Estruturas criptograficas: TP1 problema 1

Descrição do problema

1. Neste problema foi nos proposto o uso do package Cryptography e do package ascon para criar uma comunicação privada assíncrona em modo “Lightweight Cryptography” entre um agente Emitter e um agente Receiver que cubra os seguintes aspectos:

    a. Autenticação do criptograma e dos metadados (associated data) usando Ascon em modo de cifra.
    b. 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.
    c. Para implementar a comunicação cliente-servidor use o package python `asyncio`.

Implementação

`Cliente/Servidor`

A abordagem utilizada começou por criar a conexão assíncrona entre cliente(Emitter) e servidor(Receiver) com o package asyncio.

Quando criamos um receiver é iniciado o servidor em background que chama a função handle_emitter() assim que for iniciada uma conexão.

Quando iniciamos um emitter é criada uma ligação ao servidor e podemos enviar as mensagens que quisermos.

`send_message()`

Recebe como input os inicializadores do nonce e da chave

Depois de escrevermos a mensagem:

1. São criados os dados associados (comprimento da mensagem)
2. Geradas a chave e o nounce com os inicializadores passados como argumento usando Ascon em modo XOF
3. Cifrado o texto usando Ascon em modo cifra com a devida chave, nonce e dados associados
4. Anexados os dados associados ao texto cifrado e enviada a mensagem

.

```
handle_emitter()
```

Recebe como input os inicializadores do nonce e da chave

Depois de escrevermos a mensagem:

1. Lê os dados assiciados (tamanho da mensagem) com recurso a uma flag
2. Lê a mensagem usando o tamanho obtido nos dados associados
3. Geradas a chave e o nounce com os inicializadores passados como argumento usando Ascon em modo XOF
4. Decifrado o texto usando Ascon em modo cifra com a devida chave, nonce e dados associados
5. Verifica se a mensagem foi comprometida

```
import asyncio
import ascon

async def handle_emitter(key_init_receiver, nonce_init_receiver,
reader, writer):

    while True:

        # Lê os dados associados
        try:
            associateddata = await reader.readuntil(b'_ADEnd_')
        except:
            break

        size = int(associateddata.decode()[:-7])

        #Lê a mensagem
        ciphertext = await reader.readexactly(size)

        #gerar chave e nonce
        key = ascon.hash(key_init_receiver.encode(), variant="Ascon-
Xof", hashlength=16)
        nonce = ascon.hash(nonce_init_receiver.encode(),
variant="Ascon-Xof", hashlength=16)

        #decifrar a mensagem
        plaintext = ascon.decrypt(key, nonce, associateddata,
ciphertext, variant="Ascon-128")

        #verifica se a mensagem foi comprometida
```

```
        if plaintext is None:
            print("Autentication failed. Message compromised.")

        else:
            print(f"Message: {plaintext.decode()}")

            #incrementa o nonce
            nonce_init_receiver = str(int(nonce_init_receiver) + 1)

#criar receiver
async def receiver(key_init_receiver, nonce_init_receiver):
    server = await asyncio.start_server(
        lambda reader, writer: handle_emitter(
            key_init_receiver, nonce_init_receiver, reader, writer),
'127.0.0.1', 7777)

    print("Receiver ready...\n")

    async with server:
        await server.serve_forever()


#Enviar mensagem
async def send_message( key_init_emitter, nonce_init_emitter, reader,
writer):

    #input da mensagem
    plaintext = input("Enter message: ").encode()

    #se nao mandou mais mensagens fecha a conexão
    if not plaintext:
        return 0

    #dados associados = comprimento do plaintext
    associateddata = (str(len(plaintext)+16) + '_ADEnd_').encode()

    #gerar chave e nonce aliatócom Ascom em modeo XOF
    key = ascon.hash(key_init_emitter.encode(), variant="Ascon-Xof",
hashlength=16)
    nonce = ascon.hash(nonce_init_emitter.encode(), variant="Ascon-
Xof", hashlength=16)

    #cifrar texto
    ciphertext = ascon.encrypt(
        key, nonce, associateddata, plaintext, variant="Ascon-128")

    #anexar dados associados
    message = associateddata + ciphertext

    # Envia a mensagem para o servidor
```

```
    writer.write(message)
    await writer.drain()

    return 1

async def emitter(key_init_emitter, nonce_init_emitter):

        reader, writer = await asyncio.open_connection('127.0.0.1',
7777)

        while True:

            # Envia a mensagem
            result = await send_message( key_init_emitter,
nonce_init_emitter, reader, writer)
            nonce_init_emitter = str(int(nonce_init_emitter) + 1)

            if result == 0:
                break

        return reader, writer
```

Cria o receiver

```
# criar o Receiver

key_init_receive = "27082001" #chave
nonce_init_receive = "12071972" #nonce

asyncio.create_task(receiver(key_init_receive, nonce_init_receive))  #
Inicia o servidor
```

```
<Task pending name='Task-5' coro=<receiver() running at C:\Users\
Utilizador\AppData\Local\Temp\ipykernel_24248\977531213.py:37>>
```

```
Receiver ready...
```

Caso certo

```
#Emitter, caso certo

key_init_emitter = "27082001"
nonce_init_emitter = "12071972"

reader, writer = await emitter(key_init_emitter, nonce_init_emitter)
```

```
Message: Ola, como estas?
Message: teste oaldjk jdjshdhfnsn sjdjfuuej jsjdfkk sjfiksefmsifs n
snfsif s teste
```

Key errada

```
#Emitter, key errada

key_init_emitter = "27082001"
nonce_init_emitter = "12071973"

# Envia a mensagem
reader, writer = await emitter(key_init_emitter, nonce_init_emitter)

Autentication failed. Message compromised.
```

Nonce errado

```
#Emitter, nonce errado

key_init_emitter = "270801"
nonce_init_emitter = "12071973"

# Envia a mensagem
reader, writer = await emitter(key_init_emitter, nonce_init_emitter)

Autentication failed. Message compromised.
```