TP1-2

March 7, 2023

1 Trabalho Prático 1

Trabalho realizado pelo grupo 11:

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

1.0.1 Exercício 2

Para cumprir o objetivo deste exercício, foi necessário a realização das seguintes subetapas:

1. Criar um gerador pseudo-aleatório do tipo XOF ("extened output function") usando o SHAKE256, para gerar uma sequência de palavras de 64 bits.
    1. O gerador deve poder gerar até um limite de 2^n palavras (n é um parâmetro) armazenados em long integers do Python.
    2. A "seed" do gerador funciona como `cipher_key` e é gerado por um KDF a partir de uma "password" .
    3. A autenticação do criptograma e dos dados associados é feita usando o próprio SHAKE256.
2. Defina os algoritmos de cifrar e decifrar: para cada uma destas operações aplicadas a uma mensagem com blocos de 64 bits, os "outputs" do gerador são usados como máscaras XOR dos blocos da mensagem. Essencialmente a cifra básica é uma implementação do "One Time Pad".

[1]:
```
import os
import operator
from cryptography.hazmat.primitives import hashes, hmac
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
```

Gerador pseudo-aleatório do tipo XOF

[2]:
```
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)]
```

Algoritmo de KDF

[3]:
```
def pbkdf_algorithm(password, s, l, its):

    pbdkf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length= l,
        salt= s,
        iterations = its)

    key = pbdkf.derive(password)

    return key
```

[4]:
```
def auth_cryptogram(key, metadata):

    h = hmac.HMAC(key, hashesSHA256())
    h.update(metadata)

    signature = h.finalize()
    return signature
```

[5]:
```
def encrypt(mensagem, generated_seq):
    msg = mensagem.encode('utf-8')
    for x in generated_seq:
        msg = bytes(map(operator.xor, msg, x))
    return msg
```

[6]:
```
def decrypt(ciphertext, generated_seq):
    for x in generated_seq:
        ciphertext = bytes(map(operator.xor, ciphertext, x))
    return ciphertext.decode('utf-8')
```

[7]:
```
def cipher_from_PRG(msg, password):

    #o argumento length recebe em bytes, 64 bits = 8 bytes.
    length = 8
    iterations = 480000

    salt = os.urandom(16)
    seed = pbkdf_algorithm(password.encode('utf-8'), salt, length, iterations)
    generated_seq = PRG(seed, length)

    ciphertext = encrypt(msg, generated_seq)
    print("Ciphertext: ", ciphertext)
    plaintext = decrypt(ciphertext, generated_seq)
    print("Plaintext: %s" % plaintext)
```

1.0.2 Exemplos

[8]:
```
mensagem = "Criptogr"
password = "EC22-23password"

cipher_from_PRG(mensagem, password)
```

```
Ciphertext:  b'[\xfd\xb8\xd5\xba\xa1\xa6\xc6'
Plaintext: Criptogr
```

[9]:
```
mensagem = "bom dia!"
password = "07-03-2023"

cipher_from_PRG(mensagem, password)
```

```
Ciphertext:  b'\x04\xa8\x1a;s\xc4\xff\x8f'
Plaintext: bom dia!
```