INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

CRYPTOGRAPHY

Practice 4: Block Cipher - Operation modes

STUDENT:
Rosas Hernandez Oscar Andres

PROFESSOR:
Msc. Nidia Asunción Cortez Duarte

Abstract:
In this practice we show how using different operation modes affect the "encoded information", and why you shoul be really careful about using ECB.

October 16, 2019

CONTENTS

I. Introduction

In this practice we are going to talk about block ciphers, what are operations modes, and what different type are there. We are going to compare them using a BMP image to see if we could "see" some information just from the encoded image.
Also, we are going to implement a simple image cipher with Flask and React so you can upload all the images you would like and see the result.

II. Literature review

A. Operation Modes

Encryption algorithms are divided into two categories based on input type, as block cipher and stream cipher. Block cipher is an encryption algorithm which takes fixed size of input say b bits and produces a ciphertext of b bits again. If input is larger than b bits it can be divided further. For different applications and uses, there are several modes of operations for a block cipher.

1) Electronic Code Book (ECB): Electronic code book is the easiest block cipher mode of functioning. It is easier because of direct encryption of each block of input plaintext and output is in form of blocks of encrypted ciphertext. Generally, if a message is larger than b bits in size, it can be broken down into bunch of blocks and the procedure is repeated.

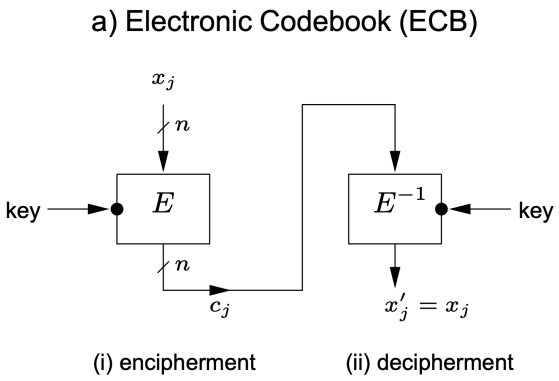


Figure 1. [3]

2) Cipher Block Chaining (CBC): Cipher block chaining or CBC is an advancement made on ECB since ECB compromises some security requirements. In CBC, previous cipher block is given as input to next encryption algorithm after XOR with original plaintext block. In a nutshell here, a cipher block is produced by encrypting a XOR output of previous cipher block and present plaintext block.

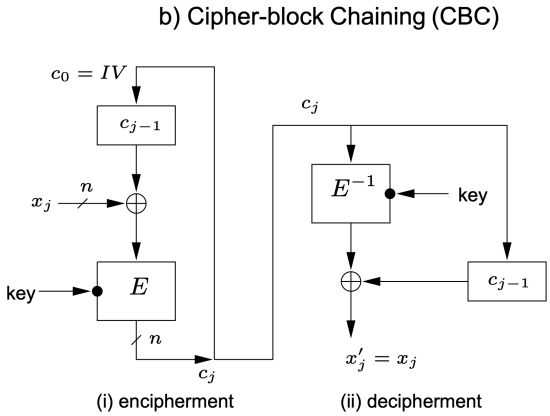


Figure 2. [3]

3) Cipher Feedback Mode (CFB): In this mode the cipher is given as feedback to the next block of encryption with some new specifications: first an initial vector IV is used for first encryption and output bits are divided as set of sandb-s bits the left hand side sbits are selected and are applied an XOR operation with plaintext bits. The result given as input to a shift register and the process continues. The encryption and decryption process for the same is shown below, both of them use encryption algorithm.

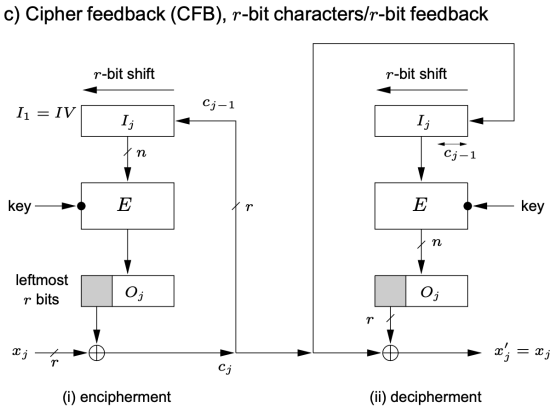


Figure 3. [3]

Output Feedback Mode (OFB)

The output feedback mode follows nearly same process as the Cipher Feedback mode except that it sends the encrypted output as feedback instead of the actual cipher which is XOR output. In this output feedback mode, all bits of the block are send instead of sending selected s bits. The Output Feedback mode of block cipher holds great resistance towards bit transmission errors. It also decreases dependency or relationship of cipher on plaintext.

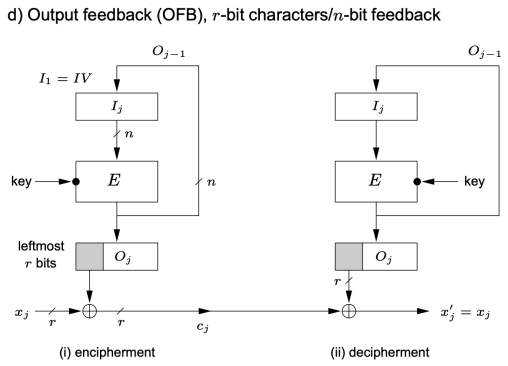


Figure 4. [3]

[1]

B. Initialization vector (IV)

An initialization vector (IV) or starting variable is a block of bits that is used by several modes to randomize the encryption and hence to produce distinct ciphertexts even if the same plaintext is encrypted multiple times, without the need for a slower re-keying process.

An initialization vector has different security requirements than a key, so the IV usually does not need to be secret. However, in most cases, it is important that an initialization vector is never reused under the same key. For CBC and CFB, reusing an IV leaks some information about the first block of plaintext, and about any common prefix shared by the two messages. For OFB and CTR, reusing an IV completely destroys security.

[2]

C. Padding

A block cipher works on units of a fixed size (known as a block size), but messages come in a variety of lengths. So some modes (namely ECB and CBC) require that the final block be padded before encryption. Several padding schemes exist. The simplest is to add null bytes to the plaintext to bring its length up to a multiple of the block size, but care must be taken that the original length of the plaintext can be recovered.
[2]

III. Sotware

- Python FLask [4]
- React JS [5]
- Python Pycryptodome [6]

IV. Procedure

- Create a simple SPA with react that is able to send a form with a file (bmp image) to a server (in this case localhost::5000/), using POST, also add the algorithm and the operation mode desired.
- Create a simple server using Flask, with just one route, the root, in the case we get a request from GET, return the react app in case there is a POST request check to see if there is a file.
- If there is, upload the file to the server and save the filename
- Create a new copy of the saved image
- Use PIL to extract the pixel data and transform that to a array of bytes.
- Use Pycryptodome to create a helper function that accepts the array, uses padding and the IV if necessary.
- Recover the algorithm and the operation mode from the client and call the apropiate function.
- Store the new image in the server, return a link to that image to the client.
- Client need to show that image.

V. Results and evidence

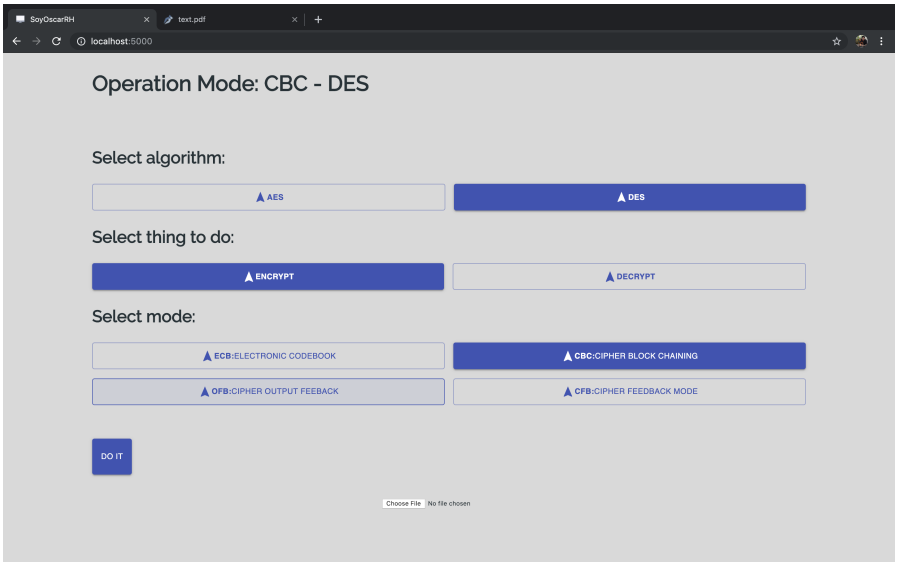


Figure 5. Screeenshots

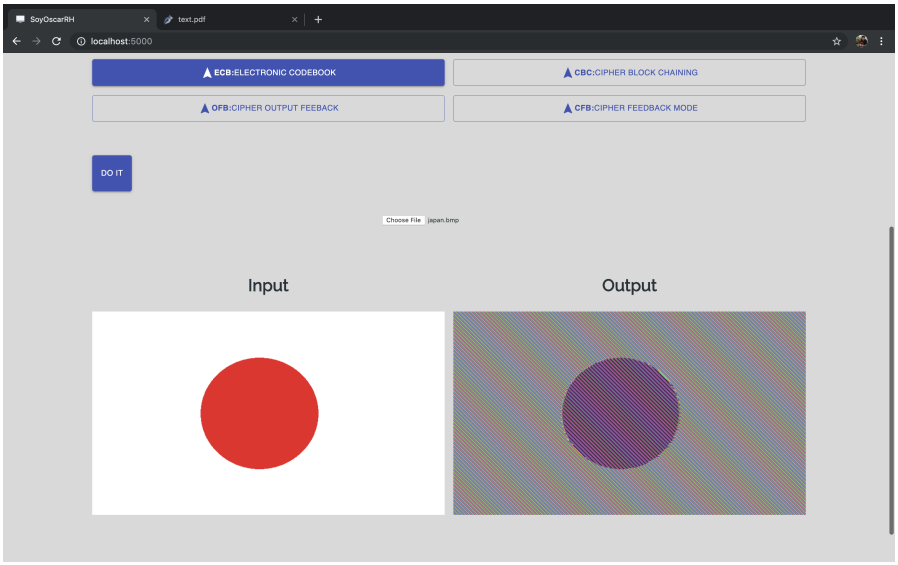


Figure 6. Screeenshots

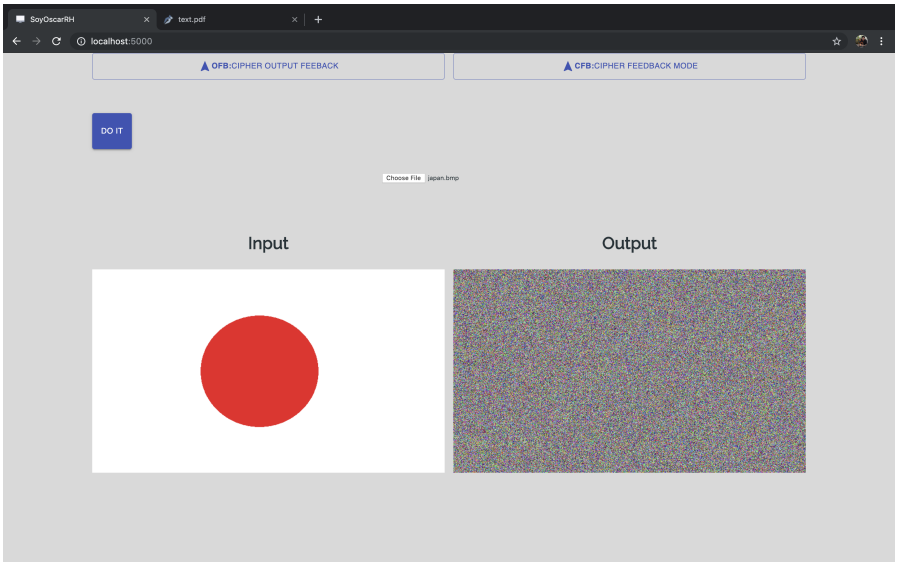
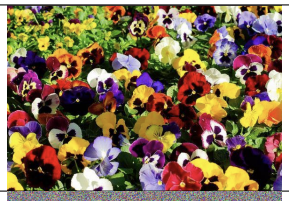
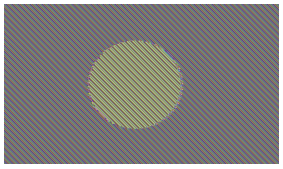
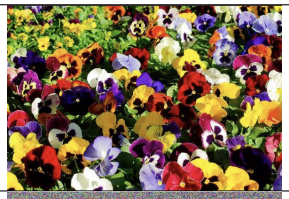


Figure 7. Screeenshots

Mode	Image A	Image B
Original		
AES - ECB		
AES - CBC		
		

Mode	Image A	Image B
Original		
DES - ECB		
DES - CBC		
DES - CFB		
DES - OFB		

VI. Discussion

We can see that with CBC parallel encryption of blocks of bits is possible, thus it is a faster way of encryption, but prone to cryptanalysis since there is a direct relationship between plaintext and ciphertext.
CBC is better resistive nature towards cryptanalsis than ECB, in general, every operation mode is better that ECB.

VII. Conclusions

I learned a lot about operation mode, what does it mean to have padding why OFB does not need it, and the different size of key and IV in AES and DES (also, that explain a little bit with it is not recomended to use DES).
The software that I developted will work with any of the former operation modes and any BMP image.

VIII. Code

See it in: https://github.com/SoyOscarRH/LibroCriptografia/tree/master/Code/OperationModes

```
import React, { StrictMode, FunctionComponent, useState, useRef } from "react"
import ReactDOM from "react-dom"

import { Button } from "@material-ui/core/"

import { Modes, MyButton } from "./Modes"
import { Section } from "./Style"

const App: FunctionComponent = () => {
  const [doIt, changeDo] = useState("encrypt")
  const [mode, changeMode] = useState("ECB")
  const [algo, changeAlgo] = useState("DES")
  const [url, changeURL] = useState("")
  const [urlOutput, changeURLOutput] = useState("")
  const formRef = useRef()

  const updateStuff = async (file: File) => {
    const formData = new FormData()
    formData.append("file", file, file.name)
    formData.append("mode", mode)
```

```
  formData.append("algo", algo)
  formData.append("do", doIt)

  changeURL(URL.createObjectURL(file))

  try {
    const response = await fetch("/", { body: formData, method: "post" })
    const json = await response.json()

    changeURLOutput(json.path)

    console.log(json)
  } catch (e) {
    console.log(e)
  }
}

return (
  <div style={Section}>
    <section>
      <h2>
        Operation Mode: {mode} - {algo}
      </h2>
    </section>

    <section>
      <h3>Select algorithm:</h3>

      <div style={{ display: "grid", gridGap: "1rem", gridAutoFlow: "column" }}>
        {["AES", "DES"].map(thisAlgo => (
          <MyButton
            key={thisAlgo}
            onClick={() => changeAlgo(thisAlgo)}
            isFull={algo === thisAlgo}
            mini={thisAlgo}
            title={""}
          />
        ))}
      </div>

      <h3>Select thing to do:</h3>

      <div style={{ display: "grid", gridGap: "1rem", gridAutoFlow: "column" }}>
        {["encrypt", "decrypt"].map(doThis => (
          <MyButton
            key={doThis}
            onClick={() => changeDo(doThis)}
            isFull={doIt === doThis}
            mini={doThis}
            title={""}
          />
        ))}
      </div>

      <h3>Select mode:</h3>

      <div style={{ display: "grid", gridGap: "1rem", gridTemplateColumns: "1fr 1fr" }}>
        {Modes.map(({ mini, title }) => (
          <MyButton
            key={mini}
            onClick={() => changeMode(mini)}
            isFull={mode === mini}
            mini={mini}
            title={title}
          />
        ))}
      </div>

      <br />
      <br />

      <Button
        onClick={() => {
          formRef &&
            formRef.current &&
            formRef.current.firstChild &&
            formRef.current.firstChild.files &&
            formRef.current.firstChild.files[0] &&
            updateStuff(formRef.current.firstChild.files[0])
        }}
        style={{ height: "4rem" }}
        size="medium"
        variant={"contained"}
        color="primary"
      >
        Do it
```

```
        </Button>
      </section>

      <section>
        <form method="post" action="" id="login" ref={formRef} style={{ textAlign: "center" }}>
          <input
            type="file"
            onChange={event => {
              const file = event.target.files && event.target.files[0]
              if (!file) return

              updateStuff(file)
            }}
          />
          <br />
          <br />
          <br />
          <section
            style={{
              textAlign: "center",
              display: "grid",
              gridTemplateColumns: "1fr 1fr",
              gridGap: "1rem",
            }}
          >
            <div>
              {url && <h3>Input</h3>}
              <img style={{ display: "block", margin: "auto auto" }} src={url} width="100%" />
            </div>

            <div>
              {urlOutput && <h3>Output</h3>}
              <img style={{ display: "block", margin: "auto auto" }} src={urlOutput} width="100%" />
            </div>
          </section>
        </form>
      </section>
    </div>
  )
}

const DOM_NODE = document.getElementById("ReactApp")
ReactDOM.render(
  <StrictMode>
    <App />
  </StrictMode>,
  DOM_NODE
)
```

```
import os
from flask import Flask, flash, request, redirect, render_template, jsonify
from werkzeug.utils import secure_filename

app = Flask(
    __name__,
    static_folder="./Distribution",
    template_folder="./",
)
app.config['UPLOAD_FOLDER'] = './Distribution/Uploads'


from Cryptodome.Util.Padding import pad, unpad
from Cryptodome.Cipher import AES, DES
from Cryptodome.Random import get_random_bytes

from PIL import Image
import sys
import datetime

def get_BMP_bytes(original_image):
    result = bytearray()
    original_data = original_image.getdata()
    for pixel in original_data:
        result.append(pixel[0])
        result.append(pixel[1])
        result.append(pixel[2])

    return result, len(original_data)


def get_pixels(bytes, original_size):
    pixels = []
    current, original_size = 0, original_size * 3

    while current + 2 < original_size:
        pixels.append((bytes[current], bytes[current + 1], bytes[current + 2]))
```

```
        current += 3

    return pixels


iv = b'\xd8r\x15\xdc|y6\xbbs\xf2\x7f:\x0b\xcb#t'
key = b'\xef\xd2\xba\xd8\x91\x9aW\xa7\x1c^\xfdO\xe1)?#'

iv_mini = b'-8B key-'
key_mini = b'-8B key-'

class Helper:
    def __init__(self, algo, mode):
        self.not_needs_padding = AES.MODE_OFB == mode or DES.MODE_OFB  == mode

        if mode == AES.MODE_ECB or mode == DES.MODE_ECB:
            self.cipher = algo.new(key_mini if algo == DES else key, mode)
        else:
            if algo == DES:
                self.cipher = algo.new(key_mini, mode, iv_mini)
            else:
                self.cipher = algo.new(key, mode, iv)



    def encrypt(self, plaintext):
        if self.not_needs_padding:
            return self.cipher.encrypt(plaintext)
        else:
            plaintext = pad(plaintext, 16)
            return self.cipher.encrypt(plaintext)


    def decrypt(self, encoded):
        if self.not_needs_padding:
            return self.cipher.decrypt(encoded)
        else:
            encoded = pad(encoded, 16)
            return self.cipher.decrypt(encoded)


def create_encoded_image(algo, mode, path_plain, path_encoded):
    x = Helper(algo, mode)
    original_image = Image.open(path_plain)

    plain, size = get_BMP_bytes(original_image)
    encoded = x.encrypt(plain)[:len(plain)]

    encoded_array = bytearray(encoded)
    print(len(plain))
    print(len(encoded))

    pixels = get_pixels(encoded_array, size)


    edited = original_image.copy()
    edited.putdata(pixels)

    edited.save(path_encoded)


def recover_image(algo, mode, path_encoded, path_plain):
    x = Helper(algo, mode)

    encoded_image = Image.open(path_encoded)

    encoded, size = get_BMP_bytes(encoded_image)
    plain = x.decrypt(encoded)

    plain_array = bytearray(plain)
    print(len(encoded))
    print(len(plain))

    pixels = get_pixels(plain_array, size)


    recovered = encoded_image.copy()
    recovered.putdata(pixels)

    recovered.save(path_plain)



DES_Modes = {
  "ECB": DES.MODE_ECB,
  "CBC": DES.MODE_CBC,
```

```
  "OFB": DES.MODE_OFB,
  "CFB": DES.MODE_CFB,
}


AES_Modes = {
  "ECB": AES.MODE_ECB,
  "CBC": AES.MODE_CBC,
  "OFB": AES.MODE_OFB,
  "CFB": AES.MODE_CFB,
}

@app.route('/', methods=['GET', 'POST'])
def server():
    if request.method == 'POST':

        if 'file' not in request.files or request.files['file'].filename == '':
            return redirect(request.url)

        file = request.files['file']
        mode = request.form['mode']
        algo = request.form['algo']
        doIt = request.form['do']

        print(doIt)

        filename = os.path.join(app.config['UPLOAD_FOLDER'], secure_filename(file.filename))
        file.save(filename)

        new_filename = f"./Distribution/Uploads/{datetime.datetime.now()}.bmp"


        if doIt == "encrypt":
            if algo == "DES":
                create_encoded_image(DES, DES_Modes[mode], filename, new_filename)
            else:
                create_encoded_image(AES, AES_Modes[mode], filename, new_filename)
        else:
            if algo == "DES":
                recover_image(DES, DES_Modes[mode], filename, new_filename)
            else:
                recover_image(AES, AES_Modes[mode], filename, new_filename)


        return jsonify(message="All is well", path=new_filename)
    else:
        return render_template('index.html')
```

REFERENCES

[1] Block Cipher modes of Operation *https://www.geeksforgeeks.org/block-cipher-modes-of-operation/*.
[2] Block cipher mode of operation - Wikipedia $https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation/$.
[3] Handbook of Applied Cryptography, Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone
[4] *http://flask.palletsprojects.com/en/1.1.x/*.
[5] *https://reactjs.org/*.
[6] *https://pycryptodome.readthedocs.io/en/latest/index.html*.