



# INSTITUTO POLITÉCNICO NACIONAL ESCUELA SUPERIOR DE CÓMPUTO



## Cryptography

### “Block Cipher/Operation Modes”

#### Abstract

Using the DES and AES Algorithm, verify how they work and how they can operate with all their operation modes, which are ECB, CBC, CFB, OFB and CTR.

**By:**

Portilla Martinez Jose David

**Professor:**

MSc. NIDIA ASUNCIÓN CORTEZ DUARTE

October 2019



# Index

## Contenido

Introduction.....	1
Literature review.....	1
Software (libraries, packages, tools).....	4
Procedure.....	5
Results.....	7
Discussion.....	9
Conclusions.....	9
References .....	9
Code.....	10

## Introduction

Within this report, the reader will learn what the DES/AES Algorithm are and the basic of some of their operation modes. Also, it'll be shown how the application was made, a quick lesson in how to use it and the outcomes given a set of tests.

## Literature review

### Block Ciphers

#### What is a Block Cipher?

There are two main types of ciphers: block and stream ciphers. In a stream cipher, the plaintext is encrypted one bit at a time. In a block cipher, the plaintext is broken into blocks of a set length and the bits in each block are encrypted together.

#### Data Encryption Standard (DES)

DES is an algorithm developed by IBM as a submission to the US National Bureau of Standards (precursor to National Institute of Standards and Technology) for a contest to select a government-approved block cipher. DES is a Feistel cipher with a 64-bit block size and a 56-bit key. Due to (legitimate) concerns about the security of a 56-bit key, it became common to run a plaintext through three DES encryptions in sequence, each using a different key (producing a 168-bit key). This variation is called 3DES or Triple DES.

#### Advanced Encryption Standard (AES)

AES was developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, as a submission to the National Institute of Technology (NIST) for a competition to replace DES. The algorithm, originally called Rijndael, uses a fixed block size of 128 bits and key sizes of 128, 192, or 256 bits. Each version uses a slightly different key schedule, which will be described in a later post.

### Modes of Operation

One of the main issues with block ciphers is that they only allow you to encrypt messages the same size as their block length. If you're using TEA, which has a block size of 64 bits, to encrypt a 65-bit message, you need a way to define how the second block should be encrypted. The solution to this is called block cipher modes of operation.

## Electronic Code Book (ECB)

Electronic Code Book (ECB) is the simplest block cipher mode of operation. In this mode, as shown in fig. 1, each block of plaintext is encrypted separately.

The "Block Cipher Encryption" in this diagram could be DES or AES cipher from above or any other block cipher. The main disadvantage to this mode is that identical plaintexts encrypted with the same key create identical ciphertexts, which allows an attacker to learn some information about the encrypted message based solely on the ciphertext.

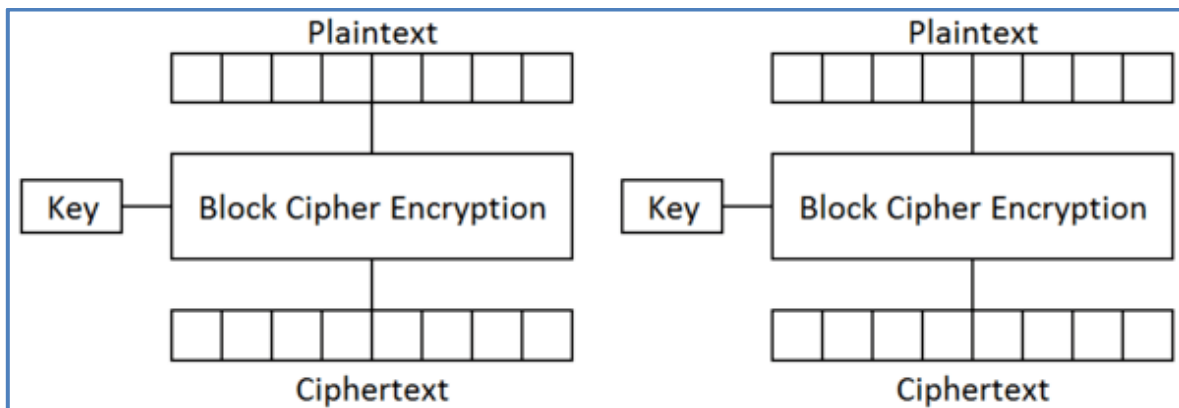


Figure 1: ECB Mode

## Cipher Block Chaining (CBC)

In the cipher block chaining (CBC) mode of operation, an initialization vector (IV) is exclusive-ored with the plaintext prior to encryption. For the first round of encryption, this is a random, public value. For subsequent rounds, it is the ciphertext of the previous round. This is intended to fix the issue with ECB mode where identical plaintext blocks create identical ciphertext blocks. The diagram can be seen in fig. 2.

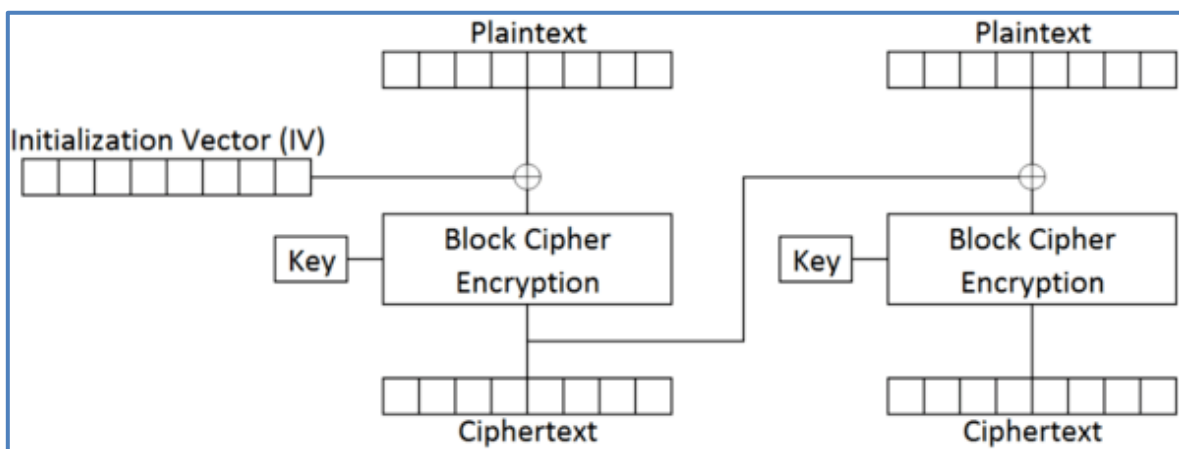


Figure 2: CBC Mode

## Cipher Feedback (CFB)

The cipher feedback (CFB) mode differs from the previous two in that the plaintext never passes through the encryption algorithm at all. Instead an initialization vector (IV) is encrypted and the result is exclusive-ored with the plaintext to create the ciphertext of a block. This is the equivalent of encrypting the plaintext with a one-time pad generating from the encryption of the IV. Like CBC mode, this IV is a random value for the first block and the previous block's ciphertext. The diagram can be seen in fig. 3.

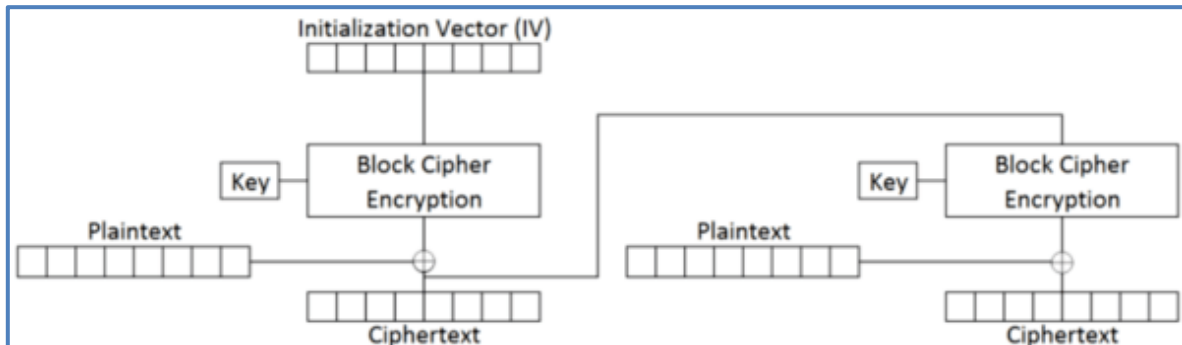


Figure 3: CFB Mode

## Output Feedback (OFB)

The output feedback (OFB) mode of operation is almost identical to cipher feedback mode. The only difference is what is used as the initialization vector for every round after the first. In cipher feedback mode, the output of the encryption is exclusive-ored with the plaintext and this value is used as the next block's IV. In output feedback mode, the output of the encryption is used as the next block's IV. As a result, encryption of the same plaintext with the same key using CFB and OFB modes will produce the same ciphertext for the first block but different ones for every other block. The diagram can be seen in fig. 4.

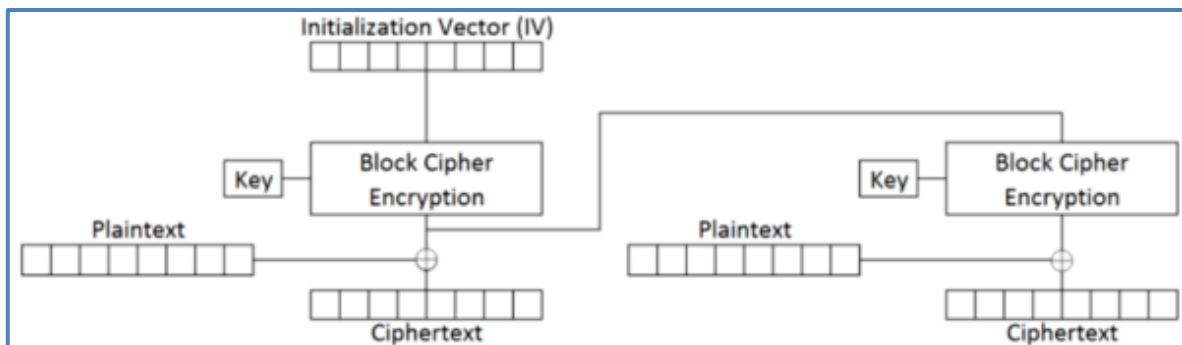


Figure 4: OFB Mode

## Counter (CTR)

The counter (CTR) mode of operation differs from all of the others that we have seen so far. Like ECB mode, every encryption operation is completely separate, which is useful for parallelization of encryption (since each block can be encrypted simultaneously). Counter mode also uses a non-plaintext output to encryption (like the feedback modes), but, instead of an initialization vector, it uses a combination of a nonce and a counter. The nonce is a random number used for all blocks of an encryption operation and the counter is exactly what it sounds like: a value that starts at zero for block zero and increments to one for block one and so on.

This combination guarantees that the same values will not pass through the encryption algorithm in the same encryption session (where every block will have the same nonce but different counter values) or the same blocks in different sessions (where every block will have the same counter value but different nonces). Like the feedback modes of operation (OFB and CFB), the plaintext is exclusive-ored with the output of the encryption operation to produce the ciphertext. The diagram can be seen in fig. 5.

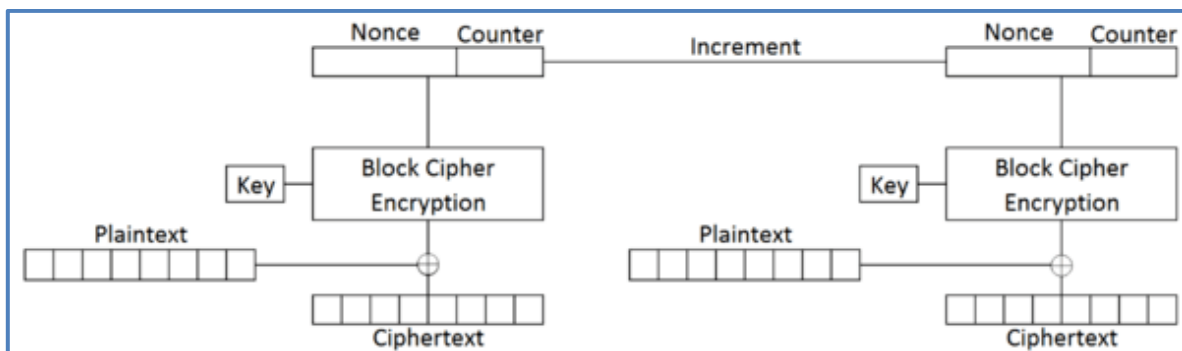


Figure 5: CTR Mode

## Software (libraries, packages, tools)

To develop this application Python 3 was used, alongside these libraries:

- Tkinter (toolkit for GUI's)
- PyCryptodome (libraries for AES/DES)

## Procedure

Block Diagram (How to use application)

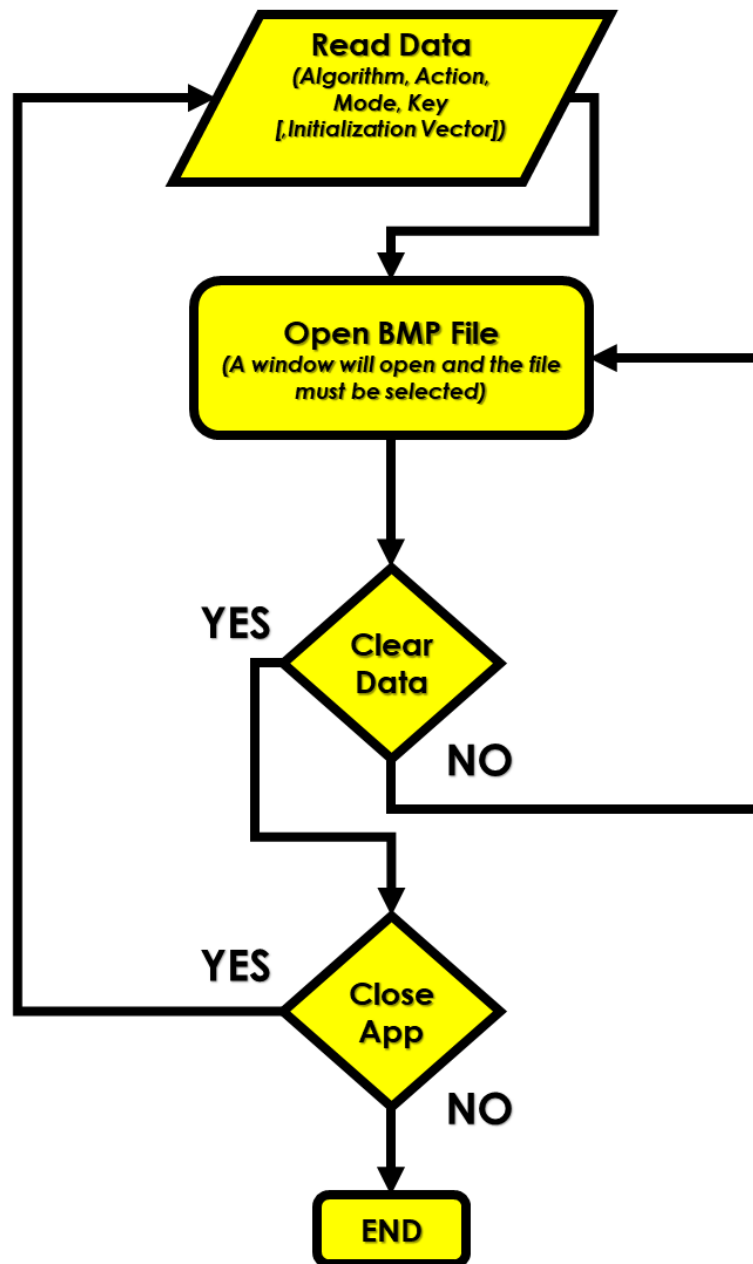


Figure 6: Diagram

No new algorithm was implemented in the making of this application. So, the use of PyCryptodome when implementing the AES/DES Algorithm was essential, next a quick guide in how to use this library:

- 1) Import libraries:

```
from Crypto.Cipher import DES, AES
```

- 2) Create an environment-like module, passing the arguments: key, mode of operation and, if needed, the initialization vector:

```
suitDES = DES.new(key, DES.MODE_ECB)
```

- 3) With the module created, now it is possible to encrypt o decrypt any information (passed as bytes):













```
cipherImage = suitDES.encrypt(plainImage)  
cipherImage = suitDES.decrypt(plainImage)
```




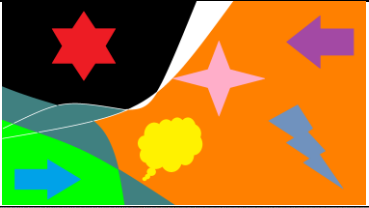

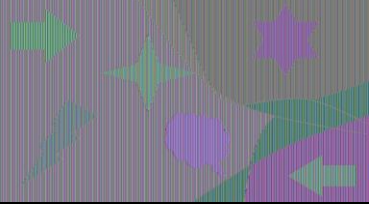



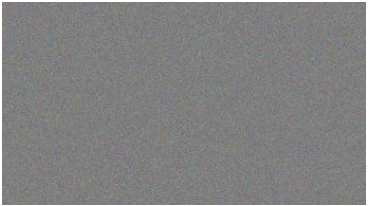




## Results

We can see if the application works when encrypting to files, two BMP images. The results of the test can be shown in the next tables:

### DES Algorithm

Operation	japan.bmp	shapes.bmp
Original		
ECB		
CBC		
CFB		
OFB		
CTR		

## AES Algorithm

Operation	japan.bmp	shapes.bmp
Original		
ECB		
CBC		
CFB		
OFB		
CTR		

## Discussion

At first glance it appears that both results (encrypting using AES and encryption using DES) are the same, at least the encrypted image but if we look closer we can see some difference in the ECB mode of operation, the color are somewhat different but other than, probably that's where the visually differences end.

Everything else can only be notice if we look at how the algorithms work, but because (as it wasn't required) the algorithms weren't implemented from scratch, if fact help was used from libraries that pretty much did all the work.

## Conclusions

Sometimes when we want to encrypt something we wonder if my information might be somewhat simplistic, and if that could affect the encryption in some way. And in fact, it affects the encryption, but only in some cases, with block cipher more specifically in one case, with the ECB mode of operation we could see how the BMP image seemed that only is was distorted but it was obvious what the image was, but when using every other mode of operation the images was completely distorted and we couldn't identify the picture anymore.

One thing that complicated the practice a little bit was de CTR mode of operation, because of it required a couple more of arguments, but nothing extremely complicated.

## References

- Paulbourke.net. (2019). *BMP files*. [online] Available at: <http://paulbourke.net/dataformats/bmp/> [Accessed 16 Oct. 2019].
- Commonlounge.com. (2019). *Block Ciphers and Modes of Operation | CommonLounge*. [online] Available at: <https://www.commonlounge.com/discussion/6747358d828a45c99f61f4c09ff2f371> [Accessed 16 Oct. 2019].
- Auth0 - Blog. (2019). *Image Processing in Python with Pillow*. [online] Available at: <https://auth0.com/blog/image-processing-in-python-with-pillow/> [Accessed 16 Oct. 2019].

## Code

### aesDES.py

```
from tkinter.filedialog import asksaveasfilename, askopenfilename
from Crypto.Cipher import DES, AES
from PIL import Image
import tkinter as tk
import struct
import ntpath
import os

blockSize = 8
paddingChar = '0'
nonce = 0

def openImage():
    path = askopenfilename(filetypes = (("BMP Image File",
    "*.bmp"), ("All Files", "*..*")), title = "Choose a image.")
    try:
        with open(path, 'rb') as bmpInfo:
            bmpInfo.read(2).decode()
            struct.unpack('I', bmpInfo.read(4))
            struct.unpack('H', bmpInfo.read(2))
            struct.unpack('H', bmpInfo.read(2))
            offset = struct.unpack('I', bmpInfo.read(4))[0]
            return [offset, path]
    except:
        return -1

def saveImage(image, imageName):
    path = asksaveasfilename(initialfile = imageName, filetypes =
    (("BMP Image File", "*.bmp"), ("All Files", "*..*")), title = "Save the
    image.")
    with open(path, 'wb') as imageFile:
        imageFile.write(image)

def encryptionDES(imagePath, offset, key, initVector, opMode, cdMode):
    imageFile = open(imagePath, 'rb')
    textImage = imageFile.read()
    header = textImage[:offset]
    plainImage = textImage[offset:] + (((blockSize -
    len(textImage[offset:])) % blockSize) * paddingChar).encode()
    if (opMode == 0):
        suitDES = DES.new(key, DES.MODE_ECB)
        cipherImage = suitDES.encrypt(plainImage)
```

```

        ecMode = 'ecb'
    if (opMode == 1):
        suitDES = DES.new(key, DES.MODE_CBC, initVector)
        cipherImage = suitDES.encrypt(plainImage)
        ecMode = 'cbc'
    if (opMode == 2):
        suitDES = DES.new(key, DES.MODE_CFB, initVector)
        cipherImage = suitDES.encrypt(plainImage)
        ecMode = 'cfb'
    if (opMode == 3):
        suitDES = DES.new(key, DES.MODE_OFB, initVector)
        cipherImage = suitDES.encrypt(plainImage)
        ecMode = 'ofb'
    if (opMode == 4):
        suitDES = DES.new(key, DES.MODE_CTR, nonce=b'',
initial_value=initVector)
        cipherImage = suitDES.encrypt(plainImage)
        ecMode = 'ctr'
    saveImage(header + cipherImage, ntpath.basename(imagePath)[:4]
+ '_des_' + cdMode + '_' + ecMode + '.bmp')

```

```

def encryptionAES(imagePath, offset, key, initVector, opMode, cdMode):
    imageFile = open(imagePath, 'rb')
    textImage = imageFile.read()
    header = textImage[:offset]
    plainImage = textImage[offset:] + (((blockSize -
len(textImage[offset:])) % blockSize) * paddingChar).encode()
    if (opMode == 0):
        suitAES = AES.new(key, AES.MODE_ECB)
        cipherImage = suitAES.encrypt(plainImage)
        ecMode = 'ecb'
    if (opMode == 1):
        suitAES = AES.new(key, AES.MODE_CBC, initVector)
        cipherImage = suitAES.encrypt(plainImage)
        ecMode = 'cbc'
    if (opMode == 2):
        suitAES = AES.new(key, AES.MODE_CFB, initVector)
        cipherImage = suitAES.encrypt(plainImage)
        ecMode = 'cfb'
    if (opMode == 3):
        suitAES = AES.new(key, AES.MODE_OFB, initVector)
        cipherImage = suitAES.encrypt(plainImage)
        ecMode = 'ofb'

```

```

        if (opMode == 4):
            suitAES = AES.new(key, AES.MODE_CTR, nonce=b'',
initial_value=initVector)
            cipherImage = suitAES.encrypt(plainImage)
            ecMode = 'ctr'
            saveImage(header + cipherImage, ntpath.basename(imagePath)[:4]
+ '_aes_' + cdMode + '_' + ecMode + '.bmp')

```

```

def decryptionAES(imagePath, offset, key, initVector, opMode, cdMode):
    imageFile = open(imagePath, 'rb')
    textImage = imageFile.read()
    header = textImage[:offset]
    plainImage = textImage[offset:] + (((blockSize -
len(textImage[offset:])) % blockSize) * paddingChar).encode()
    if (opMode == 0):
        suitAES = AES.new(key, AES.MODE_ECB)
        cipherImage = suitAES.decrypt(plainImage)
        ecMode = 'ecb'
    if (opMode == 1):
        suitAES = AES.new(key, AES.MODE_CBC, initVector)
        cipherImage = suitAES.decrypt(plainImage)
        ecMode = 'cbc'
    if (opMode == 2):
        suitAES = AES.new(key, AES.MODE_CFB, initVector)
        cipherImage = suitAES.decrypt(plainImage)
        ecMode = 'cfb'
    if (opMode == 3):
        suitAES = AES.new(key, AES.MODE_OFB, initVector)
        cipherImage = suitAES.decrypt(plainImage)
        ecMode = 'ofb'
    if (opMode == 4):
        suitAES = AES.new(key, AES.MODE_CTR, nonce=b'',
initial_value=initVector)
        cipherImage = suitAES.decrypt(plainImage)
        ecMode = 'ctr'
    saveImage(header + cipherImage, ntpath.basename(imagePath)[:4]
+ '_aes_' + cdMode + '_' + ecMode + '.bmp')

```

## imageEncryption.py

```
from aesDES import *
from tkinter import *
from tkinter import ttk
from tkinter import messagebox
import ctypes

root = Tk()

fNormal = ("Garamond", 32, "bold")
fTitle = ("Garamond", 38, "bold")
fButton = ("Century Gothic", 28)
bgColor = '#292826'
fgColor = '#F9D342'

varAlgorithm, varAction, varMode = StringVar(), StringVar(),
StringVar()
varKey, varVec = StringVar(), StringVar()
modeRadio = []

aesRadio, desRadio, encRadio, decRadio = Radiobutton(root),
Radiobutton(root), Radiobutton(root), Radiobutton(root)
for i in range(5):
    i = Radiobutton(root, variable = varMode, indicatoron = 0, font =
fButton, selectcolor = bgColor)
    modeRadio.append(i)
vecEntry = Entry(root, bd = 3, textvariable = varVec, font = fButton,
state = DISABLED, show = "*")
keyEntry = Entry(root, bd = 3, textvariable = varKey, font = fButton,
show = "*")

def clearData():
    aesRadio.deselect()
    desRadio.deselect()
    encRadio.deselect()
    decRadio.deselect()
    for i in range(5):
        modeRadio[i].deselect()
    keyEntry.delete(0, 'end')
    vecEntry.delete(0, 'end')
def closingVerified():
    request = messagebox.askquestion('Exit Application', 'Are you sure
you want to exit the app?', icon = 'warning')
    if request == 'yes':
        root.destroy()
    return
```

```

def vecNeeded():
    if (int(varMode.get())):
        vecEntry.config(state = NORMAL)
    else:
        vecEntry.delete(0, 'end')
        vecEntry.config(state = DISABLED)
    return

def startAction():
    alg, act, mod = varAlgorithm.get(), varAction.get(), varMode.get()
    key, init = varKey.get(), varVec.get()
    if (act == '' or alg == '' or mod == '' or key == ''):
        messagebox.showwarning("Data Incomplete", "You need to choose
from every field")
        return
    else:
        if (int(alg)):
            if (len(key) != 8 and key != ''):
                messagebox.showwarning("Key Invalid", "Your key's
length must be 8 characters")
                return
            else:
                if (len(key) != 16 and key != ''):
                    messagebox.showwarning("Key Invalid", "Your key's
length must be 16 characters")
                    return
                if (mod != '' and int(mod)):
                    if (init == ''):
                        messagebox.showwarning("Data Incomplete", "You need to
choose from every field")
                        return
                    if (int(alg)):
                        if (len(init) != 8):
                            messagebox.showwarning("Init Vector Invalid", "Your
init vector's length must be 8 characters")
                            return
                        else:
                            if (len(init) != 16):
                                messagebox.showwarning("Init Vector Invalid", "Your
init vector's length must be 16 characters")
                                return
                    offset, imagePath = openImage()
                    if (imagePath == -1): messagebox.showerror("Error", "Image can't
load.")

```



```

else:
    if (int(alg)):
        if (act == 'e'): encryptionDES(imagePath, offset,
key.encode(), init.encode(), int(mod), act)
        if (act == 'd'): decryptionDES(imagePath, offset,
key.encode(), init.encode(), int(mod), act)
    else:
        if (act == 'e'): encryptionAES(imagePath, offset,
key.encode(), init.encode(), int(mod), act)
        if (act == 'd'): decryptionAES(imagePath, offset,
key.encode(), init.encode(), int(mod), act)
        request = messagebox.askquestion('Operation Completed', 'Do you
want to clear the form?', icon = 'info')
        if request == 'yes':
            clearData()
    return

```

```

if __name__ == "__main__":
    root.minsize(width = 840, height = 900)
    root.maxsize(width = 840, height = 900)
    root.configure(bg = bgColor)
    root.title('Image Encryption App')
    root.iconbitmap('Icons/crypto.ico')

    mainMenu = Menu(root)
    mainMenu.add_command(label = "Close", command = quit)
    mainMenu.add_separator()
    mainMenu.add_command(label = "Clear data", command = clearData)
    root.config(menu = mainMenu)

    Label(root, text = "Image Encryption", bg = fgColor, fg = bgColor,
font = fTitle).place(x = 95, y = 30, width = 650, height = 60)
    Label(root, text = "Algorithm:", bg = bgColor, fg = fgColor, font =
fNormal).place(x = 128, y = 150)
    aesRadio.config(text = "AES", variable = varAlgorithm, value = '0',
indicatoron = 0, font = fButton, selectcolor = bgColor)
    aesRadio.place(x = 536, y = 150, width = 100, height = 60)
    desRadio.config(text = "DES", variable = varAlgorithm, value = '1',
indicatoron = 0, font = fButton, selectcolor = bgColor)
    desRadio.place(x = 436, y = 150, width = 100, height = 60)

    Label(root, text = "Action:", bg = bgColor, fg = fgColor, font =
fNormal).place(x = 128, y = 250)

```

```

    encRadio.config(text = "DECRYPT", variable = varAction, value =
'd', indicatoron = 0, font = fButton, selectcolor = bgColor)
    encRadio.place(x = 535, y = 250, width = 180, height = 60)
    decRadio.config(text = "ENCRYPT", variable = varAction, value =
'e', indicatoron = 0, font = fButton, selectcolor = bgColor)
    decRadio.place(x = 355, y = 250, width = 180, height = 60)

    Label(root, text = "Mode:", bg = bgColor, fg = fgColor, font =
fNormal).place(x = 128, y = 380)
    modeRadio[0].config(text = "ECB", value = '0', command = vecNeeded)
    modeRadio[0].place(x = 436, y = 350, width = 100, height = 60)
    modeRadio[1].config(text = "CBC", value = '1', command = vecNeeded)
    modeRadio[1].place(x = 536, y = 350, width = 100, height = 60)
    modeRadio[2].config(text = "CFB", value = '2', command = vecNeeded)
    modeRadio[2].place(x = 384, y = 410, width = 100, height = 60)
    modeRadio[3].config(text = "OFB", value = '3', command = vecNeeded)
    modeRadio[3].place(x = 484, y = 410, width = 100, height = 60)
    modeRadio[4].config(text = "CTR", value = '4', command = vecNeeded)
    modeRadio[4].place(x = 584, y = 410, width = 100, height = 60)

    Label(root, text = "Key:", bg = bgColor, fg = fgColor, font =
fNormal).place(x = 128, y = 510)
    keyEntry.place(x = 355, y = 510, width = 370)

    Label(root, text = "Init Vec:", bg = bgColor, fg = fgColor, font =
fNormal).place(x = 128, y = 610)
    vecEntry.place(x = 355, y = 610, width = 370)

    Button(root, text = "SELECT\nIMAGE", command = startAction, font =
fButton).place (x = 345, y = 760)
    ctypes.windll.shcore.SetProcessDpiAwareness(2)
    root.protocol('WM_DELETE_WINDOW', closingVerified)
    root.mainloop()

```