Trabalho Prático 1

André Freitas PG54707

Bruna Macieira PG54467

Exercício 1.

Use a package Cryptography e o package ascon (instalar daqui) para criar um comunicação privada assíncrona em modo "Lightweight Cryptography" entre um agente Emitter e um agente Receiver que cubra os seguintes aspectos:

1. Autenticação do criptograma e dos metadados (associated data) usando Ascon (ver implementação aqui) em modo de cifra.
2. 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.
3. Para implementar a comunicação cliente-servidor use o package python `asyncio`.

Usámos o pacote Ascon para autenticar o criptograma envolvido na comunicação bem como os metadados envolventes. O Ascon é uma família de algoritmos de encriptação autenticada e de hashing concebidos para serem leves e fáceis de implementar, mesmo com contramedidas adicionais contra ataques de canais laterais. Foi concebido por uma equipa de criptógrafos da Universidade de Tecnologia de Graz, da Infineon Technologies e da Universidade de Radboud: Christoph Dobraunig, Maria Eichlseder, Florian Mendel e Martin Schläffer.

O NIST anunciou a seleção da família Ascon como o padrão de criptografia leve em fevereiro de 2023, após receber feedback público num workshop. O NIST está a trabalhar com a equipa Ascon para elaborar os padrões de criptografia leve.

Asyncio é uma biblioteca para escrever código concorrente usando a sintaxe async/await, por isso tirámos proveito dessa funcionalidade para poder implementar a comunicação cliente-servidor.

```
import asyncio
import os
import ascon
from cryptography.hazmat.primitives.asymmetric import x25519
from cryptography.hazmat.primitives import serialization, hashes
from cryptography.hazmat.primitives.kdf.hkdf import HKDF
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms,
modes
from secrets import token_bytes
```

A classe AsconCipher é uma classe que encapsula operações de criptografia e descriptografia Ascon. Tem como métodos:

1. encrypt: Usa o algoritmo Ascon para criptografar o texto simples.
2. decrypt: Usa o algoritmo Ascon para descriptografar o texto cifrado.

Utilizam o Ascon-128.

```
# Define a class named AsconCipher
class AsconCipher:
    def __init__(self, key):
        self.key = key

    # Method to encrypt the plaintext using Ascon algorithm
    def encrypt(self, nonce, plaintext, associated_data):
        return ascon.ascon_encrypt(self.key, nonce, associated_data,
plaintext, variant='Ascon-128')

    # Method to decrypt the ciphertext using Ascon algorithm
    def decrypt(self, nonce, ciphertext, associated_data):
        return ascon.ascon_decrypt(self.key, nonce, associated_data,
ciphertext, variant='Ascon-128')
```

A função handle_client é uma função assíncrona que lida com a comunicação com um cliente. Usa X25519 (algoritmo de Diffie-Hellman) para troca de chaves, deriva chaves com HKDF (baseado em HMAC) e realiza criptografia/descriptografia com Ascon. A comunicação envolve a troca de chaves públicas, o estabelecimento de um segredo partilhado e a criptografia/descriptografia de mensagens.

```
# This function handles the communication with a client
async def handle_client(reader, writer):
    # Generate a private key for the server
    private_key = x25519.X25519PrivateKey.generate()
    # Get the corresponding public key and convert it to bytes
    public_key =
private_key.public_key().public_bytes(serialization.Encoding.Raw,
serialization.PublicFormat.Raw)

    # Send the public key to the client
    writer.write(public_key)
    await writer.drain()

    # Receive the public key from the client
    peer_public_key_bytes = await reader.read(32)
    peer_public_key =
x25519.X25519PublicKey.from_public_bytes(peer_public_key_bytes)

    # Perform key exchange to derive a shared key
    shared_key = private_key.exchange(peer_public_key)

    # Derive key material using HKDF
    derived_key_material = HKDF(
        algorithm=hashes.SHA256(),
```

```
        length=64,
        salt=None,
        info=b'handshake data',
        backend=default_backend()
    ).derive(shared_key)

    # Extract the cipher key from the derived key material
    cipher_key = derived_key_material[:32]

    # Create an instance of the AsconCipher class with the cipher key
    cipher = AsconCipher(cipher_key)

    # Continuously receive and decrypt messages from the client
    while True:
        # Read the nonce, ciphertext, tag, and associated data from
the client
        nonce = await reader.readexactly(16)
        ciphertext = await reader.readuntil(separator=b'|')
        tag = await reader.readexactly(16)
        associated_data = await reader.readuntil(separator=b'||')

        # Decrypt the ciphertext using the nonce, tag, and associated
data
        plaintext = cipher.decrypt(nonce, ciphertext, tag,
associated_data)

        # Send the decrypted plaintext back to the client
        writer.write(plaintext)
        await writer.drain()
```

A função main é a função principal e configura e executa o servidor.

```
# This async function is the main entry point for the server
# It creates a server using asyncio.start_server() and specifies the
handle_client function to handle client connections
# The server listens on '127.0.0.1' IP address and port 8888
# It prints the address on which the server is serving
# The server is started and runs indefinitely using
server.serve_forever()
async def main():
    server = await asyncio.start_server(
        handle_client, '127.0.0.1', 8888)

    addr = server.sockets[0].getsockname()
    print(f'Serving on {addr}')

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

A função send_public_key é a função responsável por enviar chaves públicas do cliente para o servidor.

É gerada uma chave privada X25519 para o cliente. É obtida a chave pública correspondente à chave privada gerada e converte-a para bytes usando o formato de codificação especificado. A chave pública é escrita no objeto de escrita associado ao servidor. Por fim, espera que os dados sejam realmente gravados no socket, garantindo que a escrita seja concluída antes de continuar.

```
# This async function sends the public key to the server
# It generates a private key for the client using X25519 algorithm
# It gets the corresponding public key and converts it to bytes using
the specified encoding format
# The public key is written to the writer object associated with the
server
# It waits for the data to be actually written to the socket before
continuing
async def send_public_key(writer):
    private_key = x25519.X25519PrivateKey.generate()
    public_key =
private_key.public_key().public_bytes(serialization.Encoding.Raw,
serialization.PublicFormat.Raw)
    writer.write(public_key)
    await writer.drain()
```

A função send_message é a função responsável por enviar mensagens do cliente para o servidor.

Cria um loop infinito para que o cliente possa enviar várias mensagens consecutivas. Solicita ao utilizador que insira uma mensagem a ser enviada ao servidor. Verifica se o utilizador digitou 'quit' para encerrar o loop. É então aberta uma conexão com o servidor, a função send_public_key é chamada para enviar a chave pública antes da mensagem, que depois é codificada para bytes e escrita no objeto de escrita associado ao servidor. A função writer.drain() espera que os dados sejam realmente gravados no socket antes de continuar. Quando gravados, são lidos até 1024 bytes de dados do servidor que exibe a mensagem recebida decodificando os bytes usando UTF-8. A conexão com o servidor é fechada após o envio e receção da mensagem.

A função except KeyboardInterrupt captura a exceção de interrupção do teclado (Ctrl+C) para permitir a saída controlada do cliente.

```
# This async function sends a message to the server
# It creates a loop to allow the client to send multiple consecutive
messages
# It prompts the user to enter a message to be sent to the server
# It checks if the user entered 'quit' to exit the loop
# It then opens a connection to the server, calls the send_public_key
function to send the public key before the message
# The message is then encoded to bytes and written to the writer
object associated with the server
# The writer.drain() function waits for the data to be actually
written to the socket before continuing
# Once written, up to 1024 bytes of data are read from the server,
```

```
which displays the received message by decoding the bytes using UTF-8
# The connection to the server is closed after sending and receiving
the message

# The except KeyboardInterrupt block captures the keyboard interrupt
exception (Ctrl+C) to allow for controlled client exit.
async def send_message():
    while True:
        try:
            message = input("Enter message to send (or 'quit' to
exit): ")
            if message.lower() == 'quit':
                break

            reader, writer = await
asyncio.open_connection('127.0.0.1', 8888)

            await send_public_key(writer)

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

            data = await reader.read(1024)
            print(f"Received message from server: {data.decode('utf-
8', 'ignore')}")

            writer.close()
            await writer.wait_closed()
        except KeyboardInterrupt:
            print("Exiting client...")
            break
```

A task cria e executa uma tarefa assíncrona para o servidor (main).

```
task = asyncio.create_task(main())

Serving on ('127.0.0.1', 8888)
```

Por fim, é criada e executada uma tarefa assíncrona para enviar mensagens ao servidor (send_message).

```
await asyncio.create_task(send_message())
```