Trabalho Prático 1

Parte 1

Esta fase do trabalho consiste na criação de um esquema de cifragem e decifragem de mensagens, comunicadas de forma síncrona entre um emitter e um receiver. Assim, uma mensagem será cifrada no emitter, comunicada ao receiver e decifrada no destino. Para tal, desenvolveu-se código que permite que tal comunicação fosse possível, iniciando-se pela geração de um nounce aleatório para cada comunicação, desenvolvendo-se assim a função nounceGenerator, que devolve um nounce do mesmo tamanho que recebe como argumento. Seguidamente, define-se a função de cifragem, em que se utiliza a função anteriormente mencionada para gerar tanto um *iv* aleatório a utilizar na cifragem como um salt aleatório, a utilizar na derivação de uma chave a partir de uma password (que se encontra neste caso hardcoded no código da função). A chave é derivada utilizando o SHA-256. Seguidamente, a chave é utilizada para criar um objeto encryptor, que utiliza o algoritmo AES no modo GCM (e utiliza o *iv* aleatório previamente mencionado) para recorrer à cifragem da mensagem original, criando assim um ciphertext. Depois da cifragem, são enviados o ciphertext, o *iv*, a tag do encryptor, o nounce utilizado como salt, a assinatura da chave privada usada e a public key derivada da chave privada utilizada na assinatura da chave de cifragem, e serão esses os argumentos a serem utilizados na decifragem da mensagem. A assinatura da chave é feita a partir de uma chave privada, da qual se deriva uma chave pública que é enviada ao receiver, que irá com esta recorrer à verificação da assinatura da chave. Na função de decifragem, numa primeira instância deriva-se uma chave a utilizar no algoritmo AES a partir da mesma password utilizada na cifragem, verifica-se a nova chave com recurso à chave privada do esquema de assinaturas DSA e, sendo esta verificação confirmada, procede-se à decifragem do texto cifrado. Recorre-se à biblioteca multiprocessing para a transmissão dos dados. Assim são criadas dois processos p1 (processo pai) e p2 (processo filho), onde o p1 irá enviar através de um pipe a mensagem cifrada para o processo p2. Para isto ocorrer dá-se a criação de dois objectos da classe Process, que irão posteriormente ser inicializados com o método start(). Os resultados da cifragem são enviados sequencialmente pelo pipe.

In [ ]:
```
import multiprocessing
import sys
import os
import pickle
from cryptography.hazmat.primitives.serialization import load_der_public_key
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.ciphers import (Cipher, algorithms, modes)
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.hazmat.primitives import hashes, hmac
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.asymmetric import dsa
from cryptography.hazmat.primitives.asymmetric import ec



def is_in(val, lst):
    for a in lst:
        if a==val: return True
    return False

def nounceGenerator(tam):

    with open('./used.log', 'r') as fr:
        nounces_usados = fr.readlines()
        fr.close()

    #print(nounces_usados)

    r = os.urandom(tam)
    while(is_in(str(r), nounces_usados)):
        r = os.urandom(tam)

    with open('./used.log', 'a') as fw:
        fw.write(str(r))
        fw.write('\n')
        fw.close()

    return r


def cifragem(plaintext):
    iv = nounceGenerator(12)
    password = b'chave secreta'
    nounce = nounceGenerator(16)

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

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

    ciphertext = encryptor.update(plaintext) + encryptor.finalize()

    private_key = dsa.generate_private_key(key_size=1024, )

    sign = private_key.sign(ciphertext, hashes.SHA256())

    dsa_key = load_der_public_key(private_key.public_key().public_bytes(
        serialization.Encoding.DER, serialization.PublicFormat.SubjectPublicKeyInfo))

    pp = private_key.public_key().public_bytes(serialization.Encoding.DER,
                                               serialization.PublicFormat.SubjectPublicKeyInfo)

    return ciphertext, iv, encryptor.tag, nounce, sign, pp


def emitter(conn, m):
    (ct, iv, tag, n, sig, pp) = cifragem(m)
    print('ciphertext: ', ct)
    conn.send(ct)
    conn.send(iv)
    conn.send(tag)
    conn.send(n)
    conn.send(sig)
    conn.send(pp)
    conn.close()


def decifragem(ciphertext, iv, tag, nounce, sign, public_key):
    password = b'chave secreta'

    dsa_key = load_der_public_key(public_key, backend=default_backend())

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

    dsa_key.verify(
        sign,
        ciphertext,
        hashes.SHA256()
    )

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

    return decryptor.update(ciphertext) + decryptor.finalize()


def receiver(conn):
    while 1:
        ct = conn.recv()
        iv = conn.recv()
        tag = conn.recv()
        n = conn.recv()
        sig = conn.recv()
        pp = conn.recv()
        m = decifragem(ct, iv, tag, n, sig, pp)
        print(b"Mensagem decifrada: " + m)
        break


if __name__ == '__main__':
    parent_conn, child_conn = multiprocessing.Pipe()

    while True:

        print("Write a message!!!")

        msg = msg = bytes(input(), 'utf-8')

        p1 = multiprocessing.Process(target=emitter, args=(parent_conn, msg))
        p2 = multiprocessing.Process(target=receiver, args=(child_conn,))

        p1.start()
        p2.start()

        p1.join()
        p2.join()
```

Trabalho Prático 1

Parte 1 - d)

Nesta fase o trabalho realizado foi semelhante ao apresentado anteriormento, tendo apenas sido substituido o uso de dsa por ecdsa.

In [ ]:
```
import multiprocessing
import sys
import os
import pickle
from cryptography.hazmat.primitives.serialization import load_der_public_key
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.ciphers import (Cipher, algorithms, modes)
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.hazmat.primitives import hashes, hmac
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.asymmetric import dsa
from cryptography.hazmat.primitives.asymmetric import ec


def is_in(val, lst):
    for a in lst:
        if a==val: return True
    return False

def nounceGenerator(tam):

    with open('./used2.log', 'r') as fr:
        nounces_usados = fr.readlines()
        fr.close()

    #print(nounces_usados)

    r = os.urandom(tam)
    while(is_in(str(r), nounces_usados)):
        r = os.urandom(tam)

    with open('./used2.log', 'a') as fw:
        fw.write(str(r))
        fw.write('\n')
        fw.close()

    return r


def cifragem2(plaintext):
    iv = nounceGenerator(12)
    password = b'chave secreta'
    nounce = nounceGenerator(16)

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

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

    ciphertext = encryptor.update(plaintext) + encryptor.finalize()

    private_key = ec.generate_private_key(
    ec.SECP384R1()
    )

    sign = private_key.sign(
    ciphertext,
    ec.ECDSA(hashes.SHA256())
    )

    return (ciphertext, iv, encryptor.tag, nounce, sign, private_key.public_key().public_bytes(
        serialization.Encoding.DER, serialization.PublicFormat.SubjectPublicKeyInfo))


def emitter(conn, m):
    (ct, iv, tag, n, sig, pp) = cifragem2(m)
    print('ciphertext: ', ct)
    conn.send(ct)
    conn.send(iv)
    conn.send(tag)
    conn.send(n)
    conn.send(sig)
    conn.send(pp)
    conn.close()


def decifragem2(ciphertext, iv, tag, nounce, sign, public_key):
    password = b'chave secreta'

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


    load_der_public_key(public_key,).verify(sign, ciphertext, ec.ECDSA(hashes.SHA256()))



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

    return decryptor.update(ciphertext) + decryptor.finalize()


def receiver(conn):
    while 1:
        ct = conn.recv()
        iv = conn.recv()
        tag = conn.recv()
        n = conn.recv()
        sig = conn.recv()
        pp = conn.recv()
        m = decifragem2(ct, iv, tag, n, sig, pp)
        print(b"Mensagem decifrada: " + m)
        break


if __name__ == '__main__':
    parent_conn, child_conn = multiprocessing.Pipe()

    while True:

        print("Write a message!!!")

        msg = msg = bytes(input(), 'utf-8')

        p1 = multiprocessing.Process(target=emitter, args=(parent_conn, msg))
        p2 = multiprocessing.Process(target=receiver, args=(child_conn,))

        p1.start()
        p2.start()

        p1.join()
        p2.join()
```