Instituto Politécnico Nacional

Escuela Superior de Cómputo

Cryptography

DES and 3DES

Group: 3CM15

Date: October 3, 2021

Student.

Maximiliano Cazares Martínez

Professor.

Sandra Díaz Santiago

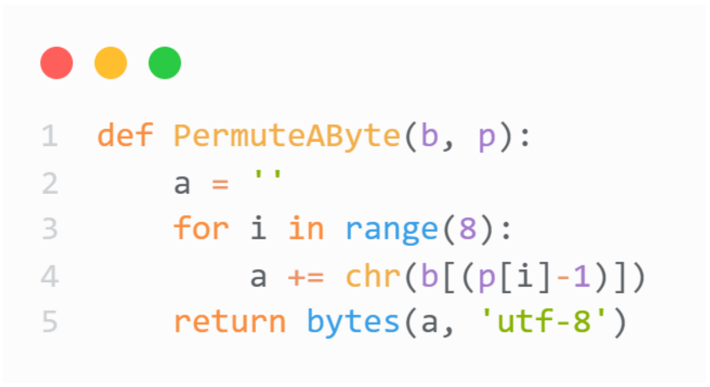
How to run the program

The practice contains 4 python scripts called permutation.py, _3DES_.py, _DES_.py and _DES_test.py. The first and last script are the only ones which are able to be run. For running them, we need to install the python module pycryptodome if it is not installed, it is necessary type the command “py -m pip install pycryptodome” in any computer with a python 3 interpreter.

Organization

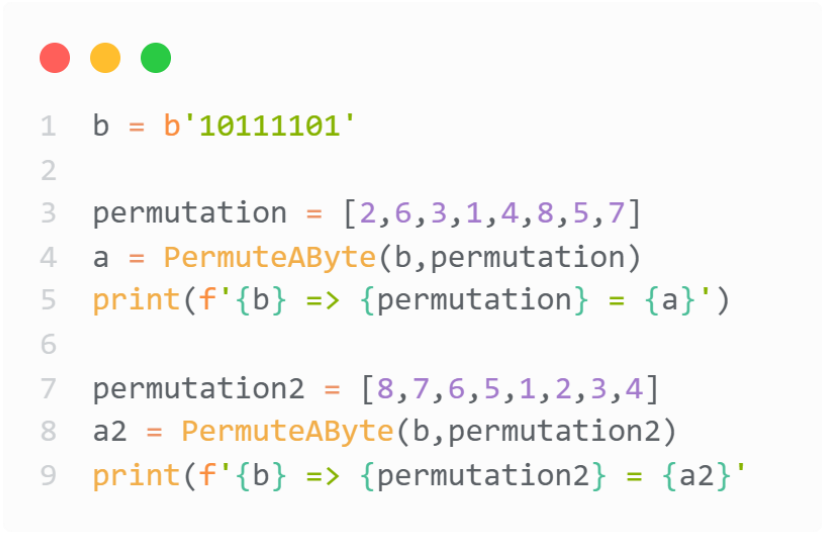
1) Permutation.py include a function named PermuteAByte which has two parameters b and p. b is a byte and p is the permutation to be applied on the byte.

   This function goes through of the byte and it is searching for the bit in the p[i] – 1 position within permutation array. Each bit is concatenated in a string and at the end it is returned a permuted byte.



```python
def PermuteAByte(b, p):
    a = ''
    for i in range(8):
        a += chr(b[(p[i]-1)])
    return bytes(a, 'utf-8')
```

We will run two examples with the same byte but different permutation array.



```python
b = b'10111101'

permutation = [2,6,3,1,4,8,5,7]
a = PermuteAByte(b,permutation)
print(f'{b} => {permutation} = {a}')

permutation2 = [8,7,6,5,1,2,3,4]
a2 = PermuteAByte(b,permutation2)
print(f'{b} => {permutation2} = {a2}'
```

This is the message shown on the terminal.

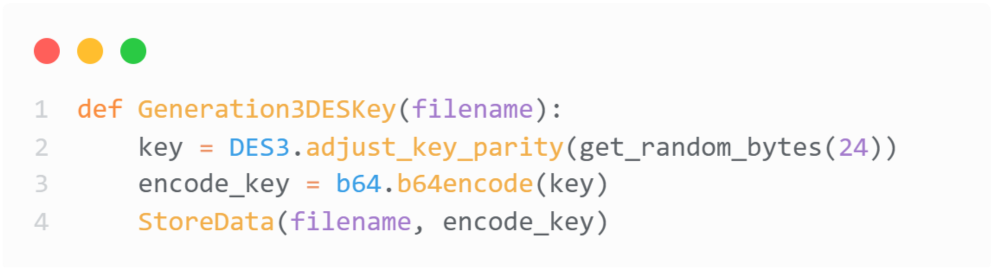
```
Maxo@Maxo MINGW64 ~/OneDrive/Documentos/ESCOM/Cryptography/codigos/DES-3DES
$ C:/Users/MaxoC/AppData/Local/Programs/Python/Python39/python.exe c:/Users/Ma
xoC/OneDrive/Documentos/ESCOM/Cryptography/codigos/DES-3DES/Permutation.py


b'10111101' => [2, 6, 3, 1, 4, 8, 5, 7] = b'01111110'
b'10111101' => [8, 7, 6, 5, 1, 2, 3, 4] = b'10111011'
```

2) We will use the library pycryptodome but unfortunately it is only able to use the EDE variant of the 3DES algorithm, so, it is proposed the EEE variant using the DES algorithm which is found on the same library.

EDE variant.

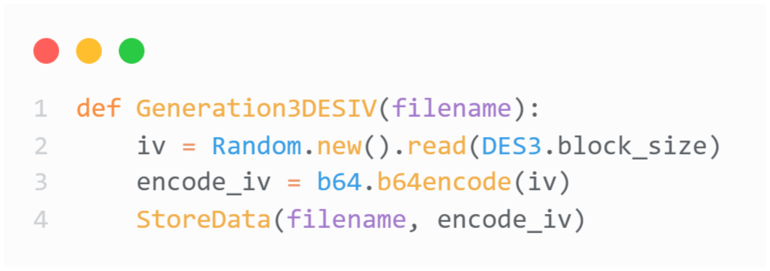
We will compute a random key for the EDE variant is like this way.

The length of the key is 24 random bytes then it will be encoded base 64 and stored in a file with a filename we decide.



```python
def Generation3DESKey(filename):
    key = DES3.adjust_key_parity(get_random_bytes(24))
    encode_key = b64.b64encode(key)
    StoreData(filename, encode_key)
```

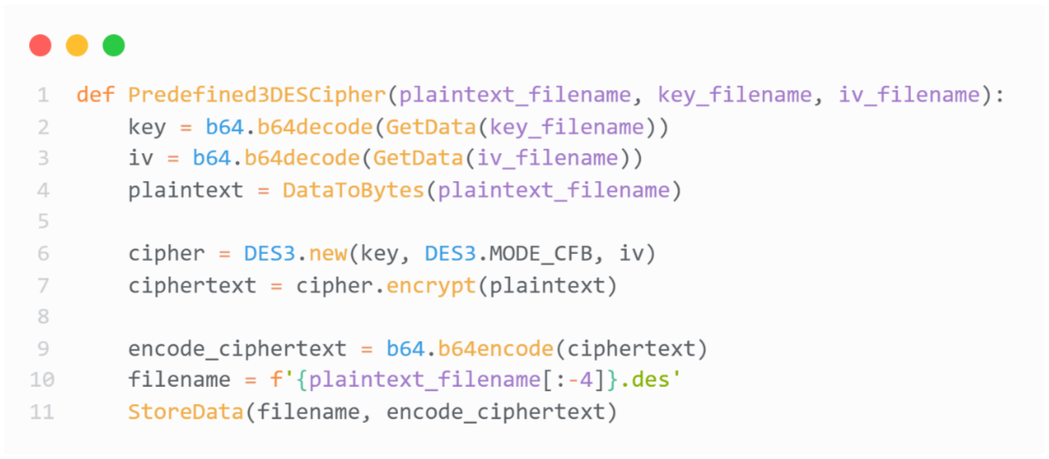
For the IV will happen the same as the key. It will be encoded base 64 and store in a file with a filename we want to.



```python
def Generation3DESIV(filename):
    iv = Random.new().read(DES3.block_size)
    encode_iv = b64.b64encode(iv)
    StoreData(filename, encode_iv)
```

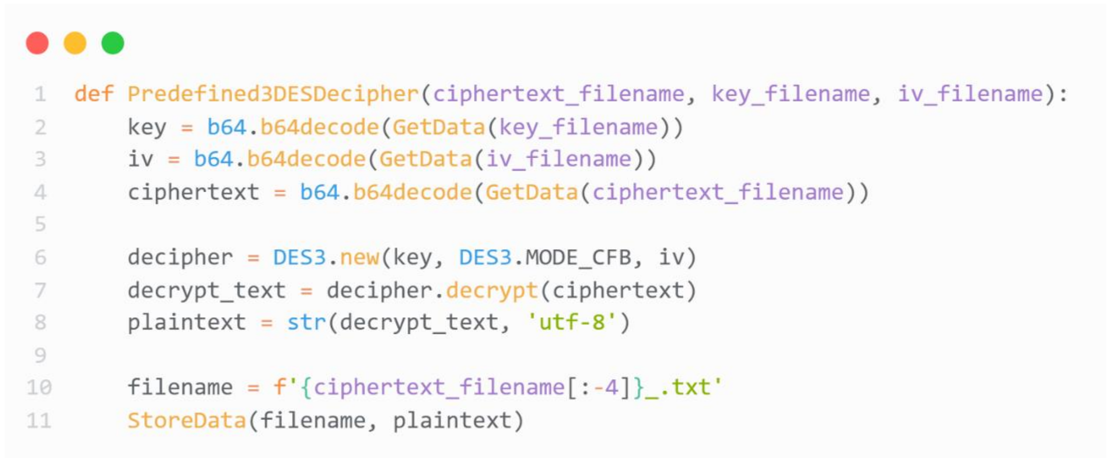
The encryption function has three parameters, the name of the text to encrypt, the name of the file with the key and the name of the file with the IV.

This function will decode the key and the IV from the files also, it will convert into bytes the text of the file we chosen. The 3DES algorithm will encrypt the text on CFB mode, next, this ciphertext will be stored in a file with the same filename of the file we chosen but with .des extension.



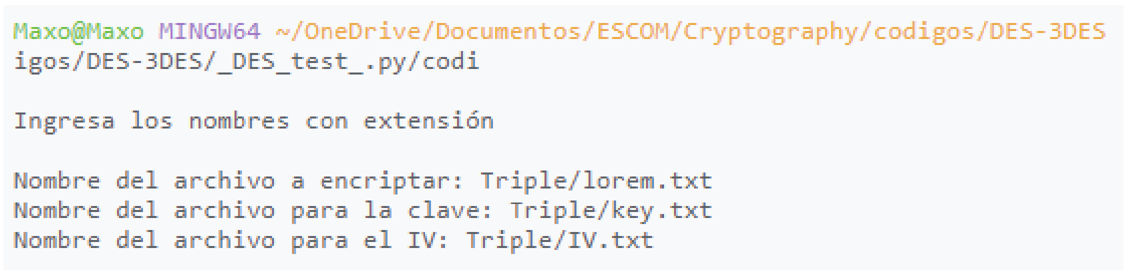
```python
def Predefined3DESCipher(plaintext_filename, key_filename, iv_filename):
    key = b64.b64decode(GetData(key_filename))
    iv = b64.b64decode(GetData(iv_filename))
    plaintext = DataToBytes(plaintext_filename)

    cipher = DES3.new(key, DES3.MODE_CFB, iv)
    ciphertext = cipher.encrypt(plaintext)

    encode_ciphertext = b64.b64encode(ciphertext)
    filename = f'{plaintext_filename[:-4]}.des'
    StoreData(filename, encode_ciphertext)
```

The decryption function has the same parameters as the encryption function. It will decode the key, the IV and the ciphertext from their files. Next, the 3DES algorithm will decrypt the ciphertext and it will be stored as a string in a file with the same name as the original file but with an "_" at the end to be easy to recognize.



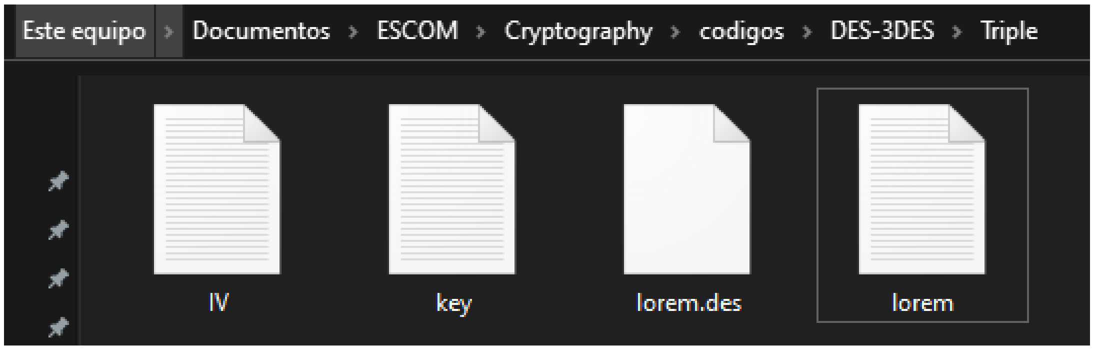
```python
def Predefined3DESDecipher(ciphertext_filename, key_filename, iv_filename):
    key = b64.b64decode(GetData(key_filename))
    iv = b64.b64decode(GetData(iv_filename))
    ciphertext = b64.b64decode(GetData(ciphertext_filename))

    decipher = DES3.new(key, DES3.MODE_CFB, iv)
    decrypt_text = decipher.decrypt(ciphertext)
    plaintext = str(decrypt_text, 'utf-8')

    filename = f'{ciphertext_filename[:-4]}_.txt'
    StoreData(filename, plaintext)
```

For a better view of the runs we will use a folder where we will store the result file. The 5Mb file to encrypt contains a text from a generator text. First, we will create the key file, the IV file and the ciphertext.

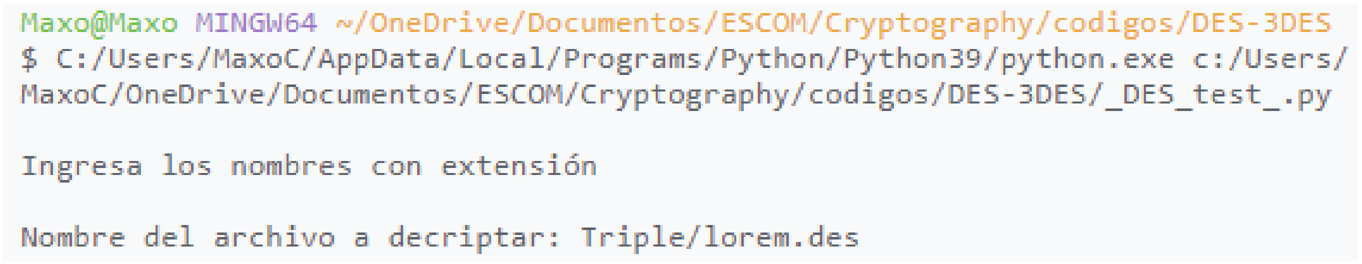


```
Maxo@Maxo MINGW64 ~/OneDrive/Documentos/ESCOM/Cryptography/codigos/DES-3DES
igos/DES-3DES/_DES_test_.py/codi

Ingresa los nombres con extensión

Nombre del archivo a encriptar: Triple/lorem.txt
Nombre del archivo para la clave: Triple/key.txt
Nombre del archivo para el IV: Triple/IV.txt
```

The files were perfectly created.

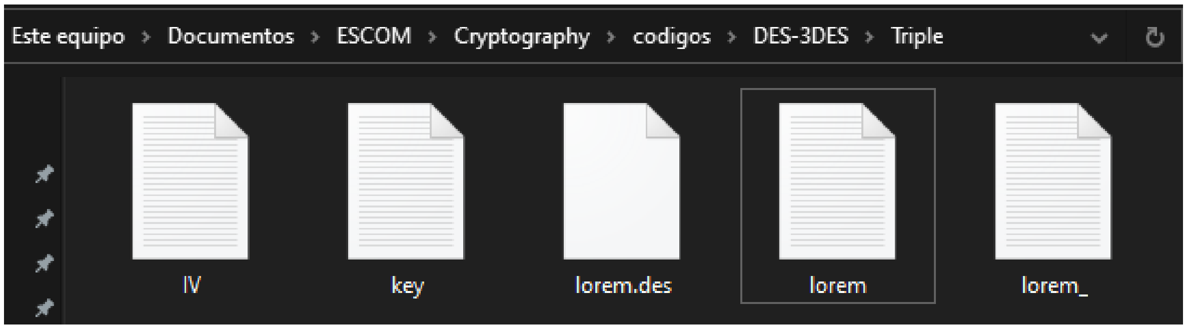

Este equipo › Documentos › ESCOM › Cryptography › codigos › DES-3DES › Triple
IV
key
lorem.des
lorem

Now, we will decrypt the ciphertext from the .des file.



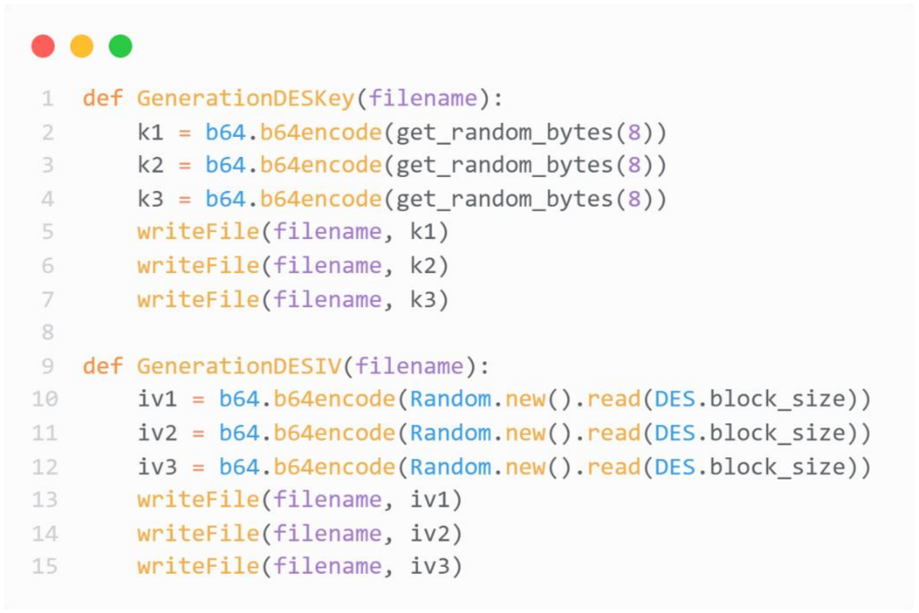
```
Maxo@Maxo MINGW64 ~/OneDrive/Documentos/ESCOM/Cryptography/codigos/DES-3DES
$ C:/Users/MaxoC/AppData/Local/Programs/Python/Python39/python.exe c:/Users/
MaxoC/OneDrive/Documentos/ESCOM/Cryptography/codigos/DES-3DES/_DES_test_.py

Ingresa los nombres con extensión

Nombre del archivo a decriptar: Triple/lorem.des
```

Finally, we have all the files.


Este equipo › Documentos › ESCOM › Cryptography › codigos › DES-3DES › Triple
IV
key
lorem.des
lorem
lorem_

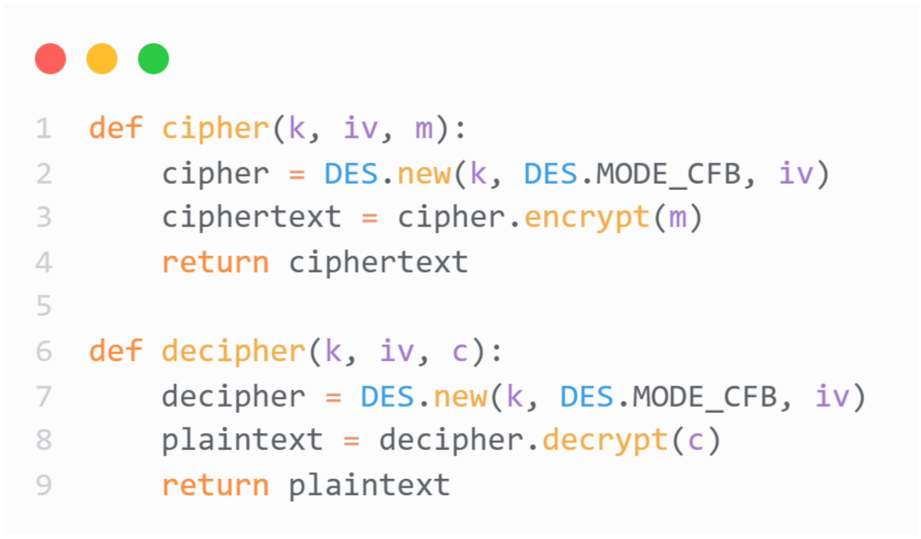
EEE variant.

The way we will calculate the key is different from EDE variant, in this case we will use three 8-length byte keys and three IV’s, they will be calculated same as the other variant. Both of the will be store in files with filenames we choose.



```python
def GenerationDESKey(filename):
    k1 = b64.b64encode(get_random_bytes(8))
    k2 = b64.b64encode(get_random_bytes(8))
    k3 = b64.b64encode(get_random_bytes(8))
    writeFile(filename, k1)
    writeFile(filename, k2)
    writeFile(filename, k3)

def GenerationDESIV(filename):
    iv1 = b64.b64encode(Random.new().read(DES.block_size))
    iv2 = b64.b64encode(Random.new().read(DES.block_size))
    iv3 = b64.b64encode(Random.new().read(DES.block_size))
    writeFile(filename, iv1)
    writeFile(filename, iv2)
    writeFile(filename, iv3)
```

We use two functions to create DES objects to encrypt and to decrypt because of we will use three of each one two make the EEE variant. Both of functions have three parameters; a corresponding key, IV and a message to encrypt or to decrypt, also, DES algorithm will use the CFB mode of operation.



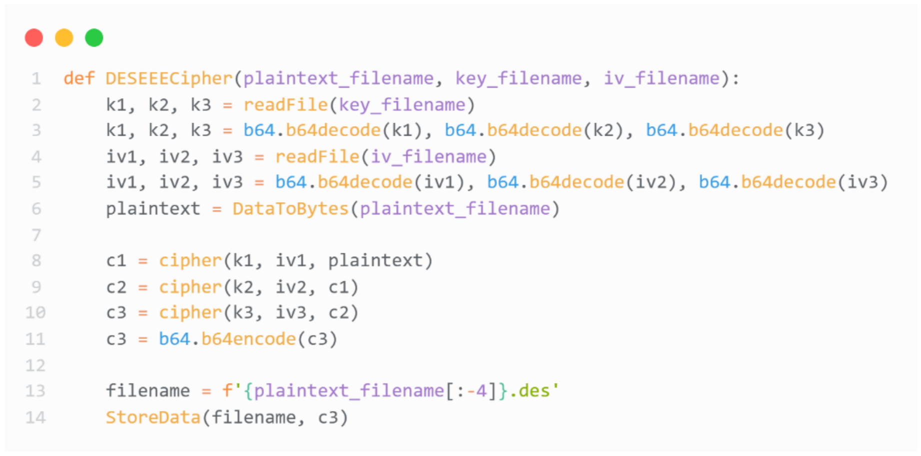
```python
def cipher(k, iv, m):
    cipher = DES.new(k, DES.MODE_CFB, iv)
    ciphertext = cipher.encrypt(m)
    return ciphertext

def decipher(k, iv, c):
    decipher = DES.new(k, DES.MODE_CFB, iv)
    plaintext = decipher.decrypt(c)
    return plaintext
```

The EEE variant encryption function has three parameters; the name of the file of the text, the name of the file of the keys and the name of the file of the IV's.

This function will decode both the keys and the IV's from their files and convert into bytes the chosen text. Then, it will encrypt the plaintext three times, each time with a different key. Finally, the ciphertext will be decode base 64 and store in a file with the same name as the original text but with the .des extension.

The next formula explains the encryption operation in a better way.

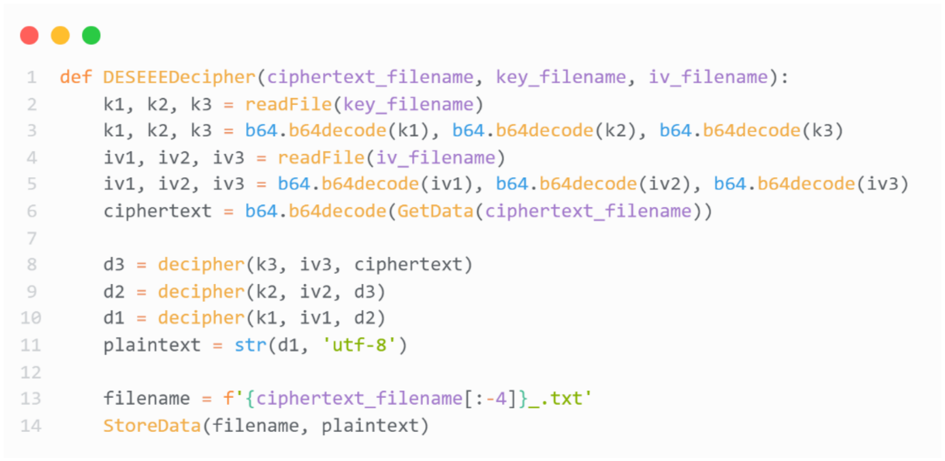
$$c = E_{k3}\,(E_{k2}\,(E_{k1}\,(m)))$$



```python
def DESEEECipher(plaintext_filename, key_filename, iv_filename):
    k1, k2, k3 = readFile(key_filename)
    k1, k2, k3 = b64.b64decode(k1), b64.b64decode(k2), b64.b64decode(k3)
    iv1, iv2, iv3 = readFile(iv_filename)
    iv1, iv2, iv3 = b64.b64decode(iv1), b64.b64decode(iv2), b64.b64decode(iv3)
    plaintext = DataToBytes(plaintext_filename)

    c1 = cipher(k1, iv1, plaintext)
    c2 = cipher(k2, iv2, c1)
    c3 = cipher(k3, iv3, c2)
    c3 = b64.b64encode(c3)

    filename = f'{plaintext_filename[:-4]}.des'
    StoreData(filename, c3)
```

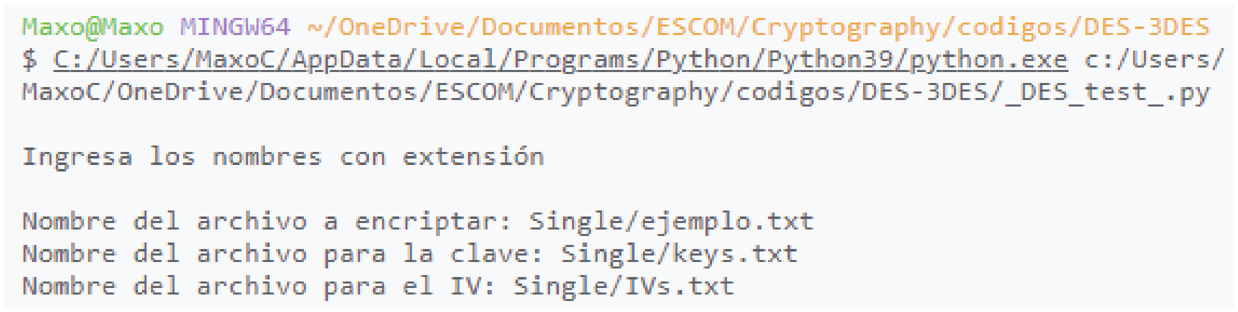
The decryption function has the same parameters as the encryption function and basically do the same as it. But it will do the inverse operation to get the plaintext, it will be store in a file with the same name as the cipher text but it will finish with a "_" and it will have .txt extension.

The decryption formula is: $m = D_{k1}\,(D_{k2}\,(D_{k3}\,(c)))$



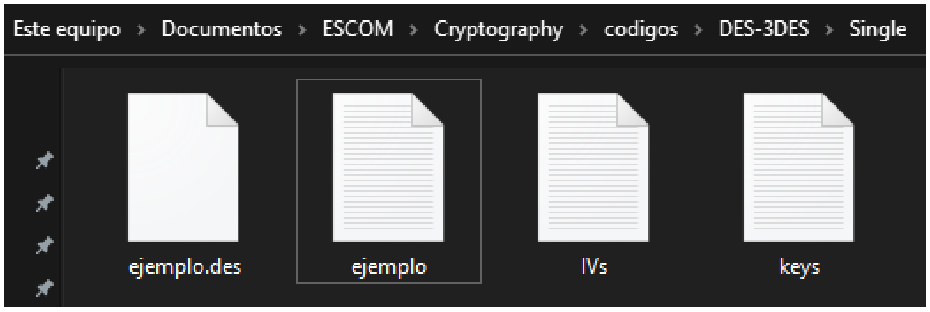
```python
def DESEEEDecipher(ciphertext_filename, key_filename, iv_filename):
    k1, k2, k3 = readFile(key_filename)
    k1, k2, k3 = b64.b64decode(k1), b64.b64decode(k2), b64.b64decode(k3)
    iv1, iv2, iv3 = readFile(iv_filename)
    iv1, iv2, iv3 = b64.b64decode(iv1), b64.b64decode(iv2), b64.b64decode(iv3)
    ciphertext = b64.b64decode(GetData(ciphertext_filename))

    d3 = decipher(k3, iv3, ciphertext)
    d2 = decipher(k2, iv2, d3)
    d1 = decipher(k1, iv1, d2)
    plaintext = str(d1, 'utf-8')

    filename = f'{ciphertext_filename[:-4]}_.txt'
    StoreData(filename, plaintext)
```

As the other variant three files will be created on the auxiliar file, which will be the keys, IV's and the .des files.

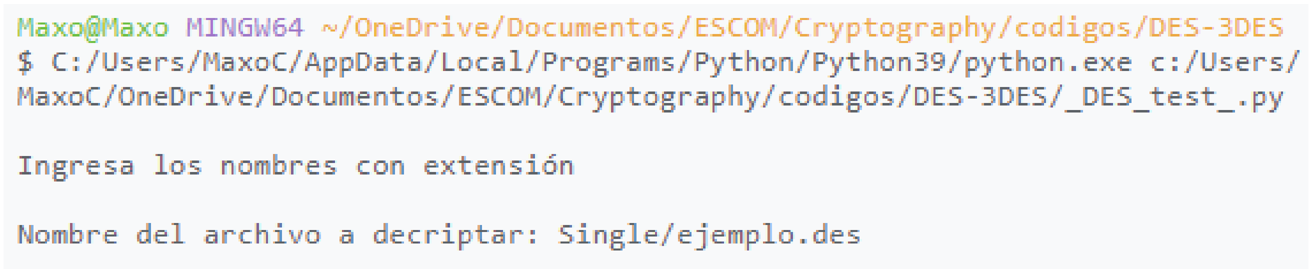


```
Maxo@Maxo MINGW64 ~/OneDrive/Documentos/ESCOM/Cryptography/codigos/DES-3DES
$ C:/Users/MaxoC/AppData/Local/Programs/Python/Python39/python.exe c:/Users/
MaxoC/OneDrive/Documentos/ESCOM/Cryptography/codigos/DES-3DES/_DES_test_.py

Ingresa los nombres con extensión

Nombre del archivo a encriptar: Single/ejemplo.txt
Nombre del archivo para la clave: Single/keys.txt
Nombre del archivo para el IV: Single/IVs.txt
```

On the folder we will have the files after we run the encryption.

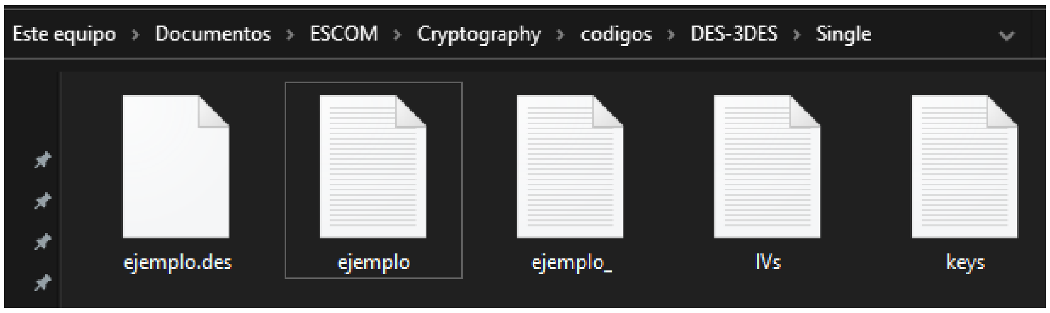

Este equipo › Documentos › ESCOM › Cryptography › codigos › DES-3DES › Single
ejemplo.des
ejemplo
IVs
keys

Now, we will run the decryption and we will have this.



```
Maxo@Maxo MINGW64 ~/OneDrive/Documentos/ESCOM/Cryptography/codigos/DES-3DES
$ C:/Users/MaxoC/AppData/Local/Programs/Python/Python39/python.exe c:/Users/
MaxoC/OneDrive/Documentos/ESCOM/Cryptography/codigos/DES-3DES/_DES_test_.py

Ingresa los nombres con extensión

Nombre del archivo a decriptar: Single/ejemplo.des
```

Finally, on the folder we will have these 5 files.


Este equipo › Documentos › ESCOM › Cryptography › codigos › DES-3DES › Single
ejemplo.des
ejemplo
ejemplo_
IVs
keys