Trabalho Prático 1

Trabalho realizado pelo grupo 11:

- Beatriz Fernandes Oliveira, PG50942
- Bruno Filipe Machado Jardim, PG49997

Exercício 3

3. Use o “package” Cryptography para
   A. Implementar uma AEAD com “Tweakable Block Ciphers” conforme está descrito na última secção do texto +Capítulo 1: Primitivas Criptográficas Básicas. A cifra por blocos primitiva, usada para gerar a “tweakable block cipher”, é o AES-256 ou o ChaCha20.
   B. Use esta cifra para construir um canal privado de informação assíncrona com acordo de chaves feito com “X448 key exchange” e “Ed448 Signing&Verification” para autenticação dos agentes. Deve incluir uma fase de confirmação da chave acordada.

In [1]:
```
from cryptography.hazmat.primitives.ciphers.algorithms import AES
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric.ed448 import Ed448PrivateKey
from cryptography.hazmat.primitives.ciphers.modes import ECB
from cryptography.hazmat.primitives import padding
from cryptography.hazmat.primitives.kdf.hkdf import HKDF
from cryptography.hazmat.primitives.asymmetric.x448 import X448PrivateKey
import os
import cryptography
from cryptography.hazmat.primitives import hashes, hmac
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import x448
import asyncio
import random
import nest_asyncio
import math
```

In [2]:
```
def cipher(tweak,key,message):
    key = tweak + key
    algorithm = algorithms.AES(key)
    cipher = Cipher(algorithm, modes.ECB())
    encryptor = cipher.encryptor()
    ct = encryptor.update(message) + encryptor.finalize()
    return ct
def decipher(tweak,key,ct):
    key = tweak + key
    cipher = Cipher(algorithms.AES(key), modes.ECB())
    decryptor = cipher.decryptor()
    pt = decryptor.update(ct) + decryptor.finalize()
    return pt
```

In [3]:
```
def xor(bitString1,bitString2):
    return bytes([ x^y for x,y in zip(bitString1,bitString2)])
```

In [4]:
```
def genTweaks(pt,nounces,length):
    n_blocks = len(pt)
    tweaks = []
    j = 0
    for i in range(n_blocks):
        tweak = nounces[i] + (i).to_bytes(7,'big') + (0).to_bytes(1,'big')
        tweaks.append(tweak)
        j = i
    last_tweak =  nounces[j+1] + (length).to_bytes(7,'big') + (1).to_bytes(1,'

    return tweaks,last_tweak

def PRG(seed,size):
    dgst = hashes.Hash(hashes.SHAKE256(2**size * 8))
    dgst.update(seed)
    nounceString = dgst.finalize()
    return [nounceString[i:i+8] for i in range(0,len(nounceString),8)]
```

In [5]:
```
def getAuth(blocks):

    if len(blocks) == 1:
        return blocks[0]
    auth = (0).to_bytes(32,'big')
    for i in range(0,len(blocks)):
        auth = xor(auth,blocks[i])
    return auth

def getB(msg,size):
    final = []
    for i in range(0,len(msg),size):
        final.append(msg[i:i+size])
    return final

def padLast(blocks):
    last = blocks[-1:][0]
    remaining = blocks[:-1]
    length = len(last)
    last += b'0'*(256-length)
    remaining.append(last)
    return remaining,length

def block2String(blocks):
    string = ""
    for block in blocks:
        string += block.decode('utf-8')
    return string
```

In [8]:
```
def genX448Keys():
    private_key = x448.X448PrivateKey.generate()

    public_key = private_key.public_key()

    public_bytes = public_key.public_bytes(

        encoding=serialization.Encoding.Raw,

        format=serialization.PublicFormat.Raw

    )
    return private_key,public_bytes

def genED448Keys():
    private_key = Ed448PrivateKey.generate()
    public_key = private_key.public_key()
    return private_key,public_key

def derive_key(private_key, peer_public_key):
    shared_key = private_key.exchange(peer_public_key)
    derived_key = HKDF(
        algorithm=hashes.SHA256(),
        length=16,
        salt=None,
        info=b'EC',
    ).derive(shared_key)
    return derived_key

def keyVerification(signKey,peer_public_bytes,signature):
    try:
        signKey.verify(signature,peer_public_bytes)
        print("Signature validated")
    except Exception as e:
        print("Invalid signature: ",e)
```

In [9]:
```
async def conProtocolEmitter(qE,qR):

    pvDerKey,pbDerKey = genX448Keys()
    pvSignKey,pbSignKey = genED448Keys()
    signature = pvSignKey.sign(pbDerKey)

    await qE.put((pbSignKey,pbDerKey,signature))

    pbkReceiver,derKey,signatureRec = await qR.get()
    print("[Emitter]: Key validation ...")
    keyVerification(pbkReceiver,derKey,signatureRec)


    pbkReceiver = x448.X448PublicKey.from_public_bytes(derKey)

    derived_key = derive_key(pvDerKey, pbkReceiver)

    return derived_key

async def conProtocolReceiver(qE,qR):
    pvDerKey,pbDerKey = genX448Keys()
    pvSignKey,pbSignKey = genED448Keys()
    signature = pvSignKey.sign(pbDerKey)

    await qR.put((pbSignKey,pbDerKey,signature))

    pbkEmitter,derKey,signatureEmi = await qE.get()
    print("[Receiver]: Key validation ...")
    keyVerification(pbkEmitter,derKey,signatureEmi)

    pbkEmitter = x448.X448PublicKey.from_public_bytes(derKey)

    derived_key = derive_key(pvDerKey, pbkEmitter)

    return derived_key

def oneTimeKey(key):
    digest = hashes.Hash(hashes.SHA3_256())
    digest.update(key)
    oneTimeKey = digest.finalize()
    return oneTimeKey

def cipherLength(key,message):
    algorithm = algorithms.AES(key)
    cipher = Cipher(algorithm, modes.ECB())
    encryptor = cipher.encryptor()
    length = encryptor.update(message) + encryptor.finalize()
    return length
def decipherLength(key,message):
    algorithm = algorithms.AES(key)
    cipher = Cipher(algorithm, modes.ECB())
    decryptor = cipher.decryptor()
    length = decryptor.update(message) + decryptor.finalize()
    return length
```

In [12]:
```
msg =  """As armas e os barões assinalados,
Que da ocidental praia Lusitana,
Por mares nunca de antes navegados,
Passaram ainda além da Taprobana,
Em perigos e guerras esforçados,
Mais do que prometia a força humana,
E entre gente remota edificaram
Novo Reino, que tanto sublimaram;

E também as memórias gloriosas
Daqueles Reis, que foram dilatando
A Fé, o Império, e as terras viciosas
De África e de Ásia andaram devastando;
E aqueles, que por obras valerosas
Se vão da lei da morte libertando;
Cantando espalharei por toda parte,
Se a tanto me ajudar o engenho e arte.

Cessem do sábio Grego e do Troiano
As navegações grandes que fizeram;
Cale-se de Alexandro e de Trajano
A fama das vitórias que tiveram;
Que eu canto o peito ilustre Lusitano,
A quem Neptuno e Marte obedeceram:
Cesse tudo o que a Musa antígua canta,
Que outro valor mais alto se alevanta.

E vós, Tágides minhas, pois criado
Tendes em mim um novo engenho ardente,
Se sempre em verso humilde celebrado
Foi de mim vosso rio alegremente,
Dai-me agora um som alto e sublimado,
Um estilo grandíloquo e corrente,
Porque de vossas águas, Febo ordene
Que não tenham inveja às de Hipocrene.

Dai-me uma fúria grande e sonorosa,
E não de agreste avena ou frauta ruda,
Mas de tuba canora e belicosa,
Que o peito acende e a cor ao gesto muda;
Dai-me igual canto aos feitos da famosa
Gente vossa, que a Marte tanto ajuda;
Que se espalhe e se cante no universo,
Se tão sublime preço cabe em verso."""
```

In [13]:
```
nest_asyncio.apply()

async def Emitter(msg,queue,qE,qR,authQ):

    derivation = asyncio.create_task(conProtocolEmitter(qE,qR))
    key = await derivation

    bmsg = msg.encode('utf-8')
    otK=oneTimeKey(key)

    length = cipherLength(otK,((len(bmsg)).to_bytes(256,"big")))


    await queue.put(length)
    await asyncio.sleep(random.random())


    nounces = PRG(key,12)

    msg = msg.encode('utf-8')
    length = len(msg)

    blocks = getB(msg,256)
    blocks, tau = padLast(blocks)

    tau = (tau).to_bytes(256,'big')

    Auth = getAuth(blocks)
    tweaks, auth_tweak = genTweaks(blocks,nounces,length)
    last = blocks[-1]

    block_Tweak = zip(blocks,tweaks)


    for block,tweak in block_Tweak:
        await asyncio.sleep(random.random())

        if block == last:
            ct = cipher(tweak,key,tau)
            last = xor(ct,block)
            await queue.put(last)

            print("[Emitter]: Sent a block")
            break

        ct = cipher(tweak,key,block)
        print("[Emitter]: Sent a block")
        await queue.put(ct)
    await queue.put(None)
    tag = cipher(auth_tweak,key,Auth)
    await asyncio.sleep(random.random())
    await authQ.put(tag)
    print("[Emitter]: Sent the Authentication Tag")
    await queue.put(None)
```

```
async def Receiver(queue,qE,qR,authQ):
    derivation = asyncio.create_task(conProtocolReceiver(qE,qR))
    key = await derivation



    length = await queue.get()
    otK=oneTimeKey(key)
    length = decipherLength(otK,length)
    length = int.from_bytes(length, "big")
    queue.task_done()



    nblocks = [[i] for i in range(int(math.ceil(length/256)))]
    nounces = PRG(key,12)

    tweaks, authTweak = genTweaks(nblocks,nounces,length)

    nblocks = len(nblocks)
    pad_size = 256-length%256
    tau = (length%256).to_bytes(256,'big')
    plaintext = []
    i = 0

    while True:
        item = await queue.get()
        if item is None:
            break
        block = item

        if i == nblocks-1:
            ct = cipher(tweaks[i],key,tau)
            last = xor(ct,block)
            plaintext.append(last[:-pad_size])
            print("[Receiver]: Deciphered last block: ")
        else:
            pt = decipher(tweaks[i],key,block)
            plaintext.append(pt)
            print("[Receiver]: Deciphered block nº: " ,i)
        queue.task_done()
        i+=1
    pt = block2String(plaintext)
    print("[Receiver]: Full deciphered text:\n\n")
    print(pt,"\n\n")

    tag = await authQ.get()

    msg = pt.encode('utf-8')
    blocks = getB(msg,256)
    Auth = getAuth(blocks)

    checksum = cipher(authTweak,key,Auth)

    if tag == checksum:
        print("[Receiver]: Message is Authenticated")
```

```
    else:
        print("[Receiver]: Message has been tampered")



async def main():
    queue = asyncio.Queue()
    qE = asyncio.Queue()
    qR = asyncio.Queue()
    authQ = asyncio.Queue()
    emitter = asyncio.create_task(Emitter(msg,queue,qE,qR,authQ))
    receiver = asyncio.create_task(Receiver(queue,qE,qR,authQ))
asyncio.run(main())
```

```
[Receiver]: Key validation ...
Signature validated
[Emitter]: Key validation ...
Signature validated
[Emitter]: Sent a block
[Receiver]: Deciphered block nº:  0
[Emitter]: Sent a block
[Receiver]: Deciphered block nº:  1
[Emitter]: Sent a block
[Receiver]: Deciphered block nº:  2
[Emitter]: Sent a block
[Receiver]: Deciphered block nº:  3
[Emitter]: Sent a block
[Receiver]: Deciphered block nº:  4
[Emitter]: Sent a block
[Receiver]: Deciphered last block:
[Receiver]: Full deciphered text:


As armas e os barões assinalados,
Que da ocidental praia Lusitana,
Por mares nunca de antes navegados,
Passaram ainda além da Taprobana,
Em perigos e guerras esforçados,
Mais do que prometia a força humana,
E entre gente remota edificaram
Novo Reino, que tanto sublimaram;

E também as memórias gloriosas
Daqueles Reis, que foram dilatando
A Fé, o Império, e as terras viciosas
De África e de Ásia andaram devastando;
E aqueles, que por obras valerosas
Se vão da lei da morte libertando;
Cantando espalharei por toda parte,
Se a tanto me ajudar o engenho e arte.

Cessem do sábio Grego e do Troiano
As navegações grandes que fizeram;
Cale-se de Alexandro e de Trajano
A fama das vitórias que tiveram;
Que eu canto o peito ilustre Lusitano,
A quem Neptuno e Marte obedeceram:
```

```
Cesse tudo o que a Musa antígua canta,
Que outro valor mais alto se alevanta.

E vós, Tágides minhas, pois criado
Tendes em mim um novo engenho ardente,
Se sempre em verso humilde celebrado
Foi de mim vosso rio alegremente,
Dai-me agora um som alto e sublimado,
Um estilo grandíloquo e corrente,
Porque de vossas águas, Febo ordene
Que não tenham inveja às de Hipocrene.

Dai-me uma fúria grande e sonorosa,
E não de agreste avena ou frauta ruda,
Mas de tuba canora e belicosa,
Que o peito acende e a cor ao gesto muda;
Dai-me igual canto aos feitos da famosa
Gente vossa, que a Marte tanto ajuda;
Que se espalhe e se cante no universo,
Se tão sublime preço cabe em verso.


[Emitter]: Sent the Authentication Tag
[Receiver]: Message is Authenticated
```

In [ ]: