CSE 433S LAB1 - Secret Key

Zihan Chen

October 2024

Contents

1 Task1: Frequency Analysis

1.1 Method

At first, I only use one letter frequency analysis to match the result, but I failed, But some of the letters are correct.

Look at the plain text below we can know that "a", "o", "t", "e" must be correct. This most frequency letters well satisfy the one letter frequency.

> tre onuahn tmhi oi nmilab wrsur neecn afomt hsgrt ayteh trsn doig nthaige awahln thsp tre faggeh yeedn dske a ioiageiahsai too

Then I use two letters frequency analysis to modify the first result. And then use the three letters frequency analysis to modify the second result.

> the osuars tmrn on smndav whiuh seecs afomt right ayter this long strange awards trip the fagger yeels like a nonagenarian too

There still have little errors in the plain text. exchange "u" and "c", exchange "m" and "u", exchange "f" and "b". We are closer to the correct answer.

> the oscars turn on sundav which seems about right ayter this long strange awards trip the bagger yeels like a nonagenarian too

Then we exchange "v" and "y", exchange "j" and "q", exchange "v" and "f". Now we can get the final answer.

1.2 PlaintText

> the oscars turn on sunday which seems about right after this long strange awards trip the bagger feels like a nonagenarian too
>
> the awards race was bookended by the demise of harvey weinstein at its outset and the apparent implosion of his film company at the end and it was shaped by the emergence of metoo times up blackgown politics armcandy activism and a national conversation as brief and mad as a fever dream about whether there ought to be a president winfrey the season didnt just seem extra long it was extra long because the oscars were moved to the first weekend in march to avoid conflicting with the closing ceremony of the winter olympics thanks pyeongchang
>
> one big question surrounding this years academy awards is how or if the ceremony will address metoo especially after the golden globes which became a jubilant comingout party for times up the movement spearheaded by powerful hollywood women who helped raise millions of dollars to fight sexual harassment around the country
>
> signaling their support golden globes attendees swathed themselves in black sported lapel pins and sounded off about sexist power imbalances from the red carpet and the stage on the air e was called out about pay inequity after its former anchor catt sadler quit once she learned that she was making far less than a male cohost and during the ceremony natalie

portman took a blunt and satisfying dig at the allmale roster of nominated directors how could that be topped

as it turns out at least in terms of the oscars it probably wont be

women involved in times up said that although the globes signified the initiatives launch they never intended it to be just an awards season campaign or one that became associated only with redcarpet actions instead a spokeswoman said the group is working behind closed doors and has since amassed million for its legal defense fund which after the globes was flooded with thousands of donations of or less from people in some countries

no call to wear black gowns went out in advance of the oscars though the movement will almost certainly be referenced before and during the ceremony especially since vocal metoo supporters like ashley judd laura dern and nicole kidman are scheduled presenters

another feature of this season no one really knows who is going to win best picture arguably this happens a lot of the time inarguably the nailbiter narrative only serves the awards hype machine but often the people forecasting the race socalled oscarologists can make only educated guesses

the way the academy tabulates the big winner doesnt help in every other category the nominee with the most votes wins but in the best picture category voters are asked to list their top movies in preferential order if a movie gets more than percent of the firstplace votes it wins when no movie manages that the one with the fewest firstplace votes is eliminated and its votes are redistributed to the movies that garnered the eliminated ballots secondplace votes and this continues until a winner emerges

it is all terribly confusing but apparently the consensus favorite comes out ahead in the end this means that endofseason awards chatter invariably involves tortured speculation about which film would most likely be voters second or third favorite and then equally tortured conclusions about which film might prevail

in it was a tossup between boyhood and the eventual winner birdman in with lots of experts betting on the revenant or the big short the prize went to spotlight last year nearly all the forecasters declared la la land the presumptive winner and for two and a half minutes they were correct before an envelope snafu was revealed and the rightful winner moonlight was crowned

this year awards watchers are unequally divided between three billboards outside ebbing missouri the favorite and the shape of water which is the baggers prediction with a few forecasting a hail mary win for get out

but all of those films have historical oscarvoting patterns against them the shape of water has nominations more than any other film and was also named the years best by the producers and directors guilds yet it was not nominated for a screen actors guild award for best ensemble and no film has won best picture without previously landing at least the actors nomination since braveheart in this year the best ensemble sag ended up going to three billboards which is significant because actors make up the academys largest branch that film while divisive also won the best drama golden globe and the bafta but its filmmaker martin mcdonagh was not nominated for best director and apart from argo movies that land best picture without also earning best director nominations are few and far between

1.3 Code

And the solution code is below:

```python
    # One Letter Frequency
letter_freq = ['E', 'T', 'A', 'O', 'I', 'N', 'S', 'H', 'R', 'D', 'L', 'U', 'C', 'M', 'W', 'F', 'Y', 'G', 'P', 'B', 'V', 'K', 'X', 'J', 'Q', 'Z']
letter_freq = [x.lower() for x in letter_freq]

# bigram
bigram_freq = ['TH', 'HE', 'IN', 'ER', 'AN', 'RE', 'ON', 'AT', 'EN', 'ND', 'TI', 'ES', 'OR', 'TE', 'OF', 'ED', 'IS', 'IT', 'AL', 'AR', 'ST', 'TO', 'NT', 'NG', 'SE', 'HA', 'AS', 'OU', 'IO', 'LE', 'VE', 'CO', 'ME', 'DE', 'HI', 'RI', 'RO', 'IC', 'NE', 'EA', 'RA', 'CE', 'LI', 'CH', 'LL', 'BE', 'MA', 'SI', 'OM', 'UR']
bigram_freq = [x.lower() for x in bigram_freq]

# trigram
trigram_freq = ['THE', 'AND', 'ING', 'ENT', 'ION', 'HER', 'FOR', 'THA', 'NTH', 'INT', 'ERE', 'TIO', 'TER', 'EST', 'ERS', 'ATI', 'HAT', 'ATE', 'ALL', 'ETH', 'HES', 'VER', 'HIS', 'STA', 'ITH', 'OTH', 'RES', 'ONT', 'ARE', 'EAR', 'THER', 'STH', 'SHE', 'WAS', 'ETHA', 'END', 'HAS', 'ONS', 'ITHA', 'WIT', 'DTH', 'THER', 'THI', 'HICH', 'FTH', 'SIN', 'TED', 'WIT', 'THER', 'HICH']
trigram_freq = [x.lower() for x in trigram_freq]

# exchange two letter in frequency
def exchange(letter1, letter2, one_freq):
    index1 = one_freq.index(letter1)
    index2 = one_freq.index(letter2)
    one_freq[index1], one_freq[index2] = one_freq[index2], one_freq[index1]

# calculate frequency
def freq(text):
    freq = {}
    for c in text:
        if c in freq:
            freq[c] += 1
        else:
            freq[c] = 1
    return freq

# calculate bigram frequency
def b_freq(text):
    freq = {}
    for i in range(len(text)-1):
        bigram = text[i:i+2]
        if bigram in freq:
            freq[bigram] += 1
        else:
            freq[bigram] = 1
    return freq

# Modify one letter frequency by bigram frequency
def modify_by_bigram(sorted_one_freq, two_freq):
    havedone = letter_freq[:4]
    for x in range(len(bigram_freq)):
        i = two_freq[x]
        index0 = sorted_one_freq.index(i[0])
        index1 = sorted_one_freq.index(i[1])
        if letter_freq[index0] not in havedone and bigram_freq[x
            ][0] not in havedone:
            if letter_freq[index0] != bigram_freq[x][0]:
                exchange(letter_freq[index0], bigram_freq[x][0],
                    letter_freq)
        if letter_freq[index1] not in havedone and bigram_freq[x
            ][1] not in havedone:
            if letter_freq[index1] != bigram_freq[x][1]:
                exchange(letter_freq[index1], bigram_freq[x][1],
                    letter_freq)
        havedone.append(letter_freq[index0])
        havedone.append(letter_freq[index1])


# calculate trigram frequency
def t_freq(text):
    freq = {}
    for i in range(len(text)-2):
        trigram = text[i:i+3]
        if trigram in freq:
            freq[trigram] += 1
        else:
            freq[trigram] = 1
    return freq

# Modify one letter frequency by trigram frequency
def modify_by_trigram(sorted_one_freq, three_freq):
    havedone = letter_freq[:4]
    for x in range(len(trigram_freq)):
        i = three_freq[x]
        index0 = sorted_one_freq.index(i[0])
        index1 = sorted_one_freq.index(i[1])
        index2 = sorted_one_freq.index(i[2])
        if letter_freq[index0] not in havedone and trigram_freq[x
            ][0] not in havedone:
            if letter_freq[index0] != trigram_freq[x][0]:
                exchange(letter_freq[index0], trigram_freq[x][0],
                    letter_freq)
        if letter_freq[index1] not in havedone and trigram_freq[x
            ][1] not in havedone:
            if letter_freq[index1] != trigram_freq[x][1]:
                exchange(letter_freq[index1], trigram_freq[x][1],
                    letter_freq)
        if letter_freq[index2] not in havedone and trigram_freq[x
            ][2] not in havedone:
            if letter_freq[index2] != trigram_freq[x][2]:
                exchange(letter_freq[index2], trigram_freq[x][2],
                    letter_freq)
        havedone.append(letter_freq[index0])
        havedone.append(letter_freq[index1])
        havedone.append(letter_freq[index2])

# sort frequency
def sort_freq(freq):
    return sorted(freq.items(), key=lambda x: x[1], reverse=True)

# from cipher text to plain text
def decrypt(sorted_freq_cipher, cipher):
    plain = ""
    for c in cipher:
        if c.isalpha():
            plain += letter_freq[sorted_freq_cipher.index(c)]
        else:
            plain += c
    return plain

with open('cipher.txt', 'r') as f:
    cipher = f.read().strip()

cipher_no_space = cipher.replace("␣", "")
cipher_no_space_newline = cipher_no_space.replace("\n", "")

one_freq = freq(cipher_no_space_newline)
two_freq = b_freq(cipher_no_space_newline)
three_freq = t_freq(cipher_no_space_newline)

sorted_one_freq = sort_freq(one_freq)
sorted_one_freq = [x[0] for x in sorted_one_freq]

sorted_two_freq = sort_freq(two_freq)
sorted_two_freq = [x[0] for x in sorted_two_freq]
modify_by_bigram(sorted_one_freq, sorted_two_freq)

sorted_three_freq = sort_freq(three_freq)
sorted_three_freq = [x[0] for x in sorted_three_freq]
modify_by_trigram(sorted_one_freq, sorted_three_freq)

#exchage u and c
exchange('u', 'c', letter_freq)

#exchange m and u
exchange('m', 'u', letter_freq)

#exchange f and b
exchange('f', 'b', letter_freq)

#exchange v and y
exchange('v', 'y', letter_freq)

#exchange j and q
exchange('j', 'q', letter_freq)

#exchange v and f
```

```python
exchange('v', 'f', letter_freq)

plain = decrypt(sorted_one_freq, cipher)

with open('plain.txt', 'w') as f:
    f.write(plain)
```

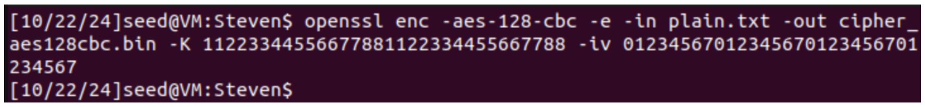
2 Task2

Use the plain file of the last task to encrypt.

2.1 First Encrypt

A plain file is encrypted by AES-128-CBC with the key 11223344556677881122334455667788 and the IV 01234567012345670123456701234567.

```
    openssl enc -aes-128-cbc -e -in plain.txt -out cipher_aes128cbc
        .bin \
    -K 11223344556677881122334455667788 \
    -iv 01234567012345670123456701234567
```

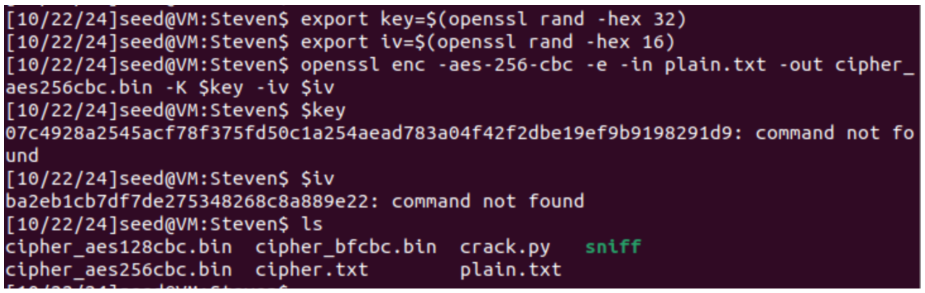


```
[10/22/24]seed@VM:Steven$ openssl enc -aes-128-cbc -e -in plain.txt -out cipher_
aes128cbc.bin -K 11223344556677881122334455667788 -iv 01234567012345670123456701
234567
[10/22/24]seed@VM:Steven$
```

2.2 Second Encrypt

A plain file is encrypted by AES-256-CBC with random key and IV.

```
    openssl enc -aes-256-cbc -e -in plain.txt -out cipher_aes256cbc
        .bin \plainheadrulewidth
    -K $key -iv $iv
```

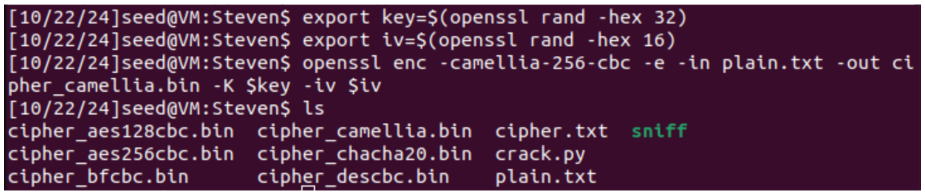


```
[10/22/24]seed@VM:Steven$ export key=$(openssl rand -hex 32)
[10/22/24]seed@VM:Steven$ export iv=$(openssl rand -hex 16)
[10/22/24]seed@VM:Steven$ openssl enc -aes-256-cbc -e -in plain.txt -out cipher_
aes256cbc.bin -K $key -iv $iv
[10/22/24]seed@VM:Steven$ $key
07c4928a2545acf78f375fd50c1a254aead783a04f42f2dbe19ef9b9198291d9: command not fo
und
[10/22/24]seed@VM:Steven$ $iv
ba2eb1cb7df7de275348268c8a889e22: command not found
[10/22/24]seed@VM:Steven$ ls
cipher_aes128cbc.bin  cipher_bfcbc.bin  crack.py   sniff
cipher_aes256cbc.bin  cipher.txt        plain.txt
```

2.3 Third Encrypt

A plain file is encrypted by camellia-128-cbc with random key and IV.

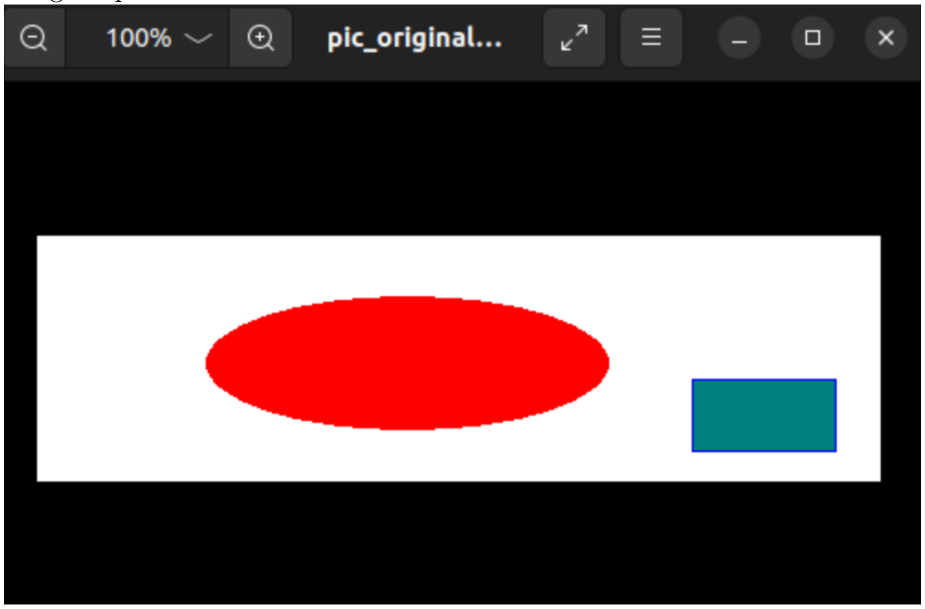
```
openssl enc -camellia-128-cbc -e -in plain.txt -out
    cipher_camellia.bin \
-K $key -iv $iv
```



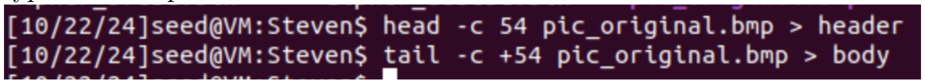
```
[10/22/24]seed@VM:Steven$ export key=$(openssl rand -hex 32)
[10/22/24]seed@VM:Steven$ export iv=$(openssl rand -hex 16)
[10/22/24]seed@VM:Steven$ openssl enc -camellia-256-cbc -e -in plain.txt -out ci
pher_camellia.bin -K $key -iv $iv
[10/22/24]seed@VM:Steven$ ls
cipher_aes128cbc.bin  cipher_camellia.bin  cipher.txt  sniff
cipher_aes256cbc.bin  cipher_chacha20.bin  crack.py
cipher_bfcbc.bin      cipher_descbc.bin    plain.txt
```

3 Task3

The original picture is shown below.



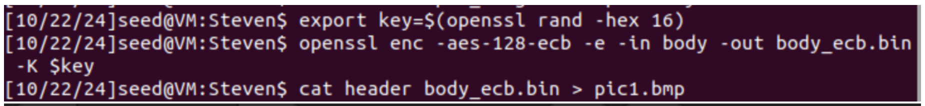
We need to divide the header and the data of the picture first, and we only encrypt the data part.



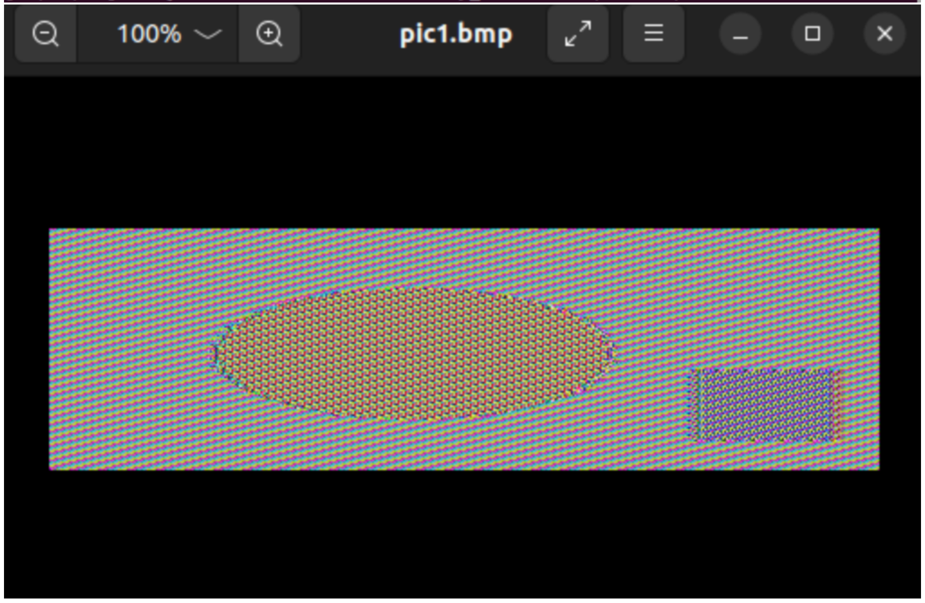
```
[10/22/24]seed@VM:Steven$ head -c 54 pic_original.bmp > header
[10/22/24]seed@VM:Steven$ tail -c +54 pic_original.bmp > body
```

3.1 First Encrypt

Use aes-128-ecb to encrypt the data part and generate pci1. We can find the pic1 has some traits of the original picture.

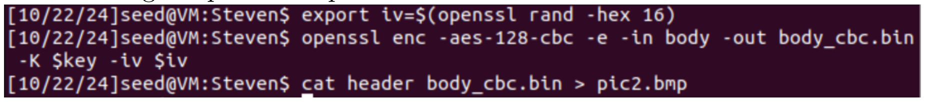


```
[10/22/24]seed@VM:Steven$ export key=$(openssl rand -hex 16)
[10/22/24]seed@VM:Steven$ openssl enc -aes-128-ecb -e -in body -out body_ecb.bin
 -K $key
[10/22/24]seed@VM:Steven$ cat header body_ecb.bin > pic1.bmp
```

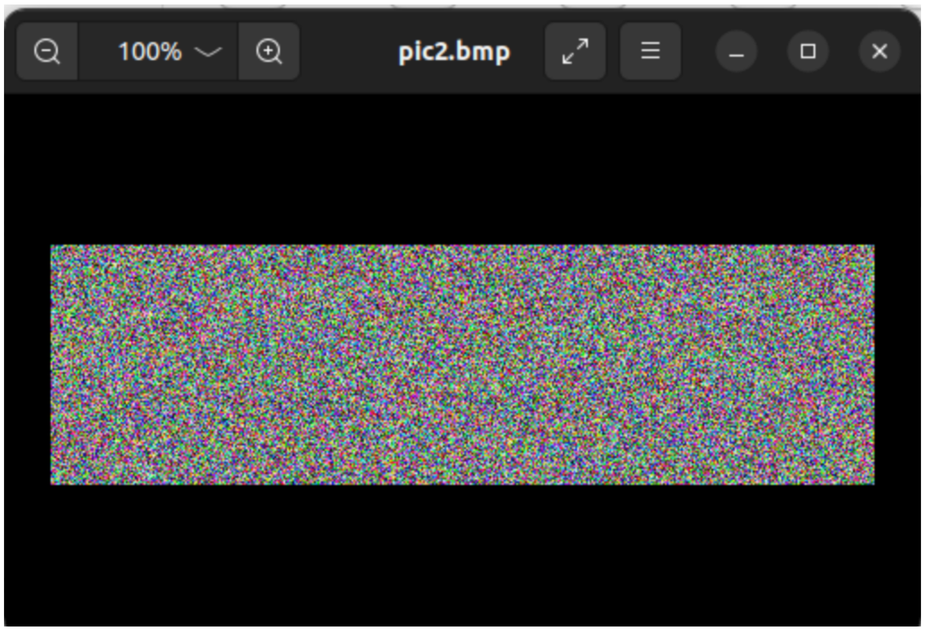


3.2 Second Encrypt

Use aes-128-cbc to encrypt the data part and generate pci2. We can't find any traits of the original picture in pic2.

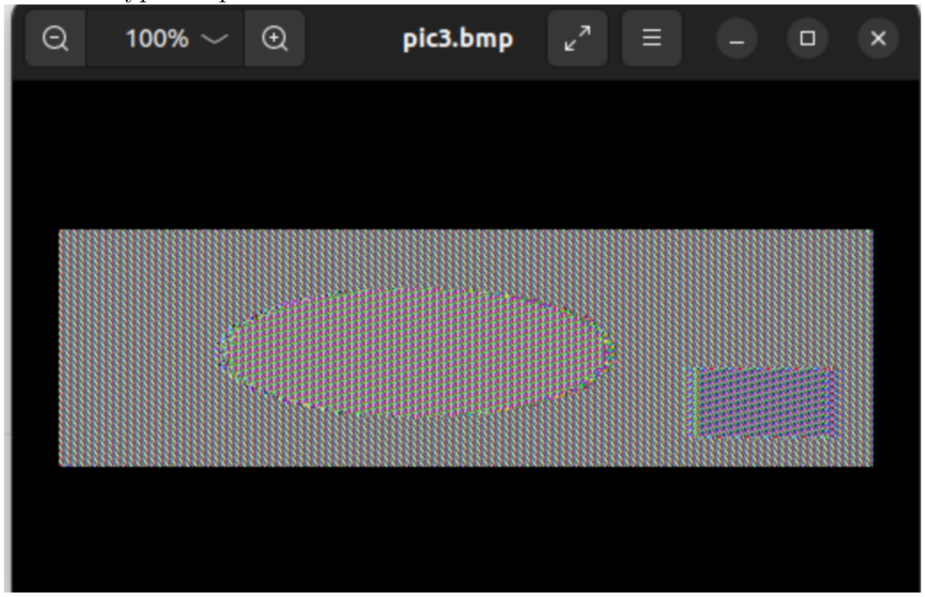


```
[10/22/24]seed@VM:Steven$ export iv=$(openssl rand -hex 16)
[10/22/24]seed@VM:Steven$ openssl enc -aes-128-cbc -e -in body -out body_cbc.bin
 -K $key -iv $iv
[10/22/24]seed@VM:Steven$ cat header body_cbc.bin > pic2.bmp
```



3.3 ECB twice

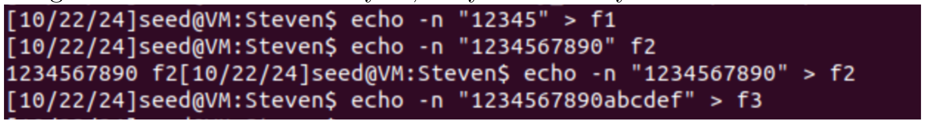
Use aes-128-ecb to encrypt the data part twice and generate pci3. We can still find some traits of the original picture in pic3. This means that ECB is not secure to encrypt the picture data.



4 Task4

For four types of encryption mode, ECB, CBC, CFB, and OFB, The ECB and CBC are block cipher modes, so they need a padding to make the data length to be a multiple of the block size. But for CFB and OFB, they are stream cipher modes, so they don't need padding.

We generate three files with 5bytes, 10bytes and 16bytes.



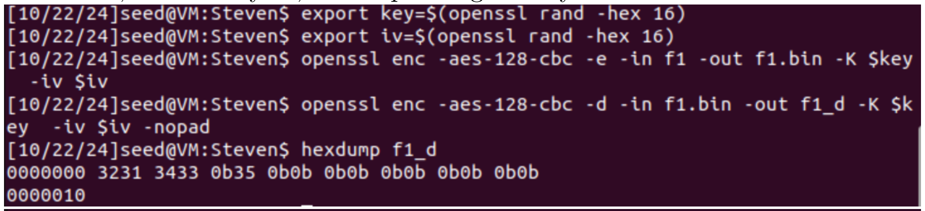
```
[10/22/24]seed@VM:Steven$ echo -n "12345" > f1
[10/22/24]seed@VM:Steven$ echo -n "1234567890" f2
1234567890 f2[10/22/24]seed@VM:Steven$ echo -n "1234567890" > f2
[10/22/24]seed@VM:Steven$ echo -n "1234567890abcdef" > f3
```

And use aes-128-cbc to encrypt them and then decrypt them. We can obviously find that the three file has different padding in the decrypted file. And one block is 16bytes, if the data length is not a multiple of 16, the padding will be added.

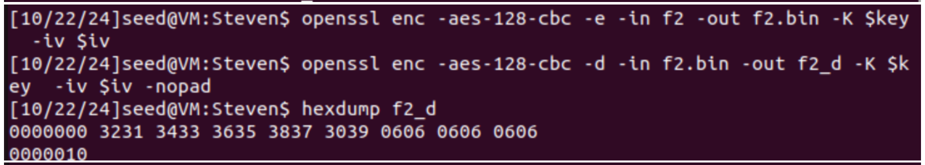
For the f1, it has 5bytes, so the padding is 11bytes and its value is 0x0b.

For the f2, it has 10bytes, so the padding is 6bytes and its value is 0x06.

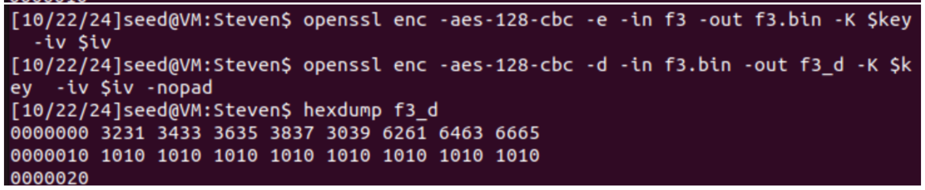
For the f3, it has 16bytes, so the padding is 16bytes and its value is 0x10.



```
[10/22/24]seed@VM:Steven$ export key=$(openssl rand -hex 16)
[10/22/24]seed@VM:Steven$ export iv=$(openssl rand -hex 16)
[10/22/24]seed@VM:Steven$ openssl enc -aes-128-cbc -e -in f1 -out f1.bin -K $key
  -iv $iv
[10/22/24]seed@VM:Steven$ openssl enc -aes-128-cbc -d -in f1.bin -out f1_d -K $k
ey  -iv $iv -nopad
[10/22/24]seed@VM:Steven$ hexdump f1_d
0000000 3231 3433 0b35 0b0b 0b0b 0b0b 0b0b 0b0b
0000010
```



```
[10/22/24]seed@VM:Steven$ openssl enc -aes-128-cbc -e -in f2 -out f2.bin -K $key
  -iv $iv
[10/22/24]seed@VM:Steven$ openssl enc -aes-128-cbc -d -in f2.bin -out f2_d -K $k
ey  -iv $iv -nopad
[10/22/24]seed@VM:Steven$ hexdump f2_d
0000000 3231 3433 3635 3837 3039 0606 0606 0606
0000010
```

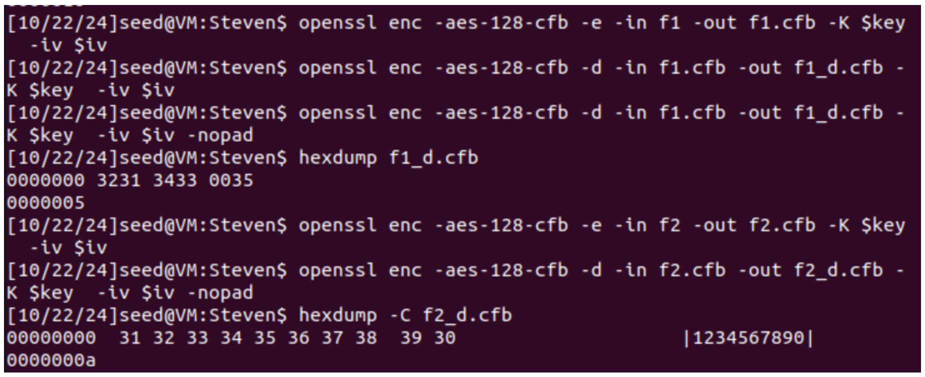


```
[10/22/24]seed@VM:Steven$ openssl enc -aes-128-cbc -e -in f3 -out f3.bin -K $key
  -iv $iv
[10/22/24]seed@VM:Steven$ openssl enc -aes-128-cbc -d -in f3.bin -out f3_d -K $k
ey  -iv $iv -nopad
[10/22/24]seed@VM:Steven$ hexdump f3_d
0000000 3231 3433 3635 3837 3039 6261 6463 6665
0000010 1010 1010 1010 1010 1010 1010 1010 1010
0000020
```

But if we use aes-128-cfb, then we can't find any padding in the decrypted file.

For the f1, it has 5bytes, and the padding is 0bytes.

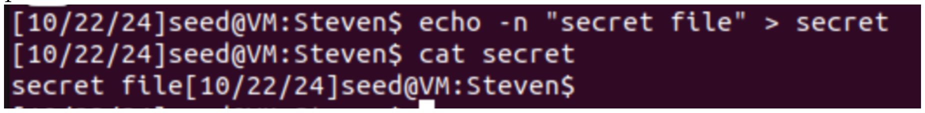
For the f2, it has 10bytes, and the padding is 0bytes.



```
[10/22/24]seed@VM:Steven$ openssl enc -aes-128-cfb -e -in f1 -out f1.cfb -K $key
  -iv $iv
[10/22/24]seed@VM:Steven$ openssl enc -aes-128-cfb -d -in f1.cfb -out f1_d.cfb -
K $key  -iv $iv
[10/22/24]seed@VM:Steven$ openssl enc -aes-128-cfb -d -in f1.cfb -out f1_d.cfb -
K $key  -iv $iv -nopad
[10/22/24]seed@VM:Steven$ hexdump f1_d.cfb
0000000 3231 3433 0035
0000005
[10/22/24]seed@VM:Steven$ openssl enc -aes-128-cfb -e -in f2 -out f2.cfb -K $key
  -iv $iv
[10/22/24]seed@VM:Steven$ openssl enc -aes-128-cfb -d -in f2.cfb -out f2_d.cfb -
K $key  -iv $iv -nopad
[10/22/24]seed@VM:Steven$ hexdump -C f2_d.cfb
00000000  31 32 33 34 35 36 37 38  39 30                    |1234567890|
0000000a
```

5 Task6

Prepare a secret file for test.

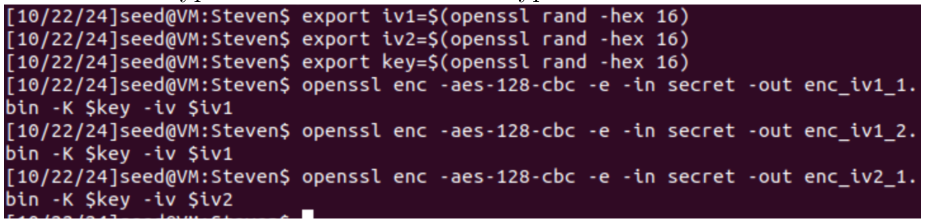


```
[10/22/24]seed@VM:Steven$ echo -n "secret file" > secret
[10/22/24]seed@VM:Steven$ cat secret
secret file[10/22/24]seed@VM:Steven$
```

5.1 Task6.1

Generate two random ivs and one random key.

Use iv1 to encrypt twice and use iv2 to encrypt once.

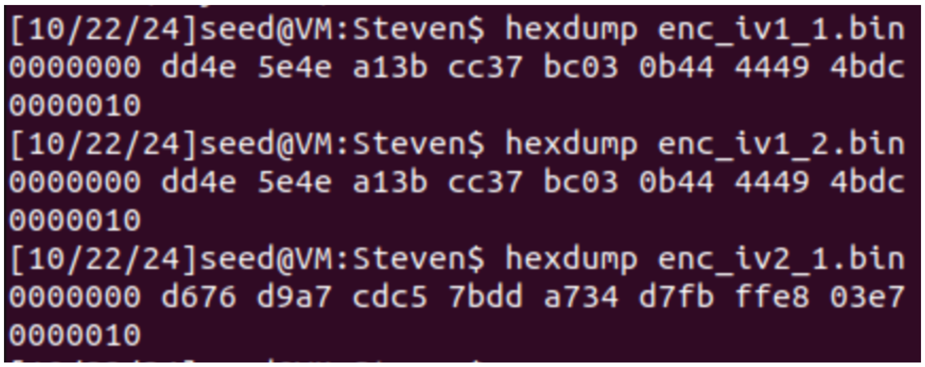


```
[10/22/24]seed@VM:Steven$ export iv1=$(openssl rand -hex 16)
[10/22/24]seed@VM:Steven$ export iv2=$(openssl rand -hex 16)
[10/22/24]seed@VM:Steven$ export key=$(openssl rand -hex 16)
[10/22/24]seed@VM:Steven$ openssl enc -aes-128-cbc -e -in secret -out enc_iv1_1.
bin -K $key -iv $iv1
[10/22/24]seed@VM:Steven$ openssl enc -aes-128-cbc -e -in secret -out enc_iv1_2.
bin -K $key -iv $iv1
[10/22/24]seed@VM:Steven$ openssl enc -aes-128-cbc -e -in secret -out enc_iv2_1.
bin -K $key -iv $iv2
```

The result is shown below. The two files encrypted by iv1 are the same, and the file encrypted by iv2 is different from the other two files. So the iv can change the result of the encryption even if the key is the same.

If iv is not unique, the attacker can build a relationship with an iv to its result, and this will leak some information.

Even worse, hacker can add some payload after the previous encrypted file and use the same iv to encrypt the new file. Then the hacker can get the new file by decrypting the new file with the previous file.



```
[10/22/24]seed@VM:Steven$ hexdump enc_iv1_1.bin
0000000 dd4e 5e4e a13b cc37 bc03 0b44 4449 4bdc
0000010
[10/22/24]seed@VM:Steven$ hexdump enc_iv1_2.bin
0000000 dd4e 5e4e a13b cc37 bc03 0b44 4449 4bdc
0000010
[10/22/24]seed@VM:Steven$ hexdump enc_iv2_1.bin
0000000 d676 d9a7 cdc5 7bdd a734 d7fb ffe8 03e7
0000010
```

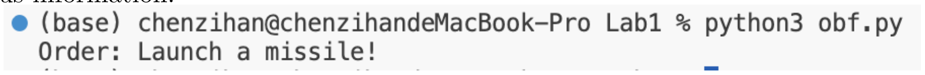
5.2 Task6.2

OFB mode use xor to encrypt the plaintext with key and iv.

So if we know the plaintext and the ciphertext, we can get the key and iv.

```python
P1 = "This is a known message!"
C1 = "a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159"
C2 = "bf73bcd3509299d566c35b5d450337e1bb175f903fafc159"

C1_bytes = bytes.fromhex(C1)
C2_bytes = bytes.fromhex(C2)

P1 = P1.encode()

# R = P1 xor C1
R = bytes([p1 ^ c1 for p1, c1 in zip(P1, C1_bytes)])

# P2 = R xor C2
P2 = bytes([r ^ c2 for r, c2 in zip(R, C2_bytes)])

print(P2.decode())
```

The data is dangerous because the hacker can decrypt the result from previous information.



```
(base) chenzihan@chenzihandeMacBook-Pro Lab1 % python3 obf.py
Order: Launch a missile!
```

If we use CFB mode, we can only get part of the P2 because the CFB mode uses the ciphertext to encrypt the plaintext. Since it is encrypt as a block, so we can reveal a block of message for P2 from the previous information.

5.3 Task6.3

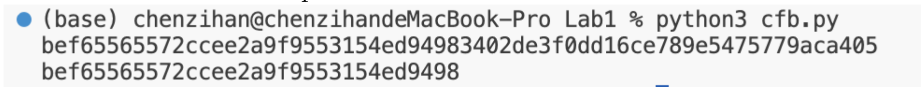
Use p2 add padding and then p2 xor iv1 xor iv2 as the new plaintext.

Send this and get the new ciphertext.

Compare the ciphertext with the previous one, if there is something similar, then they are the same.

```python
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad
import binascii

def aes_encrypt(plaintext, key, iv):
    cipher = AES.new(key, AES.MODE_CBC, iv)
    ciphertext = cipher.encrypt(pad(plaintext.encode(), AES.block_size))
    return binascii.hexlify(ciphertext).decode()

key = binascii.unhexlify('00112233445566778899aabbccddeeff')  # key
iv1 = binascii.unhexlify('31323334353637383930313233343536')  # IV1
iv2 = binascii.unhexlify('31323334353637383930313233343537')  # IV2

p1_yes = "Yes"
p1_no = "No"

c1_yes = aes_encrypt(p1_yes, key, iv1)
c1_no = aes_encrypt(p1_no, key, iv1)

p2 = "Yes"

# p2 = p2 xor iv1 xor iv2
p2_bytes = bytes(p2.encode())
# p2 add padding
p2_bytes = pad(p2_bytes, AES.block_size)
iv1_bytes = bytes(iv1)
iv2_bytes = bytes(iv2)

p2 = bytes([p2 ^ iv1 ^ iv2 for p2, iv1, iv2 in zip(p2_bytes, iv1_bytes, iv2_bytes)])

c2 = aes_encrypt(p2.decode(), key, iv2)

# print c2 and c1_yes
print(c2)
print(c1_yes)
```

So we can know that the p1 is "Yes".

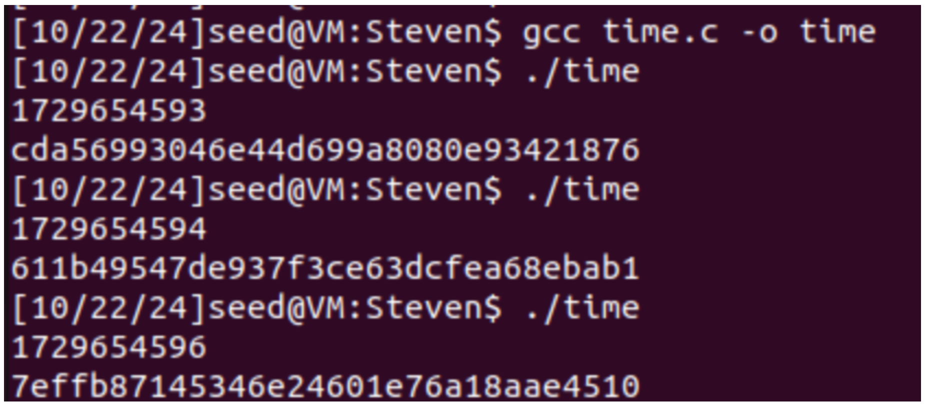


```
(base) chenzihan@chenzihandeMacBook-Pro Lab1 % python3 cfb.py
bef65565572ccee2a9f9553154ed94983402de3f0dd16ce789e5475779aca405
bef65565572ccee2a9f9553154ed9498
```

6 Topic 2: Task1

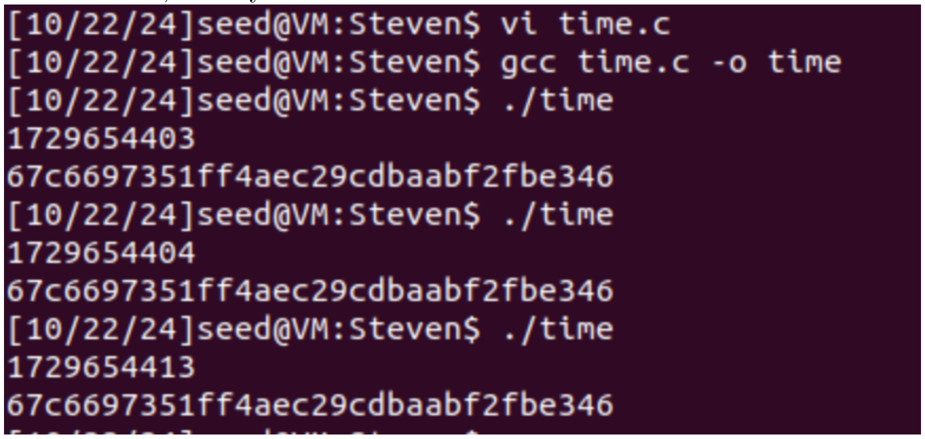
Time is a seed to generate a random number.

With time, the key is different.



```
[10/22/24]seed@VM:Steven$ gcc time.c -o time
[10/22/24]seed@VM:Steven$ ./time
1729654593
cda56993046e44d699a8080e93421876
[10/22/24]seed@VM:Steven$ ./time
1729654594
611b49547de937f3ce63dcfea68ebab1
[10/22/24]seed@VM:Steven$ ./time
1729654596
7effb87145346e24601e76a18aae4510
```

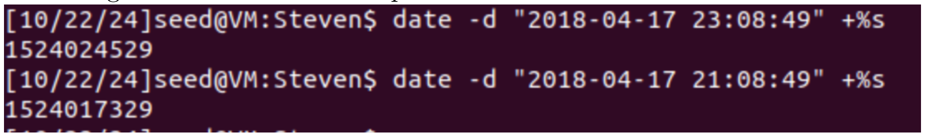
Without time, the key is the same.



```
[10/22/24]seed@VM:Steven$ vi time.c
[10/22/24]seed@VM:Steven$ gcc time.c -o time
[10/22/24]seed@VM:Steven$ ./time
1729654403
67c6697351ff4aec29cdbaabf2fbe346
[10/22/24]seed@VM:Steven$ ./time
1729654404
67c6697351ff4aec29cdbaabf2fbe346
[10/22/24]seed@VM:Steven$ ./time
1729654413
67c6697351ff4aec29cdbaabf2fbe346
```

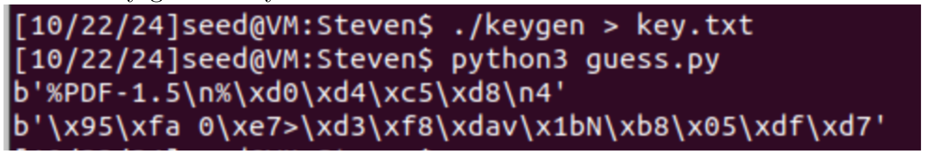
7 Topic 2: Task2

Use date to get the random number possible seed.



```
[10/22/24]seed@VM:Steven$ date -d "2018-04-17 23:08:49" +%s
1524024529
[10/22/24]seed@VM:Steven$ date -d "2018-04-17 21:08:49" +%s
1524017329
```

Successfully get the key.



```
[10/22/24]seed@VM:Steven$ ./keygen > key.txt
[10/22/24]seed@VM:Steven$ python3 guess.py
b'%PDF-1.5\n%\xd0\xd4\xc5\xd8\n4'
b'\x95\xfa 0\xe7>\xd3\xf8\xdav\x1bN\xb8\x05\xdf\xd7'
```

Generate Key code.

```
    #include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

```c
#define KEYSIZE 16
void main()
{
    int j;
    char key[KEYSIZE];
    time_t start = 1524017329;
    time_t end = 1524024529;
    for(time_t i = start; i <= end; i++) {
        srand(i);
        for (j = 0; j< KEYSIZE; j++){
            key[j] = rand()%256;
            printf("%.2x", (unsigned char)key[j]);
        }
        printf("\n");
    }
}
```

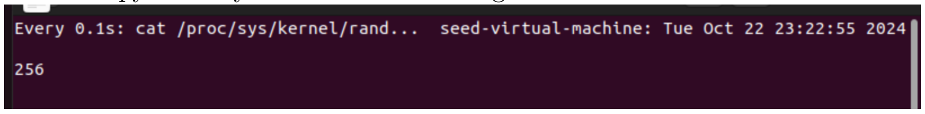
Guess key code.

```python
    import time
import random
from Crypto.Cipher import AES
from binascii import unhexlify

#Plaintext: 255044462d312e350a25d0d4c5d80a34
#Ciphertext: d06bf9d0dab8e8ef880660d2af65aa82
#IV: 09080706050403020100A2B2C2D2E2F2

P = bytes.fromhex("255044462d312e350a25d0d4c5d80a34")
C = bytes.fromhex("d06bf9d0dab8e8ef880660d2af65aa82")
IV = bytes.fromhex("09080706050403020100A2B2C2D2E2F2")

print(P)
random1 = 1524017329
random2 = 1524024529

def generate_key(seed):
    random.seed(seed)
    key = bytearray()
    for _ in range(16):
        key.append(random.randint(0, 255))
    return bytes(key)

def decrypt(ciphertext, key, iv):
    cipher = AES.new(key, AES.MODE_CBC, iv)
    return cipher.decrypt(ciphertext)

# read all key as bytes
with open("key.txt", "r") as f:
    keys = [bytes.fromhex(line.strip()) for line in f]

for key in keys:
    plaintext = decrypt(C, key, IV)
    if plaintext.startswith(b"%PDF"):
        print(key)
        break
```

8 Topic 2: Task3

Maybe because the virtual environment in Apple is different. I can't find any entropy change even if I do many different movements.

The entropy is always 256 and never changes.



```
Every 0.1s: cat /proc/sys/kernel/rand...  seed-virtual-machine: Tue Oct 22 23:22:55 2024

256
```