Kyber

May 10, 2021

1 Exercício 2 - grupo 6 - Ana Margarida Campos (A85166) , Nuno Pereira (PG42846)

O segundo exercício tinha como objetivo a implementação de duas versões, IND-CPA e IND-CCA, do protótipo **Crystalis Kyber** que foi candidato ao concurso *NIST-PQC*, neste caso da terceira ronda. De seguida, são apresentadas em duas seções (Kyber CPAPKE e Kyber CCAKEM) os resultados da resolução deste exercício acompanhado de uma explicação detalhada de cada função implementada. Este, foi desenvolvido tendo por base o documento *kyber.pdf*.

1.1 Kyber CPAPKE

Esta versão permite obter uma segurança do tipo IND-CPA, isto é, segurança contra ataques Chosen Plaintext Attacks. Numa primeira fase foi necessário implementar algumas funções auxiliares, sendo estas:

- ***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* através de *SHA3-512*;
- ***encode***: recebe um polinómio como argumento e, como resultado, retorna um *byte array*;
- ***decode***: contrário do ***encode***, ou seja, recebe um *byte array* e retorna o polinómio correspondente;
- ***compress***: comprime um polinómio e retorna os *bytes* correspondentes;
- ***decompress***: função oposta ao ***compress***, isto é, descomprime *bytes* e retorna o polinómio correspondente;
- ***transposta***: efetua 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.

Para além destas, recorremos à função ***NTT*** disponibilizada pelo docente.

[1]:
```
# imports necessários para a resolução
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]:
```
class NTT(object):
#
    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 ntt_inv(self,ff):                              ## transformada inversa
        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)
```

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

# criação 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))


# obtenção do 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

# recebe como input um conjunto de bytes e retorna o polinómio correspondente a esse conjunto;
def parse(str_bytes):
    result = []
    for i in str_bytes:
        result.append(i)
    return Rq(result)

# extendable output function, utilizando 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, utilizando 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 através de SHA3-512;
def G(d):
    digest = hashes.Hash(hashes.SHA512())
    digest.update(bytes(d))
```

```python
    r = digest.finalize()
    return r

# recebe um polinómio como argumento e retorna um byte array;
def encode(poly):
    byt=b''
    aux=1
    countX=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
        byt = byt+ int((_Z(j))).to_bytes( aux, 'big')
        byt = byt +"/-n-/".encode()
        countX =countX +1
    return byt

# recebe um byte array e retorna o polinómio correspondente;
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 e retorna os bytes correspondentes;
def compress(polinomio):
    polinomioB= encode(polinomio)
    compress = gzip.compress(polinomioB)
    return compress

# descomprime os bytes e passa para 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

# geração das chaves pública e secreta
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

# cifragem de uma mensagem
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.

[4]:
```
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))
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#]\x99`\x02\xffc\xd4\xd7\xcd\xd3\xd5g\x00\x93\x8cD
\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\x1eJH\xda\x99\xc3@AH\x92\xe7Z\x06\xbcy\x8a\
x91J\xf9\x8b\x91\xaa$\x03Y9\x8e\x92\xd4\xc80J\x92E\x02\x00\xbb\xcfd]\x00\x06\x00
\x00'

Texto cifrado =  b'\x1f\x8b\x08\x00#]\x99`\x02\xff5VyP\xd5U\x14\x86\xb7\xf1x,\x8
f\xc7\xf6\x08c\x93MB V\xc5\xd4\x80bqiH6\x85,@\t#\x96\x02e3\x98!\x1d\x10\x11\x06\
x90M\x82\x80\x0c\x02\t\x97\xd8T\x02\xc2$\r4\x91@\x86Mx(\x08\xa5 B\x0e\x9bh3\xbf\
xf3\xbd\xbf\xde7\xe7\xde{\xcew\xbe\xf3\xfd\xee}l\x8b\xf1T+\x8b(\x0b+\x19AZ\xe5/\
x84TS78\x12\x923\xd9:\xca \x8e\xce7\x89\x14\xe2\x7f\xf8C\'!\xf9\x18\xee\xbf\x84D
\x9a\xde:\x84\x14\xcd\xedB\x19\xc4\x16\x9b, \xef\xcc\x84\x88\x90\xda\xdd\x83\x7f
1\xc8x\x0f\xf2+\x9d\xe8\xa5\xb4<\xe1\xca\x1c\x85\x94\xdf\x8cE \xff\x9ap\r\x842\x
aa=\x18\xc4\xf5s3B\xae\xfe\xacV\x06\xe9\x89\x0e1\xbf\x85\x93\xf9\xc4To-\x93\x01\
xb2\x92\xee\xff@\xaa\x7f\x0e\xf5x\x8dN\xdb\x89\xde\xe0\xde&\x06\xa8n\x93A\x8d\xd
c\xd6\x1e\x06}Y\xa3\x00r\xa3+\xbf\xa2\xe7\xe1o\x9d\xa8~zb9\x03X\xb7>\x98\xa6D\xb
f\xd9>\x07\xdd\x0b\xa1\xaaT8\xa95\x81B\xc2\xcd\xe7\xc6@!\x9e;D\x07[6\xbeM!\x15\x
d5\xe6@\xda?\xe8\xbd\x84]\x7f\x874R\xd6\x18\xff\x7f(\xa4\xf0(\x14\xc9\x04\x0f\xf
7w\xa0\x8b\x82\xa5- }9\xf7\rV\xc5w>\xa1\x022\x83\xee$\x84\xc4\xcc\x1e\xfb_\x04{b
\x9e\xf79\x1b!s{\xc6=\xb4\x99\xbd\x92G<\xea,I\x01NK\xbb5\xaa\x1fo\xfc\x1d\xf9#\x
ba\xa7H\xe7\xed\xcd\xb4[0\xbf\t\xd3\x7fZu\x8bB\x99k\xd4[\xfdL>\x966\x17\x82\x9fJ
b\xf208h\x08\xbe\x82b9\xfa\xd4\x11\xbb\xady\x03\x91_\xac\xc4I5vX\x11\xadM\xf3\xb
d\xa5u\xf4\xe7\x89_i\r\x19O\x96\xf3\x9af\xc5\xf6\xf0\xad\x85\xe41\xf7u\xb1=\xdc\
xb2\x02H\xcd\xbb\x01\xb9rav5\r\xc7\x02J\x11_q\x02\'\xa7\xc2\xc9\xc4\xecKe\x018\x
f8\xf9\x9dH\x08mjSB\x0e\xc8,\xdb\x8fP\xd3\x90\x0fT\xaa\xcb\x9bFG7\x0f\x84!V\x9d\
xaf\x84\xce\x93]\x0b\x89v\x93\x99\x1f\xd5<\xbcN.b\x9b\x97I\xcdsv@:\xe6\x9d\xebOp
\xd0\xc7\x93\xc6\xc0\xd3\x9dK\x06G\xef#g\xb0\xe8\xe2\xaf\x0es\x06\x04\xbcO\xd4t\
xcc\xf5\x08\x8c\xa7nC\x03\x8d\xe1R\x03\xba\xba\xb7QM\xf5@\t\xed\xe2\x19\x94am\xb
a\xefg\xa0\xb4\x9b\xf0\x8c\x92\x9c\x1f\x9b\x90\xba\xde\xf7\xb7Q\xdd(\x98\xe4\xe3
\xceFiA\xc7\xc2\xe5U4\x1c\xbdk\x04\x1d\xf8X\xd6\xd04/\x0e\x86\xe0d\x95\xe1g\xe0\
xed\x7f\x1a\n\xf1\x17\x1fe\x03I\xc2\xe0E\xe5\x03\xfb\x1c\x90\xc4\x81E:\xcb\x16\x
a7\x9f\x03\xa5\xa3\x9f\xf61Hd"u^\xd0p=R\xd4qT\x10\xeb\xba\x8e\x16\x04\xe3KhA4;\x
ff\x18\xe6\xf3\xbdH\xdf=\xab\xa2bY\x9a\xe4e\x1c\x92\xd4\xdb\x05\x83\xe5!\xdee|;g
\x13*1\xef2\x1bs$\x89p|\tJ*69\x0c\xea\x1d\xa5k\xcfx-\x85XG\rh\x10\xd8T\xa4\x0c1\
x16\xde\x82\xc8\x1f\xef\xfe\x1ae\xdc&v@=\xed\xddMX}`\x0f\x8f\xaa\xabw\xc1`\xf2\x
97\x82\x0c\xa0\xb7\xd6\x0b8RQlm-\xfd\x80\xfdq\x0f\xcbi\xa6B\x0fat\x06\xc9,\xbcM\
x0e\xe3\xa4\xbck\x08Y\xdc\x13\xe8^d\xcfq&\xd0\xb8S\r\xce\xc9\xd7\r\xd0\xb0e\xb7$
J\'!\xd1\xc5\xfc\x05\x8b}^8p\xf8\x9e\xf4\xba\x0eX\x7f\x80\xd8\xb35\t\x94\xeaI?\x
89U\xf3\xf6\x1f\xc915\xce%X\xdcS\xae\x0f\x8e\xcdIW\x11\xcb\x1b:\x8a^\x12\x8a\x95
1\x96\x1b9n\xe0\x14ZC\x9d\xb2\xbaL\xa4\x8e\xed\xa8\xe7\x13M\xad^\x1e}#\xce\xab\x
ae\x0cH\xeb\x87\x11\x94\xc4W\xe1/\x85\xd7\xda\xb1$\xc2\x93\x16S\x10;\xbfP\x8a\xd
aE"c$\xed,\xf4\x85\x8eGX_\xa0)\x1b\x7f\xb8I\xed\xb9\xa0\r\x89\x83\xeai\x1f\xe7\x
e9H\x1e\x16S\x8b\xfc\x90n\xf9\x18\xbdP\xec\x8f*\xff\x04\xff\xd5\xf1y\xe4\x9d0$\x
89\xd8\x17J\xdeA\x01\xcd\xe3\xb8\xfcywMq\xbf\xab\xd8\xd66@"\x07\x8b\x9d\xc8\xcb7
\xa9B\x92\xeb\x8d\xb0\x85\xa0\x13\xf7-\xb7\xe7\xa7\xbd\x04\xfc\xed\xdbI\x972\x05
\x01\x0e\xee\xf3\xa4M\xacz\xb3kh\xbeE\x8c\xf7@\xf4]\xed\x0c\x9a\xef\xf6\xa2\x0b1
\xc8\x06O\xb9 c\x07=\x7f\\\x07\x17XA\xd83\xc2\x02\xed\xb0\xeaY\x88k\x9b7N\xdb\xd
es\xc6\xc3\xa9xCY\x1b\xba,>K\x022\x8f\x85\x8bD\xe3:T\xfd\xf4\x95(\xa4=%\x88fP\x0
```

2\x07\xaf;\xbfT\x8e\x1e\x04vy\xa4\x07T\x9f\x14dP\xc8\xa8p+\x18v\x9c\xa4\xfb%\xb6
\x1b\xbe\x94\x7f8\xfd\x07\xd6\xbc\x02\xddQY\x18\xef\x02\xaa)&6\xe0`e0\x89}\rq\x1
8\x8eB\xf1y\xe9\xa56z\xed\x18:\xa9\xc8\xea\x01\xc7+\x86\xf8?#\n8C\x0f\x12\xb7 \x
d2\x8a\x94}\xd5}\nb\xafO\x89\x81\x1e[\xa2\xaa\xea\xc1WhA9\x8e\x8f\xabLml\x17\xbd
\xef\xb2\xd9!\xbd\xe4\xa5!;|\xd8rY\x1c\xba\x00\xfe\x07\xcf\xd06\xed\x9b\t\x00\x0
0\x1f\x8b\x08\x00#]\x99`\x02\xff5V{8\xd5w\x18w\x1c\xe7\xee8\xee\n5\xe3\xc9"\xe7\
xd1\xd0\x93\xcb\xa3\x8cnXS\x8f\xe8\x8cR.\x85m\x14\x13\xed\t\x93\x89D\xe5RL\xad\x
cd-\xce\xd6\xc6.%\xcd}r9\xb3\t\xd34\xa5X\xf5\xb8l\x19f\xc3\x88=\xcf\xef\xfd\xfc\
xfe:\x9f\xe7\xf7\xbe\xdf\xf7\xfd|\xde\xf7\xfd\xbe\xdf\xa3\xc6\x9f\x98\xce\xb0\x9
5G\xcbm\xd5tU~\x81\x0c\xd20\xda\xd3J\x9f$N\xc3\xa5\x844wDj\x11\xd2?\xb6\x10Dns\x
f5\xc18\x99\xa1\xa5CH\xab~\xc2\x98A\xea\x8e\xbe%\xf4I\xd8\x1a\xbf\x95\x90\xd4\xc
d\xce\x81A\x1c\x07\xd5}\xf2\xda?\xe9J6Y\xbaU"2\xfd\xec\xac\x0e\x7f\xf7\xd24B:J\x
af\\Bb\xf3+\xc9\xb0zf{\x11\x0f\xeb\xf6z\n{\xed\xf5\x7f@\xc3\xc6\xe4\x03\xa0\xb14
\x15yU\x96\t\x90\xaa\xa3\xc3\x8d\x90\xe8\xc8\xc1"\x90\xbc\xff\xc5;\x84\x04oE|\x0
f\xf1\xb5\xbb\xfa\ti\xd7\x8c|\x07ks\x95\x0f\xac\xb9\xa5_B\xbd"\x9cJ\xc3M>q\x13Y\
xcdg\xed\x816GZ\xc1-\xa4\x7f\x85A|\xab\x80Z\xc4\x10\x1e\x18\x81\x98\x98\xe9\x9d\
x0c\xe2U\xdf\xfe\x1au\xe0\xb4?\x03a\x85\xd3\x1fP\xbf\xe6\xc5\xbf\xb0F\xbb\x12\xc
d,\xd3F\xd4\xc89!\x10a+8M\x10\xa8\x16z\x0f\xfe\xa1\xdd\x87 \xf5\x8eK\x1e\x83\xa2
\x1c\x8f\xc1+\xe9\x91\'!~[\x957P\xb1\xf7-*\xea\x8cO\x18\x92\xfb\xdd5!\xa4\x17rQ\
x0e\xb7F\xcbA\xd4\xe8\xc8h!)\x18\xfc\xfb\x1a\x95\x83\xf7\xfc\x12u\xb9(\xa6\x1a58
\xad\xb3\x86AmE\xee\x14}k\x88\x07\xf9\xe4\xcb\xb3\x91f\xfd\xba*p.H\xb4&\xe3\xe2\
xa3}\x10\xa9\xd8O\xbd\xe6k\x07\x9d\xc3\'\xd1\xd2s\xf2\xd2*\xa1\xc1\xe4\x16\xe6\x
90\x8d[z\x9c\xe4\xf4\xf5\xf4\xd1\x87\xd02\xaa\x07O\xbb\xd7\x88\x08H\n\x0c!J\xb5\
xe9GL\xb8T\xf9\n\x18\x94\xf4|\x8b\x1aI\x0f\x87@\xe8\xc1\x9c$\x06\xc9~\t\xa7\xa0!
\x11p\x97\x1d^~\x01\xa1\xc9\xf7\x8c1\x00\xa5\xeb\x8fB\xd8\x81\xa3=\xb0\xba\xac\x
f6\xc2\x89\xb6\xed\xe3D\xde\x7f\x19\x9f\x84uN\xa3\xc8d\x10A\xd7D\x1a\x18\xc7\x0e
\xe7\xbb\xeeP\xfd\xfe\xd4\x87\x08\xe6\x99\xf1\x06\xfc\x95\xd9\x97O\x01\x12E8\xac
\xfe\xd3/\x81NF\xd9\xe1\xecd\xdc\x05\xa8j\xe9\xd3\xc4\x89\xa4.\xcc\xa9\xaci-Z*\x
db\x9d\xa8\x02\xf5\xe6\x01}0\x88n\x80D\xcd\xa6\xd6\x16Dv87\x03\xeb\xdcW\xd8\x12\
x82\xbb\xd9\xe71"gn\xcdR\xa9\r^\xea"\\K}*\x8e\xa6\x9d\xc7m\x93.\xf4/\x93[B\x1b&U
<\x13\x80\x1eK}~\xa7U\xa3n5\x7f\x05\'\xf9Y\t\xd4J\xb9\x02\xdc\xc4\xa6e\x98cA\x8e
`\x11\'\x03o\xb0\xd3$\x97\x17\xc2:\x98\xc2\xc1\xb7\xb0\xa6n\nb\xfa\x80-\xbf\xef{
\x16\x94j!M\x01/e\xf0,\xcbC\xf49b\x84m\xf8\x93\xe8:\xae\xda\xc8\x80r\xe5\x1d\x14
5E\xccN\xeb\xed\xd8J\xa0M\rXtZ+\x93\x9f!SJ-\x95\x8d\xc39\x19\x80\x0er\xf7\xb5\xb
0U\x93\x82\xa3\xde.;d\x17\x8a\xb5V3\xc8e\xc3N\xa4\xfaix\x9eB4\xce}\x84\x10\xafN\
xa0i\x92\x9a\xd8\\v5\xe8aSk^5\xc2\xb0\xe8\xeeU\x1b\xa6\xc5\xd8\xe3\x81\xb6H\xa2\
xb6A\x9dx\xdc\xd4\x1f\xd9\x1f\xaelcPq3\x17\xb6\xda\x13XZ\xc2\t\xfdS\x94}\xd1\x82
u\xf7+\xff\x0b\x99\x8e/aPd\x0e\xdeY(\x9f\xa6\xc8\x02A\xd6\xfe\xd0\x89B\xaau\xcd\
xe1lz>=\x19\x1a^[\xdc\xd9\x9d\x19{\n\x07\xacOwP\xa7\xcc\xf2\xfc\x18\xe0\xedw\x16
\xf1sS\xeb\xb1\x7f\x9f9#\x91\xe1CjJj\x08\xb4\xe9\x07m\xa1\xb2p\x06zcq\xce\xde\xa
8\x02i\xb6g\x9aQp\xcb0\x11*[\xec\x08\xbd\x92)\xfb1\x10\x9c\xca;\x83\xf8^\x97\xe9
}Q\xf7p\xa5\xa5\xebc\x18\xc9>r=\x99\xa0<|h7\xa5L\xa9"=|7#\x1b:\xd6kF\xd7\x9e\xf7
$\xe05\xd4\x8b7\xc2\x9e\x8b\x1b\nE\xc2\x9a%l\x0cQ\xcdX\x17\xe8\xd4\xad\xa2\x8b\x
a5~]L\x9b\x8bg?\xf0\x1f\xfal\xdc\x8aA\x92v\xfbb\x15\xe8\xfb\xce\xf4!\x9a\xf9\xd0
\x12\xac\x8f\xed\xbf\xa1J[\xc9\xf1p\x0bL\xaeR\xbbx17\x9d\xc0hc|\x0c\x1a\xfd\xf8\
xb2\x1ab\xec\xa8N\x01\x8f1\xbd\x01p\xdb\xac\x10b\xb6\xc7\xaf\xe3\xd1\x97\xa4\xaf
\x9b\xa2\x9e\xfc\xfa\x94\xed\xc9\xdbO\xb0W\xc4\xbcHlU\xf1\xd3\xc2 \xa0\xf9\x06\x
bc\xd8\xfc\xf0J\xdc\x7f\xd9^\xae%\x83l\n!F;;\n\xe4\xa4\xb6\x0f\x8a\x81\xceN`j\xa

```
5\xca\xa4\x8b\xa0t\xa3\xa5\x1c\'TF\xd4\x84\xbe!\xbaE\xbc\x8f\x7f\xd3 \xd0\x99\x1
9\xc8\xc6j\xc6&W^\xa0\x15\xc1m5\xa0f\xf02:\x0b\xa0\xa8C\xb3\x9dj\x16\xaf\x81\xd1
\x95\x04\x97\xd1\x0e\xaeK{\x13\x83d\xebF/\x96\x86\xb0#\x1f=Q5a\xc3\t\x0c\x14\xec
\x02\x1e\xfd\x14\xff\x00t\xc2\xf7`\xef\x88\xc4Vt\x1f\'>\xa1_^l\x05\x95\xf5\x7fq=
\x13\xe5\xa3\t\x00\x00'

Texto limpo =  b'\x1f\x8b\x08\x00#]\x99`\x02\xff5V{P\xd4U\x14f\xd9]\x16\x16vA\x9
6\x96\x18\nE\x1e\xf2ZX\\\xb1,\xe2%ND,P8\x10\x0fa1\xde\x81(\xb9F\x89\x08\xcb\x8e\
xae<F%\x08\xc8H\xc5\xc8\x89\x08\x0c,\x04Dd \x81E\xf2\x11M\xf1\xd8\x18\x8a\x00\x1
15\xc8\xc7\x90\xa0\xcd\xfc\xce\xc7_\xfb\xcd\xbd\xbf{\xbe\xef|\xe7\xdcsW\xcfT\xdb
\xff\xd0C\xb2G\xe2\xa1g\xaaJ\xcd"\xc4\xab\x16\x7f\x03\x14u+\x9d\x90\x90=\x15O\xc
8\xb0\xc0Z\xc0 n\xa8b\x1eG\xef\xaao\x132{i\xd3\xc7\x84Lb\xee\x17\xe3h\xc4\xd25\x
ecjY\\\x06\xe9\xff5\xdf\xc3\x00V}\xf9v\xecu\xcau\xb4W\xec\xbf\x00\xa6\x86\x0b\x9
6\x84\x0c\x8e\xcbdXkR-\x13\xe2\xb3*n\x102:\xdbt\x96\x90yl\x825!A\x86\xcf\xab\xc4
\xe0\x1b\x07\x1d\xeb\x16#l\xc1\x15Sp\x00\xca\'t\xa006\xdf\xa2C\xb8\xd6\x18%v\xbf
\xce\xa9G\xe0\xad\xfcvX\xf2\xb0p\x03N\xe4\x19v\x90\x11#]\xb1\x0c\x08\xfa"\x99\xf
9e\xd7\x06\xc9\x89\xdcN\xebG\x9f\xc4L\r"\x81\x0f\x0f\xeeFR\xddud$;\xba\xae\x1f\x
84vm\x08.\xda\x13ZA\xc8b\xf3\xea[8:\xe5\xaa\xc2n\\6\xe4\x18\x0e\xc8\xdd!15\xf3.\
x12=\x92\x06gLF\xa5T\x18\xfd\x93cS0\x86\x9f\x12\x07\xb2\xe8\xa1\x1ad\xec\x9aN\x1
e\xb1WC\xccH\xaf\xce\xe7\x01\xaa7\xec\xed\x86\xccu3\x14\x96\xab\xadn\xc4\xc1\xb7
\xbbQn\xa3\x96J5\xccM\x94\xa6\x82j\xbao\x16\xbb\xbfk\x8fC\xe5\x89\x1c?\xd0\xaf\x
ff\xe3\x17\xe8M8\xf0\x0c\x14\x8fl\xad@\xbb\xe8\x84\xbcLZ*.c\xcd\xda\x1a\xe5\xb0\
xc8l\x1eG<~P\x19\xcaQ=[\xc0 1\xd7\x9cdz\r\xc1V\xc3\xac\x80G\x0c\xe2\xb8\xef\x1a\
x80\xa2\xd8\xf2\x10JyK\x881\x84?\xcf\x8f\xa4\x83\xcb\xdd\x1b\xa1\xb1[\xf9\'\x94)
\x8a\x8eQI\xbdz\xb1$\x1c?\x7f\x01\x9b\xd9a`2\xb9\xac;\x81\xea\xd6\xf8\x1d\xc4\x9
a\x9b\xa7/\x15!`\xba\x89\x08l\x1b}\x18\x90\xf0\xec\x10\xbe9j\xf3\n\x12\x8a\xe4\x
9d\x86-3\x7f\xe3"\x18k\x1c\x9a`i^\xe2\x97P\x9b\xe4\x1c\r\x91~\xaf\xa1\x19\x84\xe
b?qb\x90R*\x81Q2o)\xe2F\\\x84\x8d\x16\xdbW\xb6!ZY\xe0)0\xe8\xd4\xd4\xc2,\xa7\xf6
wq\xa0\xe4\xfa\xa7\xf8\xcc"\x07W\xc84\xde\x11WY\xa8\xb7\x1cH\xfe\xa5\x05\x17\x92
\xb7\x05\xad7\xd1\x9d\xee9\x1e\xb4\xe4!\xd1"\xbf\xf0]\xbf"Xx\xb83>KRR.\x9c\xeb\x
9e\xd4\xe1\xe2\x95{\x14\xb3\xf4\x89\x0b\xbe\xa9\x17\xbd\x83|5\\;\xf8\xe2\xd3\x99
GZ\xf3=.\xd1\xf7\xc7\x8a\xcf \xbaS\xf4O\xa8y\xdb=1\xd669\x19"\xc6\xeaF\x92_\x11\
xec\x896X\xdf\xfd\x0f\xd0d<]\x15\x96Y\xef\x19\xb4yV\x00\x03\xc6E\x98s\x06W\xae\x
9dG$\xce\xcbO\xa9\xa6sW>#\xb0Y\xe5\x03/#\x1dN\xa2/\xd4;\x15\x14\xf3\xaaB\x00\x9a
\xfdK\xef\x03%\x9d\xd6#\xf5\xba[\xab\xc86Q^G\xd4\xcd\x03k%\xb2\x9fXDB\xb9VI@\xca
\x15*=K\xb3/\x83\x1c\x14[\xd0\x9c\xe6\xd6\x0b\x83P>\x9b\xfe(\n?\xe2RJ_\x1b\x08\x
1e\xc3Ai\x19e\xc6NN\'k\xd8=\x12odV\xf6\x18}\xcb\xcf\xfd\x0f\xdd\xc1\xb3j\xd8\x8f
\xa8\xbb\x1dX\xc8\xb2\xaf\x85:\x96\xb54\xaf\xa1\x18\x8bJ4\x8c\t\xaf\x02S\x95\x1f
\x9a\xdd\x80tS\xf6\xe1\xa1\xe1\r5\xdd\xc7\xdax\xd0\x18\xa2\x89:\x9e\xa0\xc5\xc2\
xdc\xbe\xa2T\xfa\x04\x94%7\xe7\x06\x9dd\xcfkil\xb3\xab~\xa8\xc2\xd76\x9ew\x18$,\
xc7]1\xd2,\xecD\x96\x1d\xb7\x9fb\x8d\xd7\xbc\x97B\x95\x8e6\x93acW\r\xc0\\\xa5\x7
fgm\x1e\x17\xa5@\xbf\xa3\x1d\x86\x99\xd0h\x07={l\x05\x07\xedd\xf1\x06?\x17\xe4\x
d2 \\)\xf3\t_X+\x9a\x13\xd7\xa2r\x85\x9ex\x95\xcc6,\xce\x81"\xa1\xcc\x1fgY\x93t\
xa7"\x1f\\$\x1b\xede[\xc1\xeey(\x1f\xda\xf8,\xca\x93sXN\x16T\x05\x7f\x8e-u%\xfa\
xc2|\xb4\xd9\x88$N\xd9~\xcb\x00\x03\x87A\x1a+\x96\xf5\xe8\x18a\xf89z\xd1\xd9\xb1
\xe9?b\xe9_9\xa6?\x7fj/$\x0bk>Bu\xf9/\xbaV\x93c\x97d#`\x1c\xce\x10\xaf=\xe8CTqV\
xedo\xb8\x0f\xcf\x17\xe0\xebM\xbbl4iDa)\x04v\x1e^A\xd2\xef9\xa7\xc1\x88\xef\xa7\
xc3(\x84\xff\x07\xad(\x92\xb0w\x94A^\x93x]\xcc\xa4G2a\x89ZV\x08\x7f\xb7\xf1\xd6F
```

```
\x96$1\x16\'\xa3Tl\x12\xeb\x92W\x89\xda\x0f\x9e\xa2\xbaq\x1d\xad\x11_X1[\x8d\x18
?O\x96\x90\xda\xd7E\tty-\xdb0\x10\x0c\xed9\n\xa0v\xce0\x83&\xf4\xf0\x14\xf2\x1b\
xe3fH\xf4\x0b\xe2\x02\xfc}QB\xcd\xba\xd8\x92"\xa0\xae\x96H0FvaZ\n\x02G\xf0\x86\x
1b\xfb\x9e\xa3"\xe9\x1f\x9d\x03\xa5(K\xa5\x01z\xd31\x19\xfe|\xd7C\x13\xfc\x7fW\x
88\xc0\xc9\xa1\t\x00\x00'
```


1.2 Kyber CCAKEM

A segunda implementação consistiu no desenvolvimento de funções que permitissem uma segurança IND-CCA, ou seja, segurança contra ataques *Chosen Ciphertext Attacks.* Tal como anteriormente, tornou-se necessário, numa primeira fase, a implementação de funções auxiliares. São utilizadas algumas das funções criadas previamente mas com a adição de novas funções que possibilitam a implementação com KEM. Estas são as seguintes:

- ***H***: função de *hash* que recorre ao *SHA3-256*;
- ***KDF***: função do tipo *key-derivation function* que utiliza *SHAKE-256.*

Como funções principais temos:

- ***gerar_chaves_KEM***: tem como objetivo a criação das chaves pública e secreta que vão ser importantes na cifragem e decifragem respetivamente;
- ***cifragem_KEM***: recebe como *input* a chave pública, calcula o *hash* de um m e ciframos este de modo a obter o encapsulamento. Retorna o criptograma e a chave partilhada. Utiliza como recurso as funções da implementação anterior;
- ***decifragem_KEM***: recebe como argumentos o criptograma e a chave secreta e, após uma séria de cálculos, retorna a chave partilhada caso não ocorra erros.

[5]:
```python
# função de hash com recurso a SHA256
def H(pk):
    digest = hashes.Hash(hashes.SHA256())
    digest.update(pk)
    r = digest.finalize()
    return r

# key derivation function com SHAKE256
def KDF(b):
    digest = hashes.Hash(hashes.SHAKE256(int(32)))
    digest.update(b)
    r = digest.finalize()
    return r

# geração da chaves pública e secreta
def gerar_chaves_KEM():
    z = bytearray(os.urandom(32))
    pk, sk1 = gerar_chaves()
    sk = sk1 + pk + H(pk) + z
    return pk, sk
```

```python

# cifraagem e encapsulamento do m
def cifragem_KEM(pk):
    m = bytearray(os.urandom(32))
    m = H(m)
    k1 = G(m + H(pk))[:32]
    r =  G(m + H(pk))[-32:]
    c = cifragem(pk, compress(m), r)
    k = KDF(k1 + H(c))
    return c, k

# decifragem e obtenção da chave partilhada
def decifragem_KEM(c, sk):
    pk = sk + bytearray.fromhex('{:0192x}'.format(12*k*(n//8)))
    h = sk + bytearray.fromhex('{:0192x}'.format(24*k*(n//8))) + bytearray(os.
↪urandom(32))
    z = sk + bytearray.fromhex('{:0192x}'.format(24*k*(n//8))) + bytearray(os.
↪urandom(64))
    m1 = decifragem(sk, c)
    k1 = G(m1+h)[:32]
    r1 = G(m1+h)[-32:]
    c1 = cifragem(pk,m1,r1)
    if c==c1:
        return KDF(k1+H(c))
    else:
        return KDF(z + H(c))
```

De seguida são apresentados os resultados obtidos com recurso às funções anteriores.

[6]:
```python
pk, sk = gerar_chaves_KEM()
c, k2 = cifragem_KEM(pk)
print("Criptograma = ", c, "\n")
shared_key = decifragem_KEM(c,sk)
print("Shared Key = ", shared_key)
```

```
Criptograma =
b'\x1f\x8b\x08\x00$]\x99`\x02\xff5VyP\x94u\x18\x86\xbd\xf8v\x81oq\x17\x876\x0eS
\x8ea\xe3\x8aC9\x86\xa2\xc4D\xb2\xa9\xe1\x98\x98B\xb0H\x94(\x10E\x16DH\x92\x0c2\
x12GNK`\x9dI\x0cL\x84 D49\xc4\xc0\x03%\x12\x05\x16\x87c\xc0\x00\x95s8\x84f\xf6}\
xbe\xbf\xf6\x99\xdf\xef{\xaf\xe7y\xdf\xf7\xb7\xbc\x05\x8b#\x8e\xca8\xa5\xa3\x8et
\xa7\x897!6J\xddAH\xdc\xd39A\xc8\xc0\xdf&\x94\x10\x93|=\x9e\x90l\xd9I\x08[\x8b\x
f8\x93\x84\xe4\x9a\xc9\x11\xdc\x16*ja\xf1\xb7s\xaf\x16-}>\x8a;\x8d\xcf%B\x86O\x1
b\xdf"d4\x184\x0bo\xf9\x9e\xc7\xb5H7l\xe0G-(\x9e\xd7h\x7f\x05\x9a\x88\x0cx\x18JN
\x02\xb2d\x7f\xd1"^\xcdt\x0f\\\x8d\xff\xb5\x19\xee;\x02\xfd\x91D\xb6F\x87\x90\xf
e\xfe[wP(\xef\xd7j\n\xf4\xac\xf2"\x0c\x12\x9b,`\xe0\xdb5\x833\xdbG\xe7P\xdd\xce\
x8c\x9f\x10"D\xe7\x06\xceL7QrBO\xeb\n:Z\xd7=^\t\xd3}i(T./\x03\x952\xdf\xb3J2\xc8
\xf56\xc0g\x05j;B\xa2\xe9$]-\xe2?\x9e$o\xfc\xea\xb6\x8dP#r|\t\xce\xd6\xe6\\\xc0\
```

x95S+\x05\x10\xfa:\xb18\xaaK?\x86J[\xb2\xb6qtm?\x0b\xd3\xd9\x7f\x8b\xc8\xefj\x02
\x03m-\xb7|G4\x1c*\xadGB\xacy\x03jI\x1e*\x02[\xd3\x1f@>}\x83;J\xa4\x14P`N\xe1O\x
19\x85\xd3\x91\xf1\xa7\x8dT\x15\x7f.C\x05*\xdfk#e\x04/\x9f\x0f\xd0\x91\xde\xa8U*
\\\xecw}L\x97\xac\x03\x1aK|&\xa4\x1d\x96\xea\x99\xd7I\xdc5\xa1\x04\xc1\xb7N/R\xb
6^\x9d\xa4\x1e/\xca\xf3\t\xddI,\xb3\xf29\xb9\xdd8\x86\x82\r\xaa\xc8\xff\x8daJQ\x
f8g\x0c:M\xdc[1\x0c=\x1b\xd4\xb9(\xfdB_7*q\x0f\x01\x91R\xb5;\xd46\x0c=\x8d\xfe\x
10\x9fca+W\x8c\x97\xc2\xcb\xb1\xfeD\xd4W\xefx\x80\xa2:\xab\x0e\xc3\xb4.>\x0b"/Mb
\xc0\xa4y\x1fyA\xa1\xc6\xe0gD\xdb\xc3W_\xc0\x87WY\x17L\x1b\xcb\xbe!oS\xa9\xfd\xf
8\x9e\xaf\xfb\xb6\x16Ez\xdf\x87>*A\t\x90\xd5\x00\n\x94\xbc\x11\x8d\x81a\xeb|\xac
\x89\xad\x90f\xca[X\xf9\x19\xd71k\x9d\x18_\xa3\xe4\x95\xbb\x08>3\xf3\t|\\m8\x03B
B\xb7\x11\xd1\x82\t\x93\x83P\xa3\x8a)@\x80\xe3\x05\x17\xc0L~U9\x98y*\xc1\xd6\x10
\xbb2HN\xbf"I\x8c\xdb\x95\xd7~\x83c\x9d\x1c\x04c\x1bM\xf7\x90\xbaY\x97\x14\xf8\x
ac^\x1c\xa2E"{\xbb<\xf0\x91\x19\x8c\xd4\x193\x87\x7f\x10\xa1\xff.\x1au\xdd\xc2\x
89G\x94&O\xbd\x1b\x15Z\xefx\x00\x83\xa9\x94W\xb8Qo\x1e\xa4\xcf<\xb3\x1d\x91\x9b\
xa13\x0c\xc4\x0fvc\x98\xf5\xba\xcb\xe8\x96\x1f\x93\xe4Jyl\xe9\x8d\xe7j^\x0b\xa4\
xbb\xdc\x91\x8f\xc1\xa0\xd3l\x18"\x95\xcfG \x92o_\x13\x94o+G\x00YG\xa1\x1a\xfc\x
16\xd6R_\xf2\x14\xed\x90M\xa4j\xdd\x84\x8c\xce\x8bh\xf7\x08\x03R\xb1f\x06_\xa2eX
\xc1!Nzf\x12E\xc9\xef\xc7Dk\x91\xad\x90\x18\xe2\x0fN\xcd\xe1\xa3\xae"+\xaaW\x10\
x8f\x0e\xf0`n\x13\xf8\xba\x00\xbdl\xe8\x9aH\xdc\x8d\xcd\x83\x13\xe3\x83m?\x83\x9
3\xbe\xda6dU\xbc\x82i\xd1s\xf3w\x00\xc5\xf6\x13\xd8\x00\xd2\xb4 s(\xe1~\x92\xdaU
^\x7f\x02\xac\xc4\x9a\x04\x82\xa9\x1f:\xb8Fg\xc2)\x0f~\xdc\xf61\x02\xe1K)\xc8\xf
a\x80\xcbeX\xfa\x88)<?\xe7\xca$H4K\xa7\xa6\x16\x98\x8a\xe0_v\xca\x98\x16\x8b\xae
}\xad\x1f\xd1\xaaJ\x18G\x01\xe2T,r\xd6\xe6\xab8\x10v8\x0f$2U-;\xb4Hj\x97\x06\x19
\x86\x9eC\x06\xc6\xa0\x19\xeb\xc7\xe8\x0b\x93k\xf0f\x1a\xa0\x0f:R\xaaZp\xab\xd8\
x8b\x846<\x89E\xa8\x8e@l:\xbdD\x19\xf7\x18\xf5-\xe8\x01\x85n-\x86\x93\xf5j\x19\x
ceJ\xe4\xd4B<e{:\xfcnV\xd2\x03\xc1\xbb\x9c\xc9=(\xef\x7f\x18\xa3Eao\x06\xa1\x91\
xca,0\x0b\xd2\xdb\xe5 \xcd0\xe7\x16M\x85\xe8\x9d&.\x90\xa2F\x03\xaf+\x91\xb4\xae
\xd8\xc5e\xea\x83\xea=~P}\xd74\x9eP\xb9G.M\x02\xcfCe\x06\xf7\x11)\xd4\xcf\xba7U\
xf8\xcf *~\x88\xc7\x97\xadq@\x86\xa2\xd5o\xa9\x876\xfc\xbe\x0cq\xac\x86[I\xc2w\x
97\xf6\x91\xd3\x1e%v\xb7q\xd0F\x8a\xbd\xcb\x8b6\xac\xee\xf7\xb6 D4v\x14L\x1b\xfc
\xd1\x85\x97\x81\xd5\x8bF\xaa\xf2\xff\xd2\xaf\xa0\xb2\xeb\x02\xd2W\xc4\xb4\xf8R\
x1c\xb7\x8b\xa7\xb5\xa0!\xfaK\xa8vd\x14\xfa1\x99Fhz\xfd\xec\x17\xabdg\xe6\xcd\r\
x8bY2=\xe1\x82\xf9Xz>\xf87\xaf%\xd0\xc9=\x0f,:&\x8a=\n\xeaK\x16i\xdc\x04.5\xe7\x
91\xea\xde{WA\xf3d&\x0f\x0f\xcf\xc8(\xb7\x18\xad\xe8\xc8"\xdc\x139T\xda`[\xc8%\x
e9x1$~\xa5\xa4\xcf\xff\xae\x03\xfc\x96\xa6\t\x00\x00\x1f\x8b\x08\x00$]\x99`\x02\
xff=V}4\xd5w\x18\xc7\x8f\xebr/\xd7\xbd7\x84s\x0fil\xe8Fe\xc6\xa9\x83:\\%o\x1b\xc
2F\x88\x1d\xd6A\x94\x85\xac\xbc\x1fE/\\\xaa\x1d\xb1b\x9a\xa5\xf5b\xde\x9d)\xaa\x
b1\xd16\xafu\x90\x16\'&b-\xac1\xb2s~\xcf\xe7\xee\xbf\xcfy\xbe\xcf\xf7y>\xcf\xf3|
\xbe\xcf\xef\xa7\xa4u\xdb\xfd\xac\xb54Nj\xadZy\xcd\x93\x05J\xa2_\xaa\x1b\tid\x0e
$\x13\xd2\x1a\xb6Z&\xa4~}\xbf\x90\x10\xb7\xc4\xcd\x9f\x10\xbf\'\xbc\x86\x90\xd8\
xf9\x9d^B\x9a\x1f\xa4\x84\x13\xd2\xb6\x1b\xec\xc7i\x8d\xb55\x8b\x98\xb2\xce\x062
\t\xa2\x0b_\xb3HY$C\x0c\x8dq\xb1\x17!\xe1\xdc\xc40\x8b\x1c#\x87\xe0>\xf4\xe3g\x8
8u\xd7\x1c\x99\xc43%G\xc0\xa3\x95g\x80\xec\t\x1d\x89,\xd2\xfe\xfc\x04,+\x82\x83,
R\xe9\xc9\xaa\xa5\x8a\x9f\x86\xcaX`\xfe\xaa\x87|8\xe9i\xcf\xd1\x83?\x17o\x927/$\
x1di\xa2V6\x11\xe2=R\xa2\xa0\x8c\xf4\xb8*2_\xe6\xad\xc7\xa1\xe72\xfa\xc3Y\xb2%[\
xda|\x0c\xc5\xca\xbbbB\xf7\xec>rA\x9e\xba\xe7\xc6D\x86\xb3\xd3\x0f&Q\xd1\x06rOx\
xbb\x80\xaa\xe3-V(RZ\x11Yt\xba\xbd\xd1@n\x96~\x04\xfa\xdc\xf1\xf3!p0\xed\xd4\x85
\x9f\xef\xd8\x0c\xfc$\xc5-,\xe28H.\xb0@\xad\xbf`\x1bn\x0e\x17\xea\xa3\xc8\xc8\xa

```
9\x01\xd0p\xf20\xc1\xe9XV.q\xb4o\x9f\xc2`\x1a\xf7/A\x18\xb9\xa7\x16a\xeb\xb9J\xa
7*\xde}%\x94\x807\xf8\x03\x81\x94\xd9\xa3\x14\xe1\xbdY9\x0b\x9a\xa7\xda\x11|.Z1\
xd0c\r\x89\xa0\xea\xb5l\x0f\x12q7\x8c\x88\xf4\xd6\xb0z2\xad;d\xf5\x10\x87\xb9\xb
a\xb8 \x0e?\xee\x0cu\xce\xd7\xef\x86\x8a\x1e\x9f\x0f@\xf7\xb2R\xf6\xe2t\xf6\x8d\
x12\xd4P\xbc\xb0\x8fFa\xe2;F\xfa\xe3\xef\xe1\x10\xc7\xd5\x8c\xc3H\xe5<\xad\x03w\
xaf>4H\xbbU\xb2\x15\xad\x8du\x9a@Xn\x901\x88\xb8\x8d\xb8\xa3\x867\xa9w\xd0!}q*\x
fc\x8a\x1c\xc6\xe0\xa7\x96.B\xe4Q\xddW\xe8\xdfZX\x1a\xa8\xfb\xe7\x7f\x82\xbb\x7f
\xb8\xd0HUz\xafY\xa0j\xbd\xaa\x07\x98\xb2\xaa\xdf.\xb8e\x1c\xa0\xb62K{ e~\xe8kzD
j\xdd\x8b\xc1\xf0\x8f\xack\xa3Rgd\x17PMU\x99\x07\xb8-\x1c\x95\xa2j\xb5\xb1\x15p\
x8b{\x1f\xd2\x12\xcao\x04!\xc83\x0e\xde\x83z{<\x1f\x9d\xf8\xcac\x1dZ}+Q\x13\xe2\
xdfkcJ\x94\x04\x81;(k\xc1#\x92\x05\x13\xf1\xd2\x8d,\xfe\x11\xf3\x04\x92\xc2\xf0n
u\xa2\xbc\xd1\x18n\x9a\x932\x826\x15(\x14^\x96\xdf\x86\x8aMo\xdb\xd0\xd5\x96\xb1
(\\\xc8\xcc\xc9\x07\xba\xdeE\x0b\x8c\x91X\x1e\x84\xc9\x90\xeb\x8ah\xb3F\x8aM\x96
\xa3\x16\x8a\x86\xeb\xc9\xa6Yd\xc7\xc4\xa0\xcc\x7f\xa3\xa1~\xf1J%\x04\xaa9\xe9fN
\xc3\x98\x7f\x86.k\x19\xac7\xc7\x853\x8dX\x1d\xc2B\x1fz\x9a\x1c\x8b9\x08I\xac\x9
b\x8a\x0b\xeaF\xfe?\x81\x87\xbc\xbe\x82*pU\x85*D.\xcb\x02\xd0\xad\x08\x84\xde\xb
8\xbf\x07@\xc6\xa2\x80\x8b(\x9e\xf7\xb6a\x94E#\xdbd\xc8\xa9\xaf_H\xe2\xcd\xfc\xd
0\x96\xc0\xe4\xeaf\x16d<\x80\x0f\xcf\xf1\xc5e\xea\xca_Mx\x96#\x15\x918\x8bo\xa5U
\xce\x91&\x95"\x9d\x99\xbb\xa3\xa2\x03k<t\x80\xfb.\x96\x94\xb8vb\x10\xc8>\xb5\x1
3hcy \xe8\x0c\xc5\xde\x83\xedI2\xb6\xb4P\xf9\xc5Y\xd8|d\xb42\x98\xcc\xc9*Ti\xb5\
xdb\x90h\xaf-\xe5\xd1\xd9\xd7\xf2\x9d8;=\x95\rq\x94J10\xe1\x96\xbb\xe8\x85\xc8\x
d8\xaf\x98n\xfa*[B\xfa\xf5%f\xc8\x94j\x82\xe7(\xb4\x17|\xaaX8.\xf8.\tN\x1d.\x83\
xcd\xf0\xfewPB\xf0\xd4\x0e\xd8\xb4\x93\xf0\x99\xb0\x0b\x94\xe0\xc2\xa4\x0f\xad\x
d5\x11\xb3\x02\x8c\xd3
\xf8"\x94`SK\x1bQ\xed\xca2=\xfc\xf37\xc7\x91\xbb\xef\x92\x1c\x05\xd8\x06h \xba\x
a3\xca%p\x1c\xbf\x8f-\xa6sR\n\xbd\x08\xceY\xd2\xb3W}RBd\x99`\x93.\x0c\xef\xa1\xe
2%\xee\xfaF\xa1\x11\xde\xb0\x15\xe5\xde\xe4\x10\x87:\xe4k\xb4\x92\x94\xe5\x9e\xf
8\x88\x88\\\xab\x9bAhZ\x82\xfd\xaaq\xef\xd7j$\xefiB\\\xf5\xc1\x01\xb4@3\xb9\x1b\
xd4\x84\xb1!\xd8p\x9a\xfd/\xb11\xf8\xd9\xa51h\xd0\tz\x82*\xb2\x85\xdf\x88\xe4B\x
b9\x04\xb4\x03\x95\xb1\xf8\xcf\x1cC\xbd\x85\xbd\xdf\x03\xf5w\xd1\x1b\x8e\xb3\x04
1\xcd\xa5\xa0U$l\xbbJ\x88\xf98\x8b\x9e\xa6\x8a\xa1\xc7I|U\xf2h\xe2\xccS\x87\x9a\
xff\xd7.f\xca\xef\xf8\xe2\x1c\xca\xcdyLD9\xea\xa7C\xc8\x7f\xfb8Z\xa1}\'\xeao\x94
]\xbe:\x8a\x01\xfa\x1e!q1\xf9\xdf\xd2\xf6U=`@\xcb\x8c\x19\xd7\xc3\x02\x12h\xd8\x
e1\xf9\xeal\xcc\xc6\x0e\xe5\xb5\xffc\x04\xfa3\t\xf4\x87\xc0x\x95Q3\x99E\xf1-|\xc
b\'\xeah\x1e\xf36\xf8e\xd0\xaa\xdc\x1eMN_\xb6\xed\x83\xa9e3-\xae\xe6-\xa4\x9d\xf
f\x00\x17\xce<\x94\x9f\t\x00\x00'
```


Shared Key =


```
b"\xe1x\xe5e1)\xc6\xef\xe4'\xc46\xe5\x8d1tq,\xe8\xaf5|T\xf5\\@W\xa5J\xe3\xdeC"
```