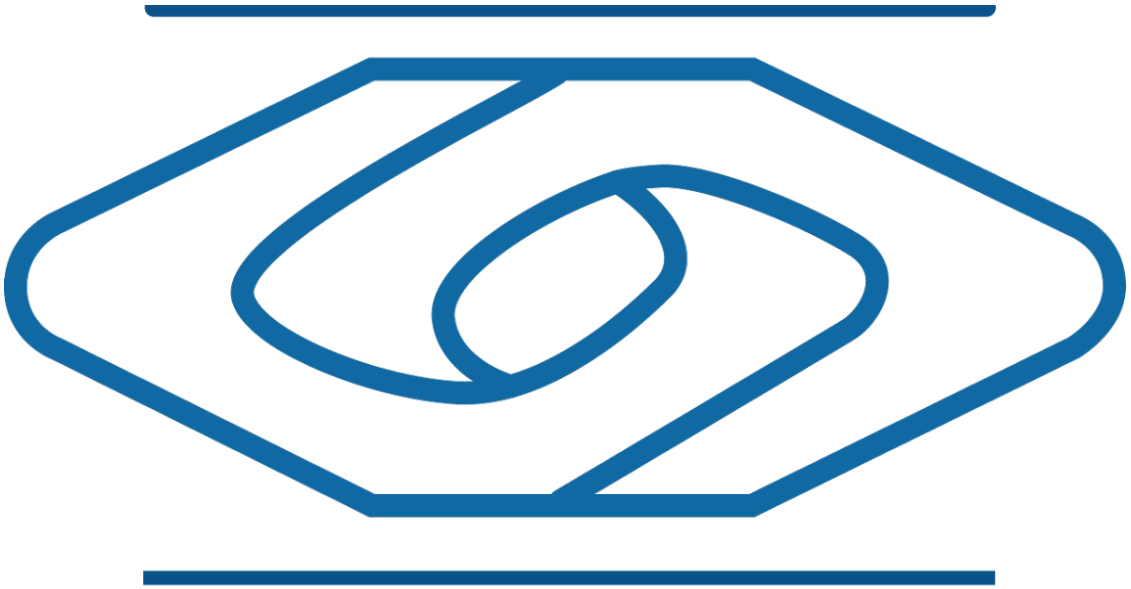

Instituto Politécnico Nacional

Escuela Superior de Computo

Cryptography
Operating modes

Abstract:
When encrypting, there exist a wide variety of algorithms and operation modes to do so, some of them are considered obsolete; therefore, in this paper we try to illustrate the difference between some of the best known algorithms by developing a program capable of showing the results of encryption and decryption in images.

By : Meza Madrid Raúl Damián
Professor: MSc. Nidia Asunción Cortez Duarte
3CM9

Contents

Introduction

The main reason for so many algorithms and operation modes is that, throughout time multiples vulnerabilities have been discovered, from being easily broken to just being bad at hiding information in specific scenarios. To make the examples more clear, we'll encrypt images.

Literature review

2.1 Triple DES

Triple DES (Data Encryption Standard), sometimes referred to as 3DES, is a block cipher standardized by NIST. Triple DES has known crypto-analytic flaws, however none of them currently enable a practical attack. Nonetheless, Triple DES is not recommended for new applications because it is incredibly slow; old applications should consider moving away from it.

2.2 AES

AES (Advanced Encryption Standard) is a block cipher standardized by NIST. AES is both fast, and cryptographically strong. It is a good default choice for encryption.

2.3 Symmetric encryption

Symmetric encryption is a way to encrypt or hide the contents of material where the sender and receiver both use the same secret key. Note that symmetric encryption is not sufficient for most applications because it only provides secrecy but not authenticity. That means an attacker can't see the message but an attacker can create bogus messages and force the application to decrypt them.

Software (libraries, packages, tools)

The program was developed in Python along with the next libraries

- tkinter
- pillow
- pyca/cryptography

Procedure

The figure 1 shows the UI for the program.
On the right, we can see the current image being used. Below it, there are two buttons, the first one, on the left, to load a new image, and the second one, on the right, to save the current image.
On the left side, we can see two inputs, one for the key, needed for all the encryption modes, and IV, used in all modes but ECB. Below we see 4 main buttons, one for each encryption mode, and below it, there's a button to toggle between encryption and decryption mode.

Figure 1: Program after loading an image

Results

The figure 2 shows the results of encrypting using the tripleDES algorithm and the different encryption modes available.
key = asegurar
IV=12345678

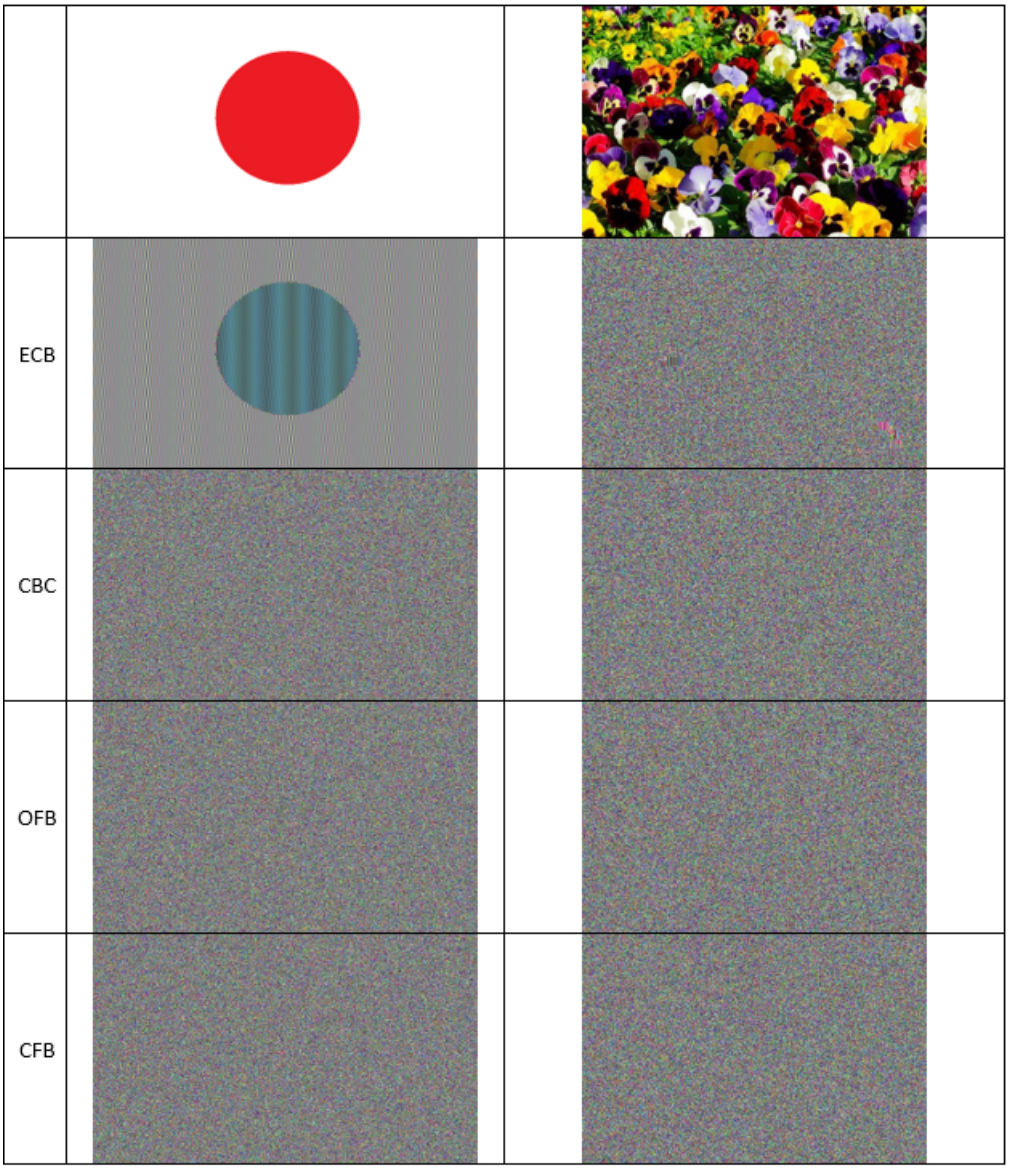

Figure 2: Two images encrypted in different modes with tripleDES algorithm

The figure 3 shows the results of encrypting using the AES algorithm and the different encryption modes available.
key = asegurar12345678
IV=asegurar12345678

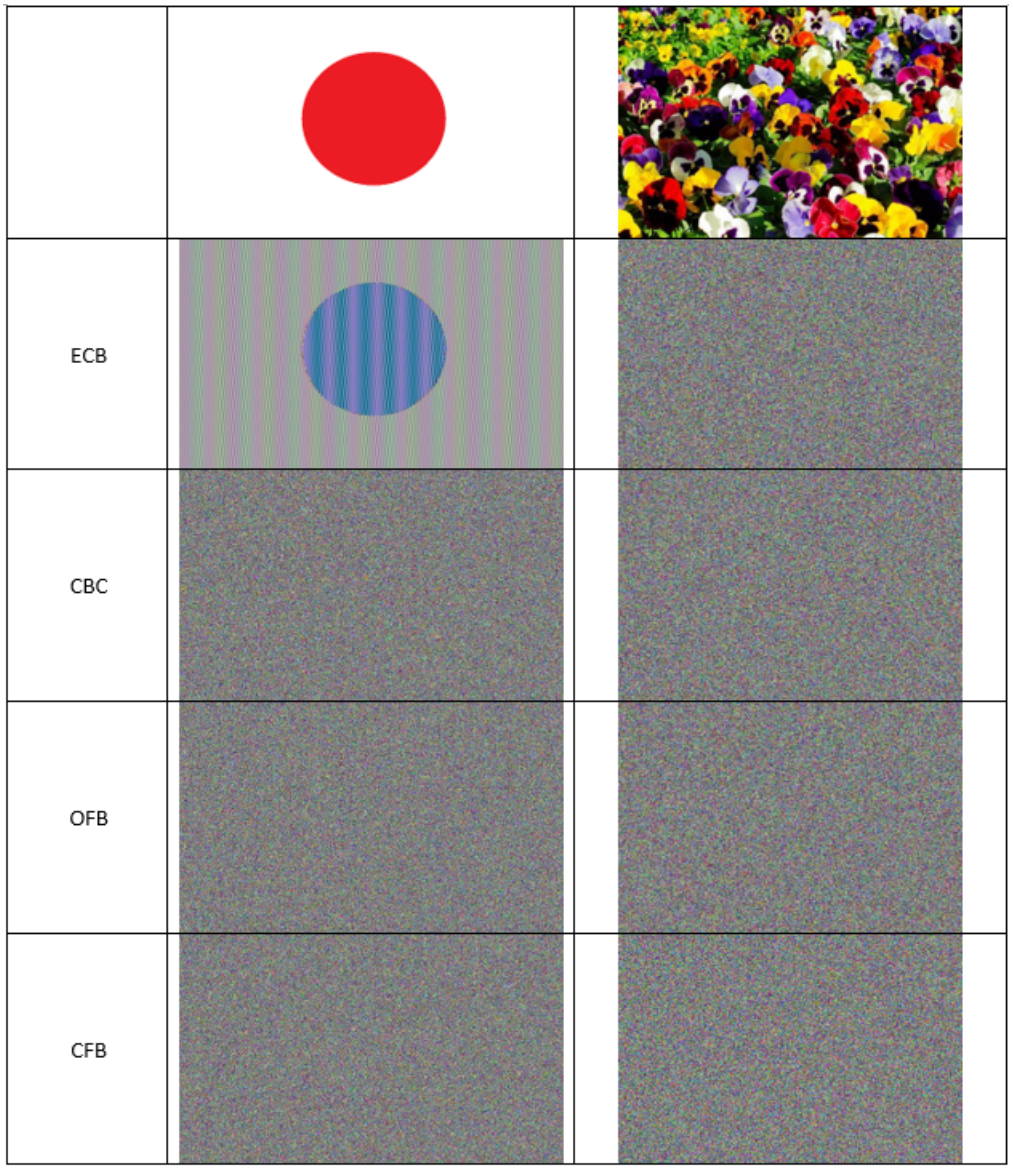

Figure 3: Two images encrypted in different modes with AES algorithm

Discussion

The two images from the results tables were used with the purpose of showing how the use of chaining and feedback allow the results to appear random, where the ECB cipher failed, since similar blocks return similar results, which allows attackers to decipher the content using old cryptanalysis. We can see how the input to the cipher is also important, since repeating patterns can be easy to see in the ECB cipher.
Another thing not shown in the table is that ciphers like OFB have the same results in encryption and decryption.

Conclusions

The result of encryption depends on many factors, from the algorithm used, mode, key, vector, and ,obviously, the input; but the reliability of it also depends in other things as the entropy of the input.

References

[1] P. C. Authority, “Symmetric encryption,” 2013.

Code

```
import os
import tkinter as tk
from PIL import Image, ImageTk
from cryptography.hazmat.primitives.ciphers import Cipher,
 ↪  algorithms, modes
from cryptography.hazmat.primitives import  padding
from cryptography.hazmat.backends import default_backend
from tkinter import Tk, Label,
 ↪  Button,filedialog,Entry,E,W,N,S,Checkbutton,Label,
 ↪  Radiobutton,messagebox

class CipherGUI:
    def __init__(self, master):
        self.padder = padding.PKCS7(128).padder()
        self.encMode = None
        self.algorithm = None
        self.keyVal = None
        self.inVect = None
        self.master = master
        self.master.title("Practica de Nidia <3")
        self.lbl_key = Label(master, text="Key", width=10)
        self.lbl_key.grid(sticky="ew")
        self.lbl_iv = Label(master, text="IV", width=10)
        self.lbl_iv.grid(sticky="ew")
        self.ent_key = Entry(master,width = 10)
        self.ent_key.grid(row=0, column=1, sticky="ew")
        self.ent_iv = Entry(master, width=10)
        self.ent_iv.grid(row=1, column=1, sticky="ew")

        self.bttn_ECB = Button(master, text="ECB",
                               command=self.encryptECB)
        self.bttn_ECB.grid(row=3, column=0, sticky="nsew")
        self.bttn_CBC = Button(master, text="CBC",
                              command=self.changeCBC)
        self.bttn_CBC.grid(row=3, column=1, sticky="nsew")
        self.bttn_OFB = Button(master, text="OFB",
                              command=self.changeOFB)
        self.bttn_OFB.grid(row=4, column=0, sticky="nsew")
        self.bttn_CFB= Button(master, text="CFB",
```

```
                                command=self.changeCFB)
        self.bttn_CFB.grid(row=4, column=1, sticky="nsew")
        self.encrypting =True
        self.bttn_Mode = Button(master, text="Encrypting",
                                command=self.changeMode)
        self.bttn_Mode.grid(row=5, column=0, columnspan=2)

        self.bttn_Open = Button(master, text="Open Image",
        ↪  command=self.openFilePath)
        self.bttn_Open.grid(row=5, column=2, sticky="nsew")
        self.bttn_Save = Button(master, text="Save",
        ↪  command=self.saveFilePath)
        self.bttn_Save.grid(row=5, column=3, sticky="nsew")
        self.label_image = Label(root)
        self.setImagePath('./japan.bmp')
        self.label_image.grid(row=0, column=2, columnspan=2,
        ↪  rowspan=5,
               sticky=W+E+N+S, padx=5, pady=5)

    def updateImage(self,p):
        self.thumbnail = Image.open(p)
        self.thumbnail = self.thumbnail.resize(
            (self.thumbnail.size[0], self.thumbnail.size[1]))
        self.tki = ImageTk.PhotoImage(self.thumbnail)
        self.label_image.configure(image = self.tki)

    def setImage(self,img):
        self.tki = ImageTk.PhotoImage(img)
        self.label_image.configure(image = self.tki)

    def setImagePath(self,p):
        self.thumbnail = Image.open(p)
        self.tki = ImageTk.PhotoImage(self.thumbnail)
        self.label_image.configure(image = self.tki)

    def openFilePath(self):
        filename = filedialog.askopenfilename(initialdir="./",
        ↪  title="Select file")
        if filename == None:
            return
```

```
73          self.updateImage(filename)
74
75      def saveFilePath(self):
76          filename =
            ↪  filedialog.asksaveasfilename(defaultextension=".bmp")
77          if filename == None:
78              return
79          self.thumbnail.save(filename,format = 'BMP')
80
81      def getkey(self):
82          if len(self.ent_key.get())!=16:
83              messagebox.showwarning('','Key value must be 16
                ↪  characters long ')
84              return None
85          else:
86              return bytes(self.ent_key.get(),'utf-8')
87
88      def getiv(self):
89          if len(self.ent_iv.get()) != 16:
90              messagebox.showwarning('', 'I Vector value must be 16
                ↪  characters long ')
91              return None
92          else:
93              return bytes(self.ent_key.get(), 'utf-8')
94
95      def changeMode(self):
96          if self.encrypting:
97              self.bttn_Mode.configure(text = 'Decrypting')
98          else:
99              self.bttn_Mode.configure(text = 'Encrypting')
100         self.encrypting = not self.encrypting
101
102     def encryptECB(self):
103         key = self.getkey()
104         if key == None:
105             return
106         cipher = Cipher(algorithms.AES(key),
107                         modes.ECB(), backend=default_backend())
108         if self.encrypting:
109             enc = cipher.encryptor()
```

```
110             padder = padding.PKCS7(128).padder()
111             data = padder.update(self.thumbnail.tobytes()) +
                ↪  padder.finalize()
112             ct = enc.update(data) + enc.finalize()
113         else:
114             dec = cipher.decryptor()
115             padder = padding.PKCS7(128).padder()
116             data = padder.update(self.thumbnail.tobytes()) +
                ↪  padder.finalize()
117             ct = dec.update(data) + dec.finalize()
118         self.thumbnail = Image.frombytes(data = ct,
119             mode=self.thumbnail.mode, size=self.thumbnail.size)
120         self.setImage(self.thumbnail)
121
122
123     def changeCBC(self):
124         key = self.getkey()
125         if key == None:
126             return
127         iv = self.getiv()
128         if iv == None:
129             return
130         cipher = Cipher(algorithms.AES(key),
131                         modes.CBC(iv), backend=default_backend())
132         if self.encrypting:
133             enc = cipher.encryptor()
134             padder = padding.PKCS7(128).padder()
135             data = padder.update(self.thumbnail.tobytes()) +
                ↪  padder.finalize()
136             ct = enc.update(data) + enc.finalize()
137         else:
138             dec = cipher.decryptor()
139             padder = padding.PKCS7(128).padder()
140             data = padder.update(self.thumbnail.tobytes()) +
                ↪  padder.finalize()
141             ct = dec.update(data) + dec.finalize()
142         self.thumbnail = Image.frombytes(data=ct,
143                                          mode=self.thumbnail.mode,
                                         ↪  size=self.thumbnail.size)
144         self.setImage(self.thumbnail)
```

```
145
146     def changeOFB(self):
147         key = self.getkey()
148         if key == None:
149             return
150         iv = self.getiv()
151         if iv == None:
152             return
153         cipher = Cipher(algorithms.AES(key),
154                         modes.OFB(iv), backend=default_backend())
155         if self.encrypting:
156             enc = cipher.encryptor()
157             ct = enc.update(self.thumbnail.tobytes()) +
                ↪  enc.finalize()
158         else:
159             dec = cipher.decryptor()
160             ct = dec.update(self.thumbnail.tobytes()
161                                     ) + dec.finalize()
162         self.thumbnail = Image.frombytes(data=ct,
163                                          mode=self.thumbnail.mode,
                                          ↪  size=self.thumbnail.size)
164         self.setImage(self.thumbnail)
165
166     def changeCFB(self):
167         key = self.getkey()
168         if key == None:
169             return
170         iv = self.getiv()
171         if iv == None:
172             return
173         cipher = Cipher(algorithms.AES(key),
174                         modes.CFB(iv), backend=default_backend())
175         if self.encrypting:
176             enc = cipher.encryptor()
177
178             ct = enc.update(self.thumbnail.tobytes()) +
                ↪  enc.finalize()
179         else:
180             dec = cipher.decryptor()
181             ct = dec.update(self.thumbnail.tobytes()
```

```
182                                ) + dec.finalize()
183         self.thumbnail = Image.frombytes(data=ct,
184                                          mode=self.thumbnail.mode,
                                              ↪  size=self.thumbnail.size)
185         self.setImage(self.thumbnail)
186
187 root = Tk()
188 my_gui = CipherGUI(root)
189 root.mainloop()
```