TP1 - 2

Objetivo

Implementar uma AEAD com "Tweakable Block Ciphers", usando *AES-256* ou *ChaCha20* e contruir um canal privado de informação assíncrona com acordo de chaves feito com "X448 key exchange" e autenticação com "Ed448 Signing&Verification"

Abordagem

Inicialmente, tanto o **cliente** quanto o **servidor** têm de gerar as suas chaves *x448* e *ed448* privadas e públicas.

Após estabelecerem uma conexão, ambos enviam a sua chave pública *ed448* **assinada** ao outro agente. Em seguida, procedem à leitura da chave recebida, verificando a sua assinatura. Depois disto, enviam a sua *x448* e assinatura e verificam a *x448* recebida.

O passo seguinte é o acordo da chave partilhada, em que ambos geram uma "shared key" usando o *exchange* do *x448*. Em seguida, é realizada a derivação de chave usando o HKDF.

Cliente

No caso do **cliente**, após derivar a chave através do HKDF, usando o algoritmo *ChaCha20*, ciframos a mensagem, e após ser assinada, enviamo-la para o servidor.

Servidor

No que toca ao **servidor**, é lido o *nonce* e o criptograma, e por fim, é decifrada e imprimida a mensagem.

Implementação

Variáveis e funções auxiliares

In [ ]:
```
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import x448
from cryptography.hazmat.primitives.kdf.hkdf import HKDF
from cryptography.hazmat.primitives.asymmetric import ed448
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes

import os
import asyncio

async def send_signed(key: ed448.Ed448PrivateKey, writer: asyncio.StreamWriter, data: bytes) -> Non
    sig = key.sign(data)
    writer.write(sig)
    writer.write(data)
    await writer.drain()

async def read_signed(auth_key: ed448.Ed448PublicKey, reader: asyncio.StreamReader, n: int) -> byte
    sig = await reader.read(114)
    data = await reader.read(n)

    auth_key.verify(sig, data)

    return data

def add_bytes(a: bytes, b:bytes) -> bytes:
```

```
    short_len = len(a) if len(a)<len(b) else len(b)
    r = bytearray(a) if len(a)>len(b) else bytearray(b)
    for i in range(0,short_len):
        r[i] = a[i]+b[i]
    return bytes(r)

client_message = b'Ola nina quero tratar de ti.'
```

Cliente

In [ ]:

```
async def client(s_ip: str, s_port: int, message: bytes):
    private_key_x448 = x448.X448PrivateKey.generate()
    public_key_x448 = private_key_x448.public_key()

    # Ed448 Signing&Verification
    private_key_ed448 = ed448.Ed448PrivateKey.generate()
    public_key_ed448 = private_key_ed448.public_key()

    print('client: Openning connection')
    reader, writer = await asyncio.open_connection(s_ip, s_port)

    # ed448 Exchange
    print('client: sending public auth key')
    await send_signed(private_key_ed448, writer, public_key_ed448.public_bytes(serialization.Encodi

    print('client: reading and verifying server auth key')
    other_auth_sig = await reader.read(114)
    other_authkey_bytes = await reader.read(57)

    other_authkey = ed448.Ed448PublicKey.from_public_bytes(other_authkey_bytes)

    other_authkey.verify(other_auth_sig, other_authkey_bytes)

    # x448 Exchange
    print('client: sending public x448 key')
    await send_signed(private_key_ed448, writer, public_key_x448.public_bytes(serialization.Encodin

    print('client: reading server public x448 key')
    other_pkey_bytes = await read_signed(other_authkey, reader, 56)

    other_pkey = x448.X448PublicKey.from_public_bytes(other_pkey_bytes)

    shared_key = private_key_x448.exchange(other_pkey)

    # Perform key derivation.
    key = HKDF(
        algorithm=hashes.SHA256(),
        length=32,
        salt=None,
        info=b'handshake data',
    ).derive(shared_key)

    print('client: derived key:',key)

    nonce = os.urandom(16)
    ct = b''
    for i in range(0, len(message), 64):
        algorithm = algorithms.ChaCha20(add_bytes(key, bytes(i)), nonce)
        cipher = Cipher(algorithm, mode=None)
        encryptor = cipher.encryptor()

        ct += encryptor.update(message[i:i+64])

    print('client: sending message')
    await send_signed(private_key_ed448, writer, nonce+ct)

    writer.close()

    await writer.wait_closed()
```

Server

In [ ]:
```
class Server:

    def __init__(self, ip: str, port: int):

        self.ip = ip
        self.port = port

        # X448 key exchange
        self.private_key_x448 = x448.X448PrivateKey.generate()
        self.public_key_x448 = self.private_key_x448.public_key()

        # Ed448 Signing&Verification
        self.private_key_ed448 = ed448.Ed448PrivateKey.generate()
        self.public_key_ed448 = self.private_key_ed448.public_key()

        self.server = asyncio.Server

    async def handle_connection(self, reader: asyncio.StreamReader, writer: asyncio.StreamWriter):
        # ed448 Exchange
        await send_signed(self.private_key_ed448, writer, self.public_key_ed448.public_bytes(serial

        other_auth_sig = await reader.read(114)
        other_authkey_bytes = await reader.read(57)

        other_authkey = ed448.Ed448PublicKey.from_public_bytes(other_authkey_bytes)

        other_authkey.verify(other_auth_sig, other_authkey_bytes)

        # x448 Exchange
        await send_signed(self.private_key_ed448, writer, self.public_key_x448.public_bytes(seriali

        other_pkey_bytes = await read_signed(other_authkey, reader, 56)

        other_pkey = x448.X448PublicKey.from_public_bytes(other_pkey_bytes)

        shared_key = self.private_key_x448.exchange(other_pkey)

        # Perform key derivation.
        key = HKDF(
            algorithm=hashes.SHA256(),
            length=32,
            salt=None,
            info=b'handshake data',
        ).derive(shared_key)

        print('server: derived key:',key)

        # Read message
        ct_data = await read_signed(other_authkey, reader, -1)
        nonce = ct_data[0:16]
        ct_message = ct_data[16:]


        plaintext = b''
        for i in range(0, len(ct_message), 64):
            algorithm = algorithms.ChaCha20(add_bytes(key, bytes(i)), nonce)
            cipher = Cipher(algorithm, mode=None)
            decryptor = cipher.decryptor()

            plaintext += decryptor.update(ct_message)

        plaintext = plaintext.decode('utf-8')

        print("server: received and decrypted:", plaintext)

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

```
        self.server.close()

    async def start_server(self):
        self.server = await asyncio.start_server(self.handle_connection, self.ip, self.port)
        print('server: started')
        async with self.server:
            await self.server.serve_forever()
```

Inicia Server

In [ ]:
```
server = Server('127.0.0.1', 9876)
server_task = asyncio.get_running_loop().create_task(server.start_server())
```

```
server: started
server: derived key: b'\xe5pG\x84\x0b\x91\xf5\x1a<\xc8y%\xf1~p\x84\x13.\xb3\xd5r\x8c\x81\x85\xeb\x0
4\xe4\x7fq\x06\xc1\xff'
server: received and decrypted: Ola nina quero tratar de ti.
```

Inicia Cliente

In [ ]:
```
#client_task = asyncio.get_running_loop().create_task(client('127.0.0.1', 9876, b'yep'))
await client('127.0.0.1', 9876, client_message)
```

```
client: Openning connection
client: sending public auth key
client: reading and verifying server auth key
client: sending public x448 key
client: reading server public x448 key
client: derived key: b'\xe5pG\x84\x0b\x91\xf5\x1a<\xc8y%\xf1~p\x84\x13.\xb3\xd5r\x8c\x81\x85\xeb\x0
4\xe4\x7fq\x06\xc1\xff'
client: sending message
```

Termina servidor

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

Out[ ]:
```
False
```

TESTE

Se alguém conseguir comprometer a integridade da chave, e assim, fizer com que o cliente receba uma chave errada, isso dará origem a um erro na verificação.

In [ ]:
```
async def teste_client(s_ip: str, s_port: int, message: bytes):
    private_key_x448 = x448.X448PrivateKey.generate()
    public_key_x448 = private_key_x448.public_key()

    # Ed448 Signing&Verification
    private_key_ed448 = ed448.Ed448PrivateKey.generate()
    public_key_ed448 = private_key_ed448.public_key()

    print('client: Openning connection')
    reader, writer = await asyncio.open_connection(s_ip, s_port)

    # ed448 Exchange
    print('client: sending public auth key')
    await send_signed(private_key_ed448, writer, public_key_ed448.public_bytes(serialization.Encodi

    print('client: reading and verifying server auth key')
    other_auth_sig = await reader.read(114)
    wrong_key = os.urandom(57)

    other_authkey_bytes = wrong_key

    other_authkey = ed448.Ed448PublicKey.from_public_bytes(other_authkey_bytes)
```

```
    other_authkey.verify(other_auth_sig, other_authkey_bytes)

    # x448 Exchange
    print('client: sending public x448 key')
    await send_signed(private_key_ed448, writer, public_key_x448.public_bytes(serialization.Encodin

    print('client: reading server public x448 key')
    other_pkey_bytes = await read_signed(other_authkey, reader, 56)

    other_pkey = x448.X448PublicKey.from_public_bytes(other_pkey_bytes)

    shared_key = private_key_x448.exchange(other_pkey)

    # Perform key derivation.
    key = HKDF(
        algorithm=hashes.SHA256(),
        length=32,
        salt=None,
        info=b'handshake data',
    ).derive(shared_key)

    print('client: key:', key)

    nonce = os.urandom(16)
    algorithm = algorithms.ChaCha20(key, nonce)
    cipher = Cipher(algorithm, mode=None)
    encryptor = cipher.encryptor()
    ct = encryptor.update(message)

    print('client: sending message')
    await send_signed(private_key_ed448, writer, nonce+ct)

    writer.close()

    await writer.wait_closed()
```

In [ ]:
```
server = Server('127.0.0.1', 9876)
server_task = asyncio.get_running_loop().create_task(server.start_server())
```

```
server: started
```

In [ ]:
```
await teste_client('127.0.0.1', 9876, b'asdf')
```

```
client: Openning connection
client: sending public auth key
client: reading and verifying server auth key
---------------------------------------------------------------------------
InvalidSignature                          Traceback (most recent call last)
Cell In[14], line 1
----> 1 await teste_client('127.0.0.1', 9876, b'asdf')

Cell In[12], line 24, in teste_client(s_ip, s_port, message)
     20 other_authkey_bytes = wrong_key
     22 other_authkey = ed448.Ed448PublicKey.from_public_bytes(other_authkey_bytes)
---> 24 other_authkey.verify(other_auth_sig, other_authkey_bytes)
     26 # x448 Exchange
     27 print('client: sending public x448 key')

InvalidSignature:
```

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

Out[ ]:
```
True
```

In [ ]:
```

```