arrayB)

        for i in range(len(arrayB)):
```

```
            fi = 0

            for j in range(l):

                fi += int(bitArray[i*l+j]) * 2**j

            f.append(fi)

        return self.Rq(f)

    #Função elimina alguns low-order bits no x
    def Compress(self,x,d) :

        coefs = x.list()

        newCoefs = []

        for coef in coefs:
            new = mod(round( int(2 ** d) / self.q * int(coef)), int(2 ** d))
            newCoefs.append(new)

        return self.Rq(newCoefs)

    #Função repõem o x parcialmente
    def Decompress(self,x,d) :

        coefs = x.list()

        newCoefs = []

        for coef in coefs:
            new = round(self.q / (2 ** d) * int(coef))
            newCoefs.append(new)

        return self.Rq(newCoefs)

    #FUNÇÃO: Gera a chave pública e privada a ser utilizada no processo de␣
↪cifra e decifra.
    def keyGen(self):

        mtx = MyMatrix()

        # d ← B32
        d = bytearray(os.urandom(32))

        # ( ,  ) := G(d)
        ro,sigma = self.G(d)
```

```
        # N := 0
        N = 0

        # Generate matrix Â  Rq in NTT domain
        A = []

        for i in range(self.k):
            A.append([])
            for j in range(self.k):
                index = i*(self.k)+j
                A[i].append(self.T.ntt(self.Parse(self.XOF(ro,j,i))))


        # Sample s  Rq from B 1
        s = []

        for i in range(self.k):

            s.insert(i,self.CBD(self.PRF(sigma,N), self.n1))
            N = N+1

        #Sample e  Rq from B 1
        e = []

        for i in range(self.k):

            e.insert(i,self.CBD(self.PRF(sigma,N), self.n1))
            N = N+1


        # ŝ := NTT(s), ê := NTT(e)
        for i in range(self.k) :
            s[i] = self.T.ntt(s[i])
            e[i] = self.T.ntt(e[i])

        # t1 := Â ∘ ŝ
        tAux = mtx.multMatrixVector(A,s,self.k,self.n)
        # t := t1 + ê
        t = mtx.sumMatrix(tAux,e, self.n)

        # pk := As + e
        pk = t, ro
        # sk := s
        sk = s

        return pk,sk
```

```
#FUNÇÃO: Cifra uma mensagem.
def encrypt(self,pk, m ,r):

    mtx = MyMatrix()

    # N := 0
    N = 0

    t, ro = pk

    # Generate matrix Â   Rq in NTT domain
    transposeA = []

    for i in range(self.k):

        transposeA.append([])

        for j in range(self.k):

            transposeA[i].append(self.T.ntt(self.Parse(self.XOF(ro,i,j))))

    # Sample r   Rq from B 1, ^r:= NTT(r)
    rr = []

    for i in range(self.k):
        rr.insert(i,self.T.ntt(self.CBD(self.PRF(r, N), self.n1)))
        N += 1

    # Sample e1   Rq from B 2
    e1 = []

    for i in range(self.k):
        e1.insert(i,self.CBD(self.PRF(r, N), self.n2))
        N += 1

    # Sample e2   Rq from B 2
    e2 = self.CBD(self.PRF(r, N), self.n2)

    #Â ∘ r̂
    uAux = mtx.multMatrixVector(transposeA, rr, self.k, self.n)

    #NTT-1( Â ∘ r̂)
    uAux2 = []
    for i in range(len(uAux)) :
        uAux2.append(self.T.invNtt(uAux[i]))
```

```
        # u := NTT( Â ∘ r̂) + e1
        uAux3 = mtx.sumMatrix(uAux2,e1, self.n)

        u = []
        for i in range(len(uAux3)) :
            u.append(self.Rq(uAux3[i]))

        #t ∘ r̂
        #t = [] + t
        vAux = mtx.multMatrix(t,rr,self.n)

        #NTT-1( t̂ ∘ r̂)
        vAux1 = self.T.invNtt(vAux)
        #NTT-1( t̂ ∘ r̂) + e2
        vAux2 = self.Rq(mtx.sumValue(vAux1,e2, self.n))

        #Decompress(m, 1)
        m1 = self.Decompress(m, 1)
        # v := NTT-1( t̂ ∘ r̂) + e2 + Decompress(m, 1)
        v = self.Rq(mtx.sumValue(vAux2,m1,self.n))

        # Compress(u, du)
        c1 = []

        for i in range(len(u)):
            c1.append(self.Compress(u[i],self.du))


        # Compress(v, dv)
        c2 = self.Compress(v,self.dv)

        return (c1,c2)


    def decrypt(self,sk, c):

        mtx = MyMatrix()

        c1, c2 = c

        #Decompress(c1,du)
        u = []

        for i in range(len(c1)):
            u.append(self.Decompress(c1[i],self.du))

        #Decompress(c2,dv)
```

```
        v = self.Decompress(c2,self.dv)

        s = sk

        #NTT(u)
        u1 = []
        for i in range(len(u)) :
            u1.append(self.T.ntt(u[i]))

        #ŝ ∘ NTT(u)
        mAux = mtx.multMatrix(s,u1,self.n)

        #v - NTT-1(ŝ ∘ NTT(u))
        mAux1 = mtx.subValue(v,self.T.invNtt(mAux), self.n)

        #Compress(v - NTT-1(ŝ ∘ NTT(u)), 1)
        m = self.Compress(self.Rq(mAux1), 1)

        return m
```

Cenário de teste De seguida apresentamos o cenário de teste para o PKE-IND-CPA, sendo que a função `Decode` que é apresentada na especificação do KYBER e implementada na classe KYBER_PKE será utilizada para transformar um byte array que representa a mensagem a enviar num polinómio $f \ Rq$ para ser utilizado na função *Decompress* de forma a conseguir implementar o algoritmo *encrypt* que é apresentado na especificação.

[179]:
```
kyber = KYBER_PKE()

public, private = kyber.keyGen()

m = "Mensagem a enviar"
print("MENSAGEM ORIGINAL: ")
print(m)

ciphertext = kyber.encrypt(public, kyber.Decode(m.encode(),1), os.urandom(32))
print("CIPHERTEXT: ")
print(ciphertext)

plaintext = kyber.decrypt(private, ciphertext)
print("DECRYPT TEXT: ")
print(plaintext)

print("**** ENCRYPT e DECRYPT funcinou? ****")
print(plaintext == kyber.Decode(m.encode(),1))
```

```
MENSAGEM ORIGINAL:
Mensagem a enviar
```

CIPHERTEXT:

```
([420*w^255 + 708*w^254 + 595*w^253 + 999*w^252 + 856*w^251 + 151*w^250 +
97*w^249 + 387*w^248 + 305*w^247 + 619*w^246 + 133*w^245 + 34*w^244 + 286*w^243
+ 330*w^242 + 604*w^241 + 818*w^240 + 522*w^239 + 244*w^238 + 755*w^237 +
196*w^236 + 44*w^235 + 295*w^234 + 833*w^233 + 570*w^232 + 541*w^231 + 843*w^230
+ 464*w^229 + 595*w^228 + 172*w^227 + 327*w^226 + 340*w^225 + 761*w^224 +
342*w^223 + 865*w^222 + 899*w^221 + 140*w^220 + 308*w^219 + 990*w^218 +
564*w^217 + 449*w^216 + 984*w^215 + 405*w^214 + 798*w^213 + 492*w^212 + 8*w^211
+ 901*w^210 + 29*w^209 + 217*w^208 + 487*w^207 + 281*w^206 + 260*w^205 +
674*w^204 + 469*w^203 + 913*w^202 + 1015*w^201 + 252*w^200 + 102*w^199 +
139*w^198 + 955*w^197 + 183*w^196 + 569*w^195 + 684*w^194 + 976*w^193 +
674*w^192 + 890*w^191 + 594*w^190 + 563*w^189 + 966*w^188 + 77*w^187 + 731*w^186
+ 696*w^185 + 426*w^184 + 236*w^183 + 986*w^182 + 906*w^181 + 1000*w^180 +
401*w^179 + 454*w^178 + 958*w^177 + 645*w^176 + 774*w^175 + 75*w^174 + 640*w^173
+ 416*w^172 + 975*w^171 + 698*w^170 + 974*w^169 + 583*w^168 + 875*w^167 +
799*w^166 + 826*w^165 + 141*w^164 + 858*w^163 + 307*w^162 + 144*w^161 +
181*w^160 + 863*w^159 + 225*w^158 + 323*w^157 + 674*w^156 + 927*w^155 +
375*w^154 + 600*w^153 + 328*w^152 + 870*w^151 + 66*w^150 + 941*w^149 + 16*w^148
+ 978*w^147 + 890*w^146 + 466*w^145 + 127*w^144 + 511*w^143 + 150*w^142 +
147*w^141 + 728*w^140 + 629*w^139 + 1009*w^138 + 29*w^137 + 129*w^136 +
427*w^135 + 561*w^134 + 261*w^133 + 635*w^132 + 329*w^131 + 410*w^130 +
420*w^129 + 454*w^128 + 92*w^127 + 1013*w^126 + 862*w^125 + 430*w^124 +
366*w^123 + 336*w^122 + 161*w^121 + 542*w^120 + 781*w^119 + 695*w^118 +
110*w^117 + 969*w^116 + 676*w^115 + 533*w^114 + 286*w^113 + 933*w^112 +
661*w^111 + 635*w^110 + 398*w^109 + 201*w^108 + 721*w^107 + 482*w^106 +
625*w^105 + 670*w^104 + 883*w^103 + 563*w^102 + 710*w^101 + 18*w^100 + 881*w^99
+ 679*w^98 + 257*w^97 + 560*w^96 + 638*w^95 + 84*w^94 + 462*w^93 + 797*w^92 +
482*w^91 + 16*w^90 + 902*w^89 + 490*w^88 + 484*w^87 + 346*w^86 + 403*w^85 +
427*w^84 + 130*w^83 + 575*w^82 + 18*w^81 + 580*w^80 + 311*w^79 + 462*w^78 +
976*w^77 + 190*w^76 + 193*w^75 + 51*w^74 + 343*w^73 + 245*w^72 + 424*w^71 +
388*w^70 + 387*w^69 + 318*w^68 + 532*w^67 + 971*w^66 + 826*w^65 + 71*w^64 +
406*w^63 + 297*w^62 + 925*w^61 + 777*w^60 + 806*w^59 + 243*w^58 + 772*w^57 +
717*w^56 + 320*w^55 + 816*w^54 + 248*w^53 + 254*w^52 + 504*w^51 + 51*w^50 +
838*w^49 + 219*w^48 + 29*w^47 + 224*w^46 + 350*w^45 + 744*w^44 + 860*w^43 +
92*w^42 + 253*w^41 + 771*w^40 + 570*w^39 + 523*w^38 + 831*w^37 + 183*w^36 +
304*w^35 + 336*w^34 + 421*w^33 + 240*w^32 + 806*w^31 + 641*w^30 + 154*w^29 +
519*w^28 + 84*w^27 + 711*w^26 + 139*w^25 + 971*w^24 + 15*w^23 + 345*w^22 +
540*w^21 + 622*w^20 + 468*w^19 + 691*w^18 + 450*w^17 + 450*w^16 + 629*w^15 +
182*w^14 + 50*w^13 + 753*w^12 + 833*w^11 + 181*w^10 + 575*w^9 + 865*w^8 + 24*w^7
+ 774*w^6 + 300*w^5 + 250*w^4 + 699*w^3 + 505*w^2 + 628*w + 81, 358*w^255 +
249*w^254 + 711*w^253 + 498*w^252 + 461*w^251 + 229*w^250 + 790*w^249 + 6*w^248
+ 970*w^247 + 643*w^246 + 211*w^245 + 37*w^244 + 543*w^243 + 585*w^242 +
208*w^241 + 261*w^240 + 928*w^239 + 616*w^238 + 372*w^237 + 688*w^236 +
663*w^235 + 47*w^234 + 376*w^233 + 882*w^232 + 629*w^231 + 911*w^230 + 418*w^229
+ 900*w^228 + 48*w^227 + 616*w^226 + 251*w^225 + 152*w^224 + 566*w^223 +
647*w^222 + 103*w^221 + 498*w^220 + 572*w^219 + 231*w^218 + 934*w^217 +
283*w^216 + 252*w^215 + 945*w^214 + 71*w^213 + 424*w^212 + 942*w^211 + 945*w^210
+ 268*w^209 + 579*w^208 + 941*w^207 + 572*w^206 + 16*w^205 + 856*w^204 +
```

```
328*w^203 + 409*w^202 + 852*w^201 + 468*w^200 + 353*w^199 + 128*w^198 +
771*w^197 + 848*w^196 + 673*w^195 + 97*w^194 + 225*w^193 + 500*w^192 + 492*w^191
+ 156*w^190 + 425*w^189 + 670*w^188 + 369*w^187 + 341*w^186 + 918*w^185 +
472*w^184 + 63*w^183 + 800*w^182 + 536*w^181 + 891*w^180 + 518*w^179 + 67*w^178
+ 639*w^177 + 60*w^176 + 967*w^175 + 817*w^174 + 184*w^173 + 540*w^172 +
953*w^171 + 122*w^170 + 910*w^169 + 167*w^168 + 366*w^167 + 760*w^166 +
718*w^165 + 962*w^164 + 19*w^163 + 212*w^162 + 912*w^161 + 1005*w^160 +
305*w^159 + 472*w^158 + 980*w^157 + 295*w^156 + 346*w^155 + 642*w^154 +
325*w^153 + 484*w^152 + 380*w^151 + 793*w^150 + 409*w^149 + 655*w^148 +
589*w^147 + 709*w^146 + 190*w^145 + 285*w^144 + 828*w^143 + 701*w^142 + 81*w^141
+ 874*w^140 + 557*w^139 + 149*w^138 + 70*w^137 + 726*w^136 + 161*w^135 +
706*w^134 + 962*w^133 + 454*w^132 + 358*w^131 + 221*w^130 + 787*w^129 +
223*w^128 + 326*w^127 + 194*w^126 + 229*w^125 + 437*w^124 + 511*w^123 +
970*w^122 + 378*w^121 + 1007*w^120 + 10*w^119 + 128*w^118 + 28*w^117 + 988*w^116
+ 665*w^115 + 143*w^114 + 337*w^113 + 293*w^112 + 104*w^111 + 826*w^110 +
947*w^109 + 963*w^108 + 974*w^107 + 888*w^106 + 998*w^105 + 732*w^104 +
389*w^103 + 130*w^102 + 676*w^101 + 317*w^100 + 294*w^99 + 880*w^98 + 194*w^97 +
853*w^96 + 426*w^95 + 785*w^94 + 3*w^93 + 132*w^92 + 552*w^91 + 915*w^90 +
40*w^89 + 1017*w^88 + 1004*w^87 + 666*w^86 + 450*w^85 + 54*w^84 + 262*w^83 +
52*w^82 + 841*w^81 + 872*w^80 + 742*w^79 + 907*w^78 + 451*w^77 + 202*w^76 +
60*w^75 + 798*w^74 + 962*w^73 + 294*w^72 + 929*w^71 + 775*w^70 + 123*w^69 +
22*w^68 + 877*w^67 + 908*w^66 + 522*w^65 + 598*w^64 + 894*w^63 + 109*w^62 +
988*w^61 + 908*w^60 + 522*w^59 + 627*w^58 + 584*w^57 + 585*w^56 + 996*w^55 +
2*w^54 + 173*w^53 + 204*w^52 + 356*w^51 + 508*w^50 + 505*w^49 + 14*w^48 +
223*w^47 + 960*w^46 + 856*w^45 + 907*w^44 + 329*w^43 + 530*w^42 + 618*w^41 +
32*w^40 + 903*w^39 + 391*w^38 + 2*w^37 + 664*w^36 + 978*w^35 + 362*w^34 +
42*w^33 + 739*w^32 + 855*w^31 + 83*w^30 + 914*w^29 + 89*w^28 + 326*w^27 +
27*w^26 + 475*w^25 + 630*w^24 + 450*w^23 + 163*w^22 + 588*w^21 + 202*w^20 +
346*w^19 + 335*w^18 + 599*w^17 + 834*w^16 + 309*w^15 + 834*w^14 + 216*w^13 +
155*w^12 + 386*w^11 + 848*w^10 + 971*w^9 + 585*w^8 + 576*w^7 + 601*w^6 + 330*w^5
+ 268*w^4 + 980*w^3 + 889*w^2 + 390*w + 676], 14*w^255 + 13*w^254 + 10*w^253 +
15*w^252 + 8*w^251 + 11*w^250 + 13*w^249 + 15*w^248 + 5*w^247 + 4*w^246 +
11*w^245 + w^244 + 9*w^243 + 3*w^242 + 9*w^241 + 6*w^240 + 15*w^239 + 9*w^238 +
9*w^237 + 13*w^236 + 2*w^235 + 7*w^234 + 4*w^233 + 6*w^232 + 11*w^231 + 5*w^230
+ w^229 + 4*w^228 + 7*w^227 + 8*w^226 + 9*w^225 + 2*w^224 + 7*w^223 + 14*w^222 +
4*w^221 + 13*w^220 + w^219 + 5*w^218 + 14*w^217 + 15*w^216 + 10*w^215 + 2*w^214
+ 13*w^213 + 3*w^212 + 3*w^211 + 11*w^210 + 12*w^209 + 3*w^208 + 13*w^207 +
4*w^206 + 9*w^205 + 4*w^204 + 8*w^203 + 6*w^202 + 6*w^201 + 7*w^200 + 3*w^199 +
11*w^198 + 15*w^196 + 7*w^195 + 9*w^193 + 5*w^192 + 6*w^191 + 9*w^190 + 7*w^189
+ 6*w^188 + 15*w^187 + 10*w^186 + 8*w^185 + 10*w^184 + 6*w^183 + 5*w^182 +
4*w^181 + 9*w^180 + 14*w^179 + 8*w^178 + 2*w^176 + 10*w^175 + 10*w^174 +
14*w^173 + w^172 + 13*w^171 + 10*w^170 + 14*w^169 + 13*w^168 + 5*w^167 + 4*w^166
+ 9*w^165 + 6*w^164 + 3*w^163 + 14*w^162 + 11*w^161 + 5*w^160 + 13*w^159 +
5*w^158 + 5*w^157 + 15*w^156 + 5*w^155 + 6*w^154 + 13*w^153 + 10*w^152 + w^151 +
2*w^150 + 13*w^149 + 2*w^148 + 3*w^147 + 5*w^146 + 8*w^145 + 15*w^144 + 12*w^143
+ 6*w^142 + 15*w^141 + 3*w^140 + 7*w^139 + w^138 + 12*w^137 + 15*w^136 +
15*w^135 + 10*w^134 + 11*w^133 + 5*w^132 + 14*w^131 + 10*w^130 + 11*w^129 +
8*w^128 + 7*w^127 + 3*w^126 + 12*w^125 + 9*w^124 + 14*w^122 + 7*w^121 + 8*w^120
```

```
+ 8*w^118 + 8*w^117 + 2*w^116 + 6*w^115 + 4*w^114 + 11*w^113 + 6*w^112 + 4*w^111
+ 3*w^110 + 2*w^109 + 13*w^108 + 10*w^107 + 12*w^106 + 12*w^105 + 5*w^104 +
15*w^103 + 9*w^102 + 2*w^101 + 10*w^100 + 2*w^99 + w^98 + 10*w^97 + 6*w^96 +
w^95 + w^94 + 11*w^93 + 6*w^92 + 12*w^91 + 5*w^89 + 2*w^88 + 14*w^87 + 8*w^86 +
10*w^85 + 7*w^84 + 8*w^83 + w^82 + w^81 + 3*w^80 + 2*w^79 + 12*w^78 + 6*w^77 +
10*w^75 + 7*w^74 + 15*w^73 + 3*w^72 + 4*w^71 + 12*w^70 + 4*w^69 + 2*w^68 +
6*w^67 + 6*w^66 + 3*w^65 + 13*w^64 + 6*w^63 + 14*w^62 + 13*w^60 + 9*w^59 +
14*w^58 + 10*w^57 + 14*w^55 + 3*w^54 + 2*w^53 + 15*w^52 + 7*w^51 + 4*w^50 + w^49
+ 3*w^48 + 8*w^47 + 13*w^46 + 15*w^44 + 11*w^43 + 10*w^42 + 8*w^41 + 14*w^40 +
14*w^39 + 8*w^38 + 14*w^37 + 5*w^36 + 13*w^35 + 13*w^34 + 6*w^33 + 14*w^32 +
7*w^31 + 12*w^29 + 3*w^27 + 7*w^26 + 14*w^25 + 8*w^24 + 5*w^23 + 3*w^22 +
12*w^21 + 4*w^20 + w^19 + 7*w^17 + 15*w^16 + 5*w^15 + 12*w^14 + 11*w^13 + 5*w^12
+ 15*w^11 + 3*w^10 + 6*w^9 + 15*w^8 + 3*w^7 + 4*w^6 + 13*w^5 + 15*w^4 + 14*w^3 +
10*w^2 + 2*w + 4)
DECRYPT TEXT:
w^15 + w^13 + w^12 + w^10 + w^9 + w^8 + w^6 + w^5 + w^3 + w + 1
**** ENCRYPT e DECRYPT funcinou? ****
True
```

1.1.3 Implementação KEM (IND-CPA)

Após a implementação do PKE-IND-CPA procedeu-se ao desenvolvimento do KEM que se encontra definido na classe KYBER_KEM. Tal como era pedido no enunciado, esta implementação do KEM consistiu no desenvolvimento de funções que permitissem uma segurança IND-CPA, ou seja, segurança contra ataques *Chosen Plaintext Attack*. Posto isto foram desenvolvidas três funções principais:

Geração do par de chaves: A função `keygen` tem como objetivo a criação do par de chaves pública e secreta que vão ser importantes no encapsulamento e desencapsulamento da chave. Simplesmente, foi utilizada a função *keygen* definida na classe KYBER_PKE.

Encapsulamento da chave: A função `encapsulate` recebe como *input* a chave pública e calcula a *hash* de um valor nonce aleatório $m \quad Rq$ utilizando a função `h` que irá representar a chave secreta *key*. De seguida fazemos a cifragem recorrendo à função *encrypt* da classe KYBER_PKE utilizando a variável m e a chave pública recebida como *input*. Desta função é retornado o criptograma c e a chave *key*.

Desencapsulamento da chave: A função `decapsulate` recebe como argumentos o criptograma e a chave secreta. De seguida, recorremos à função desenvolvida na classe KYBER_PKE para fazer a decifragem do criptograma. De seguida revela-se da chave secreta através da *hash* do resultado da decifragem. Caso a chave seja igual à gerada no processo de encapsulamento então a chave foi desencapsulada com sucesso.

[180]:
```python
class KYBER_KEM:

    #Função de inicialização das variaveis a usar nos métodos
    def __init__(self):
        self.pke = self.setup()
```

```
    #Parâmetros da técnica KYBER 512
    def setup(self):

        pke = KYBER_PKE()

        return pke

    def H(self,b):

        r = hashes.Hash(hashes.SHA3_256())
        r.update(b)
        return r.finalize()

    def encapsulate(self,pk):

        #nonce aleatório
        m = self.pke.Rq([choice([0, 1]) for i in range(kyber.n)])
        key = self.H(dumps(m))

        r = os.urandom(256)
        c = self.pke.encrypt(pk,m,r)

        return c, key

    def decapsulate(self,c,sk):

        m = self.pke.decrypt(sk,c)

        key = self.H(dumps(m))

        return key


    def keygen(self):

        pk,sk = self.pke.keyGen()

        return pk,sk
```

Cenário de Teste

[181]:
```
kem = KYBER_KEM()

pk, sk = kem.keygen()

c, k = kem.encapsulate(pk)
```

```
print("CIPHERTEXT: ")
print(c)
print()
print("ENCAP KEY: ")
print(k)

k2 = kem.decapsulate(c,sk)
print("DECAP KEY: ")
print(k2)

print("**** ENCAP e DECAP funcinou? ****")
print(k2 == k)
```

```
CIPHERTEXT:
([315*w^255 + 591*w^254 + 50*w^253 + 481*w^252 + 87*w^251 + 523*w^250 +
971*w^249 + 994*w^248 + 793*w^247 + 167*w^246 + 648*w^245 + 836*w^244 +
950*w^243 + 293*w^242 + 604*w^241 + 186*w^240 + 896*w^239 + 187*w^238 +
213*w^237 + 859*w^236 + 207*w^235 + 509*w^234 + 276*w^233 + 959*w^232 +
448*w^231 + 862*w^230 + 912*w^229 + 347*w^228 + 877*w^227 + 506*w^226 +
335*w^225 + 403*w^224 + 672*w^223 + 42*w^222 + 349*w^221 + 362*w^220 + 560*w^219
+ 983*w^218 + 826*w^217 + 120*w^216 + 734*w^215 + 370*w^214 + 540*w^213 +
304*w^212 + 1012*w^211 + 325*w^210 + 141*w^209 + 245*w^208 + 779*w^207 +
869*w^206 + 363*w^205 + 824*w^204 + 118*w^203 + 469*w^202 + 450*w^201 +
398*w^200 + 60*w^199 + 770*w^198 + 334*w^197 + 75*w^196 + 640*w^195 + 498*w^194
+ 31*w^193 + 7*w^192 + 476*w^191 + 834*w^190 + 213*w^189 + 648*w^188 + 92*w^187
+ 747*w^186 + 537*w^185 + 130*w^184 + 856*w^183 + 553*w^182 + 875*w^181 +
28*w^180 + 703*w^179 + 141*w^178 + 48*w^177 + 563*w^176 + 689*w^175 + 833*w^174
+ 732*w^173 + 411*w^172 + 583*w^171 + 260*w^170 + 66*w^169 + 457*w^168 +
12*w^167 + 982*w^166 + 696*w^165 + 481*w^164 + 671*w^163 + 493*w^162 + 928*w^161
+ 373*w^160 + 250*w^159 + 321*w^158 + 967*w^157 + 856*w^156 + 670*w^155 +
826*w^154 + 528*w^153 + 714*w^152 + 310*w^151 + 483*w^150 + 907*w^149 +
289*w^148 + 239*w^147 + 897*w^146 + 364*w^145 + 954*w^144 + 887*w^143 +
338*w^142 + 562*w^141 + 877*w^140 + 646*w^139 + 290*w^138 + 222*w^137 +
721*w^136 + 386*w^135 + 6*w^134 + 346*w^133 + 3*w^132 + 567*w^131 + 980*w^130 +
55*w^129 + 95*w^128 + 442*w^127 + 777*w^126 + 847*w^125 + 307*w^124 + 796*w^123
+ 44*w^122 + 627*w^121 + 270*w^120 + 553*w^119 + 20*w^118 + 14*w^117 + 214*w^116
+ 461*w^115 + 616*w^114 + 265*w^113 + 590*w^112 + 632*w^111 + 744*w^110 +
764*w^109 + 338*w^108 + 792*w^107 + 983*w^106 + 817*w^105 + 484*w^104 +
654*w^103 + 65*w^102 + 658*w^101 + 110*w^100 + 125*w^99 + 92*w^98 + 356*w^97 +
116*w^96 + 1017*w^95 + 929*w^94 + 194*w^93 + 409*w^92 + 977*w^91 + 925*w^90 +
1019*w^89 + 293*w^88 + 312*w^87 + 398*w^86 + 250*w^85 + 751*w^84 + 717*w^83 +
249*w^82 + 1022*w^81 + 949*w^80 + 332*w^79 + 837*w^78 + 606*w^77 + 642*w^76 +
712*w^75 + 427*w^74 + 416*w^73 + 985*w^72 + 634*w^71 + 404*w^70 + 294*w^69 +
482*w^68 + 54*w^67 + 992*w^66 + 827*w^65 + 708*w^64 + 482*w^63 + 813*w^62 +
283*w^61 + 358*w^60 + 108*w^59 + 637*w^58 + 135*w^57 + 565*w^56 + 181*w^55 +
196*w^54 + 326*w^53 + 939*w^52 + 994*w^51 + 426*w^50 + 969*w^49 + 748*w^48 +
57*w^47 + 822*w^46 + 76*w^45 + 607*w^44 + 961*w^43 + 441*w^42 + 29*w^41 +
```

```
571*w^40 + 807*w^39 + 360*w^38 + 191*w^37 + 676*w^36 + 703*w^35 + 765*w^34 +
656*w^33 + 696*w^32 + 678*w^31 + 652*w^30 + 306*w^29 + 331*w^28 + 552*w^27 +
390*w^26 + 372*w^25 + 816*w^24 + 404*w^23 + 704*w^22 + 345*w^21 + 51*w^20 +
240*w^19 + 199*w^18 + 383*w^17 + 814*w^16 + 682*w^15 + 714*w^14 + 708*w^13 +
446*w^12 + 793*w^11 + 594*w^10 + 640*w^9 + 176*w^8 + 651*w^7 + 143*w^6 + 319*w^5
+ 864*w^4 + 766*w^3 + 735*w^2 + 239*w + 455, 468*w^255 + 205*w^254 + 654*w^253 +
387*w^252 + 303*w^251 + 907*w^250 + 405*w^249 + 427*w^248 + w^247 + 573*w^246 +
859*w^245 + 881*w^244 + 284*w^243 + 313*w^242 + 17*w^241 + 214*w^240 + 564*w^239
+ 923*w^238 + 253*w^237 + 280*w^236 + 791*w^235 + 421*w^234 + 191*w^233 +
756*w^232 + 317*w^231 + 542*w^230 + 344*w^229 + 251*w^228 + 861*w^227 +
753*w^226 + 151*w^225 + 101*w^224 + 232*w^223 + 544*w^222 + 1018*w^221 +
990*w^220 + 838*w^219 + 938*w^218 + 734*w^217 + 185*w^216 + 549*w^215 +
448*w^214 + 751*w^213 + 1014*w^212 + 650*w^211 + 901*w^210 + 381*w^209 +
358*w^208 + 477*w^207 + 698*w^206 + 176*w^205 + 562*w^204 + 26*w^203 + 777*w^202
+ 428*w^201 + 129*w^200 + 330*w^199 + 671*w^198 + 652*w^197 + 850*w^196 +
562*w^195 + 691*w^194 + 662*w^193 + 772*w^192 + 677*w^191 + 431*w^190 +
458*w^189 + 370*w^188 + 508*w^187 + 1003*w^186 + 218*w^185 + 362*w^184 +
656*w^183 + 446*w^182 + 824*w^181 + 525*w^180 + 295*w^179 + 931*w^178 +
355*w^177 + 814*w^176 + 276*w^175 + 275*w^174 + 1014*w^173 + 1001*w^172 +
713*w^171 + 236*w^170 + 147*w^169 + 508*w^168 + 904*w^167 + 35*w^166 + 956*w^165
+ 553*w^164 + 91*w^163 + 432*w^162 + 913*w^161 + 330*w^160 + 480*w^159 +
532*w^158 + 502*w^157 + 236*w^156 + 965*w^155 + 426*w^154 + 292*w^153 +
1020*w^152 + 580*w^151 + 513*w^150 + 744*w^149 + 259*w^148 + 754*w^147 +
856*w^146 + 398*w^145 + 997*w^144 + 553*w^143 + 145*w^142 + 838*w^141 +
822*w^140 + 846*w^139 + 192*w^138 + 95*w^137 + 541*w^136 + 1010*w^135 +
439*w^134 + 697*w^133 + 6*w^132 + 984*w^131 + 494*w^130 + 195*w^129 + 906*w^128
+ 229*w^127 + 755*w^126 + 990*w^125 + 898*w^124 + 825*w^123 + 579*w^122 +
656*w^121 + 23*w^120 + 228*w^119 + 110*w^118 + 264*w^117 + 420*w^116 + 615*w^115
+ 1015*w^114 + 365*w^113 + 806*w^112 + 421*w^111 + 476*w^110 + 187*w^109 +
373*w^108 + 402*w^107 + 197*w^106 + 529*w^105 + 698*w^104 + 307*w^103 +
235*w^102 + 285*w^101 + 288*w^100 + 931*w^99 + 339*w^98 + 532*w^97 + 389*w^96 +
379*w^95 + 224*w^94 + 408*w^93 + 372*w^92 + 273*w^91 + 155*w^90 + 940*w^89 +
600*w^88 + 227*w^87 + 562*w^86 + 842*w^85 + 811*w^84 + 142*w^83 + 402*w^82 +
975*w^81 + 293*w^80 + 125*w^79 + 545*w^78 + 31*w^77 + 526*w^76 + 12*w^75 +
702*w^74 + 212*w^73 + 45*w^72 + 631*w^71 + 77*w^70 + 536*w^69 + 417*w^68 +
789*w^67 + 890*w^66 + 572*w^65 + 144*w^64 + 1005*w^63 + 33*w^62 + 135*w^61 +
865*w^60 + 392*w^59 + 161*w^58 + 1008*w^57 + 280*w^56 + 483*w^55 + 898*w^54 +
988*w^53 + 888*w^52 + 817*w^51 + 51*w^50 + 419*w^49 + 154*w^48 + 396*w^47 +
452*w^46 + 79*w^45 + 601*w^44 + 419*w^43 + 129*w^42 + 343*w^41 + 655*w^40 +
312*w^39 + 909*w^38 + 429*w^37 + 737*w^36 + 961*w^35 + 284*w^34 + 139*w^33 +
835*w^32 + 274*w^31 + 696*w^30 + 187*w^29 + 47*w^28 + 428*w^27 + 96*w^26 +
203*w^25 + 390*w^24 + 127*w^23 + 479*w^22 + 303*w^21 + 655*w^20 + 587*w^19 +
575*w^18 + 469*w^17 + 522*w^16 + 599*w^15 + 951*w^14 + 736*w^13 + 328*w^12 +
189*w^11 + 577*w^10 + 454*w^9 + 418*w^8 + 695*w^7 + 382*w^6 + 893*w^5 + 644*w^4
+ 438*w^3 + 864*w^2 + 945*w + 371], 15*w^255 + 9*w^254 + 5*w^253 + 14*w^252 +
7*w^251 + 4*w^250 + 10*w^248 + 7*w^247 + 6*w^246 + 13*w^245 + 4*w^244 + 6*w^243
+ 5*w^242 + 10*w^241 + 10*w^240 + 14*w^239 + 12*w^238 + w^237 + 8*w^236 +
13*w^235 + 3*w^234 + 6*w^233 + 11*w^232 + 8*w^231 + 13*w^230 + 12*w^229 +
```

```
10*w^228 + 4*w^227 + 11*w^226 + 3*w^225 + 2*w^224 + 11*w^223 + 14*w^222 +
12*w^221 + 4*w^220 + 12*w^219 + 15*w^218 + 2*w^217 + 7*w^216 + 10*w^215 +
8*w^214 + 8*w^213 + 11*w^212 + 5*w^211 + 5*w^210 + 12*w^209 + 5*w^208 + 14*w^207
+ 3*w^206 + 3*w^205 + 6*w^203 + 8*w^202 + 5*w^201 + 8*w^200 + w^199 + 9*w^198 +
8*w^197 + 4*w^195 + 10*w^194 + 3*w^193 + 9*w^191 + 9*w^190 + 3*w^189 + 5*w^188 +
6*w^187 + 6*w^186 + 13*w^184 + 10*w^183 + 3*w^182 + 11*w^181 + 13*w^180 +
11*w^178 + 6*w^177 + 5*w^176 + 4*w^175 + 12*w^174 + 8*w^173 + 6*w^172 + 6*w^171
+ 6*w^170 + 2*w^169 + 7*w^168 + 14*w^167 + w^166 + 8*w^165 + 12*w^164 + 3*w^163
+ 9*w^162 + 10*w^161 + 3*w^160 + w^158 + 9*w^157 + 11*w^156 + 3*w^155 + 5*w^154
+ 13*w^153 + 9*w^152 + 13*w^150 + w^149 + w^148 + 15*w^147 + 6*w^146 + 9*w^145 +
3*w^144 + 14*w^143 + w^142 + 12*w^141 + 2*w^140 + 5*w^139 + 8*w^138 + 4*w^137 +
8*w^136 + 12*w^135 + 12*w^134 + 3*w^132 + 3*w^131 + 14*w^130 + w^129 + 7*w^127 +
9*w^126 + 9*w^125 + 6*w^124 + 4*w^123 + 11*w^122 + 12*w^121 + 15*w^120 + w^119 +
5*w^118 + 9*w^117 + 12*w^116 + 7*w^115 + 9*w^114 + 4*w^113 + 3*w^112 + 3*w^111 +
8*w^110 + 6*w^109 + 2*w^108 + w^107 + 3*w^106 + 13*w^105 + 11*w^103 + 2*w^102 +
w^101 + 11*w^99 + 2*w^98 + 3*w^97 + 14*w^96 + 5*w^95 + 8*w^94 + 9*w^93 + 10*w^92
+ 4*w^91 + 5*w^90 + 12*w^89 + 7*w^88 + 12*w^87 + 4*w^85 + 10*w^83 + 9*w^82 +
w^81 + 12*w^80 + 10*w^79 + 4*w^78 + 6*w^77 + 13*w^75 + 4*w^74 + 7*w^72 + w^71 +
12*w^70 + 6*w^69 + 15*w^68 + 3*w^67 + 11*w^66 + 13*w^65 + 14*w^64 + w^63 +
3*w^62 + 2*w^60 + 5*w^59 + 6*w^57 + 10*w^55 + 4*w^54 + 14*w^53 + 3*w^52 +
15*w^51 + 4*w^50 + 4*w^49 + 13*w^48 + 13*w^47 + 6*w^46 + 10*w^45 + 12*w^43 +
4*w^42 + 10*w^41 + 14*w^40 + 3*w^39 + 8*w^38 + 6*w^37 + 9*w^36 + 6*w^35 + 6*w^34
+ 11*w^33 + 14*w^32 + 11*w^31 + 9*w^30 + w^29 + 2*w^28 + 9*w^26 + 11*w^25 +
5*w^24 + 6*w^23 + 2*w^22 + 3*w^21 + w^20 + 10*w^18 + 8*w^17 + 2*w^16 + 11*w^15 +
15*w^14 + 2*w^13 + 15*w^12 + 9*w^10 + 15*w^9 + 8*w^8 + 13*w^7 + 13*w^6 + 13*w^5
+ 2*w^4 + w^3 + 12*w^2 + 11*w + 11)

ENCAP KEY:
b'\xfa\xd0\xff\\\xe2\xf0\x11b\xc6d\xd5\x979\xd4\xef\x85\x87n\n\xb5\x18%\x97#P\xc
cT\x82K6n9'
DECAP KEY:
b'\xfa\xd0\xff\\\xe2\xf0\x11b\xc6d\xd5\x979\xd4\xef\x85\x87n\n\xb5\x18%\x97#P\xc
cT\x82K6n9'
**** ENCAP e DECAP funcinou? ****
True
```

1.1.4 Implementação PKE (IND-CCA)

Tal como é pedido no enunciado, na classe abaixo é implementado o algoritmo PKE-IND-CCA que irá cifrar uma mensagem utilizando para isso as transformação FOT (Fujisaki-Okamoto) para assim transformar um PKE-IND-CPA em PKE-IND-CCA. Assim sendo, será necessário implementar 3 funcionalidades principais:

Geração do par de chaves: A função de geração de chaves será dada pela função *keyGen* da classe KYBER_PKE

Cifra: Para implementação da função `encryptCCA` implementou-se as transformações FOT segunda

a expressão:

$$E(x) \equiv \vartheta\, r \leftarrow h \;.\; \vartheta\, y \leftarrow x \oplus g(r) \;.\; \vartheta\, c \leftarrow f(r, h(r \,\|\, y)) \;.\; (y\,,\, c)$$

Nesta função começamos por gerar um valor aleatório r Rq e a sua hash apartir da função `g`. Este valor será que será utilizado juntamente, com a mensagem a cifrar x, na operação `xor` para formar a variariavel y. Esta variável será concatenada com o valor de r e assim formar a *hash* $a = g(r \,\|\, y)$. É de notar que tanto a função h como a função g que aparecem na expressão acima são implementas utilizando a mesma função hash. Tanto a variavel a com a r serão utilizadas pela função f que corresponde à função *encrypt* implementada na classe KYBER_PKE. No final obtemos os criptogramas parciais c e y com $c = f(r,a)$.

Decifra: Para implementação da função `decryptCCA` implementou-se as transformações FOT segunda a expressão:

$$D(y, c) \equiv \vartheta\, r \leftarrow D(c) \;.\; \text{if } \; c \neq f(r, h(r \| y)) \; \text{ then } \; \perp \text{ else } \; y \oplus g(r)$$

Este algoritmo começa por decifrar o criptograma parcial c utilizando para isso a função *decrypt* da classe KYBER_PKE, devolvendo como resultado o valor de r. Este valor vai ser novamente utilizado juntamente com a variável y para calcular a variável $aux = f(r, h(r \| y))$, sendo esta função f dada pela função *encrypt* da classe KYBER_PEK. De seguida, confirma-se se o processo de decifra correr como era esperado comparando o valor de c e de aux. Caso seja igual, então significa que a operação correr bem e determinamos a mensagem que foi cifrada através da função *xor*. Caso não seja igual, então a operação não teve sucesso e por isso o processo de decifra falhou.

[182]:
```
class KYBER_PKE_CCA:

    #Função de inicialização das variaveis a usar nos métodos
    def __init__(self):
        self.pke, self.pk, self.sk = self.setup()

    #Parâmetros da técnica KYBER 512
    def setup(self):

        pke = KYBER_PKE()

        pk, sk = pke.keyGen()

        return pke,pk,sk

    #FUNÇÃO: Operação de XOR
    def xor(self,key, text):

        return bytes(a ^^ b for a, b in zip(key, text))

    #Função de hash h e g
    def g(self,b):

        r = hashes.Hash(hashes.SHA3_256())
        r.update(b)
        return r.finalize()
```

```
    #Função de encrypt para PKE-CCA
    def encryptCCA(self,x):

        #  r←h : valor aleatório a pretencer ao anel Rq
        r = self.pke.Rq([choice([0, 1]) for i in range(self.pke.n)])

        #  y← x   g(r)
        y = self.xor(x, self.g(bytes(r)))

        # c←f(r,h(r y)), f = encrypt do KYBER_PKE
        c = self.pke.encrypt(self.pk,r,self.g(bytes(r)+y))

        return (y, c)

    #Função de decrypt para PKE-CCA
    def decryptCCA(self,y,c):

        #  r←D(c), D = decrypt do KYBER_PKE
        r = self.pke.decrypt(self.sk, c)

        # f(r,h(r y))
        aux = self.pke.encrypt(self.pk,r,self.g(bytes(r)+y))

        # c   f(r,h(r y))
        if c[0] != aux[0]:

            #
            return None
        else:

            # y   g(r)
            return self.xor(y, self.g(bytes(r)))
```

Cenário de Teste

[183]:
```
pkeCCA = KYBER_PKE_CCA()

m = "Mensagem a enviar em modo CCA"
print("MENSAGEM ORIGINAL: ")
print(m)

parcialCipher = pkeCCA.encryptCCA(m.encode())
print("CIPHERTEXT: ")
print(parcialCipher)

(y,c) = parcialCipher
```

```
result = pkeCCA.decryptCCA(y,c)

if result == None:
    print("**** ENCRYPT-CCA e DECRYPT-CCA não funcinou ****")
else:
    print("**** ENCRYPT-CCA e DECRYPT-CCA funcinou ****")
    print(result.decode())
```

```
MENSAGEM ORIGINAL:
Mensagem a enviar em modo CCA
CIPHERTEXT:
(b'\x8c\xccK\n\x90\xf6\xdd\x83a\xc0-\xd0uR\xe7;\xe9\xbf\x9fH\xf7\x94\xda\x80~\x9
ei^"', ([383*w^255 + 388*w^254 + 697*w^253 + 153*w^252 + 635*w^251 + 549*w^250 +
778*w^249 + 255*w^248 + 851*w^247 + 77*w^246 + 949*w^245 + 1010*w^244 +
733*w^243 + 632*w^242 + 369*w^241 + 19*w^240 + 931*w^239 + 218*w^238 + 251*w^237
+ 400*w^236 + 902*w^235 + 432*w^234 + 66*w^233 + 243*w^232 + 825*w^231 +
617*w^230 + 999*w^229 + 892*w^228 + 668*w^227 + 536*w^226 + 697*w^225 +
103*w^224 + 190*w^223 + 953*w^222 + 149*w^221 + 424*w^220 + 510*w^219 +
212*w^218 + 469*w^217 + 836*w^216 + 132*w^215 + 717*w^214 + 664*w^213 +
273*w^212 + 991*w^211 + 408*w^210 + 556*w^209 + 742*w^208 + 976*w^207 +
679*w^206 + 396*w^205 + 464*w^204 + 823*w^203 + 266*w^202 + 45*w^201 + 245*w^200
+ 852*w^199 + 30*w^198 + 451*w^197 + 839*w^196 + 824*w^195 + 385*w^194 +
256*w^193 + 433*w^192 + 442*w^191 + 169*w^190 + 534*w^189 + 751*w^188 +
266*w^187 + 787*w^186 + 957*w^185 + 310*w^184 + 814*w^183 + 201*w^182 +
884*w^181 + 624*w^180 + 125*w^179 + 477*w^178 + 1000*w^177 + 471*w^176 +
873*w^175 + 635*w^174 + 232*w^173 + 827*w^172 + 537*w^171 + 396*w^170 +
343*w^169 + 925*w^168 + 31*w^167 + 53*w^166 + 702*w^165 + 416*w^164 + 937*w^163
+ 830*w^162 + 794*w^161 + 965*w^160 + 356*w^159 + 23*w^158 + 59*w^157 +
859*w^156 + 234*w^155 + 861*w^154 + 873*w^153 + 360*w^152 + 908*w^151 +
477*w^150 + 186*w^149 + 361*w^148 + 617*w^147 + 604*w^146 + 891*w^145 +
346*w^144 + 163*w^143 + 771*w^142 + 187*w^141 + 646*w^140 + 230*w^139 +
595*w^138 + 999*w^137 + 546*w^136 + 912*w^135 + 593*w^134 + 624*w^133 + 2*w^132
+ 542*w^131 + 365*w^130 + 728*w^129 + 95*w^128 + 413*w^127 + 402*w^126 +
561*w^125 + 237*w^124 + 330*w^123 + 358*w^122 + 772*w^121 + 325*w^120 +
658*w^119 + 140*w^118 + 268*w^117 + 947*w^116 + 118*w^115 + 288*w^114 +
595*w^113 + 920*w^112 + 156*w^111 + 410*w^110 + 566*w^109 + 748*w^108 +
696*w^107 + 675*w^106 + 1003*w^105 + 943*w^104 + 448*w^103 + 69*w^102 +
879*w^101 + 272*w^100 + 680*w^99 + 211*w^98 + 849*w^97 + 427*w^96 + 243*w^95 +
567*w^94 + 461*w^93 + 213*w^92 + 976*w^91 + 624*w^90 + 854*w^89 + 25*w^88 +
675*w^87 + 917*w^86 + 761*w^85 + 412*w^84 + 499*w^83 + 94*w^82 + 330*w^81 +
761*w^80 + 824*w^79 + 507*w^78 + 764*w^77 + 469*w^76 + 286*w^75 + 2*w^74 +
159*w^73 + 401*w^72 + 935*w^71 + 261*w^70 + 241*w^69 + 511*w^68 + 566*w^67 +
150*w^66 + 224*w^65 + 471*w^64 + 441*w^63 + 689*w^62 + 85*w^61 + 253*w^60 +
294*w^59 + 562*w^58 + 24*w^57 + 871*w^56 + 90*w^55 + 842*w^54 + 356*w^53 +
736*w^52 + 783*w^51 + 568*w^50 + 53*w^49 + 743*w^48 + 963*w^47 + 621*w^46 +
871*w^45 + 183*w^44 + 502*w^43 + 115*w^42 + 743*w^41 + 406*w^40 + 857*w^39 +
```

```
438*w^38 + 907*w^37 + 6*w^36 + 829*w^35 + 608*w^34 + 258*w^33 + 690*w^32 +
271*w^31 + 665*w^30 + 388*w^29 + 35*w^28 + 7*w^27 + 489*w^26 + 382*w^25 +
119*w^24 + 177*w^23 + 210*w^22 + 298*w^21 + 530*w^20 + 116*w^19 + 244*w^18 +
52*w^17 + 940*w^16 + 583*w^15 + 794*w^14 + 877*w^13 + 502*w^12 + 10*w^11 +
781*w^10 + 630*w^9 + 81*w^8 + 56*w^7 + 692*w^6 + 513*w^5 + 94*w^4 + 113*w^3 +
92*w^2 + 329*w + 692, 801*w^255 + 999*w^254 + 685*w^253 + 838*w^252 + 737*w^251
+ 320*w^250 + 5*w^249 + 79*w^248 + 641*w^247 + 177*w^246 + 992*w^245 + 295*w^244
+ 8*w^243 + 55*w^242 + 541*w^241 + 102*w^240 + 552*w^239 + 234*w^238 + 219*w^237
+ 964*w^236 + 682*w^235 + 484*w^234 + 998*w^233 + 201*w^232 + 798*w^231 +
489*w^230 + 879*w^229 + 497*w^228 + 12*w^227 + 808*w^226 + 315*w^225 + 569*w^224
+ 323*w^223 + 449*w^222 + 219*w^221 + 312*w^220 + 809*w^219 + 471*w^218 +
157*w^217 + 668*w^216 + 50*w^215 + 234*w^214 + 1010*w^213 + 10*w^212 +
1001*w^211 + 749*w^210 + 9*w^209 + 300*w^208 + 564*w^207 + 757*w^206 + 644*w^205
+ 68*w^204 + 688*w^203 + 369*w^202 + 440*w^201 + 259*w^200 + 537*w^199 +
27*w^198 + 523*w^197 + 305*w^196 + 744*w^195 + 889*w^194 + 477*w^193 + 563*w^192
+ 807*w^191 + 399*w^190 + 213*w^189 + 802*w^188 + 567*w^187 + 556*w^186 +
641*w^185 + 902*w^184 + 725*w^183 + 91*w^182 + 882*w^181 + 691*w^180 + 640*w^179
+ 962*w^178 + 691*w^177 + 196*w^176 + 1006*w^175 + 487*w^174 + 907*w^173 +
626*w^172 + 239*w^171 + 223*w^170 + 968*w^169 + 124*w^168 + 881*w^167 +
124*w^166 + 93*w^165 + 199*w^164 + 853*w^163 + 856*w^162 + 247*w^161 + 132*w^160
+ 648*w^159 + 186*w^158 + 857*w^157 + 542*w^156 + 231*w^155 + 103*w^154 +
843*w^153 + 263*w^152 + 72*w^151 + 274*w^150 + 531*w^149 + 435*w^148 + 442*w^147
+ 954*w^146 + 456*w^145 + 667*w^144 + 185*w^143 + 899*w^142 + 46*w^141 +
977*w^140 + 400*w^139 + 107*w^138 + 781*w^137 + 356*w^136 + 932*w^135 +
898*w^134 + 705*w^133 + 524*w^132 + 145*w^131 + 291*w^130 + 133*w^129 +
508*w^128 + 738*w^127 + 451*w^126 + 105*w^125 + 49*w^124 + 461*w^123 + 168*w^122
+ 924*w^121 + 13*w^120 + 537*w^119 + 606*w^118 + 710*w^117 + 395*w^116 +
571*w^115 + 797*w^114 + 598*w^113 + 429*w^112 + 886*w^111 + 200*w^110 +
727*w^109 + 354*w^108 + 780*w^107 + 350*w^106 + 128*w^105 + 948*w^104 +
913*w^103 + 168*w^102 + 508*w^101 + 362*w^100 + 763*w^99 + 432*w^98 + 392*w^97 +
393*w^96 + 75*w^95 + 955*w^94 + 393*w^93 + 93*w^92 + 699*w^91 + 376*w^90 +
938*w^89 + 553*w^88 + 168*w^87 + 977*w^86 + 707*w^85 + 161*w^84 + 105*w^83 +
538*w^82 + 899*w^81 + 489*w^80 + 968*w^79 + 494*w^78 + 323*w^77 + 607*w^76 +
408*w^75 + 600*w^74 + 874*w^73 + 793*w^72 + 563*w^71 + 397*w^70 + 51*w^69 +
232*w^68 + 145*w^67 + 909*w^66 + 742*w^65 + 761*w^64 + 861*w^63 + 420*w^62 +
495*w^61 + 776*w^60 + 725*w^59 + 426*w^58 + 275*w^57 + 537*w^56 + 955*w^55 +
786*w^54 + 93*w^53 + 131*w^52 + 1020*w^51 + 989*w^50 + 782*w^49 + 645*w^48 +
666*w^47 + 82*w^46 + 689*w^45 + 878*w^44 + 1005*w^43 + 578*w^42 + 610*w^41 +
889*w^40 + 814*w^39 + 980*w^38 + 1004*w^37 + 996*w^36 + 593*w^35 + 146*w^34 +
448*w^33 + 777*w^32 + 724*w^31 + 243*w^30 + 513*w^29 + 202*w^28 + 887*w^27 +
785*w^26 + 638*w^25 + 838*w^24 + 473*w^23 + 695*w^22 + 947*w^21 + 121*w^20 +
731*w^19 + 551*w^18 + 916*w^17 + 300*w^16 + 357*w^15 + 502*w^14 + 141*w^13 +
451*w^12 + 210*w^11 + 818*w^10 + 169*w^9 + 495*w^8 + 10*w^7 + 415*w^6 + 581*w^5
+ 525*w^4 + 526*w^3 + 766*w^2 + 282*w + 958], 3*w^255 + 2*w^254 + 5*w^253 +
15*w^252 + 12*w^251 + 6*w^249 + 15*w^248 + 14*w^247 + 6*w^246 + 14*w^245 +
6*w^244 + 13*w^243 + 4*w^242 + 5*w^240 + 5*w^239 + 13*w^238 + 3*w^237 + 4*w^236
+ 2*w^235 + 8*w^234 + w^233 + 10*w^232 + 5*w^231 + w^230 + 12*w^228 + 13*w^227 +
5*w^225 + 6*w^223 + 12*w^222 + 8*w^221 + 13*w^219 + 14*w^218 + 9*w^217 +
```

```
13*w^216 + 11*w^215 + 10*w^214 + 6*w^213 + 5*w^211 + 10*w^210 + 5*w^209 +
13*w^208 + 8*w^207 + 7*w^206 + 14*w^204 + 15*w^203 + 11*w^202 + 8*w^201 +
8*w^200 + 14*w^199 + 4*w^198 + 5*w^197 + 13*w^195 + 10*w^194 + 3*w^193 +
13*w^192 + 12*w^190 + 13*w^189 + 10*w^188 + w^187 + 4*w^186 + 6*w^185 + 3*w^184
+ 15*w^183 + w^182 + 12*w^181 + 8*w^180 + 15*w^179 + 3*w^178 + 9*w^177 + 6*w^176
+ 2*w^175 + 9*w^174 + 13*w^173 + 5*w^172 + 14*w^171 + 4*w^170 + 11*w^169 + w^168
+ 7*w^167 + 3*w^166 + 2*w^165 + 3*w^164 + 14*w^163 + 7*w^162 + 14*w^161 +
5*w^160 + 12*w^159 + 4*w^158 + 10*w^157 + 14*w^156 + 8*w^155 + 11*w^153 +
10*w^152 + 4*w^151 + 7*w^150 + w^149 + 15*w^148 + 15*w^147 + 5*w^146 + 11*w^145
+ w^144 + 13*w^143 + 12*w^142 + 2*w^141 + 3*w^140 + 7*w^139 + 6*w^138 + 5*w^137
+ 13*w^136 + 7*w^135 + 11*w^134 + 4*w^133 + 15*w^132 + w^131 + 7*w^130 +
15*w^129 + 9*w^128 + 6*w^127 + 2*w^126 + 15*w^125 + 10*w^124 + 5*w^123 + 3*w^122
+ 13*w^120 + 5*w^119 + 7*w^118 + 3*w^117 + w^116 + 4*w^115 + 15*w^113 + 11*w^112
+ 5*w^111 + 7*w^110 + 14*w^109 + 8*w^108 + 12*w^107 + w^106 + 8*w^105 + 11*w^104
+ 14*w^103 + 12*w^101 + w^100 + 4*w^99 + 15*w^98 + 12*w^97 + 7*w^96 + 14*w^95 +
2*w^94 + 15*w^93 + 7*w^92 + 8*w^91 + 8*w^90 + 5*w^89 + 12*w^88 + 6*w^87 + 9*w^86
+ 5*w^85 + 10*w^84 + 9*w^83 + 8*w^82 + 6*w^81 + 14*w^80 + 13*w^79 + 12*w^77 +
12*w^75 + 4*w^73 + 10*w^72 + 15*w^71 + 9*w^70 + 12*w^69 + 15*w^67 + w^66 +
10*w^65 + 13*w^63 + 12*w^62 + 10*w^61 + 11*w^60 + 4*w^59 + 13*w^58 + 12*w^57 +
4*w^56 + 2*w^55 + 3*w^54 + 13*w^53 + 15*w^52 + 7*w^51 + 3*w^50 + 3*w^49 + 4*w^48
+ 10*w^47 + 14*w^46 + 13*w^44 + 4*w^43 + 2*w^42 + 13*w^40 + 3*w^39 + 13*w^38 +
7*w^37 + 12*w^36 + 10*w^35 + 6*w^34 + 11*w^33 + 3*w^32 + 9*w^31 + 13*w^30 +
13*w^29 + 9*w^28 + 4*w^27 + 2*w^26 + 7*w^25 + 8*w^24 + 6*w^23 + w^22 + 15*w^21 +
2*w^20 + 15*w^19 + w^18 + 6*w^17 + 3*w^16 + 6*w^15 + 14*w^14 + 8*w^13 + 2*w^12 +
13*w^11 + 10*w^10 + 9*w^9 + 15*w^8 + 7*w^7 + 11*w^6 + 6*w^5 + 4*w^4 + 7*w^3 +
11*w^2 + 14*w + 14))
**** ENCRYPT-CCA e DECRYPT-CCA funcinou ****
Mensagem a enviar em modo CCA
```