TP2-KYBER

May 2, 2022

1 Trabalho prático 2 - Estruturas Criptográficas

Autores: Ariana Lousada (PG47034), Cláudio Moreira (PG47844)

Grupo 12

1.1 Kyber

A última técnica a implementar tem como objetivo a implementação de um KEM IND-CPA e um PKE IND-CCA, do protótipo **Crystalis Kyber**. De seguida, são apresentadas em duas seções (PKE-IND-CPA e PKE-IND-CCA) os resultados da implementação de cada ténica. Esta resolução foi construída com base no documento *kyber.pdf* mais recente do site disponibilizado pela equipa docente.

1.1.1 KEM-IND-CPA

Esta versão permite obter uma segurança do tipo IND-CPA (Chosen Plaintext Attacks). Em primeiro lugar foram desenvolvidas funções auxiliares de implementação de aritmética, encode, decode entre outras:

- ***parse***: recebe como *input* um conjunto de *bytes* e retorna o polinómio correspondente a esse conjunto;
- ***XOF***: corresponde a uma função do tipo *extendable output function*, utilizando *SHAKE-128*;
- ***PRF***: corresponde a uma função do tipo *pseudorandom function*, utilizando o *SHAKE-256*;
- ***G***: função de *hash* construída com base no *SHA3-512*;
- ***encode***: recebe um polinómio como argumento e retorna um *byte array* correspondente ao polinómio;
- ***decode***: função inversa da ***encode*** - recebe um *byte array* e retorna o polinómio correspondente;
- ***compress***: comprime um polinómio e retorna os *bytes* correspondentes;
- ***decompress***: função inversa da ***compress***. Descomprime *bytes* e retorna o polinómio correspondente;
- ***transposta***: calcula a transposta de uma matriz.

As funções principais centram-se na geração de chaves, cifragem da mensagem passada como parâmetro e posterior decifragem do texto cifrado, obtendo deste modo, a mensagem original:

- ***gerar_chaves***: com recurso às funções anteriormente apresentadas, ocorre a geração das chaves pública e secreta. A primeira é essencial para a cifragem da mensagem e a segunda para a decifragem do criptograma;
- ***cifragem***: tem como objetivo principal a cifragem de uma mensagem. Desta forma, recebe como parâmetros a chave pública, a mensagem e *coins* (*bytes* aleatórios) e dá como *output* o texto cifrado;
- ***decifragem***: tem como objetivo decifrar um criptograma, obtendo como resultado o texto limpo correspondente. Recebe como argumentos a chave secreta e o criptograma.

[17]:
```
import os
import random as rn
from sympy import ntt
from cryptography.hazmat.primitives import hashes
import numpy
from sympy import intt
import gzip
import struct
```

[18]:
```
# constantes Kyber
n = 256
q = 343576577
k = 2
n1 = 3
n2 = 2
du, dv = 10, 4

# criação dos anéis
_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))


# tamanho necessário para o decompress
def tamanho(stringB, numberS):
    count = 10
    auxCount = 1
    i = 0
    while i < len(stringB):
        if numberS == auxCount:
            i = i + 10
            while (i < len(stringB)) and (stringB[i] != 31 or stringB[i + 1] !=␣
 ↪139 or stringB[i + 2] != 8 or stringB[i + 3] != 0  ):
                count = count + 1
                i = i + 1
```

```
            auxCount = auxCount + 1

        i = i + 1
        if (i + 10) < len(stringB) and (stringB[i] == 31 and stringB[i + 1] == 
↪139 and stringB[i + 2] == 8 and stringB[i + 3] == 0 ):
            auxCount = auxCount + 1
        if auxCount > numberS:
            break
    return count

# input: conjunto de bytes; output: polinómio do conjunto de bytes inserido;
def parse(str_bytes):
    result = []
    for i in str_bytes:
        result.append(i)
    return Rq(result)

# XOF, com o SHAKE-128
def XOF(p,i,j):
    digest = hashes.Hash(hashes.SHAKE128(int(32)))
    digest.update(p)
    digest.update(bytes(i))
    digest.update(bytes(j))
    r = digest.finalize()
    return r

# pseudorandom function com SHAKE-256
def PRF(s,b):
    digest = hashes.Hash(hashes.SHAKE256(int(32)))
    digest.update(s)
    digest.update(bytes(b))
    r = digest.finalize()
    return r

# função de hash com SHA3-512;
def G(d):
    digest = hashes.Hash(hashes.SHA512())
    digest.update(bytes(d))
    r = digest.finalize()
    return r

# input: polinómio; output: byte array;
def encode(poly):
    byt=b''
    aux=1
    countX=0
    for j in poly:
```

```python
        if(j>255):
            aux=2
        if (j > 65025):
            aux = 3
        if (j > 16581375):
            aux = 4
        if (j > 4228250625):
            aux = 5
        byt = byt+ int((_Z(j))).to_bytes( aux, 'big')
        byt = byt +"/-n-/".encode()
        countX =countX +1
    return byt

# input: byte array; output: polinómio correspondente ao byte array do input;
def decode(byt):
    listaCoef = []
    byteAux = b''
    listAux = []
    desc=0
    while desc <byt.__len__():
        if byt[desc] == 47 and byt[desc+1]==45 and byt[desc+2]==110 and␣
 ↪byt[desc+3]==45 and byt[desc+4] == 47 :
            desc = desc+4
            listaCoef.append(int.from_bytes(byteAux, 'big'))
            byteAux = b''
        else:
            byteAux = byteAux + bytearray([int(_Z(byt[desc]))])
        desc = desc+1
    return listaCoef

# comprime um polinómio em bytes
def compress(polinomio):
    polinomioB= encode(polinomio)
    compress = gzip.compress(polinomioB)
    return compress

# descomprime os bytes e transfora-os num polinómio
def decompress(compress):
    unpack = gzip.decompress(compress)
    return Rq(decode(unpack))

# calcula a transposta de uma matriz
def transposta(matrix):
    zipped_rows = zip(*matrix)
    transpose_matrix = [list(row) for row in zipped_rows]
    return transpose_matrix
```

```python
# geração do par de chaves (publickey, secretkey)
def gerar_chaves():
    d = bytearray(os.urandom(32))
    p = G(d)[:32]
    teta = G(d)[-32:]
    N = 0
    A = [[ 0 for x in range(k-1)] for y in range(k-1)]
    for i in range(0,k-1):
        for j in range(0,k-1):
            A[i][j] =parse(XOF(p,i,j))
    s = []
    for i in range(0,k-1):
        s.append(parse(PRF(teta,N)))
        N = N + 1
    e = []
    for i in range(0,k-1):
        e.append(parse(PRF(teta,N)))
        N = N + 1
    s1 = Rq(T.ntt(s[0]))
    e1 = Rq(T.ntt(e[0]))
    t = A[0][0].lift() * s1.lift() + e1.lift()
    pk = encode(t) + p
    sk = encode(s1)
    return pk, sk

# cifra uma mensagem m
def cifragem(pk, m, coins):
    N = 0
    t2 = pk[:len(pk)-32]
    t= decode(t2)
    p = pk[-32:]
    A = [[ 0 for x in range(k-1)] for y in range(k-1)]
    for i in range(0,k-1):
        for j in range(0,k-1):
            A[i][j] =parse(XOF(p,i,j))
    AT = transposta(A)
    r = []
    for i in range(0,k-1):
        r.append(parse((PRF(bytearray(r),bytearray([N])))))
        N = N + 1
    e1 = []
    for i in range(0,k-1):
        e1.append(parse(PRF(bytearray(r[i]),bytearray([N]))))
        N = N + 1
    e2 = parse(PRF(bytearray(r[0].list()),N))
    r1= Rq(T.ntt(r[0]))
    u= Rq(T.ntt_inv(A[0][0].lift()*r1.lift()))+e1[0]
```

```
    v = Rq(T.ntt(Rq(t).lift() * r1.lift())) + e2 + decompress(m)
    c1 = compress(u)
    c2 = compress(v)
    c = c1 + c2
    return c

# decifragem do criptograma
def decifragem(sk, ciphertext):
    u = decompress(ciphertext[:tamanho(ciphertext,1)])
    v = decompress(ciphertext[-tamanho(ciphertext,2):])
    s1 = Rq(decode(sk))
    m = compress(v.lift() - (T.ntt_inv(s1.lift()*Rq(T.ntt(u)).lift())))
    return m
```

De seguida são apresentados os resultados obtidos com recurso às funções anteriores. Neste caso, estamos a considerar uma mensagem fixa. Note-se que houve uma dificuldade acrescida nesta implementação pelo que, deparamos-nos com alguns erros não identificados que impossibilitam o correto funcionamento do programa.

[19]:
```
pk, sk = gerar_chaves()

m = Rq([1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1,
           0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1,
           1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1])

print("Mensagem = ", compress(m), "\n")
coins = bytearray(os.urandom(32))
ciphertext = cifragem(pk, compress(m),coins)
print("Texto cifrado = ", ciphertext, "\n")
texto = decifragem(sk, ciphertext)
print("Texto limpo = ", texto)
```


```
Mensagem =  b'\x1f\x8b\x08\x00\xf3\xc8mb\x02\xffc\xd4\xd7\xcd\xd3\xd5g\x00\x93\x
8cD\x90\x0cD\x88 \x8bc\x92\xa4\x9a\xc6@\xa2\x0bq\xe9b\xa0\xd8w\xe4\x99\xc9H\x96\
x1f\x19(\x88\x17\x06\x12}DL<R+m\x10\xef\x1eF\x12\xd3\xd5(9J\x8e\x92\xa4\x92\x00\
xe2{\x855\x00\x06\x00\x00'

Texto cifrado =  b'\x1f\x8b\x08\x00\xf3\xc8mb\x02\xff5UyT\xd4U\x14\x1ef\x06fa\x1
6`\x06\xf5\x90,\xa1\x89\xec)\xc3"t\x12\xd9\x86MBb
\xc0\x8e\xa4Dp\x0c\xa23%\x18\x11\x8bMA R\x08X"d*\xc6\x91E\x8cP@\x16#\x01\x01\tA\
x8aS\x90$\xa6@\x94\x88,\x02\x99\x9d\xf3\xbb\x1f\xff}\xe7\xde\xf7\xbew\xefw\x97\x
c7\x12\x06\xa4}co\x9bhk\xcf\xd2\xf3\xb3?MH\xf7Y\xf6\xaf\xb0i\x86\xbe"$\xec\xce*%
\xa4\xef(\x08d\x10{\xc7\xbdz2\xe9\x1c\x9eo`\x10\xa7p\xf5\x00\xf9\xae\x9dO#\xcbx\
```

x8a\x01X\x95\xcf\xf6\x93\xef\xe3\xccq\x98\xd2\x97\x94\x0cr\x1b\x90\x93E\xd6n\xf0
>\x83\x8aT\xff\x90\xc5`b\xac\x1d\xaf\x1c\xffw\x9e8]\x85Md\x12\x07g\xa4#\xd2\xc5\
xa9`B\x92\xe8C\xcb\x0c\xdaz\xe2\x0bz\xefl\xc6\x0cB\xcf\x99,\x00k\xf9Nor*6=O\xa4\
x99\x92\xef\xf1\xcej\x8f\x8c|%jW\xb07\xeb\x871\x88k\xe4\xd3F&>\xeb\x8a\t\x83\xb4
\xcd\xf3\x11\xbch\xd0\xf2?\xc45\xd7\x90\x8d\x9b\xaa\xb87\t\xc9C\x17\xa3\x18\xa4\
x95\xe3TEdO\xfc\x7f$\x9f4\xe0@\x16\x02kZ\xa7\x86\x14\xe7\x8e\xf70\xe8\xfc\xc29\x
04\xd6\xd8\\I\x0ce\xeaad\xe4e\xf8\x03\ndg\xda\x0c\xb2\xc2\x9btA\xbb\xc9u\x84L<\x
db~d":\x9a\xf0\x84\x9cy%\'P3\xa9\x11\x85Sw\x19\n\x88\x94[\xcf\x80+\xa3m\x01\x8f\
xab\x8e}\n\xefL\xf4)$\xe7\x19\x11\x83\x84\xfd\xfb>\x04\xaa\x8asB\xc2\xb6\xdc\x8b
\x88\xad\xc1\xec,\x02\xf9\xc5\xd2\x02\xe7:L\xe2P0\xeb[\xbb\xc1\x1c\xe9s\x04\xcc\
xc2\xf6\xfb\x88`Lp\x08,\n\x974H\xa3j|\r\xb6\xd2G\x9f\x80\xaf\xcd\xd8\x0bo\xe4\x8
d/\xa1\xbf*\x8cZ\xc0wzl\'\xce\x15;\xbf\x81\xbbI5\xbd\x88\xe0b\xf2(^{8HQ\xb1\xbf\
xd6\xa3\xb6c\xc7\x86\xec\x01Gu\x9c\n\xe7G:H\x1f\x1dy5F@j\x9d\xe7\x08\xd4.Cq$\x1e
\xfc@<\xb5\xd0\x95\xbd&\x94\xe2[\xb7T\x07\xdc\xa8\r\x8a$$\xb0P"m\xfe\xe8\x83L\x
948\xd4\xf7g\xaa\xcf|?5\x04\xa7\xdbe\x1f\x12<\xd9\xff\x1e8\xde=\x99\x0f\x11o\x18
\xdf\xc3\xfb\xfa\x7fP\x82\x9c\xc6L\xb4\x90\xa8\'"\x9aLE9\x8b\xe0\xc8\xb9r\x07/\x
1ds\xc7\xdcK\xdd\xea\xed\xc0\xf1[\xf8,\xe9`u\xd9\x96\xc0u\xd96\xf8\x06\xb4Q.q\xc
8\xdf\x12\xea\xcc\xba-\xebQ#\xdf\x0eTK\xae\xb7\xa9\n\xaf\xab\xa5>\xd4|w\x921\xb7
R\x1bk\x1aj\x1d\xef&w\xd4\xef\x94\xf8- \xbbh<\xa5+S\x99\x82W9\x8cJ\xeaz\xb9\xff\
x0e[V\x0b\r\x12\xe7H\xb77\xf4\x8b\xb3A\x93\xca\xe4\xbe\xb4R8\xe6\xcb\xa4\x1f\x9b
\xd7I\x179\x9d\xb3\x13\x881\xe1%)..\xdb\x9b\xe3\xa2\x8d\x1d\xc9\xcc\xe9kX\x9bN\x
8f}$\xa40\xf9.%\xa2\xf0\xa3\xd2y\x8e\x86\xe3\x88\x9b{\x0e\xddz\xae\xf4*r\x8c\xc8
\x82\xfc|\x07G\x16\x836\x1aiH/\xad\xa4\xd7q\xaf9\xd7\x13\xc7\x07C\x92\xd0qe\x967
\x18\x14\xd7\xf7\x1dq.\xf3y\x04\xccb\xc2P\xb4\xfd\x85\x02\x94\xe3\x85bL\x00\xff/
\xbdD(\xceQ\xaeP\xa0\xdbG\xa6\xe1\xf4\xb6\xe7A\xdeU\x16\xed+\x8e\xed\x91\xbd\x14
N_k\nId:\xa6\xc0\xdc?P\xb8\x10\xc3\xed\x02H$\xa8\x7f\x84\xae\x93\x05\x94\xa2c\xe
4Q\x0e+\x08\xc3\xe9\xd65t}y.\xce\xf1v\r\x87\xe2F\xc3\xd0+\xf4\xd4\x8bm\xf8n\xc4\
x9a\xebe\xc8<\x9e\xe3G\xce`\x15\x96\xbe\xbe\xdd,mWn\xeaF1\xc8T\xc5\x98~\xc1\xee@
|^\x92\x89\x9c\x1d\xb0MU\xd1\xb6\xcf\xaaA\x03\xf0\xad\xc2\xd0Ob\xbf\xd0\xcf\x18\
xd4\xba|\x13\\\x83\x1b\xb0\xcf%\xf7e.`\x98\xfe \x96A]\xd6O\x11\xd7\xc1,\x8c\xb4N
X\x82\x06\x81\xfdT\xb8\xe6u\xf9\xf3!)\xb4ys\x06\xb2\xac\\\x98\x02Y\xc1m\xfa\xec\
xb8\xb9\x05$\xb6\xd6j?\r\x81\xd6\xf6B,\x05Y\x82Z\x87dOH\xc0/&JwN\x85\xb3V\x8e\x7
f\xd3\xbc\x85>Wn\xb4\x13\xba\\<\xa5m\x89V\xc9\x9c\xc3/\xc4\xdb3M\r\xcc\xe5E\xd5\
xe2<\xaf\x02#\x93\xe6J\xcb\x99S7\x89\x95#\xba\xaa\xf9\x1c\xb9%\xb5\x1e\xc6\xe7mF
\xbf\x05\xf7h\x0f\xcd$\xbbY\x0b\xbd,Z\x19z\x99L\x9d\xa2\xbdP.>\x19i\x0b\xe3g"\xa
0yei5\x83\xd4\xe1%\xd0+R\x0c\r\r6$Z!\xd4\xf5\x81w\x81l\x9e:3h\xc4B\x02U\xb7\x05\
x97S\x10Kr\xb4\xb5\xf0\xcc\x1c\x0b\xce\x81\x0b\x86\xc8H\xb3%\x08Z\\\xea\xa5\xd5\
xc3\x9e\x0cx\x07\x17\x1e\xcf\xe2C\xd6sJB\xb4\xb2\xa0K\xaf"\x8e\xb7\xb3cQ\xa4\xde
\xc1\x8f\x10eL\r\x96\xa1\xae\x89\x07M\x18\xe7\xa0q\x11\x81\xc7\x89P#\xbf\xe2K\xa
8\xd1\xe5}\x81|\xd9\x86\xfeT\xd4u\xbb\xacA\x95\x94\x87\x8d,\x0f\xf3\xa4Z\x8e\x07
v\xc0b\xdaL\xf1k\xcb%\x18\x16A\x8ay\xc6\xda\xc2\xd7P\xd4\xff\x03?ak\x9a\x9d\t\x0
0\x00\x1f\x8b\x08\x00\xf3\xc8mb\x02\xff5VyP\xd4e\x18\xde\x85ewY\xf6\xde\x05\x83\
x19\x8e\x81(\xcc\xb8D\x11+\x8e\x8c\x151\x8e\x00\x1bE&\\Y\n,.Ga\x91!\xd1H\x08\x14
E!\x12\x87#\x96\xc3Q@ABG\x11\xec0\x84d\x85i(0!a\'\x8e\x80\x81P\x04D\xd0f\xf6}\xf
6\xbfg\xbe\xef\xb7\xcf\xf3\xbe\xcf{|\xcb\xe0Y\x9e\xb9\xee\xea\x9c\xe4\xec\xca\x1
0\xed\xfb\xef\x9c\x1e\x99\xbc\x9a\xb8CG\xecB\x86\x05\x1d\x15p\x86\xf5`\xfd\n\x9f
\xaed\x8c\x80\xf3\x84\xf8\xac\xd5iB\xa6\xa13^\x848\xa2\x9b\xb7\x08\t\xa3\xfdv\xe
3\x17aG\xa1$lz;\x8f\x90Y\x9a\xf5s=2\xd2Io\xe2\xf2\xf1\x94\x1b!\xf9\xfeBs\xba\x8c

```
\xba\xa0\xa5#q\x8dM#\x14\xf2\xfa\xe2\x08\t\x86\xf2\x95\x84\xb8\xaa53\xc4\xf4\xf0
\xaf.\xa8\xbe\x9e\xf2\x05\xce\xaa\xcaw\x80eY\x02\x16\xf1\xa8\xcb\x06BR\xc7\x16\x
7f\xa0\xad\x1fg#\xfd\xcc*9\xce\xa6N\xaf\x81\xaf`\xe9gD0\xddU\x81\xb3\r\x83\xdd\x
84$\x9f7x\xc3\x8b=/\x907g\xb8t\x0e|\x17\x8d\x9e\x02)\x1fh\x08\xf1n4\xfe\x03/2\xe
3N\xe9\x11[\xa2|\x06/\xbc\x97\xf3\xf5\x88\xe9\xf7\xe6o\x88\xb8\xc8t\x13\xea\xf5\
x8d\xcc\x9a\x8a\xa3\x1d\xbeJ_\x8d\xfc\x98\x8a\\\xbb\x9b\xaa\r\x9a*2\x96Y\xc5v\xd
5\x03\xe3\x97\x0b\xdb\xc1\xa0-\xae\xd1#VV\xe0g\x08V\xec\x80\xb0\x05\x81\xbe\x942
3\xe2t\x08\xf2\x0c\xef\xc8\x85\x1f\xe6\x91\xe9\xe0\x0fm\xad\xd7\xa3c\xe5\xc9\xd0
\x1e\xb8=\x04\x0f\xde\x19\x9f \xc9\x18.\x1d\xb1B\x13O@(\xfd]8\xcfN\xfc$\x14^d\xf
48\xc0\x0bE\'\xc2\x10\xda\x98\xb8\x90+\xe2!?D\x16r\xe1\x1e\xa4b\xd7\xfe\x06\xc9\
xcb\xd5\x16\x18\xa4\xa9\xf4\x84|\xd9\x11\n\x89\xf5t\xf0"xK\x17\xd0\x83\xdc\x01[T
Eh\xaeS#\xbf\xbb\x1f\xa1\xca\x9c\xcd#\xc1\xe4m\xccc[\x04\xd7\xab@\x9b\x89\x1d\x1
9\xedPH\xbe\xb4\x02\xfd\x84\xa2\x17\xb0\xe6\xfe\xf8s\xd8\xfb\xef\xfb\x90\x95\x07
\xf4\x81\x85{v\xbe\x00g\x1evT5\x9f\x01\x1a\x1a\xe6\x13V\x1f\x12\x1c\x1c\xa5:\x9a
\xdc\t\xbd\x84\xc0U\x9buP?\x90K=Q\x19VH\x1f\xd9\xfe\x81\x8e\x90F\xd4\xbc\x02J\r"
O\x8d\xf7N$\x10\xbb\x97\xca\x08Tv\x8b\xe8t\xcee\x1b\x05\x19\xd4\x16y\x1b1o\xf4D\
xf3\x8b[\xab1$f\xdc\xf6@\xe28re/\x81\xc9<\x06\x84v\xaeR\xf9\xb5q\xf3\xf0)\xcb\xc
3\x19v>\xca9D\xf4\x8b\x96\x93\xe8\xec\xb2\xfdHq\xd9\x19\x03\xca\xefdV"\x9c-\x8d\
xeb\xe0\xab\xd2\x07\x03/*\xcfF\x9b\xf0:J$\xc4V\x9fd(u\xf2qw\xd4\x90;u\x0c$\x9b\x
9c~\x05Iz_+$\x82]\xb1\x17\x04q\xedb\x04w\xa2\xb9\x1f,?\xf8\xa3\x9c\xf2-\xbeg\xc9
\xd2\x9eH{J\xd4Q\x83\x89\xe20>,\x81\x96\xb6\xff{\xb0\xed\x99\x10\x80\xe3\xb5\x93
\xbf\xe0\xac\xf8n\x07-\xad\x99\xedpD\xa8(\xe3\x10\xad\xcf\x15\xd4@2\xd7:dPWY\xe3
\xb3\xd9\x8a\x1c\xc4\xb1\x8d\x8b\xe9\xe5D\x0f|\xadGuV[\x91U}\n\xa6\x91\xeb
\x85\x0f\xb2\x94\xc2 \xaa\xf6r\x0bU\x83\xd9\xca.\xc5W\xea\xaa}p\xa1\x8bg\x07\x8e
\x98)\x14YV\xeb\x86\xc2K2\x9e\xd0d\x9ad8#\x0c\xf1\x82\x8e\x87\x9f\xbas]P\xef\xde
\xef\x90\x15\xa7\x82S\x8b\xca\x1ce~I)\xa7\xf2\r\x19l\xec\xc1\x96\xe4\xb9h\xbf\xa
2\x90\xe2\xab\xd3`\xd1\xca\x98aM\x0b5R\xc8_^\xd5\x19\x16\xca\x1b\x8bPx\xef\xd3sH
b\xaa\xe5*\xceJF\xff\x04qO\xddO\xb8\xcd\x9d\x1bEb)N\xd4\xdaF\xd7\x870z\x92\xf0\x
df\x13\x0ctMi\xd8\x97\xd7Pp\xa9M\x00\xd6*7\xdf\x9d,1.\x8bF\xff\x88\xda\xfck \x15
q\x83\x0b\x94\xea\x85\x85#\x1bnN!\xb6Y\x1bD)\x0c\x99\xff\x16\x97\xbc,!\xf5\xea\x
8e\xfb\xdb\x88\xf6\xe0#<Wb\xeb]\x99@\xa7\x0e5\x935c\xd7f\x11\xe3\xee\xa5r\x90i\x
92i\xfcN>\xa0\x8fL\x8c\x1a\xf0h\xf2\xe3\xd5\nxU\x94\xbf\x13\x99\xab\xc7Y\x949\xa
f\x13\xdb\x9d\x7f\xb0\xef\x00\x0c\x8f*E?\x89\x9e\xb5`\xc4\xc4uQXS\\\x91\x12/\x90
\xf88\x0bo\xa5Y}\xf0\x88\xe1E\xf3@\xb7K\xef5\xa2\x1dd\x91I\xab\xf8E\xb7\xd5\x07p
f&1\x822\xd5.\x95 \xf7i\x95\xc1\xb5\xf3\x06\x8e\x18\'\xcc\xa9L\x1d\xbf\x1e\n"_SD
7\x99\x1db(s\x90\x04\xb1\'0\xa90,\xe5<\xfd\xd4\xd8we\x17\xee<x\xb1t\xa7\xae+\xa6
\x05\xe3\x16No(\xcb\xa2_g\xf8\xdb\xe1\xdb\x81`\xd9c\x87\xc9#\x0b\xef^\\*\xda\xd0
D\xd2\xb2I+\xf2\xb9>\x16\x8f\xb4\xa9\xa5\x14\x8bU&gaL\xa4o\xddZ\x80mK\x1a\x9aj\x
a6\x95\x03\xb5\xa4q\xedCl2\xb33\x91\xd8nB\x81k\x18P\xd6aX%\xf3L\x0f\x00\xdb\x8a}
3\xb4\xa4i\xf8\xcf\xc5\xd3\x8da\xe5\x88}\x1b1k\x82\xe1\x1c\xbc\x03\xf2u\xf6\rz\x
f4?\xd9\x1d\xbb^\xb9\t\x00\x00'
```



```
Texto limpo =  b'\x1f\x8b\x08\x00\xf3\xc8mb\x02\xff5V{P\xd4e\x14e\x7f\x0b\xbb\xe
e\x8be\x17\x88\x08\x080\x84\x14y#\x08\x08\x94\x81\x85\xa2c;\xe8\x08\xab\x18\x0e\
xa1\xc0\x08!\x0fMM\xd8\x14\xe3\xfd\x88\xe1\xd5\x103P\x10J<\x84\x8cYSy\x19P)e\x19
o)^\x8a\x1bDE1" \xcd\xfc\xee\xd9\xbf8s\xbf\xefw\xee\xb9\xe7\xde\xef.\x9cV{C\'\x8
78\x07\'\x1da\xe6\xac\x07!Qnq\x1d\x8b\xf4\xb89\x1a\n\t\x9c\x8b\'Xd\xa1\x8c@\xc4\
xcbz\x95.5\xa6\xe8\xb1 c\xa5\x84\x8e\x0cC[l\tIF\xeb\xad\x08\xc9O\x84\xac\xb3\x88
```

```
\xb9\xc5\xc4Q\xc8\xa0v\xd9\x98B\x1a\xdb!\xa2\x12>\xfd\x1e\x12\x0el\n $\xb3U\x8f\
xd1-\xa9k\x1b\x85\xf8\xe5^\x1c\xb0&\xb45\xb1\x88\xf3VQ\rX\xb9\xc7\xce#yg\xeafh\x
b5\xe4\xb6\x13\xc7\xc6\xe6x\x1c\x06\xa4\xfcJH?\xf6q6\xd0\xe1\xe2\xdd\x84x*\x8f;\
xa8E=\xae@\xd6\x1b\xa1\xb9,\x1a\xd9r\x00\xb4a~T\x00\xb7mG#\x81F\xd7\x0fQ\xc0\xcd
\x8c\xdfQ\xc0\xa9\xab\xa1`\xf8\xb2\xf9)\xbc\xce\xb9P\x8d\x98M\xe9I\x16\xe9\xca\x
04\xff"\xb4\xdb)\x84\xec\xa8\x0c\x8f\x85\x9eK\xe7\xca)\x81\xc2/\x8aB\x1bz\xae\x1
d\x85D\xcd\x82\rU\xe7\xafxF\\\x9a\xe0\xdf \xf1\xd0\xa0?!#\xff\x81\xd7Y\xf4j\xf7\
x10\xca\xfd31\x91E\x85\xe1\xa7\xc0\xb9l\x0f\xd7\rT\xa7}\xc0`w|\x8a,~\xefJ>\x81\x
f5\xa1[(cDU\x87\xfb#\xc1S`\xcd\x1c\xd8BH\xdc\xa2\x9cFE\xbav\xd6\xf4i\x89\xe6*B3\
xad\xf6\xf0\xc7\xd2k\'\x95\xfbZ\xc1\xc7h\xeb\x8e&\x8c\x92\xd0\xdb\xe8\x02\x1d\x1
6\xacFRm\x9c\x9d+d\x05\x7f\xe5"Eb\x8b1\xc2\xbcz\xe5qx\x12\xdb\xac\x0f=\xed\xdeNh
y\xe2W\xdf\x00\xcd\xb9\xa2bY\x1f\xe7\x1f\xd4\x9et\x9b\x9a\xcf\xe99\xb1\x8ck_;\x1
0\x89^\xd5\xd6,\xe8\x91u\xeb\xe0~\xfb\xca+\xc8\xb0Pb\x01\x8f\xa5\xcf3\xd1~\xfb\x
b2?PKPx\x07\x9c\xbc$\x0c\x01\xea\xb7\x13\xc3\x86/d4&\\M\xaf%\xa5_,\x98\x84\xa7V\
x91\x9f\xe2\xd6\xf5\xa6~\x16\xb99/\xb0\x7fy\xbb\x14\x8e\xd4\xf2\xa2\x8bKPq\xa6K\
t\xdf=\x15\x88\x19\xd6xv\x81\xa1\xae\x04s#RL\xccRQ\xf6\x83\x94\x88\xe9\x9c\xd9H\
x1a^\xc4\xd0\x0b\xa3\x1f\x98\x90\xb5\xde~\x90\xc0\xcb\x9a4b\xd1\xb0O\x1f\x8a\xa8
\xccB9\xb2\xdaj\x0c\x0b\xbf&\x0b]\x15M;\x07\xe2\xf4\xdd$\xbc\x04i^\x04\xc5\x98\x
e4\xbe\n*V\x18F\xac\xba\xdf\xdd;\x82\xebn\xae\xb42\xb8\x8b\x8d\xf4\xde\xf4\x06\x
02\x1e\xd0gy\xbe\x85`\x8f\xf4\xc9E\x8f<\xca\xb5#\x18\xf8-\x99\xca\xa4\x8b\xd2X0>
\x8f#\xfeC0\xe8\x18\xae\x97y\xc2\xa5.+\tN\xeb\xf7\x1a\x83\xb6t\xdbs\xa0\xd9\xb0f
\xf8ZU\xb8\t\xcan\xa8,\xc1\xa2R\xd2\x0c2r\x81)\x89\x8d*\xbd\x8e\xfb\xee\xeaVp\xf
8\xeaF\xc3\xaa\xfcG\xc5tmN\x8d\x1d&\x9cJ9\x88A\xba\xb2\xb4O;H}x0\x92\xfd\xf2\xfb
\xd4hsKlK\x99Mz\x03\xe5<g5H\x80\x11\xd3\xb6T4\xfakM\x7f\x84\xdc\xb2\xecgd\x95n^\
x0fC@\xe2\x85W+M\x8f\xc1\x0b\x158\x1e\xfc\x19_v$\xf3!\xe2N\x05$\xf2\x9a\xda\xad1
\xc3O\xaa+\xe0Z\x83\xf0,\xd1\x15}@?\r\\\xb7I\x14"\xc9\xb4z\x81B\xca!\x03\x84\xa4
k/\xa3;\xe1\x0f\xb1oEi\xd3<\x9c::\xc3x\xc9\x8c\xc5\x00\xa6\xdf\x0f\xfd\x91g\'\\\
x83Ai\x9f\xffH\x05\xbb\xba\xc3\x1f\x81Y\x86#|d\xa2{\xe9\xcb\xea\xcb\xd0!\xa8?\xa
3\x1d\x0e\xcd\xe2\x9b\xe8\xf1h\x94/\xdeh\x08\x8fZ6\x93\xe7@\x1f\xfeg\xac\xdd\xd9
KJz\x1aLd\xcf\x1ar\x87\xee\xa3\x15\xcc\x0bl\xf1\x03}\x94\x1d\xe5\xe6\xdcso\x81\x
88\xb2\xf6\x8fp?\xa4\x19\x1bH\x1a\xf7\xf6O4\xbe\xdd\xe7\xf7\x82\x9f{\x7f\x146&\x
fdE\xee1\xa1m\xb4\xfe\rb\xfb\xb5;\xac\xe54\xd4\xa7\x06U\x91\x9f\x05\xa9\xef@\xfc
\xe6\x89y\xa8\x10\xcf\x98 \xd1B;\x96\xb6$L\x9d\x8f\x98C2\x8d\xaf\xaey:\x89\xe5V\
x8a\xb0\xd5\xc4\x81c\xf0_:Tk\x066\xa7V\x11\xdd\x0f\xfdA\x859\x08\xea\x85w\xf2\xe
d\xf1xv\xbc\x9c\x9b\xd8\xcbR\xbb\xf8HH2U\x0c\xe3U\x98\xa9\x8b!$\xa6-\x06,O\xc4\x
f3Z\x8f\xb2\xf1\xef\x85\xe0\xf6\x91=@\x91\xe6\xb49\xe7S\x0ei7\xdd\xc9N\x98\x15\x
ec2\x07\xc1a\x1d\x18"y\xc7<\xb2\xcaJ\x8e6`\xb0|\xcf"\x17\xffX\x12\xde\x81\xec\xe
e06\xb2\xa1\xe7\xfb\xb0\\\xce\xa8\xe8?\x98\xdaA\xda\x8e\xdc\x927h\xa9\xf3L\xd6`\
x88L]\x9f\x8c\xec\x87wI\x81"f\xb6\xa3\x82\xf1=\xa6`\x1d\xf8\x1b\x1bv\x83~\xa6\x0
bb\xdbz_"\x0bW\x1a\x12X0\xe6\x02\xdb\xf8\x8f\xad\xf2 \xc2\xce\x91\x86\x94\xf9d\x
ff/,P&\xd0\x1b\xe3\xde\xfd\x0c\xb3\xad\xbf\xba\x95~\xbc\xff\x07\x0b\xe4\xb5\x87\
x9b\t\x00\x00'
```


1.2 Kyber CPAPKE

Começou-se com a implementação do algoritmo que permite uma segurança do tipo IND-CPA, isto é, contra Chosen Plaintext Attacks.

Sendo assim, começou-se com a implementação de funções auxiliares:

- ***parse***: Através de um conjunto de bytes, devolve um polinómio.
- ***XOF***: Corresponde a uma função do tipo extendable output function.
- ***PRF***: Corresponde a uma função do tipo pseudorandom function.
- ***G***: Realiza o *hash* através do SHA3-512;
- ***encode***: Devolve um array de bytes através de um polinómio.
- ***decode***: Possui um comportamento contrário ao encode.
- ***compress***: Realiza a compressão de um polinómio.
- ***decompress***: Função oposta ao ***compress***.
- ***transposta***: Calcula a transposta de uma matriz.

De seguida, realizou-se a construção das funções principais:

- ***gerar_chaves***: Realiza a geração de chaves privadas e públicas.
- ***cifragem***: Realiza a cifragem da mensagem através da chave pública.
- ***decifragem***: Decifra um criptograma, devolvendo um texto limpo.

No entanto, houveram dificuldades na construção da função *ntt* e *ntt_inversa* o que impossibilitou o bom funcionamento destas funções.

[1]:
```
import os
import random as rn
from sympy import ntt
from cryptography.hazmat.primitives import hashes
import numpy
from sympy import intt
import gzip
import struct
```

[2]:
```
n = 256
q = 343576577
T = NTT(n,q)
k = 2
n1 = 3
n2 = 2
du, dv = 10, 4

_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))
```

```
def ntt():
    return

def ntt_inv():
    return

def tamanho(stringB, numberS):
    contador = 10
    auxContador = 1
    i = 0
    while i < len(stringB):
        if numberS == auxContador:
            i = i + 10
            while (i < len(stringB)) and (stringB[i] != 31 or stringB[i + 1] !=␣
↪139 or stringB[i + 2] != 8 or stringB[i + 3] != 0  ):
                countador = countador + 1
                i = i + 1
            auxContador = auxContador + 1

        i = i + 1
        if (i + 10) < len(stringB) and (stringB[i] == 31 and stringB[i + 1] ==␣
↪139 and stringB[i + 2] == 8 and stringB[i + 3] == 0 ):
            auxContador = auxContador + 1
        if auxContador > numberS:
            break
    return countador

def parse(str_bytes):
    result = []
    for i in str_bytes:
        result.append(i)
    return Rq(result)

def XOF(p,i,j):
    digest = hashes.Hash(hashes.SHAKE128(int(32)))
    digest.update(p)
    digest.update(bytes(i))
    digest.update(bytes(j))
    r = digest.finalize()
    return r

def PRF(s,b):
    digest = hashes.Hash(hashes.SHAKE256(int(32)))
    digest.update(s)
    digest.update(bytes(b))
    r = digest.finalize()
    return r
```

```
def G(d):
    digest = hashes.Hash(hashes.SHA512())
    digest.update(bytes(d))
    r = digest.finalize()
    return r

def encode(poly):
    byte=b''
    aux=1
    countadorX=0
    for j in poly:
        if(j>255):
            aux=2
        if (j > 65025):
            aux = 3
        if (j > 16581375):
            aux = 4
        if (j > 4228250625):
            aux = 5
        byte = byte+ int((_Z(j))).to_bytes( aux, 'big')
        byte = byte +"/-n-/".encode()
        countadorX =countadorX +1
    return byt

def decode(byt):
    listaCoeficiente = []
    byteAux = b''
    desc=0
    while desc <byt.__len__():
        if byt[desc] == 47 and byt[desc+1]==45 and byt[desc+2]==110 and␣
↪byt[desc+3]==45 and byt[desc+4] == 47 :
            desc = desc+4
            listaCoeficiente.append(int.from_bytes(byteAux, 'big'))
            byteAux = b''
        else:
            byteAux = byteAux + bytearray([int(_Z(byt[desc]))])
        desc = desc+1
    return listaCoeficiente

def compress(polinomio):
    polinomioB= encode(polinomio)
    compress = gzip.compress(polinomioB)
    return compress

def decompress(compress):
    unpack = gzip.decompress(compress)
```

```
    return Rq(decode(unpack))

def transposta(matrix):
    zipped_rows = zip(*matrix)
    transpose_matrix = [list(row) for row in zipped_rows]
    return transpose_matrix

def gerar_chaves():
    d = bytearray(os.urandom(32))
    p = G(d)[:32]
    teta = G(d)[-32:]
    N = 0
    A = [[ 0 for x in range(k-1)] for y in range(k-1)]
    for i in range(0,k-1):
        for j in range(0,k-1):
            A[i][j] =parse(XOF(p,i,j))
    s = []
    for i in range(0,k-1):
        s.append(parse(PRF(teta,N)))
        N = N + 1
    e = []
    for i in range(0,k-1):
        e.append(parse(PRF(teta,N)))
        N = N + 1
    s1 = Rq(T.ntt(s[0]))
    e1 = Rq(T.ntt(e[0]))
    t = A[0][0].lift() * s1.lift() + e1.lift()
    pk = encode(t) + p
    sk = encode(s1)
    return pk, sk

def cifragem(pk, m, coins):
    N = 0
    t2 = pk[:len(pk)-32]
    t= decode(t2)
    p = pk[-32:]
    A = [[ 0 for x in range(k-1)] for y in range(k-1)]
    for i in range(0,k-1):
        for j in range(0,k-1):
            A[i][j] =parse(XOF(p,i,j))
    AT = transposta(A)
    r = []
    for i in range(0,k-1):
        r.append(parse((PRF(bytearray(r),bytearray([N])))))
        N = N + 1
    e1 = []
    for i in range(0,k-1):
```

```
        e1.append(parse(PRF(bytearray(r[i]),bytearray([N]))))
        N = N + 1
    e2 = parse(PRF(bytearray(r[0].list()),N))
    r1= Rq(T.ntt(r[0]))
    u= Rq(T.ntt_inv(A[0][0].lift()*r1.lift()))+e1[0]
    v = Rq(T.ntt(Rq(t).lift() * r1.lift())) + e2 + decompress(m)
    c1 = compress(u)
    c2 = compress(v)
    c = c1 + c2
    return c

def decifragem(sk, ciphertext):
    u = decompress(ciphertext[:tamanho(ciphertext,1)])
    v = decompress(ciphertext[-tamanho(ciphertext,2):])
    s1 = Rq(decode(sk))
    m = compress(v.lift() - (T.ntt_inv(s1.lift()*Rq(T.ntt(u)).lift())))
    return m
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipykernel_4688/1247484197.py in <cell line: 3>()
      1 n = Integer(256)
      2 q = Integer(343576577)
----> 3 T = NTT(n,q)
      4 k = Integer(2)
      5 n1 = Integer(3)

NameError: name 'NTT' is not defined
```

1.3 Resultados obtidos

[3]:
```
pk, sk = gerar_chaves()

m = Rq([1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1,␣
 ↪0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1,
            0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0,␣
 ↪1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1,
            1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1,␣
 ↪1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
            1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1,␣
 ↪0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1,␣
 ↪1, 1])
print("Mensagem = ", compress(m), "\n")
coins = bytearray(os.urandom(32))
```

```
texto_cifrado = cifragem(pk, compress(m),coins)
print("Texto cifrado = ", texto_cifrado, "\n")
texto = decifragem(sk, texto_cifrado)
print("Texto limpo = ", texto)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipykernel_4688/3069925417.py in <cell line: 1>()
----> 1 pk, sk = gerar_chaves()
      2 
      3 m = Rq([Integer(1), Integer(0), Integer(1), Integer(1), Integer(1),␣
 ↪Integer(1), Integer(1), Integer(1), Integer(0), Integer(1), Integer(1),␣
 ↪Integer(0), Integer(1), Integer(1), Integer(1), Integer(0), Integer(0),␣
 ↪Integer(0), Integer(0), Integer(1), Integer(1), Integer(0), Integer(1),␣
 ↪Integer(1), Integer(0), Integer(1), Integer(1), Integer(0), Integer(0),␣
 ↪Integer(1), Integer(1), Integer(1), Integer(1), Integer(1), Integer(1),␣
 ↪Integer(1), Integer(0), Integer(0), Integer(1), Integer(0), Integer(1),␣
 ↪Integer(1), Integer(1), Integer(1), Integer(1),
      4              Integer(0), Integer(1), Integer(1), Integer(0), Integer(1),␣
 ↪Integer(0), Integer(0), Integer(1), Integer(0), Integer(1), Integer(1),␣
 ↪Integer(1), Integer(1), Integer(1), Integer(1), Integer(1), Integer(1),␣
 ↪Integer(0), Integer(1), Integer(1), Integer(0), Integer(0), Integer(1),␣
 ↪Integer(0), Integer(0), Integer(1), Integer(1), Integer(1), Integer(1),␣
 ↪Integer(1), Integer(1), Integer(0), Integer(1), Integer(0), Integer(1),␣
 ↪Integer(0), Integer(1), Integer(1), Integer(0), Integer(1), Integer(0),␣
 ↪Integer(1), Integer(0), Integer(1), Integer(1),
      5              Integer(1), Integer(0), Integer(1), Integer(1), Integer(1),␣
 ↪Integer(1), Integer(1), Integer(1), Integer(0), Integer(1), Integer(1),␣
 ↪Integer(0), Integer(1), Integer(1), Integer(0), Integer(1), Integer(1),␣
 ↪Integer(0), Integer(1), Integer(0), Integer(1), Integer(1), Integer(1),␣
 ↪Integer(1), Integer(1), Integer(1), Integer(1), Integer(1), Integer(1),␣
 ↪Integer(1), Integer(1), Integer(0), Integer(1), Integer(1), Integer(1),␣
 ↪Integer(1), Integer(1), Integer(1), Integer(1), Integer(1), Integer(1),␣
 ↪Integer(1), Integer(1), Integer(0), Integer(1),

NameError: name 'gerar_chaves' is not defined
```