Information Security

Assignment _01

Umer Fazal 22k-4726

Muhammad Ali 22k-4691

## Task 1: Frequency Analysis

Decrypted text using the key mentioned below then use tr command to substitude .

```
[09/16/24]seed@VM:~/.../Files$ tr 'abcdefghijklmnopqrstuvwxyz' 'cfmypvbrlqxwiezd
sgkhnajotu' <ciphertext.txt> plaintext.txt
[09/16/24]seed@VM:~/.../Files$ cat plaintext.txt
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
```

## Task 2: Encryption using Different Ciphers and Modes

These are the encrypted file

```
cipher_aes128cbc.bin
cipher_aes128cfb.bin
cipher_aes128ctr.bin
```

Using –aes-128-cbc, aes-128-cfb, aes-128-ctr.

```
[09/17/24]seed@VM:~/.../Files$ cat keys.txt
key=F8B270218BEDD6E6FEE5ED99787C1AC5
iv =480D8F2424A07C1B4C453605F5A4213C
iv(1)=5059250eb1842febb0b7754bb3dcb6b3
[09/17/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in plain.txt -out ci
pher_aes128cbc.bin -K F8B270218BEDD6E6FEE5ED99787C1AC5 -iv 480D8F2424A07C1B4C453
605F5A4213C
[09/17/24]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -e -in plain.txt -out ci
pher_aes128cfb.bin -K F8B270218BEDD6E6FEE5ED99787C1AC5 -iv 480D8F2424A07C1B4C453
605F5A4213C
[09/17/24]seed@VM:~/.../Files$ openssl enc -aes-128-ctr -e -in plain.txt -out ci
pher_aes128ctr.bin -K F8B270218BEDD6E6FEE5ED99787C1AC5 -iv 480D8F2424A07C1B4C453
605F5A4213C
```

Key and IV is generated randomly

```
[09/17/24]seed@VM:~/.../Files$ openssl rand -hex 16
```

## Task 3: Encryption Mode – ECB vs. CBC

First I encrypt the picture using aes128cbc and aes128ecb ciphers by using openssl command

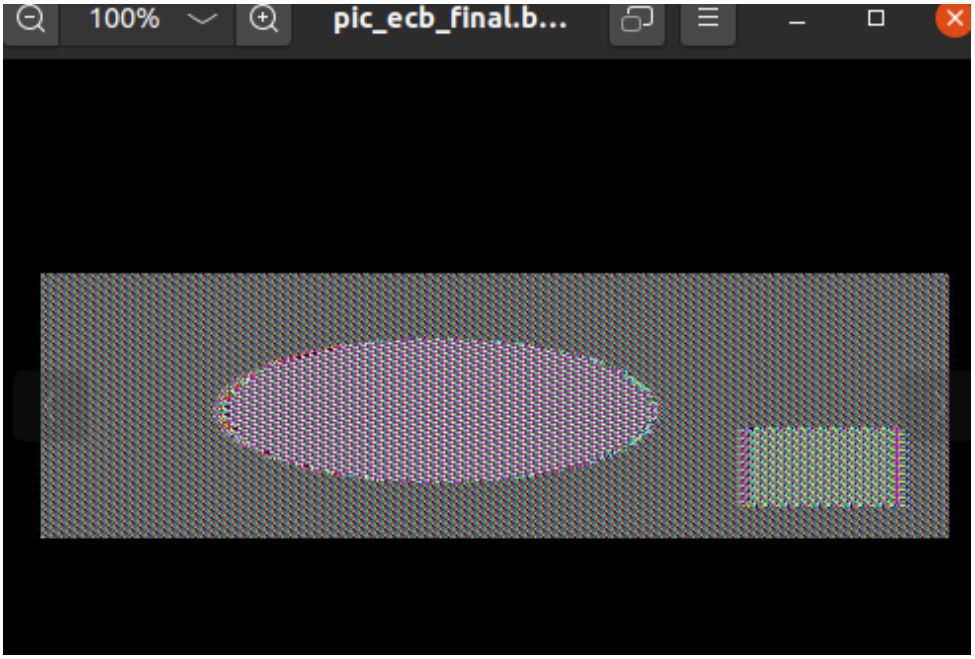
```
[09/17/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -in pic_original.bmp -ou
t encrytedpic_cbc.bmp -K F8B270218BEDD6E6FEE5ED99787C1AC5 -iv 480D8F2424A07C1B4C
453605F5A4213C
[09/17/24]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -in pic_original.bmp -ou
t encrytedpic_ecb.bmp -K F8B270218BEDD6E6FEE5ED99787C1AC5
```

Then, extract the header of original pic and save it into header, same for encrypted pic and then combine it accordingly .

```
[09/17/24]seed@VM:~/.../Files$ head -c 54 pic_original.bmp > header
[09/17/24]seed@VM:~/.../Files$ tail -c +55 encryptedpic_cbc.bmp > body_cbc
[09/17/24]seed@VM:~/.../Files$ tail -c +55 encryptedpic_ecb.bmp > body_ecb
[09/17/24]seed@VM:~/.../Files$ cat header body_cbc > pic_cbc_final.bmp
[09/17/24]seed@VM:~/.../Files$ cat header body_ecb > pic_ecb_final.bmp
```

This image is of CBC cipher.

This Image is of ECB cipher.




In the ECB mode encrypted image, you will likely still be able to see a **pattern** of the original image. This is because ECB encrypts identical blocks of plaintext into identical blocks of ciphertext, allowing patterns to remain visible. This is why ECB is generally not recommended for encrypting images or other files where visual patterns are important.

In the CBC mode encrypted image, you will likely not be able to make out any patterns from the original image. This is because CBC uses an initialization vector (IV) and each block is XOR'd with the previous block before encryption, which breaks up any repeating patterns in the plaintext. The result should appear much more random and provide better security against analysis.

## Task 4: Padding

These are the files,

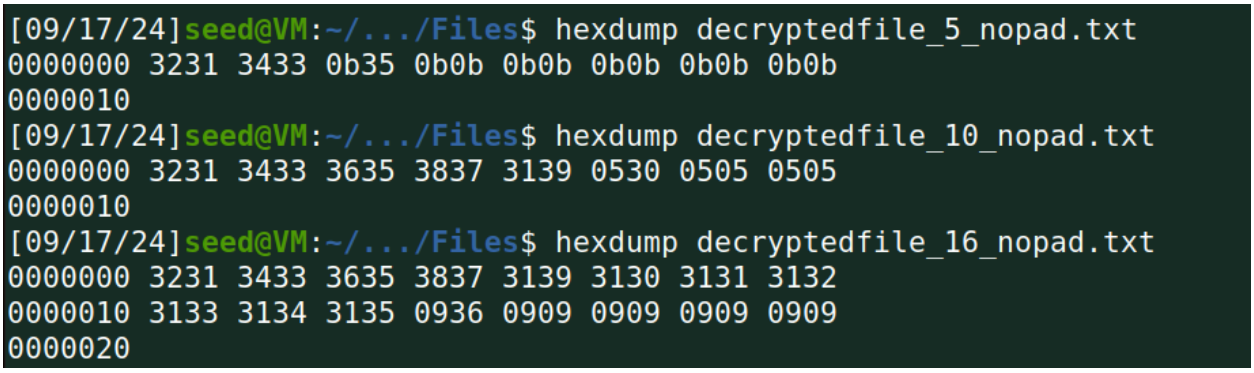
```
[09/17/24]seed@VM:~/.../Files$ cat file_5.txt
12345[09/17/24]seed@VM:~/.../Files$ cat file_10.txt
12345678910[09/17/24]seed@VM:~/.../Files$ cat file_16.txt
12345678910111213141516[09/17/24]seed@VM:~/.../Files$
```

Encrypted by using openssl enc command using aes128cbc cipher.

```
[09/17/24]seed@VM:~/.../Files$ cat encryptedfile_16.txt
5~Q!K;p}`Ze
[09/17/24]seed@VM:~/.../Files$ cat encryptedfile_10.txt
J RO[09/17/24]seed@VM:~/.../Files$ cat encryptedfile_5.txt
e5Dn vS[09/17/24]seed@VM:~/.../Files$
```

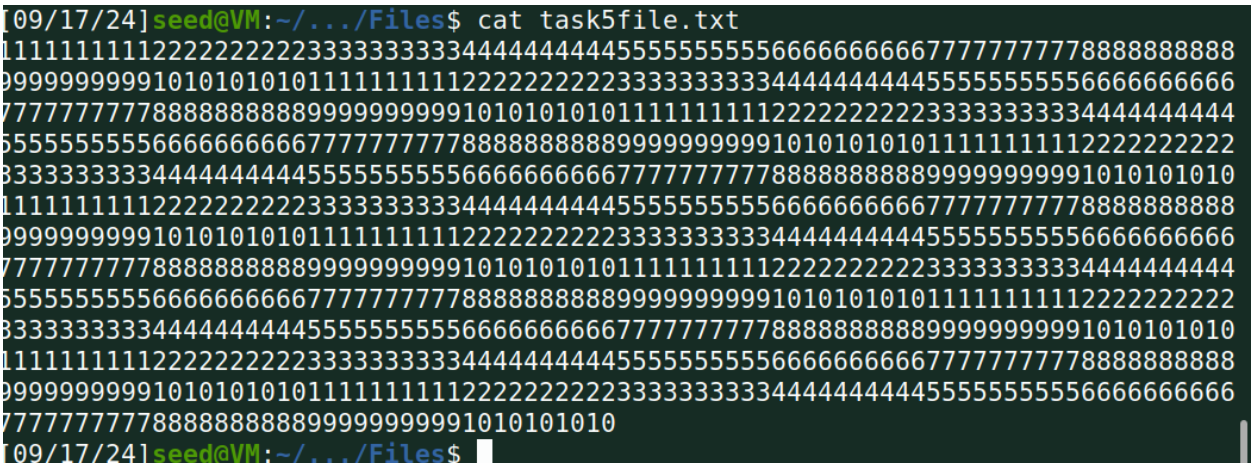
These images of hex encrypted file are with padding


These image of hex without padding

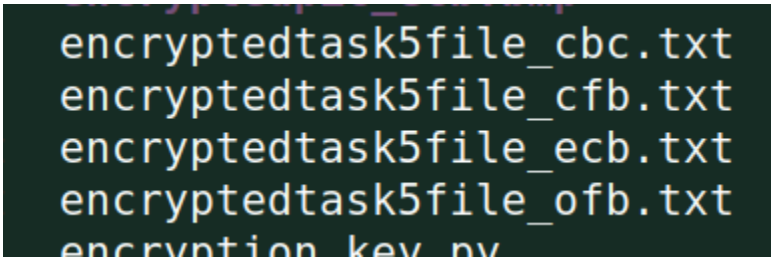



## Task 5: Error Propagation – Corrupted Cipher Text

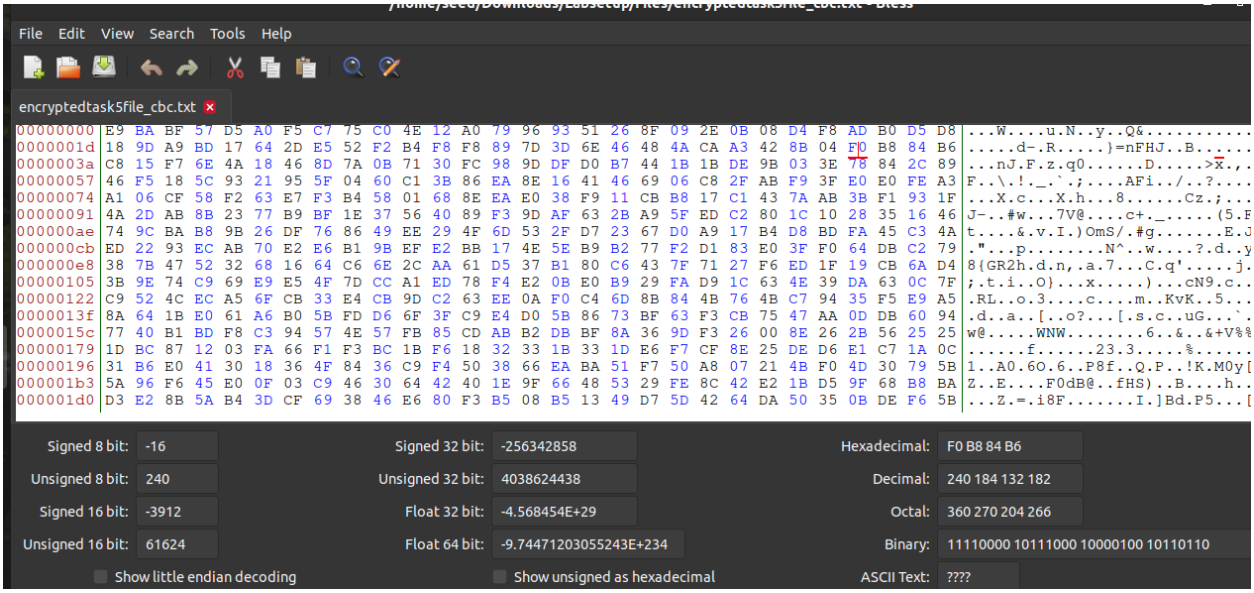
This is a file.



Encrypted using cbc,cfb,ecb,ofb cipher aes128.

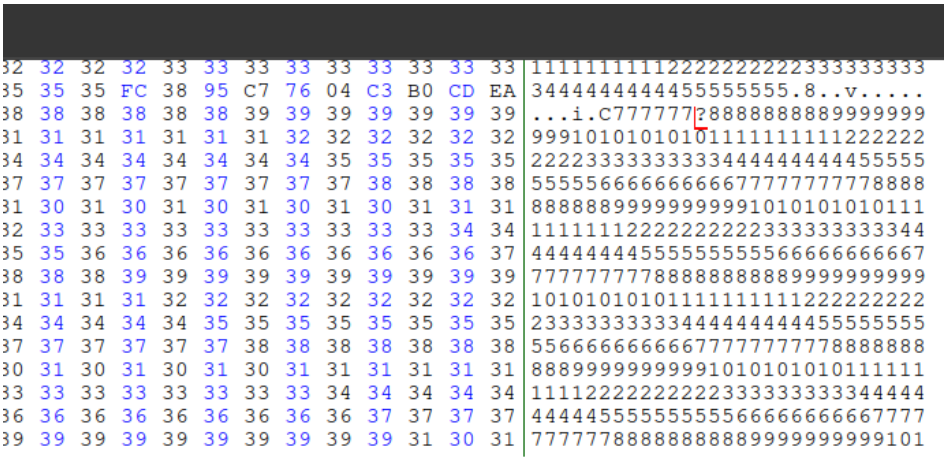



Open bless tool to edit hex decimal value for all the cipher




corrupting the 55th byte and then decrypting the files again.

In CBC 14 block corrupted,



In CFB, 16 block corrupted,

```
33 11111111112222222222333333333
36 34444444444555555555566664666
39 667777...V.a.@h..g....9999999
32 99910101010101111111111222222
35 22223333333333444444444455555
38 55555666666666677777777778888
31 88888899999999991010101010111
34 11111112222222222333333333344
37 44444444555555555566666666667
39 77777777788888888889999999999
32 10101010101111111111222222222
35 23333333333444444444455555555
38 55666666666677777777778888888
31 88899999999991010101010111111
34 11112222222222333333333344444
37 44444555555555566666666667777
31 77777788888888889999999999101
```

In ECB 15 block corrupted,

```
33 33 33 11111111112222222222333333333
01 FA DF 34444444444555555555.........
39 39 39 f...W.77777788888888889999999
32 32 32 99910101010101111111111222222
35 35 35 22223333333333444444444455555
38 38 38 55555666666666677777777778888
31 31 31 88888899999999991010101010111
33 34 34 11111112222222222333333333344
36 36 37 44444444555555555566666666667
39 39 39 77777777788888888889999999999
32 32 32 10101010101111111111222222222
35 35 35 23333333333444444444455555555
38 38 38 55666666666677777777778888888
31 31 31 88899999999991010101010111111
34 34 34 11112222222222333333333344444
37 37 37 44444555555555566666666667777
31 30 31 77777788888888889999999999101
```

In OFB, The corruption affects only the specific byte corresponding to the corrupted ciphertext.

```
11111111222222222333333333
444444445555555555666>666
77777777788888888889999999
91010101010111111111222222
22333333333344444444455555
55566666666677777777778888
```

## Task 6: Initial Vector (IV) and Common Mistakes

### Task 6.1. IV Experiment

```
iv =480D8F2424A07C1B4C453605F5A4213C
iv(1)=5059250eb1842febb0b7754bb3dcb6b3
```

These are two different IV, first I encrypted the file using two different IV by using openssl command and then view its hex value (for my clarification).

```
09/17/24]seed@VM:~/.../Files$ hexdump encryptedfile_61.txt
0000000 6a10 c38c f8cd d91c c5d7 ae34 5f68 287a
0000010 2b90 7805 4cda 88a3 5340 7ffd 5bc4 8f36
0000020 9896 0525 cfb0 0015 dc65 4b7d 34da 1c45
0000030 4fc8 9d75 a73b e7d8 8f77 2a0e 6412 e6ee
0000040 3dfe f5c5 6ca3 20e3 3507 2656 19bd 5951
0000050 5207 a2e0 b198 cc9f 07e4 e7eb 736a c05a
0000060 e26d 2e00 9c32 f87f 3373 b237 6fd5 a314
0000070 757a 32fd 7c3d 186b 7960 12ad 4ec0 99d2
```

```
[09/17/24]seed@VM:~/.../Files$ hexdump encryptedfile_611.txt
0000000 0268 06ca 65b0 cfb6 b071 6cf4 478b 93b4
0000010 a28a db39 856c e488 2f44 276b aba9 9126
0000020 2684 6c26 1093 b122 6102 2e38 76bb e5bf
0000030 c956 b0fa daac 328b db78 5f04 e0f0 24bb
0000040 6ac6 1c4c 4d71 8c65 6a7a 1bdd a433 e09f
0000050 c3aa 57a5 fa25 2c35 048e 3414 4a1d f99c
0000060 886e aa4e 6762 93d8 438d 2efd a8e5 9f6d
0000070 afa2 7328 b3d7 f3bc e840 1842 89ce 5b34
0000080 2b7f 4fb3 5fe2 2818 c7a6 1250 2533 db72
0000090 1922 aeef 7d02 f18a a355 03e0 db3f d24a
00000a0 8bf5 8359 7fdc 81a0 d519 a5f3 b274 7e52
```

```
[09/17/24]seed@VM:~/.../Files$ diff encryptedfile_61.txt encryptedfile_611.txt
Binary files encryptedfile_61.txt and encryptedfile_611.txt differ
```

When different IVs are used, the ciphertexts are distinct even if the plaintexts are the same.

When the same IV is used, the ciphertext is identical for the same plaintext.

## Task 6.2. Common Mistake: Use the Same IV

I used a python code to recovered the plain text,

### Code:

```
def xor_strings(s1, s2):
    # Perform XOR on byte arrays
    return bytes(a ^ b for a, b in zip(s1, s2))

P1 = "This is a known message!"
C1_hex = "a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159"
C2_hex = "bf73bcd3509299d566c35b5d450337e1bb175f903fafc159"

# Convert P1 to bytes
P1_bytes = P1.encode('utf-8')

# Convert hex to bytes
C1 = bytes.fromhex(C1_hex)
C2 = bytes.fromhex(C2_hex)

# XOR C1 and P1_bytes to get the keystream
keystream = xor_strings(P1_bytes, C1)

# XOR keystream with C2 to get P2
P2 = xor_strings(keystream, C2)

# Convert P2 from bytes back to a string (if it's a valid UTF-8 string)
try:
    P2_text = P2.decode('utf-8')
    print("Recovered P2:", P2_text)
except UnicodeDecodeError:
    print("Recovered P2 (bytes):", P2)
```

```
[09/17/24]seed@VM:~/.../Files$ python3 pythoncode.py
Recovered P2: Order: Launch a missile!
```

## Task 6.3. Common Mistake: Use a Predictable IV

Using Docker,

```
[09/17/24]seed@VM:~/.../encryption_oracle$ sudo service docker status
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset>
     Active: active (running) since Mon 2024-09-16 06:33:48 EDT; 21h ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 841 (dockerd)
      Tasks: 8
     Memory: 76.5M
```

```
[09/17/24]seed@VM:~/.../encryption_oracle$ sudo docker run -it -p 3000:3000 encr
yption-oracle
Server listening on 3000 for known_iv
Connect to 3000, launching known_iv
```

```
[09/17/24]seed@VM:~/.../encryption_oracle$ nc localhost 3000
Bob's secret message is either "Yes" or "No", without quotations.
Bob's ciphertex: 7f1fcd90419d2424fe878be5a795897b
The IV used    : 9d2b5be41f76d97cc7ee5a832339be24

Next IV        : 361ff90f2076d97cc7ee5a832339be24
Your plaintext : aabbccdeeff
Invalid hex string

Next IV        : acc7f86e2076d97cc7ee5a832339be24
Your plaintext : 112233445566aabbccddee
Your ciphertext: 272231c0c20b999ba3d8546ff0db545d

Next IV        : d63f33ca2076d97cc7ee5a832339be24
Your plaintext : 223344556677bbccddeeff
Your ciphertext: 5e98e207175ca946a1de2ad12af5ada1

Next IV        : a7ab00ff2076d97cc7ee5a832339be24
Your plaintext : 11aa22bb33cc44dd55ee66ff
Your ciphertext: 6c5478f7ce37e152bdc6747255755cff

Next IV        : ff8d2b6d2176d97cc7ee5a832339be24
```

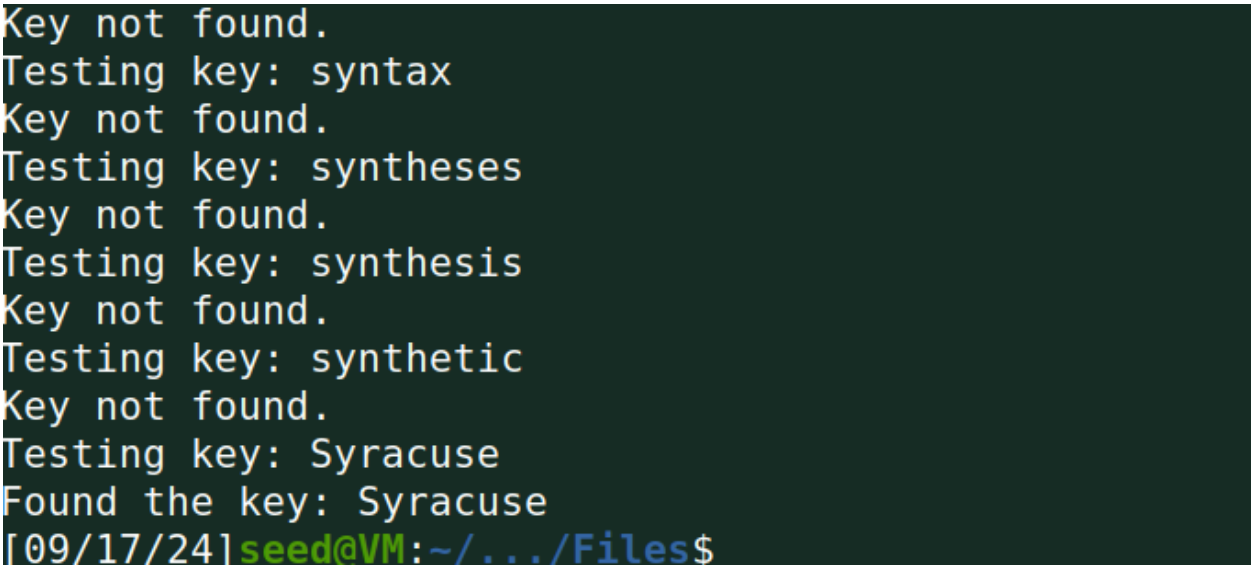
## Task 7: Programming using the Crypto Library

So in this created a python script where original text is “This is a top secret” Where hex and iv is given have to find the key where aes-128-cbc encryption is applied have to decrypt the text and find the key through the wordlist if matched.

## Code:

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
import binascii

# Function to pad the key with '#' to 16 bytes
def pad_key(word):
    word = word.strip()  # Remove any whitespace/newlines
    key = word.encode('utf-8')  # Convert the word to bytes
    if len(key) < 16:
        key += b'#' * (16 - len(key))  # Pad with '#' to make it 16 bytes
    return key[:16]  # Return exactly 16 bytes (cut extra if longer)

# Function to decrypt the ciphertext using AES-128-CBC
```

```
def decrypt_aes(ciphertext, key, iv):
    cipher = AES.new(key, AES.MODE_CBC, iv)  # Initialize AES with key and IV
    decrypted = cipher.decrypt(ciphertext)
    try:
        return unpad(decrypted, AES.block_size).decode('utf-8')  # Remove padding and decode
    except (ValueError, UnicodeDecodeError):
        return None  # Return None if the decrypted text is not valid or padding is incorrect

# Given data (ciphertext, IV, and known plaintext)
ciphertext_hex = '764aa26b55a4da654df6b19e4bce00f4ed05e09346fb0e762583cb7da2ac93a2'
iv_hex = 'aabbccddeeff00998877665544332211'
known_plaintext = 'This is a top secret.'

# Convert the hex strings to bytes
ciphertext = binascii.unhexlify(ciphertext_hex)
iv = binascii.unhexlify(iv_hex)

# Open the wordlist and try each word
with open('words.txt', 'r') as wordlist:
    for word in wordlist:
        word = word.strip()  # Remove any trailing newlines or spaces
        key = pad_key(word)  # Pad the word to form the 16-byte key

        # Print the current key being tested
        print("Testing key: {}".format(word))

        # Decrypt the ciphertext with the current key
        decrypted_text = decrypt_aes(ciphertext, key, iv)

        # Check if the decrypted text matches the known plaintext
        if decrypted_text == known_plaintext:
            print("Found the key: {}".format(word))
            break  # Stop once the key is found
        else:
            print("Key not found.")
```


Key not found.
Testing key: syntax
Key not found.
Testing key: syntheses
Key not found.
Testing key: synthesis
Key not found.
Testing key: synthetic
Key not found.
Testing key: Syracuse
Found the key: Syracuse
[09/17/24]seed@VM:~/.../Files$