Addis Ababa University

Institute of technology

Department of electrical and computer engineering

Computer Network Security Assighment-2

Assignment II (OTP, 3DES, AES Implementation)

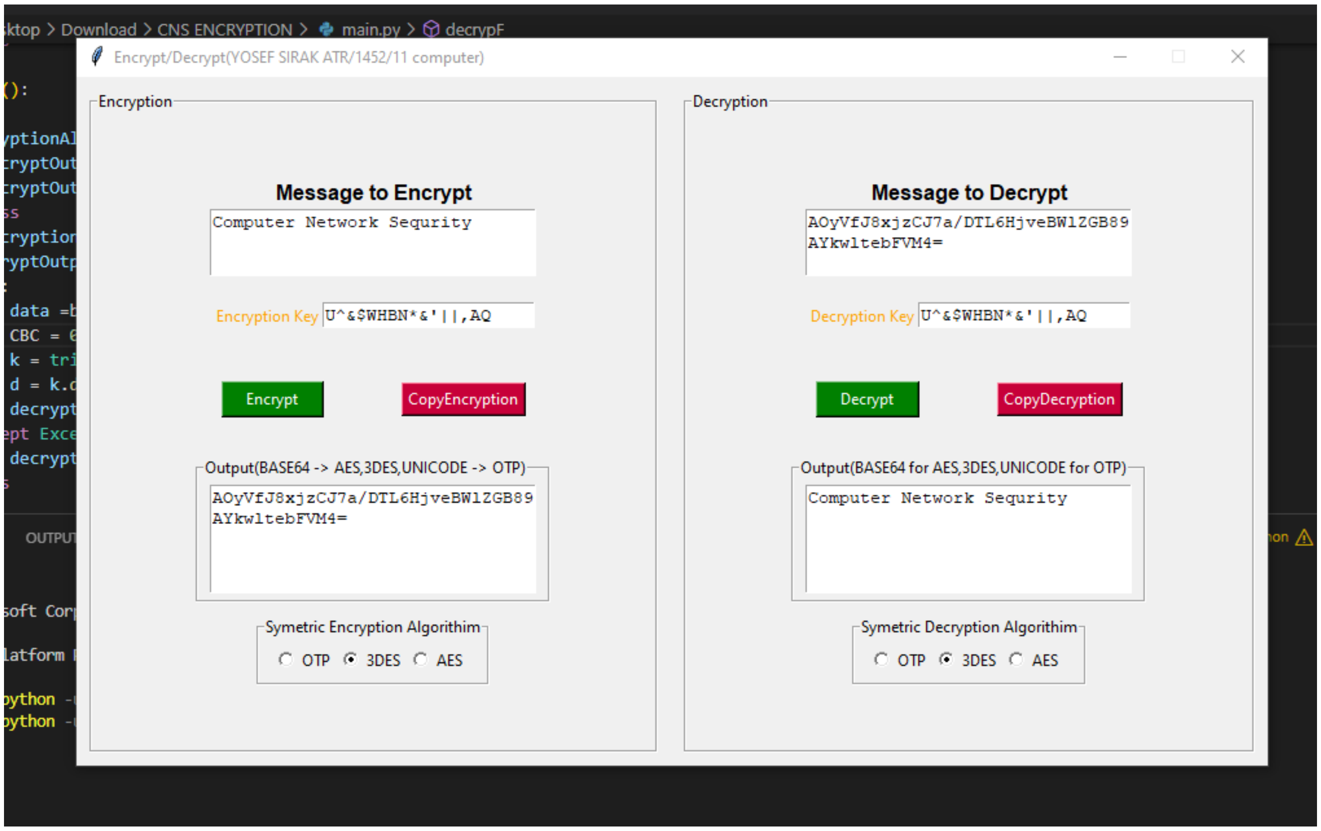
By: Yosef Sirak ATR/1452/11

Instructor: Mr Kinde.M

What I’ve learned from this assignment

- Various encoding techniques UNICODE, BASE64, HEX…
- Implementation of Rijndael (AES Encryption Algorithm paper)
- PKCS#7 padding and other padding techniques
- Cypher block chaining modes of operation (CBC, ECB…)
- Workings of triple des and one time pad
- Python desktop development

<u>GUI</u>


ktop > Download > CNS ENCRYPTION > main.py > decrypF
Encrypt/Decrypt(YOSEF SIRAK ATR/1452/11 computer)
Encryption
Decryption
Encrypt
CopyEncryption
Decrypt
CopyDecryption

Message to Encrypt

```
Computer Network Sequrity
```

Encryption Key `U^&$WHBN*&'||,AQ`

Output(BASE64 -> AES,3DES,UNICODE -> OTP)

```
AOyVfJ8xjzCJ7a/DTL6HjveBWlZGB89
AYkwltebFVM4=
```

Symetric Encryption Algorithim: OTP [✓ 3DES] AES

Message to Decrypt

```
AOyVfJ8xjzCJ7a/DTL6HjveBWlZGB89
AYkwltebFVM4=
```

Decryption Key `U^&$WHBN*&'||,AQ`

Output(BASE64 for AES,3DES,UNICODE for OTP)

```
Computer Network Sequrity
```

Symetric Decryption Algorithim: OTP [✓ 3DES] AES

This project is made as python desktop app with the tkinter gui library. The output of the encryption is BASE64 (because the encryption of aes, 3des results in characters that cannot be displayed) for AES, 3DES and Unicode for OTP.

Gui consists of encryption and decryption frames with their included functionality's the user enters whatever message to encrypt and proceeds to specify a key and choses one of symmetric encryption algorithm choices labeled at the bottom (implemented as radio buttons on the GUI).If any error occurs like the key supplied is not 16 byte for AES or the key length is not equal to message length for one time pad , error is displayed on the output section for encryption frame and the corresponding is true for the decryption frame.

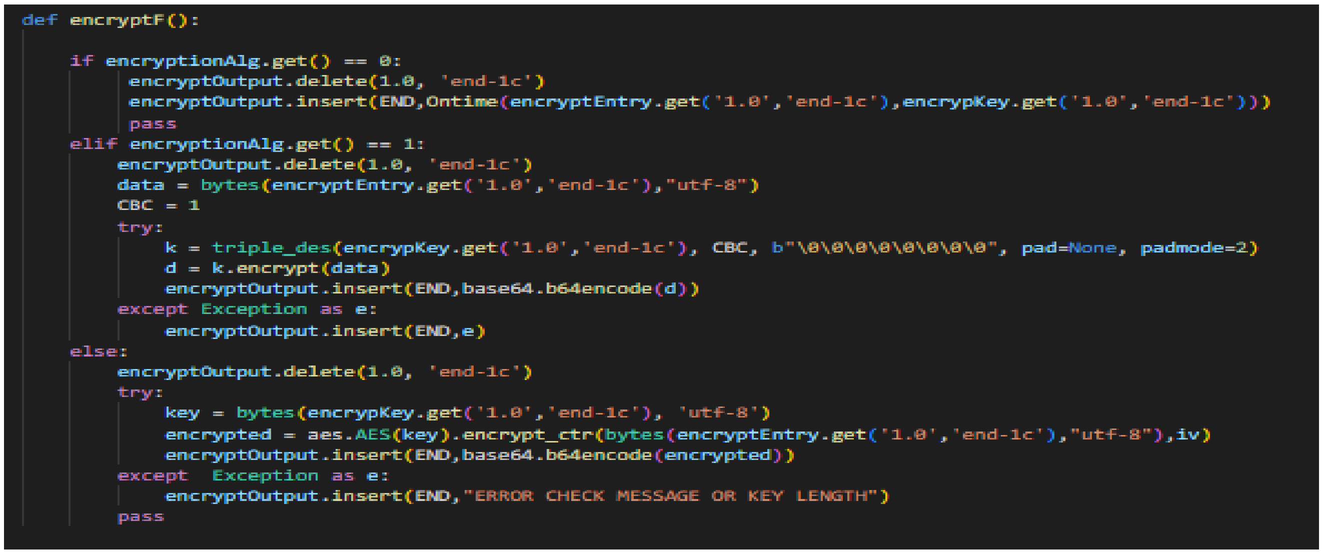
Code Review

This project is divided into 4 modules main.py OntimePad.py TripleDes.py aes.py each module consists of a specific functionality.

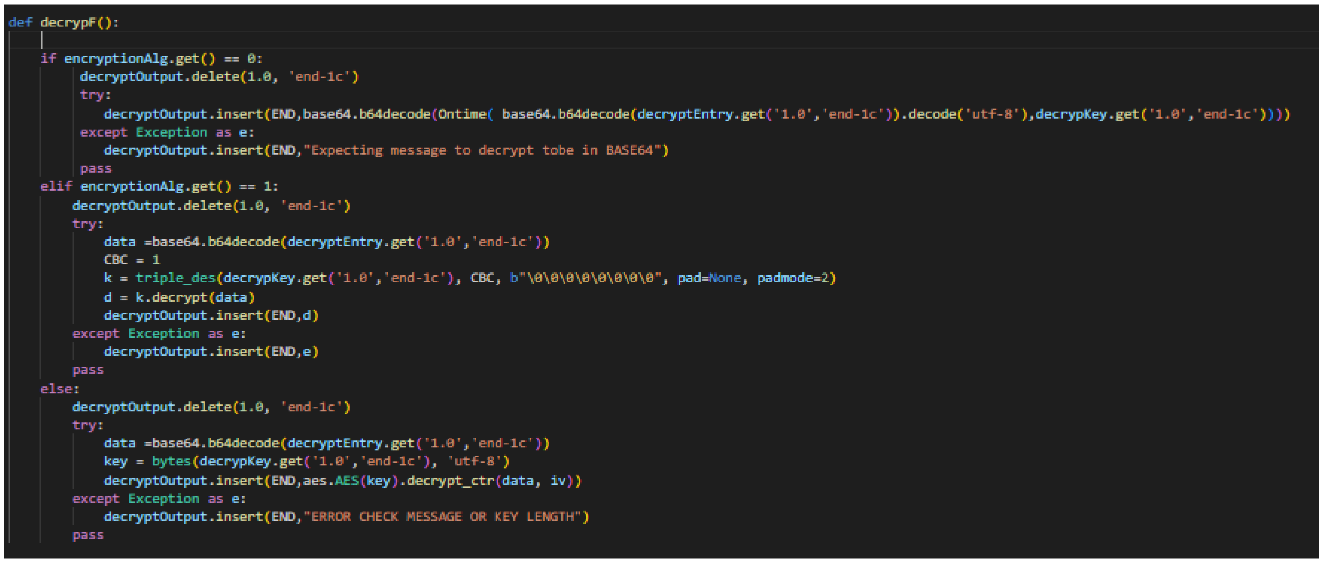
main.py

Main.py is the main part of the program where every part of the ui is draw and the core logic that links ui with its respective functionality's such as encryption, decryption, copy encryption, copy decryption.

Here we find main core logic functions encryptF and decrypF that takes the inserted values from the UI (user input) and sends them to one of the three encryption algorithms (Depending of the choice of the user on the radio buttons) after performing encryption the result is shown on lower label found at encryption box same is true for the decryption



```python
def encryptF():

    if encryptionAlg.get() == 0:
        encryptOutput.delete(1.0, 'end-1c')
        encryptOutput.insert(END,Ontime(encryptEntry.get('1.0','end-1c'),encrypKey.get('1.0','end-1c')))
        pass
    elif encryptionAlg.get() == 1:
        encryptOutput.delete(1.0, 'end-1c')
        data = bytes(encryptEntry.get('1.0','end-1c'),"utf-8")
        CBC = 1
        try:
            k = triple_des(encrypKey.get('1.0','end-1c'), CBC, b"\0\0\0\0\0\0\0\0", pad=None, padmode=2)
            d = k.encrypt(data)
            encryptOutput.insert(END,base64.b64encode(d))
        except Exception as e:
            encryptOutput.insert(END,e)
    else:
        encryptOutput.delete(1.0, 'end-1c')
        try:
            key = bytes(encrypKey.get('1.0','end-1c'), 'utf-8')
            encrypted = aes.AES(key).encrypt_ctr(bytes(encryptEntry.get('1.0','end-1c'),"utf-8"),iv)
            encryptOutput.insert(END,base64.b64encode(encrypted))
        except  Exception as e:
            encryptOutput.insert(END,"ERROR CHECK MESSAGE OR KEY LENGTH")
        pass
```

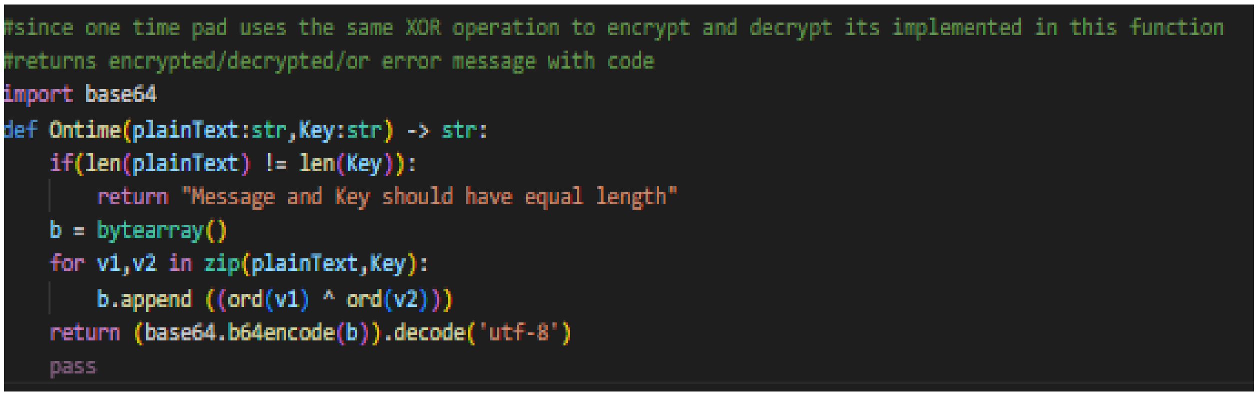


```python
def decrypF():

    if encryptionAlg.get() == 0:
        decryptOutput.delete(1.0, 'end-1c')
        try:
            decryptOutput.insert(END,base64.b64decode(Ontime( base64.b64decode(decryptEntry.get('1.0','end-1c')).decode('utf-8'),decrypKey.get('1.0','end-1c'))))
        except Exception as e:
            decryptOutput.insert(END,"Expecting message to decrypt tobe in BASE64")
        pass
    elif encryptionAlg.get() == 1:
        decryptOutput.delete(1.0, 'end-1c')
        try:
            data =base64.b64decode(decryptEntry.get('1.0','end-1c'))
            CBC = 1
            k = triple_des(decrypKey.get('1.0','end-1c'), CBC, b"\0\0\0\0\0\0\0\0", pad=None, padmode=2)
            d = k.decrypt(data)
            decryptOutput.insert(END,d)
        except Exception as e:
            decryptOutput.insert(END,e)
        pass
    else:
        decryptOutput.delete(1.0, 'end-1c')
        try:
            data =base64.b64decode(decryptEntry.get('1.0','end-1c'))
            key = bytes(decrypKey.get('1.0','end-1c'), 'utf-8')
            decryptOutput.insert(END,aes.AES(key).decrypt_ctr(data, iv))
        except Exception as e:
            decryptOutput.insert(END,"ERROR CHECK MESSAGE OR KEY LENGTH")
        pass
```

I ve also added copy encryption and decryption functions to copy the encrypted and decrypted output respectively.

OntimePad.py

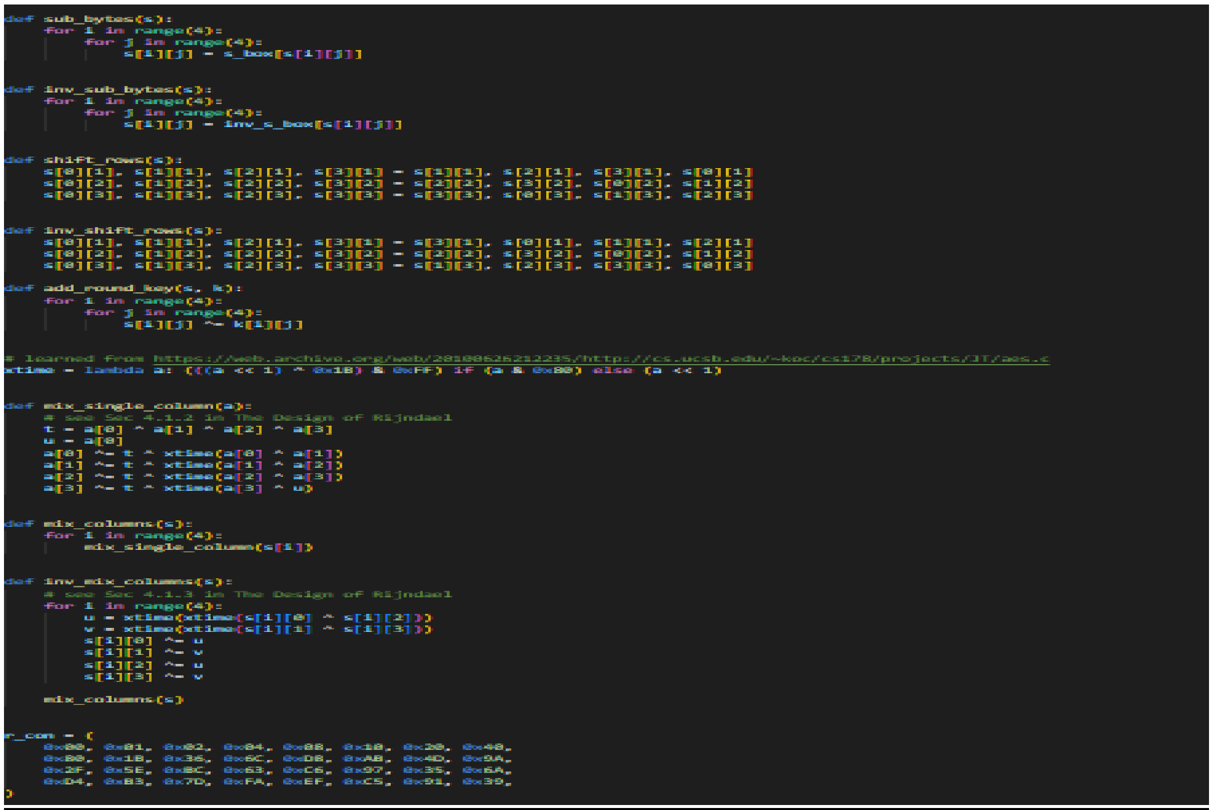
This module contains **my** implementation of onetime pad encryption/decryption algorithm which outputs its result in BASE64 because of the nature of onetime pad encrypted output characters might not be printable that's why I've converted the output to base64.It XORS plaintext with key(character by character) and returns an error if the key length and plaintext length is not equal.



```python
#since one time pad uses the same XOR operation to encrypt and decrypt its implemented in this function
#returns encrypted/decrypted/or error message with code
import base64
def Ontime(plainText:str,Key:str) -> str:
    if(len(plainText) != len(Key)):
        return "Message and Key should have equal length"
    b = bytearray()
    for v1,v2 in zip(plainText,Key):
        b.append ((ord(v1) ^ ord(v2)))
    return (base64.b64encode(b)).decode('utf-8')
    pass
```

aes.py

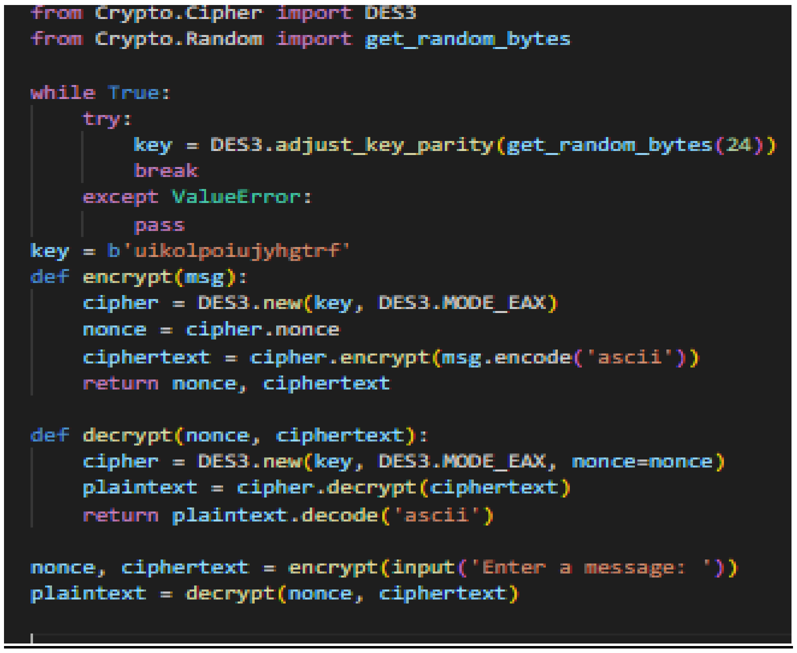
This module contains a **cryptographic library implementation** of AES (symmetric encryption algorithm) that I've used **.**This implementation shows all the steps of aes algoritim and has embedded links showing different articles to learn specific steps of aes which I found it to be helpful.



```python
def sub_bytes(s):
    for i in range(4):
        for j in range(4):
            s[i][j] = s_box[s[i][j]]


def inv_sub_bytes(s):
    for i in range(4):
        for j in range(4):
            s[i][j] = inv_s_box[s[i][j]]


def shift_rows(s):
    s[0][1], s[1][1], s[2][1], s[3][1] = s[1][1], s[2][1], s[3][1], s[0][1]
    s[0][2], s[1][2], s[2][2], s[3][2] = s[2][2], s[3][2], s[0][2], s[1][2]
    s[0][3], s[1][3], s[2][3], s[3][3] = s[3][3], s[0][3], s[1][3], s[2][3]


def inv_shift_rows(s):
    s[0][1], s[1][1], s[2][1], s[3][1] = s[3][1], s[0][1], s[1][1], s[2][1]
    s[0][2], s[1][2], s[2][2], s[3][2] = s[2][2], s[3][2], s[0][2], s[1][2]
    s[0][3], s[1][3], s[2][3], s[3][3] = s[1][3], s[2][3], s[3][3], s[0][3]
def add_round_key(s, k):
    for i in range(4):
        for j in range(4):
            s[i][j] ^= k[i][j]


# learned from https://web.archive.org/web/20100626212235/http://cs.ucsb.edu/~koc/cs178/projects/JT/aes.c
xtime = lambda a: (((a << 1) ^ 0x1B) & 0xFF) if (a & 0x80) else (a << 1)


def mix_single_column(a):
    # see Sec 4.1.2 in The Design of Rijndael
    t = a[0] ^ a[1] ^ a[2] ^ a[3]
    u = a[0]
    a[0] ^= t ^ xtime(a[0] ^ a[1])
    a[1] ^= t ^ xtime(a[1] ^ a[2])
    a[2] ^= t ^ xtime(a[2] ^ a[3])
    a[3] ^= t ^ xtime(a[3] ^ u)


def mix_columns(s):
    for i in range(4):
        mix_single_column(s[i])


def inv_mix_columns(s):
    # see Sec 4.1.3 in The Design of Rijndael
    for i in range(4):
        u = xtime(xtime(s[i][0] ^ s[i][2]))
        v = xtime(xtime(s[i][1] ^ s[i][3]))
        s[i][0] ^= u
        s[i][1] ^= v
        s[i][2] ^= u
        s[i][3] ^= v

    mix_columns(s)


r_con = (
    0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40,
    0x80, 0x1B, 0x36, 0x6C, 0xD8, 0xAB, 0x4D, 0x9A,
    0x2F, 0x5E, 0xBC, 0x63, 0xC6, 0x97, 0x35, 0x6A,
    0xD4, 0xB3, 0x7D, 0xFA, 0xEF, 0xC5, 0x91, 0x39,
)
```

TripleDes.py and TripleDes2.py

Triple des2.py contains **my** implementation of triple DES algorithm using python package named PyCryptodome**(** PyCryptodome is a self-contained Python package of low-level cryptographic primitives.**)**



```python
from Crypto.Cipher import DES3
from Crypto.Random import get_random_bytes

while True:
    try:
        key = DES3.adjust_key_parity(get_random_bytes(24))
        break
    except ValueError:
        pass
key = b'uikolpoiujyhgtrf'
def encrypt(msg):
    cipher = DES3.new(key, DES3.MODE_EAX)
    nonce = cipher.nonce
    ciphertext = cipher.encrypt(msg.encode('ascii'))
    return nonce, ciphertext

def decrypt(nonce, ciphertext):
    cipher = DES3.new(key, DES3.MODE_EAX, nonce=nonce)
    plaintext = cipher.decrypt(ciphertext)
    return plaintext.decode('ascii')

nonce, ciphertext = encrypt(input('Enter a message: '))
plaintext = decrypt(nonce, ciphertext)
```

TripleDes.py module contains **standard cryptographic library implementation** Triple DES and DES also shows how each step of the encryption. Decryption process.This is the module I linked with the ui to perform triple des encryptions and decryptions because of its speed and reliability

How to Run

The app is compiled into exe and is main folder with the name encrypt_decrypt