Exercício 1 - Trabalho Prático 1

Grupo 6:

Ruben Silva - pg57900

Luís Costa - pg55970

Problema:

1. Use a package **cryptography** para criar um comunicação privada assíncrona entre um agente *Emitter* e um agente *Receiver* que cubra os seguintes aspectos:
   A. Comunicação cliente-servidor que use o package python `asyncio`.
   B. Usar como cifra AEAD o "hash" SHAKE-256 em modo XOFHash
   C. As chaves de cifra e os "nounces" são gerados por um gerador KDF . As diferentes chaves para inicialização KDF são inputs do emissor e do receptor.

Implementação do Problema

Import

1. Instalar/importar as funcionalidades necessárias do `crypography`
2. Instalar/importar o `asyncio` para ser possível a criação do cliente-servidor assíncrono

In [2]:
```
%pip install cryptography asyncio
import asyncio

from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.kdf.hkdf import HKDF
```

```
Requirement already satisfied: cryptography in c:\users\ruben\desktop\minho\mei\c
si\ec\.venv\lib\site-packages (44.0.1)
Requirement already satisfied: asyncio in c:\users\ruben\desktop\minho\mei\csi\ec
\.venv\lib\site-packages (3.4.3)
Requirement already satisfied: cffi>=1.12 in c:\users\ruben\desktop\minho\mei\csi
\ec\.venv\lib\site-packages (from cryptography) (1.17.1)
Requirement already satisfied: pycparser in c:\users\ruben\desktop\minho\mei\csi
\ec\.venv\lib\site-packages (from cffi>=1.12->cryptography) (2.22)
Note: you may need to restart the kernel to use updated packages.
```

Inicialmente é necessário criar uma variável global pra garantir que os nounces gerados são únicos e não se repetem

In [14]:
```
nounce_list=[]
```

Funções Importantes

1. É definido a função "shake256XOF" com return do hash final
   A. Verificação se a strig passada está em bytes
   B. Executar o código que implementa o SHAKE256XOF

Esta função iniciliza o modelo sponge, sendo seguido do `absorve` e do `squeeze`, o que implica que é `XOF`

Fonte - Documentação de "cryptography"

In [15]:
```python
def shake256XOF(text , length=32):
    if isinstance(text, str):
        text = text.encode('utf-8')
    elif not isinstance(text, bytes):
        raise TypeError("Input must be string or bytes")


    digest = hashes.Hash(hashes.SHAKE256(length),  backend=default_backend()) #s
    digest.update(text) #Absorve
    return digest.finalize() #Squeeze
```

1. É definido a função "derive_nounce" com return do nounce final
   A. Verificação se a strig passada está em bytes
   B. Executar

Função muito semelhante à anterior, mas desta vez usando SHA256

Fonte - Documentação de "cryptography"

In [16]:
```python
def derive_nounce(text: bytes):
    if isinstance(text, str):
        text = text.encode('utf-8')
    elif not isinstance(text, bytes):
        raise TypeError("Input must be string or bytes")

    #Sponge
    hkdf = HKDF(
        algorithm=hashes.SHA256(),
        length=12,
        salt=None,
        info=b"Nonce Generation",
        backend=default_backend()
    )
    return hkdf.derive(text) #Absorve n Squeeze
```

1. É definido a função "cipher_ex1" para cifrar
   A. Verificação se a strig passada está em bytes
   B. Obter um `nounce` **único** evita ataques de repitação para os mesmos *inputs* dando origem a um `keystream` **único**
   C. É gerado o `keystream` usando a função previamente criada (*shake256XOF*)

D. É criado o `ciphertext` aplicando `XOR` entre o *plaintext* e *keystream*. A utilização permite que seja facilmente revertido, **SENDO FUNDAMENTAL**, que o *keystream* seja não reutilizado (unicicidade)

E. `TAG` - É aplicado `shake256XOF` com o a concatenação entre *key*, *nounce*, *ciphertext* para garantir que é uma cifra `AEAD` e assim evitar ataques de modificação obtendo assim conhecimento se existir uma `violação da integridade` da mensagem

Fonte - Estruturas Criptograficas UM

In [17]:
```python
def cipher_ex1(plaintext: str, key: bytes):
    global nounce_list

    if isinstance(plaintext, str):
        plaintext = plaintext.encode('utf-8')

    nounce = derive_nounce(key)
    while nounce in nounce_list:
        nounce = shake256XOF(nounce)
    nounce_list.append(nounce)

    try:

        keystream = shake256XOF(key + nounce, length=len(plaintext))

        ciphertext = bytes(p ^ k for p, k in zip(plaintext, keystream))

        tag_input = key + nounce + ciphertext
        tag = shake256XOF(tag_input)

        return nounce, ciphertext, tag

    except Exception as e:
        print(e)
        return None
```

1. É definido a função "de_cipher_ex1" para decifrar
   A. `TAG` - É aplicado `shake256XOF` com o a concatenação entre *key*, *nounce*, *ciphertext* para garantir que é uma cifra `AEAD` e assim evitar ataques de modificação obtendo assim conhecimento se existir uma `violação da integridade` da mensagem.
   B. Verificar se efetivamente a mensagem sofreu `violação de integrdidade`
   C. Se não sofreu, é necssário voltar a gerar a `keystream` tendo em cosideração que o hashing é um processo deterministico.
   D. É descuberto o `plaintext` aplicando `XOR` entre o *ciphertext* e *keystream*

Fonte - Estruturas Criptograficas UM

In [18]:
```python
def de_cipher_ex1(ciphertext: bytes, key: bytes, nounce: bytes, tag: bytes):
    try:

        tag_input = key + nounce + ciphertext
        expected_tag = shake256XOF(tag_input)
```

```
        if tag != expected_tag:
            raise ValueError("Authentication failed - Tag mismatch")

        keystream = shake256XOF(key + nounce, length=len(ciphertext))

        plaintext = bytes(c ^ k for c, k in zip(ciphertext, keystream))

        return plaintext

    except ValueError as e:
        print(e)
        return None
    except Exception as e:
        print(e)
        return None
```

Emitter

É definido a função `Emitter` que envia o *input* do utilizador encriptado seguindo as diretrizes pedidas no enunciado

A mensagem `"0"` permite desligar este programa

Fonte - Documentação de "asyncio"

In [19]:
```
async def emitter(queue,seed):
    key = shake256XOF(seed)
    print(f"Emitter Key: {key}")
    message_id = 0
    loop = asyncio.get_event_loop()
    while True:
        message=await loop.run_in_executor(None, input, "MESSAGE: ")

        if message=="0":
            for task in asyncio.all_tasks():
                task.cancel()  # Isso vai cancelar todas as tasks, incluindo a m
            break

        print("*"*50)
        print(f"{message_id} - ")
        print(f"Message From EMITTER: {message}")

        print("-"*20)
        out=cipher_ex1(message,key)

        message_id += 1
        await queue.put(out)
```

Receiver

É definido a função `Receiver` que recebe o *input* do utilizador encriptado e irá desencripta-lo mostrando a resposta final no terminal

A mensagem `"0"` permite desligar este programa

Fonte - Documentação de "asyncio"

In [20]:
```
async def receiver(queue,seed):
    key = shake256XOF(seed)
    print(f"Receiver Key: {key}")
    while True:
        nounce, ciphertext, tag = await queue.get()
        message=de_cipher_ex1(ciphertext,key,nounce, tag)
        if message=="0":
            for task in asyncio.all_tasks():
                task.cancel()  # Isso vai cancelar todas as tasks, incluindo a m
            break

        print(f"Received cipher: {ciphertext}")
        print(f"Received tag: {tag}")
        print(f"Received nounce: {nounce}")
        print(f"The real message: {message}")
```

Run

As funções previamente criadas irão correr de forma assíncrona tendo como ponto de comunicação a class `Queue` da libraria *asyncio*

In [21]:
```
async def main():
    try:
        queue = asyncio.Queue()
        seed=input("Digite uma frase (seed):")

        # Emitter & Receiver Tasks
        emitter_task = asyncio.create_task(emitter(queue,seed))
        receiver_task = asyncio.create_task(receiver(queue, seed))

        await asyncio.gather(emitter_task, receiver_task) # Espera as tasks term

        main_task = asyncio.create_task(main()) # Para matar a main task

        await main_task
    except asyncio.CancelledError:
        print("\n\nAcabou!")
```

In [22]:
```
await main()
```

```
Emitter Key: b'\xd0\xef\t\x14Z\x83(\xcd\nz\xc3\x90\xc6o\xb6^\x88\x9c\x0e\n\xe3\xc
9\x13qj\x1b\x11\xbf\xe8\xdb_\xe8'
Receiver Key: b'\xd0\xef\t\x14Z\x83(\xcd\nz\xc3\x90\xc6o\xb6^\x88\x9c\x0e\n\xe3\x
c9\x13qj\x1b\x11\xbf\xe8\xdb_\xe8'
**********************************************
0 -
Message From EMITTER: Ola, o meu nome e pedro e gosto muito de voar
--------------------
Received cipher: b'e\xfd_\xb9\xdf\xd2\xca\xa7\x9d\x85\x81\x91\t\xa3\xeb\x88\xe5\x
a4O;<\xba9\xc1\xf5\x04\xbb\xfc$Q\x18+\x16\xe8\x84y\x00\x96\x02|\xc35\x8a-@'
Received tag: b';\x02\x9fE%\xa3\xd1\x81\x80\xb6siq\x11\x1dB\xccL\x0b\xe9\x02\xc5u
\xa6l\x9c8\xd3\xe09\xb3b'
Received nounce: b'\xf4\xf0\x97\xe4\x19#2\xf0K\xa7\xb1\xa8'
The real message: b'Ola, o meu nome e pedro e gosto muito de voar'
**********************************************
1 -
Message From EMITTER: a serio???? Podemos simplesmente voar entao
--------------------
Received cipher: b'5\xc4\xc4N\x01\xec3\xf4\x8d\x82\x8a\x1ev\x8cp*s&tk\xf30\xf6\x8
9Y\xefOyX\xd8&\x08\x9e\x0c\xfd\xae\x8c\xe2/\xbf\xd7\x9d\xc6'
Received tag: b'J\xa1\xa6[d\xfaKZ\t\xe4\xd3\xaf;i\x02\xb6\xcb5Y\xb5b\xe0[\xb8\x1a
\x13\xcd\x0bL\x8cU\xbc'
Received nounce: b'\x95j\xd6I@x\xa0[\x1cD\xe8\xd3+\xb7[77\xd7\xe5\xccp\xd8\x18:M
\xd9b[I}^-'
The real message: b'a serio???? Podemos simplesmente voar entao'
**********************************************
2 -
Message From EMITTER: ola mundo
--------------------
Received cipher: b'\xf0\t\x9ab\xd1\xceB>\xcb'
Received tag: b'\xc5\x16"\xa8\xe8\xc6\xa3^dq\xe4\xfe\x97\xb1\xdai\xfaA\x14Fc\x0co
\xbci\xb4\xbd\xf1\t\xff\xe5t'
Received nounce: b'\xf4s\x87\xcf\xb9\x87\x9a\xd8\xcd\xa9\x87\xe51\x97\xa8\x00dX\x
cc\x91U\x8b\x12DT\xca\x13\x15\xa3\xf0\xfa\xf6'
The real message: b'ola mundo'
**********************************************
3 -
Message From EMITTER: ola mundo
--------------------
Received cipher: b'\xc4:\xbb\xcfI\xec\xe7Rs'
Received tag: b'\nU*\xfe2E\x17\xd0\xce\xba\xecS\x9a\xa2LkB\xe0\x1e\r=\xafZ\xda%\x
e1g\x9a\xfcB\xde9'
Received nounce: b'\xd0\x8bO<(!\x81E\xfbVk,K\xb1v\xad\xc6\xba\xe0V`\xa8I6\xfd\xb8
\x15\xf2\xf3\xe7\x07n'
The real message: b'ola mundo'
**********************************************
4 -
Message From EMITTER: ola
--------------------
Received cipher: b'\xc9\x872'
Received tag: b'\xd1z[*xQ\xbd\x96\x9d\x82\x8e\xdd\xf2\xdcl\x86^\xfb\xe9X\x87\xaag
1\xa5\x08E\x95+\x17\x0c4'
Received nounce: b'\xbe\xe8\xe3\xf2\xb7:\xb4\xc6\x01\x9e\x9d\xb6NZ\xf3@\x99\xf2X
\xc3\x95\x86\x9cm\xe0&+$s\x0f\xd6\x85'
The real message: b'ola'


Acabou!
```