TP1 - 1

Objetivo

Criar uma comunicação privada assíncrona e autenticada entre um agente *Emitter* e um Agente *Receiver* usando a família de algoritmos *Ascon*

1. Autenticação do criptograma e dos metadados usando *Ascon* em modo cifra
2. Geração de chaves de cifra, chaves de autenticação e nonces usando *Ascon* em modo XOF. Emissor e receptor recebem como input chave de inicialização para o gerador.
3. Usar o *package* **asyncio** para implementar a comunicação cliente-servidor

Abordagem

Criamos uma função emitter, que envia mensagens cifradas, e uma classe Receiver que as recebe e imprime.

Ambos recebem como argumento uma *seed* usada pelo gerador para gerar as chaves e os nonces, garantido que iniciam com os mesmos dados. Essa seed é também concatenada com um contador para gerar nonces diferentes para cada mensagem a ser cifrada/mensagem recebida, garantindo também que ambos os agentes geram os mesmos nonces.

Implementação

Variáveis

1. **seed:** valor usado para gerar chaves e nonces
2. **message:** texto limpo a ser cifrado
3. **loop_forever:** define se os agentes continuam a interagir

In [ ]:
```
import ascon
import asyncio

seed = b'yep'
message = b'yo'
loop_forever = True
```

Receiver

Define e inicia receiver de forma assincrona

In [ ]:
```
class Receiver:
    key: bytes
    nonce: bytes
    associated_data: bytes
    cntr: int
    seed: bytes
    message: bytes
    loop_forever: bool
    server: asyncio.Server

    def __init__(self, seed, message, loop_forever) -> None:
        self.cntr = 0
        self.key = ascon.hash(seed+bytes(self.cntr), 'Ascon-Xof', 16)
        self.associated_data = b''
        self.seed = seed
        self.message = message
        self.loop_forever = loop_forever
```

```
    async def handle_connection(self, reader: asyncio.StreamReader, writer: asyncio.StreamWriter)
        self.cntr+=1
        print('receiver: counter:', self.cntr)

        self.nonce = ascon.hash(self.seed+bytes(self.cntr), 'Ascon-Xof', 16)

        data = await reader.read(-1)
        dc = ascon.decrypt(self.key, self.nonce, self.associated_data, data, 'Ascon-128')

        print(f'receiver: received and decrypted: {dc}\n')

        writer.close()
        await writer.wait_closed()

        if not self.loop_forever:
            self.server.close()

    async def start_server(self):
        self.server = await asyncio.start_server(self.handle_connection, '127.0.0.1', 8098)
        print('receiver: started')
        async with self.server:
            await self.server.serve_forever()

server = Receiver(seed, message, loop_forever)

server_task = asyncio.get_running_loop().create_task(server.start_server())
```

```
receiver: started
receiver: counter: 1
receiver: received and decrypted: b'yo'

receiver: counter: 2
receiver: received and decrypted: b'yo'

receiver: counter: 3
receiver: received and decrypted: b'yo'

receiver: counter: 4
receiver: received and decrypted: b'yo'

receiver: counter: 5
receiver: received and decrypted: b'yo'
```

Emitter

Define e inicia emitter de forma assincrona

In [ ]:
```
async def emitter(seed, message, loop_forever):
    cntr = 0

    r_bytes = ascon.hash(seed+bytes(cntr), 'Ascon-Xof', 32)
    cntr += 1

    key = r_bytes[0:16]
    nonce = r_bytes[16:32]
    associated_data = b''

    while True:
        reader, writer = await asyncio.open_connection('127.0.0.1', 8098)

        print('emitter: counter: ', cntr)
        nonce = ascon.hash(seed+bytes(cntr), 'Ascon-Xof', 16)
        cntr += 1

        crypt = ascon.encrypt(key, nonce, associated_data, message, 'Ascon-128')
        writer.write(crypt)
        await writer.drain()
```

```
        print(f'emitter: sent enc: {crypt}\n')

        writer.close()
        await writer.wait_closed()

        if not loop_forever:
            break

        await asyncio.sleep(2)

client_task = asyncio.get_running_loop().create_task(emitter(seed,message,loop_forever))
```

```
emitter: counter:  1
emitter: sent enc: b'\xba\xf8D\x10T~.i\x0eN|\t\x8b\x15\x06Y\xc9\x00'

emitter: counter:  2
emitter: sent enc: b"o\x035a\x0e\xa7\xdd?\xf2\xe4T\xd58\xc6.\xc9'-"

emitter: counter:  3
emitter: sent enc: b'\x80!\x9dj=\xc8\xf3b\xe3\xbc~d\xeb\x1b\xbd\xaa\x8dw'

emitter: counter:  4
emitter: sent enc: b'\xd1\x11\xe8\xe5\x9d8\xb4F!^_\x12<\x05\xe1Oo\x9b'

emitter: counter:  5
emitter: sent enc: b'\xee\x81%\x07\x89\xe1\x90\xa1\xd5\xc6\x1f\x8b\x9f\xf6x\xbc\xe6\xc2'
```

Paragem do loop

Usado para parar emitter e receiver, no case de **loop_forever** ser *True*

In [ ]:
```
client_task.cancel()
server_task.cancel()
```

Out[ ]:
```
True
```