TP1 - Exercício 1

February 27, 2024

Nota: Devido ao módulo **asyncio**, o código disponibilizado deve ser executado através de ficheiros diferentes (para o *emitter* e para o *receiver*) e não no presente *notebook*.

1 Estruturas Criptográficas

1.1 Trabalho Prático 1 - Exercício 1

1.1.1 José de Matos Moreira - PG53963

1.1.2 Pedro Freitas - PG52700

1.2 Enunciado do problema

Use o *package* **Cryptography** e o *package* **ascon** para criar uma comunicação privada assíncrona em modo **"Lightweight Cryptography"** entre um agente *Emitter* e um agente *Receiver* que cubra os seguintes aspetos: * autenticação do criptograma e dos metadados (*associated data*) usando **ascon** em modo de cifra * as chaves de cifra, autenticação e os *"nounces"* são gerados por um gerador pseudo aleatório (PRG) usando o **ascon** em modo **XOF**. As diferentes chaves para inicialização do PRG são *inputs* do emissor e do receptor * para implementar a comunicação cliente-servidor use o *package* python **asyncio**

1.3 Resolução

Em primeiro lugar, procedeu-se ao *import* dos módulos necessários, em ambos os ficheiros *python* utilizados:

[ ]:
```
import asyncio
import ascon
```

1.3.1 Receiver

Inicialmente, começou-se pela escrita da função principal do *receiver*. A **receive_message** é responsável pelo corpo mais importante do programa. Deste modo, a mesma assegura o seguinte: * leitura dos dados recebidos * pedido de uma *seed* para geração da chave do **MAC**; usando o **ascon** em modo **XOF**, geração do mesmo e comparação com o **MAC** presente nos dados obtidos * em caso de sucesso no processo de autenticação, pedido das *seeds* para criação da chave de cifra e do *nounce*, caso contrário, impressão do erro obtido * geração dos parâmetros anteriormente referidos, através do **ascon**, em modo **XOF** * tentativa de decifragem dos dados recebidos, relativos ao *plaintext* * em caso de sucesso na decifragem, *print* do conteúdo obtido e, em caso de erro, apresentação do mesmo

```
[ ]: async def receive_message(reader, _):
         data = await reader.read()

         ciphertext = data[:-16]
         mac = data[-16:]

         mac_seed = input("[receiver] Type a seed to generate the mac: ")
         mac_key = ascon.hash(mac_seed.encode(), variant='Ascon-Xof', hashlength=16)

         r_mac = ascon.mac(mac_key, ciphertext, variant="Ascon-Mac", taglength=16)
         if mac != r_mac:
             print('[receiver] Error in authentication!')

         else:
             try:
                 key_seed = input("[receiver] Type a seed to generate the key: ")
                 nounce_seed = input("[receiver] Type a seed to generate the nonce:␣
      ↪")

                 key = ascon.hash(key_seed.encode(), variant='Ascon-Xof',␣
      ↪hashlength=16)
                 nounce = ascon.hash(nounce_seed.encode(), variant='Ascon-Xof',␣
      ↪hashlength=16)

                 plaintext = ascon.decrypt(key, nounce, 'ec2324'.encode(),␣
      ↪ciphertext, variant="Ascon-128").decode()

                 print("[receiver] Message received: " + plaintext)

             except Exception:
                 print("[receiver] Error in decryption!")
```

Deste modo, apresenta-se a primeira função a ser chamada, a **main**. Assim, a mesma procede à criação de um servidor assíncrono simples, que escuta no *localhost*, na porta 8888, chamando a função **receive_message** para lidar com as várias conexões recebidas. Reitera-se que a função **main** é executada continuamente durante o tempo de vida do processo.

```
[ ]: async def main():
         server = await asyncio.start_server(receive_message, 'localhost', 8888)

         async with server:
             await server.serve_forever()
```

Esta mesma função apresentada é chamada pelo módulo **asyncio**, recorrendo ao **run**.

```
[ ]: asyncio.run(main())
```

1.3.2 Emitter

O *emitter* possui uma função que produz a maior parte do seu trabalho, a **send_message**. Assim e, à semelhança do que foi feito anteriormente, apresenta-se as várias funcionalidades implementadas na mesma: * pedido de *input* da mensagem a ser cifrada e, posteriormente, enviada * pedido de *input* das *seeds* de geração da chave de cifra, do *nounce* e da chave do **MAC** * criação dos três parâmetros referidos, através das *seeds*, recorrendo ao módulo **ascon**, em modo **XOF** * cifragem da mensagem inserida * geração do **MAC** * retorno do resultado composto por *ciphertext* + **MAC**

[ ]:
```
async def send_message():
    plaintext = input("[emitter] Type the message: ")

    key_seed = input("[emitter] Type a seed to generate the key: ")
    mac_seed = input("[emitter] Type a seed to generate the mac: ")
    nounce_seed = input("[emitter] Type a seed to generate the nounce: ")

    key = ascon.hash(key_seed.encode(), variant='Ascon-Xof', hashlength=16)
    mac_key = ascon.hash(mac_seed.encode(), variant='Ascon-Xof', hashlength=16)
    nounce = ascon.hash(nounce_seed.encode(), variant='Ascon-Xof',␣
↪hashlength=16)

    ciphertext = ascon.encrypt(key, nounce, 'ec2324'.encode(), plaintext.
↪encode(), variant="Ascon-128")
    mac = ascon.mac(mac_key, ciphertext, variant="Ascon-Mac", taglength=16)

    return ciphertext + mac
```

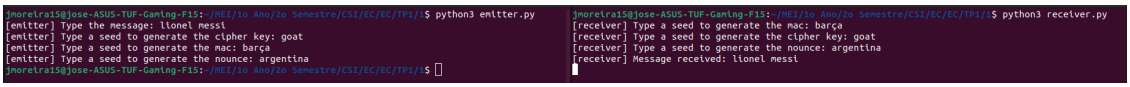
À semelhança do que acontece no *receiver*, existe uma função principal, a **main**, que é responsável por fazer a chamada à função anteriormente apresentada. Porém, a **main** possui outras funcionalidades, sendo as mesmas: * conexão assíncrona ao *localhost*, na porta 8888 * chamada da função **send_message**, guardando o resultado da mesma numa variável * envio do conteúdo obtido, recorrendo ao **write** e ao **drain**

[ ]:
```
async def main():
    _, writer = await asyncio.open_connection('localhost', 8888)

    data = await send_message()

    writer.write(data)
    await writer.drain()

    writer.close()
    await writer.wait_closed()
```

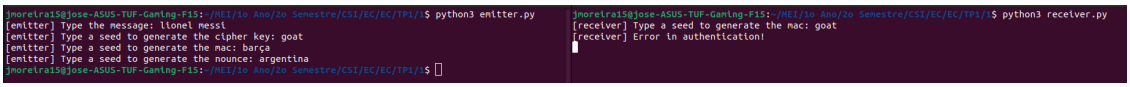
Analogamente, esta mesma função é chamada pelo **run**, do **asyncio**.

[ ]:
```
asyncio.run(main())
```

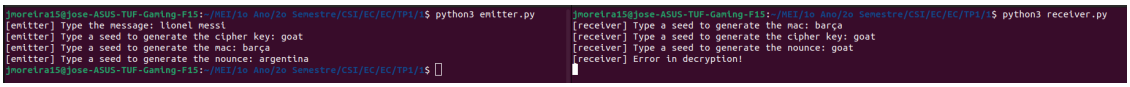
1.3.3 Testes de aplicação



Correta execução de ambos os agentes, ou seja, possuindo as mesmas *seeds* usadas na criação da chave de cifra, do *nounce* e do MAC



Introdução errada da *seed* usada para geração do MAC (pelo receiver)



Introdução correta da *seed* usada para geração do MAC, mas introdução errada de pelo menos uma *seed* de geração dos parâmetros de decifragem (chave de cifra ou *nounce*, pelo receiver)