Problema 2

O problema para o qual se apresenta a seguinte solução, é o da criação de uma cifra a partir de um gerado pseudoaleatório de palavras de 64 bits. Para a concretização de tal solução foi necessária a criação de um gerador pseudoaleatório que faça uso do algoritmo SHAKE256, de modo a gerar uma sequência de palavras de 64 bits. Este output irá ser usado para cifrar uma mensagem com blocos de 64 bits, onde as palavras geradas pelo gerador pseudoaleatório servirão como máscaras XOR destes últimos. Por fim foi calculada a eficiência desta cifra comparativamente à cifra do Problema 1.

Imports

In [9]:
```
import time
import socket
import math
import sys
import os
from cryptography.hazmat.primitives.hashes import SHAKE256
from cryptography.hazmat.primitives import hashes, hmac
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.hazmat.primitives.ciphers import (
    Cipher, algorithms, modes
)
```

Gerador de palavras pseudoaleatórias de 64 bits

In [10]:
```
def generator(N, key):
    tamanho = 8 * pow(2, N) #Cálculo do número de bytes que queremos q
    palavras = list() #Onde se vão armazenar as palavras de 64 bits

    digest = hashes.Hash(hashes.SHAKE256(tamanho))
    digest.update(key)
    x = digest.finalize() #Resultado do SHAKE256

    chonk = len(x) // pow(2,N)
    i=0
    while i < pow(2,N): #Divide-se o valor da variavel x em blocos de
        palavras.append(x[:chonk])
        x=x[chonk:]
        i+=1
    return palavras
```

Função de cifragem

In [11]:
```
def cifra(msg):
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=b'salt',
        iterations=100000,
        backend=default_backend()
    )
    k = kdf.derive(b"the greatest password") #Derivação de uma chave a
    tam = math.ceil(math.sqrt(len(msg.encode('utf-8')))) # Cálculo do
    key=generator(tam, k) # guardamos em key o valor do generator
    chonk = len(key[0]) # guardamos aqui o tamanho da primeira palavra
                        # outras, excetuando eventualmente o da última
    i=0
    temp = '' # Resultado da cifragem
    while len(msg) > 0:
        if(len(msg) > chonk): # Para cada bloco de mensagem original,
            temp += cif_aux(msg[:chonk], key[i])
            msg=msg[chonk:]
        else:
            temp += cif_aux(msg, key[i])
            msg=b''
        i+=1
    return temp
```

Função de decifragem

In [12]:
```
def decifra(msg): #Análoga à função de cifragem, aplicando a operação
                  #calcula com recurso à função generator
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=b'salt',
        iterations=100000,
        backend=default_backend()
    )
    k = kdf.derive(b"the greatest password") #Derivação de uma chave a

    tam = math.ceil(math.sqrt(len(msg.encode('utf-8'))))
    key=generator(tam, k)
    chonk = len(key[0]) #8
    i=0
    temp = ''
    while len(msg) > 0:
        if(len(msg) > chonk):
            temp += cif_aux(msg[:chonk], key[i])
            msg=msg[chonk:]
        else:
            temp += cif_aux(msg, key[i])
            msg=''
        i+=1
    return temp
```

Função auxiliar à cifragem/decifragem

Função que itera duas strings, aplicando a operação de XOR em cada iteração entre as duas strings

In [13]:
```
def cif_aux(str1, str2):
    i = 0
    r = ''
    while i < len(str1):
        r += chr(ord(str1[i])^str2[i])
        i+=1
    return r
```

Casos de teste e medição do tempo que demora a executar

In [14]:
```
i=0
start = time.time()
while i < 1:
    a=cifra('Uma mensagem fantastica e grande')
    b=decifra(a)
    i+=1
end = time.time()
print(a)
print(b)
print(end - start)

i=0
start = time.time()
while i < 10:
    a=cifra('Uma mensagem fantastica e grande')
    b=decifra(a)
    i+=1
end = time.time()
#print(a)
#print(b)
print(end - start)

i=0
start = time.time()
while i < 50:
    a=cifra('Uma mensagem fantastica e grande')
    b=decifra(a)
    i+=1
end = time.time()
#print(a)
#print(b)
print(end - start)
```

```
k_□(□ÞYL^lØo□□□*ö»□k"w¨ØôD;[àú/à
Uma mensagem fantastica e grande
0.16455864906311035
1.509648084640503
7.7533347606658936
```

In [ ]:

In [ ]:

In [ ]: