TP1-1

March 7, 2023

1 Trabalho Prático 1

Trabalho realizado pelo grupo 11:

- Beatriz Fernandes Oliveira, PG50942
- Bruno Filipe Machado Jardim, PG49997

1.0.1 Exercício 1

Use o package Criptography para:

1. Criar um comunicação privada assíncrona entre um agente Emitter e um agente Receiver que cubra os seguintes aspectos:
   1. Autenticação do criptograma e dos metadados (associated data). Usar uma cifra simétrica num modo HMAC que seja seguro contra ataques aos "nounces".
   2. Os "nounces" são gerados por um gerador pseudo aleatório (PRG) construído por um função de hash em modo XOF.
   3. O par de chaves `cipher_key`, `mac_key` , para cifra e autenticação, é acordado entre agentes usando o protocolo ECDH com autenticação dos agentes usando assinaturas ECDSA.

[1]:
```
import os
import cryptography
from cryptography.hazmat.primitives.ciphers.aead import AESGCM
```

Função de geração pseudo aleatório com base na função de hash **SHAKE256** em modo *XOF*

[2]:
```
def PRG(seed,size):
    dgst = hashes.Hash(hashes.SHAKE256(2**size * 8))
    dgst.update(seed)
    nounceString = dgst.finalize()
    return [nounceString[i:i+8] for i in range(0,len(nounceString),8)]
```

Funções de cifração e decifração com o método **AES** a utilizar o modo *Galois Counter Mode*

[3]:
```
def cipher(key,nounce,message,metadata):
    aesgcm = AESGCM(key)
    ct = aesgcm.encrypt(nounce, message, metadata)
    return ct
```

```
def decipher(key,nounce,ct,aad):
    aesgcm = AESGCM(key)
    plaintext = aesgcm.decrypt(nounce, ct, aad)
    return plaintext
```

Função que verifica a autenticidade da mensagem utilizando **HMAC** *(Hash-based message authentication codes)*

[4]:
```
from cryptography.hazmat.primitives import hashes, hmac
def HMAC(key,metadata):
    h = hmac.HMAC(key, hashes.SHA256())
    h.update(metadata)
    signature = h.finalize()
    return signature

def HMACVerify(key,plaintext,tag):
    h = hmac.HMAC(key, hashes.SHA256())
    h.update(plaintext)
    h.verify(tag)
    print("[Receiver]: Message Authenticated")
```

[5]:
```
from cryptography.hazmat.primitives.asymmetric import dh
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.kdf.hkdf import HKDF
from cryptography.hazmat.primitives.serialization import load_pem_public_key
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import ec
parameters = dh.generate_parameters(generator=2, key_size=2048)
def DHKeyGen():
    private_key = ec.generate_private_key(ec.SECP384R1())
    public_key = private_key.public_key().public_bytes(encoding=serialization.
 ↪Encoding.PEM,
                                                          format=serialization.
 ↪PublicFormat.SubjectPublicKeyInfo)
    return (private_key,public_key)

def derive_key(private_key,public_key):
    #print("Started")
    shared_key = private_key.exchange(ec.ECDH(), public_key)
    derived_key = HKDF(
        algorithm=hashes.SHA256(),
        length=32,
        salt=None,
        info=b'EC',
    ).derive(shared_key)
    return derived_key
```

```
def keyVerification(pbk,signature):
    pbkEC = load_pem_public_key(pbk)
    try:
        pbkEC.verify(signature,pbk,ec.ECDSA(hashes.SHA256()))
        print("Signature validated")
    except Exception as e:
        print("Invalid signature: ",e)
```

[6]:

```
async def connectionProtocolEm(qCe,qMe,qCr,qMr):

    pvkCipher,pbkCipher = DHKeyGen()
    signatureC = pvkCipher.sign(pbkCipher, ec.ECDSA(hashes.SHA256()))

    await qCe.put((pbkCipher,signatureC))

    pvkMac,pbkMac = DHKeyGen()
    signatureM = pvkMac.sign(pbkMac, ec.ECDSA(hashes.SHA256()))

    await qMe.put((pbkMac,signatureM))
    #await asyncio.sleep(1)
    pbkC,sC = await qCr.get()
    pbkM,sM = await qMr.get()

    print("[Emitter]: Cipher key validation")
    keyVerification(pbkC,sC)
    print("[Emitter]: HMAC key validation")
    keyVerification(pbkM,sM)

    #print(pbkCipher,pbkC)

    pbkC = load_pem_public_key(pbkC)
    pbkM = load_pem_public_key(pbkM)



    derivedCipher = derive_key(pvkCipher,pbkC)

    derivedMac = derive_key(pvkMac,pbkM)
    return (derivedCipher,derivedMac)
async def connectionProtocolRe(qCe,qMe,qCr,qMr):

    pvkCipher,pbkCipher = DHKeyGen()
    signatureC = pvkCipher.sign(pbkCipher, ec.ECDSA(hashes.SHA256()))

    await qCr.put((pbkCipher,signatureC))
    pvkMac,pbkMac = DHKeyGen()
    signatureM = pvkMac.sign(pbkMac, ec.ECDSA(hashes.SHA256()))

    await qMr.put((pbkMac,signatureM))
    #await asyncio.sleep(1)
    pbkC,sC = await qCe.get()
    pbkM,sM = await qMe.get()
    print("[Receiver]: Cipher key validation")
    keyVerification(pbkC,sC)
    print("[Receiver]: HMAC key validation")
    keyVerification(pbkM,sM)

    #print(pbkCipher,pbkC)

    pbkC = load_pem_public_key(pbkC)
    pbkM = load_pem_public_key(pbkM)



    derivedCipher = derive_key(pvkCipher,pbkC)

    derivedMac = derive_key(pvkMac,pbkM)
    return (derivedCipher,derivedMac)
```

[9]:

```
import asyncio
import random
import nest_asyncio
nest_asyncio.apply()

async def Emitter(queue,qCe,qMe,qCr,qMr):
    derivations = asyncio.create_task(connectionProtocolEm(qCe,qMe,qCr,qMr))
    kC,kM = await derivations
    msg_list = [
        "As armas e os barões assinalados,",
        "Que da ocidental praia Lusitana,",
        "Por mares nunca de antes navegados,",
        "Passaram ainda além da Taprobana,",
        "Em perigos e guerras esforçados,",
        "Mais do que prometia a força humana,",
        "E entre gente remota edificaram",
        "Novo Reino, que tanto sublimaram;"
    ]
    nounceList = PRG(kC,10)
    for i in range(len(msg_list)):
        msg = msg_list[i].encode('utf-8')
        aad = HMAC(kM,msg)
        ct = cipher(kC,nounceList[i],msg,aad)
        await asyncio.sleep(random.random())
        await queue.put((ct,aad))
        print(f"[Emitter]: Sent message")
    await queue.put(None)


async def Receiver(queue,qCe,qMe,qCr,qMr):
    derivations = asyncio.create_task(connectionProtocolRe(qCe,qMe,qCr,qMr))

    kC,kM = await derivations
    #print(kC)
    nounceList = PRG(kC,10)
    i = 0
    while True:
        item = await queue.get()
        #print(item)
        if item is None:
            break
        ct, aad = item
        pt = decipher(kC,nounceList[i],ct, aad)
        print("[Receiver]: Received -> ", pt.decode('utf-8'))
        HMACVerify(kM,pt,aad)
        queue.task_done()
        i+=1
    print("[Receiver]: End of messages")


async def main():

    queue = asyncio.Queue()
    qCe = asyncio.Queue()
    qMe = asyncio.Queue()
    qCr = asyncio.Queue()
    qMr = asyncio.Queue()

    emitter = asyncio.create_task(Emitter(queue,qCe,qMe,qCr,qMr))
    receiver = asyncio.create_task(Receiver(queue,qCe,qMe,qCr,qMr))

asyncio.run(main())
```

```
[Receiver]: Cipher key validation
Signature validated
[Receiver]: HMAC key validation
Signature validated
[Emitter]: Cipher key validation
Signature validated
[Emitter]: HMAC key validation
```

```
Signature validated
[Emitter]: Sent message
[Receiver]: Received ->  As armas e os barões assinalados,
[Receiver]: Message Authenticated
[Emitter]: Sent message
[Receiver]: Received ->  Que da ocidental praia Lusitana,
[Receiver]: Message Authenticated
[Emitter]: Sent message
[Receiver]: Received ->  Por mares nunca de antes navegados,
[Receiver]: Message Authenticated
[Emitter]: Sent message
[Receiver]: Received ->  Passaram ainda além da Taprobana,
[Receiver]: Message Authenticated
[Emitter]: Sent message
[Receiver]: Received ->  Em perigos e guerras esforçados,
[Receiver]: Message Authenticated
[Emitter]: Sent message
[Receiver]: Received ->  Mais do que prometia a força humana,
[Receiver]: Message Authenticated
[Emitter]: Sent message
[Receiver]: Received ->  E entre gente remota edificaram
[Receiver]: Message Authenticated
[Emitter]: Sent message
[Receiver]: Received ->  Novo Reino, que tanto sublimaram;
[Receiver]: Message Authenticated
[Receiver]: End of messages
```