Problema 1

A solução apresentada de seguida é referente ao problema da comunicação privada assíncrona entre um emissor e um recetor. Para a realização desta solução foi necessário implementar duas funções: emitter e receiver. A função emitter têm como objectivo o envio e cifragem da mensagem e autenticação dos respectivos metadados, bem como da criação da chave usada na cifra e a sua assinatura. A função receiver têm como objectivo a decifragem da mensagem e metadados enviados pelo emitter, bem como da autenticação destes metadados e verificação da chave usada pelo emitter.

Imports

In [1]:
```
import socket
import time
import os
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
)
```

Globals

In [2]:
```
########## Variáveis globais para comunicação assíncrona
salt = os.urandom(16)

iv = os.urandom(12)

ciphertext = b''

tag = b''

associated_data = b'data adicional'

sign = b''
#####################################
```

Cifragem

In [3]:
```
def emitter(plaintext):
    global salt
    global sign
    global tag
    global associated_data
    global ciphertext

    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=salt,
        iterations=100000,
        backend=default_backend()
    )

    key = kdf.derive(b"my great password") #Derivação de uma chave a pa

    h = hmac.HMAC(key, hashes.SHA256(), backend=default_backend())
    h.update(key)
    sign = h.finalize() #Assinatura da chave derivada


    encryptor = Cipher(
        algorithms.AES(key),
        modes.GCM(iv),
        backend=default_backend()
    ).encryptor()


    encryptor.authenticate_additional_data(associated_data)
    ciphertext = encryptor.update(plaintext) + encryptor.finalize() #ci
    tag = encryptor.tag
    return ciphertext
```

Decifragem

In [4]:
```
def receiver():
    global salt
    global ciphertext
    global iv
    global tag
    global associated_data
    global sign

    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=salt,
        iterations=100000,
        backend=default_backend()
    )

    key = kdf.derive(b"my great password") #Derivação de uma chave a pa



    ##### Verificação da assinatura da chave
    h = hmac.HMAC(key, hashes.SHA256(), backend=default_backend())
    h.update(key)
    h.verify(sign)
    ############################


    decryptor = Cipher(
        algorithms.AES(key),
        modes.GCM(iv, tag),
        backend=default_backend()
    ).decryptor()
    decryptor.authenticate_additional_data(associated_data)
    return decryptor.update(ciphertext) + decryptor.finalize() #decifra
```

Casos de teste do código e medição de tempos de execução

In [8]:
```
i=0
start = time.time()
while i < 1:
    a = emitter(b'Uma mensagem fantastica e grande')
    b = receiver()
    i+=1
end = time.time()
print(a)
print(b)
print(end - start)

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

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

```
b'\xc6\xc2\xe8+\x8aXI\xdf\xb4f\xc2U\xc91\x12%;\x89\xb4F\x0e\xdb\xc9E\x8
5x\xef\x10\x03\xb6X\x1b'
b'Uma mensagem fantastica e grande'
0.1565842628479004
1.5362012386322021
7.494379043579102
```

In [ ]:

In [ ]: