HW3 README

分工

湯濬澤 ： 解密 / ShareFunction / 自定義模式
陳冠文 ： 加密 / ShareFunction / 自定義模式

建置環境

Windows 10 64bits 專業版
Python 3.7

依賴套件

numpy
opencv-python
pycryptodome
argparse

操作方式

makefile:

使用 make 來安裝環境，有二個參數(enc、dec)可以輸入：

make

make enc

make dec

enc.py:

安裝好套件後，可以通過 python enc.py 來操作，後面有三個參數可選輸入。

也可以通過 make enc 來自動完成動作。

python enc.py [--input FILENAME] [--mode MODE] [--output FILENAME]

optional arguments:

-h, --help show this help message and exit

--input INPUT input image path, default: ./linux.jpg

--mode MODE encrypt mode, ECB, CTR, Custom. default: ECB

--output OUTPUT output image path, default: ./[mode].png

※請注意，output 的資料夾必須存在才會輸出東西。

或

make enc

※請注意，test_enc 資料夾必須存在，make enc 才會輸出東西

（若 test_enc 資料夾不存在，可以先跑 make 來建置）

Dec.py

安裝好套件，並且確認有 test_dec / test_enc 兩個資料夾以及解密所需

的 ./test_enc/ECB.png, ./test_enc/CTR.png, ./test_enc/Custom.png, iv.bin / key.bin 檔案後，跑 make dec 完成解密，或是透過手動執行

python dec.py -di [INPUT_FILE_PATH] -mode [ECB / CTR / Custom] -o [OUTPUT_FILE_NAME]

※請注意，請不要生成加密圖片後更改 iv.bin 及 key.bin 或是跑 make install，這兩個檔案存有當初加密時的金鑰，做以上操作會使金鑰更動

程式碼解說

實作解密的方法如下，首先我們透過 opencv 將圖片以 RGB 的方式讀入，每個 R / G / B 像素作為 1 Byte 去將整張圖壓成一維陣列。

```
#(高, 寬, bytes)
def readImage(path: str):
    im = cv2.imread(path)
    if(type(im) == type(None)):
        print("Can't open picture.")
        exit(1)
    imFlat = im.flatten()
    return (im.shape[0], im.shape[1], bytes(imFlat))
```

接下來讀入 key.bin，這檔案存有加密時所使用的金鑰

```
def readKey(path: str, lens: int = 16) -> bytes:
    f = open(path, "rb")
    byte = f.read(lens)
    f.close()
    return bytes(byte)
```

然後因為要讀取 Padding 的資料資訊，因此用了 State 來判斷讀取的資料為何

```python
# Get 2 X coordinate
stage = 0
# stage 0 : 等待最後的<\n> / 1 : 開始讀第一組 Ascii / 2 : 開始讀第二組 Ascii / 3 : 結束
x_real_data = 0
num_bits = 0
x_cryed_data = 0
byteLen = len(picture_data)
for i in range(byteLen-1, -1, -1):
    if stage == 0:
        # Get last <\n>
        if picture_data[i] == shareFunction.char2Ascii('\n'):
            stage = 1
    elif stage == 1:
        # Get crypted data position
        if picture_data[i] == shareFunction.char2Ascii('\n'):
            stage = 2
            num_bits = 0
        else:
            deAsciiNum = shareFunction.ascii2Int(picture_data[i])
            x_cryed_data += deAsciiNum * (10**num_bits)
            num_bits += 1
    elif stage == 2:
        # Get real data position
        if picture_data[i] == shareFunction.char2Ascii('\n'):
            stage = 3
            # end the loop here
            break
        else:
            deAsciiNum = shareFunction.ascii2Int(picture_data[i])
            x_real_data += deAsciiNum * (10**num_bits)
            num_bits += 1
    elif i == 0:
        print("Something going wrong, this picture might not be a crypted picture!")
```

那因為解密這會獲得真實資料的位置以及加密後的資料位置，因此長度都會是 128 的倍數，可以整數裁切，因此 ECB 僅 For loop 即可搞定

```python
if mode == "ECB":
    for i in range(byteIterTime):
        decryed += cipher.decrypt(encryed[i * 16:(i + 1) * 16])
```

CTR 則需要多讀取 initial Vector，並在每一個 Block 都加一

```python
elif mode == "CTR":
    initialVec = shareFunction.readKey('./iv.bin')
    for i in range(byteIterTime):
        # print("Round :", i)
        initialVec = shareFunction.byteAddOne(initialVec)
        decryed += cipher.encrypt(initialVec)
    decryed = shareFunction.xor(decryed, encryed)
```

加一因為 Python 沒有實作 Operator 因此就自己寫，遇到 255 就填 0 進位繼續跑，遇到其他數字就+1

```python
def byteAddOne(byteKey : bytes):
    isDone = False
    lastIndex = len(byteKey) - 1
    newBytes = bytes(0)
    for i in range(lastIndex, -1, -1):
        if isDone :
            newBytes = bytes([byteKey[i]]) + newBytes
        else:
            if int(byteKey[i]) == 255:
                newBytes = bytes([0]) + newBytes
            else:
                newBytes = bytes([(int(byteKey[i]) + 1)]) + newBytes
                isDone = True
    return newBytes
```

Xor 也自己寫，轉成 int 做完 xor 後再轉回 Bytes

```python
def xor(byteA : bytes, byteB : bytes):
    newBytes = bytes(0)
    for aByteA, aByteB in zip(byteA, byteB):
        newBytes += bytes([aByteA ^ aByteB])
    return newBytes
```

那 Custom mode 其實只是多了讀取柯 P 中指圖片以及傳入參數獲

取像素資訊而已，主要架構可沿用 CTR

```python
elif mode == "Custom":
    initialVec = shareFunction.readKey('./iv.bin')
    img = shareFunction.readGrayImage("./middle_finger.png")
    count = 0
    for i in range(byteIterTime):
        # print("Round :", i)
        count += 1
        newIV = shareFunction.getImgVal(initialVec, count, img)
        decryed += cipher.encrypt(newIV)
    decryed = shareFunction.xor(decryed, encryed)
```

獲取像素資訊

```python
# Return 16 bytes
def getImgVal(IV: bytes, count: int, imgInfo: (int, int, bytes)) -> bytes:
    ivInt = int.from_bytes(IV[-30:], byteorder='big')
    XIndex = (ivInt + count) % imgInfo[1]
    YIndex = count % imgInfo[0]
    value = imgInfo[2][YIndex * imgInfo[1] + XIndex]
    newBytes = IV[0:15]
    newBytes += bytes([value])
    # [y * imgInfo[1] (image_x) + x]
    return newBytes
```

最後將資料擷取到原始資料的長度即完成

```python
real_data = decryed[0 : x_real_data]
```

那麼 Argument 的部分就是用套件處理

```python
# parser
parser = argparse.ArgumentParser()
parser.add_argument("-di",
                    '--decInput',
                    type=str,
                    help="這是解密的輸入圖片",
                    default="./ecb.png")
parser.add_argument("-mode",
                    '--mode',
                    type=str,
                    help="這是解密的模式",
                    default="ECB")
parser.add_argument("-o",
                    '--outputName',
                    type=str,
                    help="這是解密的輸出檔名",
                    default="res.png")
args = parser.parse_args()
```

Padding 方式

由於當初設計時不知道僅需要加解密 linux.jpg，因此有考慮到不同圖片可能會需要 padding 的問題，因此我們的 padding 做法為在圖片的最後幾格像素塞入位置資訊。

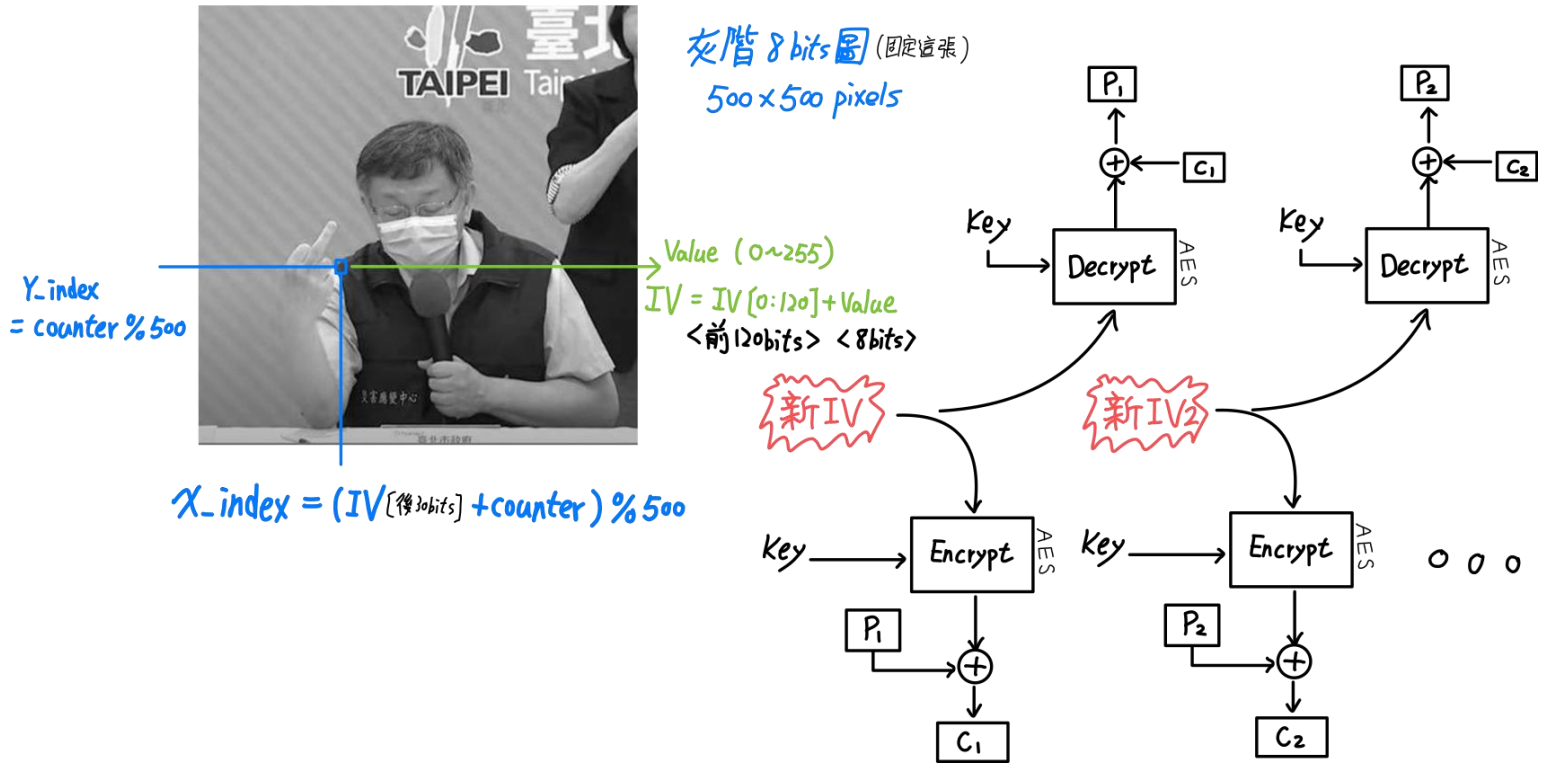
首先我們想要在加密後，依然能夠維持圖片寬度，但高度可增加(儲存 Padding 資訊)，所以會需要兩次 Padding，第一次是將資料填滿 128bits，這樣才能送進套件做 AES 加/解密。第二次則是將這坨加密的資訊依據每個像素點都塞入 3 個 Byte(RGB)的原則填入後，再將圖片以黑色填滿，畢竟這坨資訊不可能剛好可以填滿整張圖片，那在圖片的最後塞入位置資訊，格式如下

```
<\n><Real data X position | ASCII code><\n>
<After first padding X position | ASCII code><\n>
```

角括號僅為方便觀看用，不具意義。也就是透過三個換行 ASCII 包住兩組位置資訊，前者為實際有效資料的位置 Index，後者為第一次 padding 後的 Index 位置，舉例來說我有一筆 120bits 長度的資料被拿去加密，那麼第一筆位置資訊就會為 120，那第二筆位置資訊就會為 128，那因為這兩個數字可能會很大，因此我們將每個位元都拆開用 Ascii 來表示，也就是 120 表示成 49, 50, 48，三個 Byte，這樣不管何種大小的圖片都可以順利加密。

Block Cipher 架構圖

〈不要在外面吃東西加密法〉



我們的設計理念就是：好玩就好！

因為經過討論後，發現要超越現有的 Block Cipher 模式，找到一個更加完美的 Block Cipher 是很困難的，所以我們決定更改 CTR 模式用自己的方式來。

首先我們想到的是要表達自己對作業的痛恨，最好的方式就是用圖片了，所以我們使用了透過獲取中指圖片某像素點的顏色值作為類似 S-Box 的查表機制(當然兩者的效果相差距大)。並且由於我們認為 CTR 模式本身就十分完美，所以採用了與 CTR 模式相似的

pattern，也更方便重用我們寫過的程式碼。

那麼要將圖片取代 CTR 模式的哪一部分呢？我們唯一能找到的就只有 IV+counter 的地方。我們決定將原本的 IV+counter 作為 X 座標，Counter 作為 Y 座標，去獲得這張柯 P 比中指的灰階圖中的某個像素點的值，在將這值貼到原本 IV 的後 8 位元。最後在將這個新的 IV 放入 CTR 模式的後半段繼續處理。

此模式被命名為 **<不要在外面吃東西模式>**。

至於這模式有什麼優點呢? 老實說除了好玩好像真的沒有什麼優點就是。

困難與心得

湯濬澤 ：

其實前面 Padding 時遇到的困難比較大一些，因為那時忙著想如何在圖片大小不一的情況下去標示實際資料的長度並且加密後還要將這些資訊放在圖片裡，討論了一段時間，有想過填了 3 筆資料，就塞 333，填了 4 筆就 4444，但考慮到有些情況會有例外，最後就生出了 ASCII 填入像素內再讀出來這種方法，所幸這個方法還挺有效的，不過等我們實作完後才發現原來好像只需要處理 linux.jpg 這張圖就好，而這張圖完全不需要 Padding...

那接下來就是實作 ECB / CTR 部分，老實說除了 Python 的 Int Byte 轉換比較討厭以外，其實問題不大，並且由於轉換 ppm 這格式實在是有點麻煩，乾脆一點直接用 opencv 讀取 RGB 值方便許多。至於最後的困難莫過於自己的 Block 模式了，我們首先決定了發展方向<可平行運算><避免 Repeat attack>，那在維持可平行運算的情況下其實可做的操作就不多了，所以在想了很久以後決定就來個創意一點的方法就好，於是就出現了不要在外面吃東西模式，也算是抒發一下整天被關在家的心情。也從這過程理解到前人發明的這個 CTR 模式真的厲害呀。

其實這次作業還算好玩啦，除了最後又要弄最討厭的 Linux 外，實

作 ECB / CTR 以及不要在外面吃東西模式，都能夠讓我們更好的理解這些東西之所以被這麼設計的原因，而不是單純的就只是寫作業，當然對於我們來說沒有作業會是個更令人振奮的方式就是 XD