Criptografia pós-quântica -> CRYSTALS-KYBER

Enunciado do Problema

Este problema é dedicado às candidaturas finalistas ao concurso NIST Post-Quantum Cryptography na categoria de criptosistemas PKE-KEM. Em Julho de 2022 foi selecionada para “standartização” a candidatura KYBER.

```
1. O objetivo deste trabalho é a criação de protótipos em Sagemath para o algoritm
o KYBER.
2. Para esta técnica pretende-se implementar um KEM, que seja IND-CPA seguro, e um
PKE que seja IND-CCA seguro.
```

Descrição do problema

Este trabalho é dedicado às candidaturas finalistas do concurso NIST Post-Quantum Cryptography na categoria de sistemas criptográficos PKE-KEM. Em julho de 2022, a candidatura KYBER foi selecionada para "standardização", sendo considerada uma das soluções mais promissoras para a segurança pós-quântica. O objetivo deste trabalho consiste em criar um protótipo em Sagemath para o algoritmo KYBER, implementando um KEM, que seja IND-CPA seguro, e um PKE que seja IND-CCA seguro.

Neste contexto, foram desenvolvidas classes Python/SageMath, que incluem as versões KEM-IND-CPA e PKE-IND-CCA. Estas implementações foram construídas com base nas especificações fornecidas em https://www.dropbox.com/sh/mx4bybl0d6e9g1m/AACKK0WvVdJiqd2LgDGR-wmva/Kyber-20201001?dl=0&subfolder_nav_tracking=1, em https://pq-crystals.org/kyber/index.shtml e em https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

Abordagem e código

Esta primeira parte do código do algoritmo CRYSTALS-KYBER começa por importar as bibliotecas necessárias e por declarar os parâmetros. Em seguida, define anéis e quocientes para trabalhar com polinômios. A classe NTT é definida para realizar o Number Theoretic Transform, que é uma operação fundamental no algoritmo Kyber. A classe inclui métodos para calcular a NTT e a inversa da NTT, bem como gerar polinômios aleatórios.

Várias funções auxiliares também são definidas para executar operações em objetos NTT, como adição, subtração e multiplicação. As funções compress e decompress são implementadas de acordo com a documentação do algoritmo Kyber, permitindo comprimir e descomprimir polinômios. Funções adicionais são também implementadas para aplicar compressão e decompressão de forma recursiva em matrizes ou em arrays.

O código também inclui a implementação de várias funções criptográficas, como XOF, H, G, PRF e KDF, seguindo a documentação do Kyber. A função parse é usada para converter uma sequência de bytes num polinômio e a função CBD (Centered Binomial Distribution) é implementada para gerar distribuições centradas binomiais.

Por fim, são implementadas funções para realizar operações de multiplicação e adição entre matrizes e vetores de objetos NTT. Estas funções são usadas no algoritmo Kyber para realizar operações com chaves e mensagens cifradas.

In [34]:

```
# Importação das bibliotecas necessárias
```

```
import os
from hashlib import shake_128, shake_256, sha256, sha512
from bitstring import BitArray
from random import choice
import ast


# Declaração dos pârametros
n = 256

q = next_prime(3*n)
while q % (2*n) != 1:
    q = next_prime(q+1)


# Declaração dos anéis necessários
_Z.<w> = ZZ[]
R.<w> = QuotientRing(_Z, _Z.ideal(w^n + 1))

_q.<w> = GF(q)[]
_Rq.<w> = QuotientRing(_q, _q.ideal(w^n + 1))

Rq = lambda x : _Rq(R(x))


# Implementação da Classe NTT (Number Theoretic Transform)
class NTT(object):

    def __init__(self, n=128, q=None):
        if not  n in [32,64,128,256,512,1024,2048]:
            raise ValueError("Argumento inválido",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("O valor '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 ntt_inv(self,ff):
        return sum([ff[i]*self.base[i] for i in range(self.n)])

    def random_pol(self,args=None):
        return (self.R).random_element(args)


# Função que executa a função ntt_inv para todos os elementos de uma matriz ou de um arra
y
def my_ntt_inv(f):
    if type(f[0]) is list:
        res = []
        for i in range(len(f)):

            if type(f[i][0]) is list:
                res.append([])
                for j in range(len(f[i])):
                    res[i].append(T.ntt_inv(f[i][j]))

            else:
                res.append(T.ntt_inv(f[i]))
    else:
        res = T.ntt_inv(f)

    return res


# Função que executa o ntt para todos os elementos de uma matriz ou de um array
def my_ntt(f):

    if type(f) is list:
        res = []
        for i in range(len(f)):

            if type(f[i]) is list:
                res.append([])
                for j in range(len(f[i])):
                    res[i].append(T.ntt(f[i][j]))

            else:
                res.append(T.ntt(f[i]))
    else:
        res = T.ntt(f)

    return res


# Função que realiza a multiplicação entre dois objetos ntt
def my_mult(ff1, ff2, N=n, Q=q):
    res = []

    for i in range(N):
        res.append((ff1[i] * ff2[i]) % Q)

    return res


# Função que realiza a soma de dois objetos ntt
def my_add(ff1, ff2, N=n, Q=q):
    res = []

    for i in range(N):
        res.append((ff1[i] + ff2[i]) % Q)

    return res


# Função que realiza a subtração de dois objetos ntt
def my_sub(ff1, ff2, N=n, Q=q):
    res = []
```

```
    for i in range(N):
        res.append((ff1[i] - ff2[i]) % Q)

    return res


# Função compress, seguindo o algoritmo da documentação
def compress(x, d, q):
    coefs = x.list()

    new_coefs = []
    _2power = int(2 ** d)

    for coef in coefs:
        new_coef = round(_2power / q * int(coef)) % _2power
        new_coefs.append(new_coef)

    return Rq(new_coefs)


# Função compress aplicada a todos os elementos de uma matriz ou de um array
def compress_rec(f, d, q):
    if type(f) is list:
        res = []
        for i in range(len(f)):
            if type(f[i]) is list:
                res.append([])
                for j in range(len(f[i])):
                    res[i].append(compress(f[i][j], d, q))

            else:
                res.append(compress(f[i], d, q))
    else:
        res = compress(f, d, q)

    return res


# Função decompress, seguindo o algoritmo da documentação
def decompress(x, d, q):
    coefs = x.list()

    new_coefs = []
    _2power = 2 ** d

    for coef in coefs:
        new_coef = round(q / _2power * int(coef))
        new_coefs.append(new_coef)

    return Rq(new_coefs)


# Função decompress aplicada a todos os elementos de uma matriz ou de um array
def decompress_rec(f, d, q):
    if type(f) is list:
        res = []
        for i in range(len(f)):
            if type(f[i]) is list:
                res.append([])
                for j in range(len(f[i])):
                    res[i].append(decompress(f[i][j], d, q))

            else:
                res.append(decompress(f[i], d, q))
    else:
        res = decompress(f, d, q)

    return res


# Inicialização de um objeto ntt
```

```
T = NTT(n=n, q=q)


# Função que efetua xor entre duas strings binárias
def xoring(key, text):
    # Se o text for maior do que a key, então a key é multiplicada as vezes que forem pre
cisas
    if len(text) > len(key):
        t1 = len(text) / len(key)
        key *= ceil(t1)

    # Retorna o XOR
    return bytes(a ^^ b for a, b in zip(key, text))


# Instanciação de funções, seguindo a documentação
def XOF(p,i,j):
    return shake_128(str(i).encode() + str(j).encode() + str(p).encode()).digest(int(200
0))

def H(s):
    return sha256(str(s).encode()).digest()

def G(a,b=""):
    digest = sha512(str(a).encode() + str(b).encode() ).digest()
    return digest[:32], digest[32:]

def PRF(s,b):
    return shake_256(str(s).encode() + str(b).encode()).digest(int(2000))

def KDF(a,b=""):
    return shake_256(str(a).encode() + str(b).encode()).digest(int(2000))


# Função parse, seguindo a documentação
def parse(b, q, n):
    i = 0
    j = 0
    a = []

    while j < n and i + 2 < len(b):
        d1 = b[i] + 256 * b[i + 1] % 16
        d2 = b[i+1]//16 + 16 * b[i + 2]

        if d1 < q:
            a.append(d1)
            j += 1

        elif d2 < q and j < n:
            a.append(d2)
            j += 1

        i += 3

    return Rq(a)


# Função Centered Binomial Distribution, seguindo a documentação
def CBD(byte_array, base):
    f = []

    bit_array = BitArray(bytes=byte_array).bin[2:]
    for i in range(256):
        a = 0
        b = 0

        for j in range(base):
            a += 2**j if int(bit_array[2*i * base + j]) else 0
            b += 2**j if int(bit_array[2*i * base + base + j]) else 0

        f.append(a-b)
```

```
    return R(f)


# Função que realiza a multiplicação entre uma matriz e um vetor, ambos objetos ntt
def mult_mat_vec(mat, vec, k=2, n=n):
    for i in range(len(mat)):
        for j in range(len(mat[i])):
            mat[i][j] = my_mult(mat[i][j], vec[j])

    tmp = [[0] * n] * k
    for i in range(len(mat)):
        for j in range(len(mat[i])):
            tmp[i] = my_add(tmp[i], mat[i][j])

    return tmp


# Função que realiza a multiplicação entre dois vetores, ambos objetos ntt
def mult_vec(vec1, vec2, n=n):
    for i in range(len(vec1)):
        vec1[i] = my_mult(vec1[i], vec2[i])

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

    return tmp


# Função que realiza a soma entre dois vetores, ambos os objetos ntt
def sum_vec(vec1, vec2):
    for i in range(len(vec1)):
        vec1[i] = my_add(vec1[i], vec2[i])

    return vec1

# Função que realiza a subtração entre dois vetores, ambos os objetos ntt
def sub_vec(vec1, vec2):
    for i in range(len(vec1)):
        vec1[i] = my_sub(vec1[i], vec2[i])

    return vec1
```

KEM IND-CPA seguro

Nesta parte implementamos o esquema KEM IND-CPA seguro. A classe possui vários métodos, incluindo a geração de chaves (keygen), cifragem (encrypt), decifragem (decrypt), encapsulamento (encaps), desencapsulamento (decaps), cifragem com KEM (encrypt_kem) e decifragem com KEM (decrypt_kem). Esta implementação segue a documentação fornecida pelo site oficial e utiliza várias funções auxiliares, como compressão e descompressão de polinômios e transformações NTT.

In [37]:

```
# Classe que implementa o KEM IND-CPA seguro
class Kyber:
    def __init__(self, n, k, q, n1, n2, du, dv):
        self.n = n
        self.k = k
        self.q = q
        self.n1 = n1
        self.n2 = n2
        self.du = du
        self.dv = dv


    # Função que gera a chave, seguindo a documentação
    def keygen(self):
        d = _Rq.random_element()
```

```python
        p, o = G(d)

        N = 0

        # Inicializa a matriz
        A = [0, 0]
        # Gera a matriz A
        for i in range(self.k):
            A[i] = []
            for j in range(self.k):
                A[i].append(T.ntt(parse(XOF(p, j, i), self.q, self.n)))

        # Gera o array "s" e  o "e"
        s = [0] * self.k
        for i in range(self.k):
            s[i] = T.ntt(CBD(PRF(o, N), self.n1))
            N += 1

        e = [0] * self.k
        for i in range(self.k):
            e[i] = T.ntt(CBD(PRF(o, N), self.n1))
            N += 1

        mult = mult_mat_vec(A, s)
        t = sum_vec(mult, e)

        self.pk = t, p
        self.sk = s

        return self.sk, self.pk


    # Função para cifrar, seguindo a documentação
    def encrypt(self, pk, m, coins):
        N = 0
        t, p = pk

        # Inicializa a matriz
        A = [0, 0]
        # Gera a matriz A
        for i in range(self.k):
            A[i] = []
            for j in range(self.k):
                A[i].append(T.ntt(parse(XOF(p, i, j), self.q, self.n)))

         # Gera o array "r" e  o "e1"
        r = [0] * self.k
        for i in range(self.k):
            r[i] = T.ntt(CBD(PRF(coins, N), self.n1))
            N += 1

        e1 = [0] * self.k
        for i in range(self.k):
            e1[i] = T.ntt(CBD(PRF(coins, N), self.n2))
            N += 1

        e2 = T.ntt(CBD(PRF(coins, N), self.n2))

        mult = mult_mat_vec(A, r)
        u = sum_vec(mult, e1)

        t = [] + t
        mult = mult_vec(t, r)
        v = my_add(mult, e2)
        v = my_add(v, T.ntt(m))

        u = my_ntt_inv(u)
        v = my_ntt_inv(v)

        c1 = compress_rec(u, self.du, self.q)
        c2 = compress_rec(v, self.dv, self.q)
```

```
        return (c1, c2)


    # Função para decifrar, seguindo a documentação
    def decrypt(self, c):
        u, v = c
        u = decompress_rec(u, self.du, q)
        v = decompress_rec(v, self.dv, q)

        u = my_ntt(u)
        v = my_ntt(v)

        s = [] + self.sk

        mult = mult_vec(s, u)
        m = my_sub(v, mult)

        return compress(T.ntt_inv(m), 1, q)


    # Função para o encapsulamento
    def encaps(self, pk):
        # Gera o polinómio para o encapsulamento
        m1 = Rq([choice([0, 1]) for i in range(n)])
        coins = os.urandom(256)

        # Obtem o criptograma
        e = self.encrypt(pk, decompress(m1, 1, q), coins)
        # Obtem a chave partilhada
        k = H(m1)

        return e, k


    # Função para o desencapsulamento
    def decaps(self, c):

        # Obtem polinómio gerado no encapsulamento
        m = self.decrypt(c)

        # Obtem a chave partilhada
        k = H(m)

        return k


    # Função para cifrar com o KEM
    def encrypt_kem(self, pk, m):
        # Obtem o criptograma da chave partilhada e a chave partilhada
        e, k = self.encaps(pk)

        # Obtem o criptograma
        c = xoring(k, m.encode('utf-8'))

        return e, c

    # Função para decifrar com o KEM
    def decrypt_kem(self, e, c):
        # Obtem chave partilhada
        k = self.decaps(e)

        # Obtem a mensagem
        m = xoring(k, c).decode('utf-8')

        return m
```

Exemplo de execução do KEM

In [44]:

```
# Cria uma instância da classe Kyber
```

```python
# Cria uma instância da classe Kyber
kyber = Kyber(n, 2, q, 3, 2, 10, 4)

# Gera um par de chaves
sk, pk = kyber.keygen()

# Encapsulamento: gera o criptograma e a chave partilhada
e1, k_encaps = kyber.encaps(pk)

# Cifra a mensagem
e, c = kyber.encrypt_kem(pk, "Unidade Curricular de Estruturas Criptográficas")

# Desencapsulamento: obtem a chave partilhada do criptograma
k_decaps = kyber.decaps(e1)

#Verifica se as chaves obtidas são iguais
if k_encaps == k_decaps:
    print("As chaves partilhadas são iguais.")
else:
    print("erro")

# Decifra a mensagem
m = kyber.decrypt_kem(e, c)

if m == "Unidade Curricular de Estruturas Criptográficas":
    print("Cifragem e decifragem bem sucedida!!")
else:
    print("erro")

print("Mensagem original:", "Unidade Curricular de Estruturas Criptográficas")
print("Mensagem decifrada:", m)
print()
print(e1)
```

```
As chaves partilhadas são iguais.
Cifragem e decifragem bem sucedida!!
Mensagem original: Unidade Curricular de Estruturas Criptográficas
Mensagem decifrada: Unidade Curricular de Estruturas Criptográficas

([168*w^255 + 945*w^254 + 137*w^253 + 556*w^252 + 324*w^251 + 630*w^250 + 780*w^249 + 766
*w^248 + 264*w^247 + 912*w^246 + 44*w^245 + 594*w^244 + 319*w^243 + 805*w^242 + 186*w^241
+ 134*w^240 + 298*w^239 + 484*w^238 + 430*w^237 + 1016*w^236 + 703*w^235 + 652*w^234 + 38
6*w^233 + 408*w^232 + 611*w^231 + 749*w^230 + 293*w^229 + 86*w^228 + 666*w^227 + 976*w^22
6 + 745*w^225 + 424*w^224 + 824*w^223 + 917*w^222 + 210*w^221 + 334*w^220 + 837*w^219 + 6
18*w^218 + 123*w^217 + 236*w^216 + 43*w^215 + 1020*w^214 + 894*w^213 + 645*w^212 + 552*w^
211 + 835*w^210 + 536*w^209 + 71*w^208 + 801*w^207 + 966*w^206 + 247*w^205 + 870*w^204 +
980*w^203 + 202*w^202 + 902*w^201 + 110*w^200 + 384*w^199 + 576*w^198 + 931*w^197 + 657*w
^196 + 885*w^195 + 978*w^194 + 347*w^193 + 563*w^192 + 803*w^191 + 598*w^190 + 170*w^189
+ 491*w^188 + 408*w^187 + 978*w^186 + 67*w^185 + 790*w^184 + 221*w^183 + 530*w^182 + 612*
w^181 + 477*w^180 + 853*w^179 + 283*w^178 + 310*w^177 + 340*w^176 + 35*w^175 + 37*w^174 +
446*w^173 + 838*w^172 + 473*w^171 + 564*w^170 + 401*w^169 + 457*w^168 + 252*w^167 + 777*w
^166 + 377*w^165 + 78*w^164 + 696*w^163 + 199*w^162 + 187*w^161 + 65*w^160 + 288*w^159 +
519*w^158 + 994*w^157 + 376*w^156 + 702*w^155 + 252*w^154 + 960*w^153 + 97*w^152 + 358*w^
151 + 147*w^150 + 61*w^149 + 52*w^148 + 894*w^147 + 574*w^146 + 687*w^145 + 843*w^144 + 1
60*w^143 + 958*w^142 + 1023*w^141 + 994*w^140 + 590*w^139 + 165*w^138 + 532*w^137 + 790*w
^136 + 825*w^135 + 975*w^134 + 205*w^133 + 651*w^132 + 39*w^131 + 876*w^130 + 252*w^129 +
528*w^128 + 274*w^127 + 387*w^126 + 245*w^125 + 337*w^124 + 1008*w^123 + 318*w^122 + 834*
w^121 + 641*w^120 + 787*w^119 + 343*w^118 + 285*w^117 + 90*w^116 + 75*w^115 + 21*w^114 +
797*w^113 + 1021*w^112 + 880*w^111 + 680*w^110 + 3*w^109 + 201*w^108 + 850*w^107 + 684*w^
106 + 27*w^105 + 162*w^104 + 245*w^103 + 536*w^102 + 247*w^101 + 712*w^100 + 172*w^99 + 5
71*w^98 + 468*w^97 + 642*w^96 + 962*w^95 + 164*w^94 + 333*w^93 + 412*w^92 + 257*w^91 + 76
1*w^90 + 697*w^89 + 891*w^88 + 302*w^87 + 673*w^86 + 329*w^85 + 948*w^84 + 354*w^83 + 989
*w^82 + 135*w^81 + 401*w^80 + 563*w^79 + 622*w^78 + 842*w^77 + 79*w^76 + 674*w^75 + 520*w
^74 + 470*w^73 + 264*w^72 + 156*w^71 + 600*w^70 + 475*w^69 + 277*w^68 + 136*w^67 + 864*w^
66 + 588*w^65 + 292*w^64 + 728*w^63 + 552*w^62 + 121*w^61 + 436*w^60 + 623*w^59 + 822*w^5
8 + 574*w^57 + 92*w^56 + 75*w^55 + 598*w^54 + 843*w^53 + 504*w^52 + 905*w^51 + 799*w^50 +
360*w^49 + 857*w^48 + 588*w^47 + 440*w^46 + 734*w^45 + 605*w^44 + 900*w^43 + 296*w^42 + 7
62*w^41 + 355*w^40 + 815*w^39 + 293*w^38 + 167*w^37 + 203*w^36 + 524*w^35 + 870*w^34 + 18
2*w^33 + 645*w^32 + 49*w^31 + 41*w^30 + 373*w^29 + 185*w^28 + 600*w^27 + 707*w^26 + 410*w
^25 + 605*w^24 + 653*w^23 + 104*w^22 + 209*w^21 + 594*w^20 + 128*w^19 + 583*w^18 + 344*w^
17 + 852*w^16 + 259*w^15 + 245*w^14 + 277*w^13 + 611*w^12 + 213*w^11 + 69*w^10 + 225*w^9
+ 837*w^8 + 386*w^7 + 88*w^6 + 426*w^5 + 192*w^4 + 393*w^3 + 862*w^2 + 766*w + 119, 379*w
^255 + 323*w^254 + 307*w^253 + 207*w^252 + 289*w^251 + 828*w^250 + 611*w^249 + 605*w^248
```

```
+ 815*w^247 + 192*w^246 + 38*w^245 + 939*w^244 + 447*w^243 + 320*w^242 + 559*w^241 + 264*
w^240 + 741*w^239 + 294*w^238 + 855*w^237 + 818*w^236 + 807*w^235 + 342*w^234 + 120*w^233
+ 68*w^232 + 922*w^231 + 414*w^230 + 13*w^229 + 369*w^228 + 488*w^227 + 38*w^226 + 896*w^
225 + 579*w^224 + 59*w^223 + 929*w^222 + 27*w^221 + 864*w^220 + 616*w^219 + 688*w^218 + 5
48*w^217 + 363*w^216 + 17*w^215 + 481*w^214 + 624*w^213 + 560*w^212 + 426*w^211 + 641*w^2
10 + 702*w^209 + 799*w^208 + 894*w^207 + 6*w^206 + 361*w^205 + 551*w^204 + 387*w^203 + 13
7*w^202 + 445*w^201 + 514*w^200 + 367*w^199 + 1002*w^198 + 371*w^197 + 910*w^196 + 363*w^
195 + 48*w^194 + 820*w^193 + 278*w^192 + 204*w^191 + 539*w^190 + 64*w^189 + 756*w^188 + 9
08*w^187 + 500*w^186 + 52*w^185 + 681*w^184 + 791*w^182 + 74*w^181 + 99*w^180 + 194*w^179
+ 981*w^178 + 925*w^177 + 459*w^176 + 291*w^175 + 654*w^174 + 668*w^173 + 994*w^172 + 627
*w^171 + 766*w^170 + 657*w^169 + 406*w^168 + 491*w^167 + 186*w^166 + 636*w^165 + 612*w^16
4 + 144*w^163 + 431*w^162 + 881*w^161 + 608*w^160 + 508*w^159 + 567*w^158 + 407*w^157 + 6
54*w^156 + 848*w^155 + 216*w^154 + 909*w^153 + 445*w^152 + 448*w^151 + 282*w^150 + 917*w^
149 + 430*w^148 + 6*w^147 + 868*w^146 + 188*w^145 + 581*w^144 + 513*w^143 + 484*w^142 + 3
42*w^141 + 744*w^140 + 574*w^139 + 140*w^138 + 640*w^137 + 100*w^136 + 654*w^135 + 1016*w
^134 + 110*w^133 + 784*w^132 + 215*w^131 + 892*w^130 + 127*w^129 + 936*w^128 + 390*w^127
+ 441*w^126 + 458*w^125 + 540*w^124 + 606*w^123 + 962*w^122 + 960*w^121 + 909*w^120 + 229
*w^119 + 779*w^118 + 610*w^117 + 523*w^116 + 340*w^115 + 263*w^114 + 716*w^113 + 611*w^11
2 + 46*w^111 + 545*w^110 + 715*w^109 + 629*w^108 + 865*w^107 + 895*w^106 + 741*w^105 + 43
7*w^104 + 308*w^103 + 201*w^102 + 790*w^101 + 515*w^100 + 740*w^99 + 952*w^98 + 50*w^97 +
459*w^96 + 800*w^95 + 840*w^94 + 25*w^93 + 908*w^92 + 132*w^91 + 429*w^90 + 948*w^89 + 2*
w^88 + 632*w^87 + 579*w^86 + 577*w^85 + 883*w^84 + 395*w^83 + 609*w^82 + 211*w^81 + 907*w
^80 + 106*w^79 + 49*w^78 + 675*w^77 + 268*w^76 + 446*w^75 + 554*w^74 + 777*w^73 + 225*w^7
2 + 995*w^71 + 890*w^70 + 767*w^69 + 731*w^68 + 545*w^67 + 926*w^66 + 99*w^65 + 67*w^64 +
129*w^63 + 206*w^62 + 874*w^61 + 293*w^60 + 106*w^59 + 135*w^58 + 712*w^57 + 144*w^56 + 2
64*w^55 + 119*w^54 + 53*w^53 + 681*w^52 + 538*w^51 + 907*w^50 + 424*w^49 + 878*w^48 + 33*
w^47 + 155*w^46 + 432*w^45 + 830*w^44 + 150*w^43 + 711*w^42 + 475*w^41 + 554*w^40 + 365*w
^39 + 786*w^38 + 247*w^37 + 390*w^36 + 624*w^35 + 997*w^34 + 28*w^33 + 981*w^32 + 483*w^3
1 + 341*w^30 + 578*w^29 + 593*w^28 + 499*w^27 + 105*w^26 + 679*w^25 + 608*w^24 + 435*w^23
+ 1006*w^22 + 484*w^21 + 828*w^20 + 836*w^19 + 312*w^18 + 553*w^17 + 63*w^16 + 767*w^15 +
818*w^14 + 88*w^13 + 842*w^12 + 644*w^11 + 42*w^10 + 133*w^9 + 515*w^8 + 432*w^7 + 467*w^
6 + 120*w^5 + 589*w^4 + 237*w^3 + 742*w^2 + 967*w + 829], 14*w^255 + 7*w^254 + 9*w^253 +
w^252 + 10*w^251 + 6*w^250 + 14*w^249 + 8*w^248 + 6*w^246 + 12*w^245 + 13*w^244 + 6*w^243
+ 14*w^242 + 15*w^241 + 9*w^240 + 6*w^239 + 15*w^237 + 7*w^236 + 10*w^235 + 8*w^234 + 8*w
^233 + 5*w^232 + 13*w^231 + 7*w^230 + 6*w^229 + 15*w^228 + 2*w^227 + 3*w^226 + 7*w^225 +
w^223 + 9*w^222 + w^221 + w^220 + w^219 + 7*w^218 + 15*w^217 + 14*w^216 + w^215 + w^214 +
6*w^213 + 12*w^212 + 13*w^211 + 2*w^210 + 10*w^209 + 14*w^208 + w^206 + 2*w^205 + 4*w^204
+ w^203 + 6*w^202 + 2*w^201 + 5*w^200 + 8*w^199 + 4*w^198 + 5*w^197 + 8*w^196 + 11*w^195
+ 10*w^194 + 15*w^193 + 12*w^192 + 2*w^191 + 10*w^190 + 10*w^189 + 10*w^188 + 9*w^187 + 1
5*w^186 + 9*w^185 + 8*w^184 + 13*w^183 + w^182 + 13*w^181 + 14*w^180 + 3*w^179 + 13*w^178
+ 13*w^177 + w^176 + 5*w^175 + 5*w^174 + 14*w^173 + 6*w^172 + 2*w^171 + 5*w^170 + 2*w^169
+ 9*w^168 + 3*w^167 + 14*w^166 + 14*w^165 + 5*w^164 + 5*w^163 + 9*w^162 + 5*w^161 + 4*w^1
60 + 15*w^159 + 8*w^158 + 6*w^157 + 11*w^156 + 2*w^155 + 14*w^154 + 9*w^153 + 6*w^152 + 9
*w^151 + 8*w^150 + w^149 + 8*w^148 + 4*w^147 + 8*w^146 + 13*w^145 + 13*w^144 + 15*w^143 +
9*w^142 + 12*w^141 + 7*w^140 + 3*w^139 + 14*w^138 + 3*w^137 + 5*w^136 + w^135 + 2*w^134 +
9*w^133 + 10*w^132 + 6*w^131 + 6*w^129 + 13*w^128 + 15*w^127 + 11*w^126 + 9*w^125 + 13*w^
124 + 7*w^123 + 4*w^122 + 2*w^121 + 6*w^120 + 13*w^119 + 14*w^118 + 8*w^117 + 3*w^116 + 1
3*w^115 + 4*w^114 + 2*w^113 + 6*w^112 + 9*w^111 + 9*w^110 + 13*w^109 + 12*w^108 + 15*w^10
7 + 13*w^106 + w^105 + 5*w^104 + 13*w^103 + 5*w^102 + w^101 + 3*w^100 + 2*w^99 + 14*w^98
+ 8*w^97 + 2*w^96 + 3*w^95 + 7*w^94 + 9*w^93 + 5*w^92 + 3*w^91 + 2*w^90 + 5*w^89 + 7*w^8
8 + 7*w^87 + 10*w^86 + 14*w^85 + 5*w^84 + 7*w^83 + 2*w^82 + 14*w^81 + 7*w^79 + 10*w^78 +
15*w^77 + 3*w^75 + 9*w^74 + 4*w^73 + 15*w^72 + 2*w^71 + 9*w^70 + 8*w^69 + w^66 + 11*w^65
+ 10*w^64 + 3*w^63 + 9*w^62 + 10*w^61 + 2*w^60 + 9*w^59 + 4*w^58 + 7*w^56 + 15*w^55 + 4*
w^53 + 8*w^52 + 3*w^51 + 12*w^50 + 7*w^49 + 5*w^48 + 9*w^47 + 7*w^46 + 12*w^45 + 12*w^43
+ 6*w^42 + 6*w^41 + 11*w^40 + 6*w^39 + 10*w^38 + 4*w^37 + 2*w^36 + 13*w^35 + 5*w^34 + 7*
w^33 + 6*w^32 + 5*w^31 + 2*w^29 + 5*w^28 + 15*w^27 + 2*w^26 + 11*w^25 + 13*w^24 + 14*w^23
+ 2*w^22 + 5*w^21 + 2*w^20 + 7*w^19 + 2*w^17 + 12*w^15 + 7*w^14 + 8*w^13 + 9*w^12 + 4*w^
11 + 8*w^9 + 8*w^8 + 4*w^7 + 11*w^6 + 8*w^5 + 7*w^4 + 12*w^3 + 8*w^2 + 7)
```

PKE IND-CCA seguro

Esta parte implementa uma classe "Kyber_CCA", que realiza o esquema PKE IND-CCA seguro. Esta classe utiliza a classe "Kyber" anteriormente implementada e adiciona métodos para cifragem e decifragem usando a transformação Fujisaki-Okamoto.

In [31]:

```
# Classe que implementa o PKE-IND-CCA seguro
```

```python
class Kyber_CCA:
    def __init__(self, n, k, q, n1, n2, du, dv):
        self.n = n
        self.k = k
        self.q = q
        self.n1 = n1
        self.n2 = n2
        self.du = du
        self.dv = dv

        self.kyber = Kyber(n, k, q, n1, n2, du, dv)

    # Função para gerar a chave, usando a função keygen da classe anterior
    def keygen(self):
        self.sk, self.pk = self.kyber.keygen()

        return self.sk, self.pk


    # Função para cifrar, usando a função encrypt da classe anterior
    def encrypt(self, pk, r, y):
        # Obtem a hash r||y
        ry = H(bytes(r) + y)

        # Cifra r e a hash r||y
        c = self.kyber.encrypt(pk, decompress(r, 1, self.q), ry)

        return c


    # Função para decifrar, usando a função decrypt da classe anterior
    def decrypt(self, c):
        r = self.kyber.decrypt(c)

        return r

    # Função para cifrar com a transformação Fujisaki-Okamoto
    def encrypt_fo(self, m, pk):
        r = Rq([choice([0, 1]) for i in range(n)])

        g = H(r)

        y = xoring(g, bytes(m, encoding='utf-8'))

        c = self.encrypt(pk, r, y)

        return y, c

    # Função para decifrar com a transformação Fujisaki-Okamoto
    def decrypt_fo(self, y, c):
        r = self.decrypt(c)

        _c = self.encrypt(pk, r, y)

        if c != _c:
            raise Exception("Mensagem não pode ser decifrada")

        g = H(r)

        m = xoring(g, y)

        return m.decode('utf-8')
```

Exemplo de execução do PKE

In [47]:

```python
# Cria uma instância da classe Kyber_CCA
kyber = Kyber_CCA(n, 2, q, 3, 2, 10, 4)
```

```python
# Gera um par de chaves
sk, pk = kyber.keygen()

# Cifra a mensagem
y, c = kyber.encrypt_fo("Trabalho prático número 3", pk)

# Decifra a mensagem
m = kyber.decrypt_fo(y, c)

#Verifica se as mensagens são iguais
if m == "Trabalho prático número 3":
    print("Cifragem e decifragem bem sucedida!!")
else:
    print("erro")

print("Mensagem original:", "Trabalho prático número 3")
print("Mensagem decifrada:", m)
```

```
Cifragem e decifragem bem sucedida!!
Mensagem original: Trabalho prático número 3
Mensagem decifrada: Trabalho prático número 3
```

In [ ]: