SECRET-KEY ENCRYPTION REPORT

Computer Security

Task 1:

Step 1: create python file with the following code:

```
#!/bin/env python3
import random

s = "abcdefghijklmnopqrstuvwxyz"
list = random.sample(s, len(s))
key = ''.join(list)
print(key)
```

Output Screenshot:

```
[09/27/24]seed@VM:~/.../Files$ python3 task1.py
xqcmogfvelyubhsiznjdkawrp
```

```
[09/27/24]seed@VM:~/.../Files$ nano article.txt
[09/27/24]seed@VM:~/.../Files$  tr [:upper:] [:lower:] < article.txt > lowercase.txt
[09/27/24]seed@VM:~/.../Files$ nano lowercase.txt
```

Step 2: Change from uppercase to lowercase, remove punctuation and numbers:

1. *Upper to lower case*

```
  GNU nano 4.8
hi, my name is veezish! i'm 22 years old.
```

2. *Remove punctuation and numbers:*

Step 3: Encrypt letters, while leaving the space and return characters alone.

```
  GNU nano 4.8
hi my name is veezish im  years old
```

Frequency: Count how many times letters come up in the encrypted file, output:

```
[09/27/24]seed@VM:~/.../Files$  ./freq.py
----------------------------------
1-gram (top 20):
b: 4
w: 4
v: 3
l: 3
q: 2
z: 2
s: 2
g: 1
u: 1
c: 1
a: 1
k: 1
p: 1
r: 1
```

```
3-gram (top 20):
gsv: 1
svw: 1
uww: 1
wwc: 1
wcb: 1
cbl: 1
blq: 1
zws: 1
wsa: 1
sal: 1
kpr: 1
```

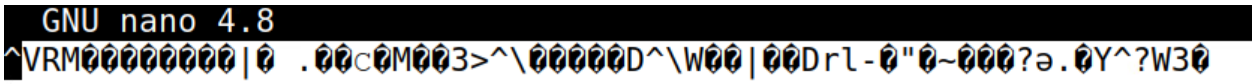
```
--------------------
2-gram (top 20):
bl: 2
qb: 1
vz: 1
gs: 1
sv: 1
vw: 1
uw: 1
ww: 1
wc: 1
cb: 1
lq: 1
bv: 1
zw: 1
ws: 1
sa: 1
al: 1
kp: 1
pr: 1
--------------------
```

Task 2: Try 3 Different Encryptions using openssl:

1. aes-256-cbc

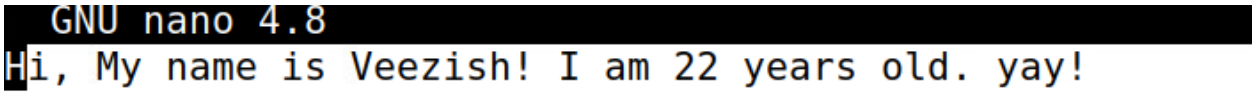
```
[09/27/24]seed@VM:~/.../Files$ openssl enc -aes-256-cbc -e -in plain.txt -out cipher.bin \
> -K 00112233445566778889aabbccddeeff00112233445566778889aabbccddeeff \
> -iv 01020304050607080102030405060708
```

a. Encryption



b. Decryption

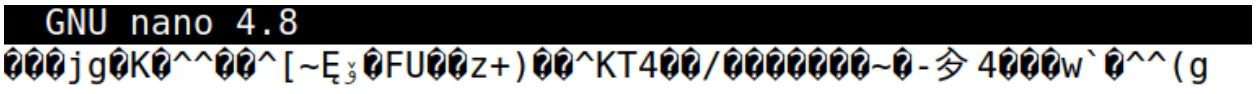
```
[09/27/24]seed@VM:~/.../Files$ openssl enc -aes-256-cbc -d -in cipher.bin -out cipherdecrypt.bin -K 00112233445566778889aabb
45566778889aabbccddeeff -iv 01020304050607080102030405060708
[09/27/24]seed@VM:~/.../Files$ nano cipherdecrypt.bin
```



2. -bf-cbc

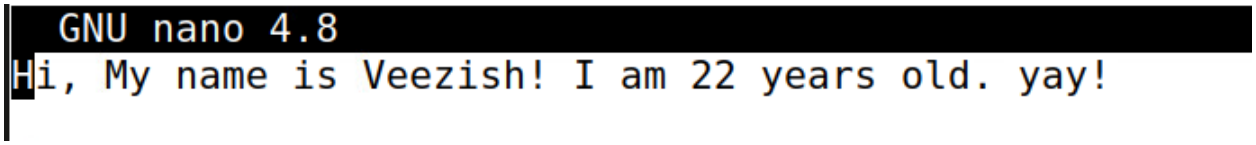
a. Encryption

```
[09/27/24]seed@VM:~/.../Files$ openssl enc --bf-cbc -e -in plain.txt -out cipher.bin -K 00112233445566778889aabbccddeeff0011
abbccddeeff -iv 01020304050607080102030405060708
```



b. Decryption

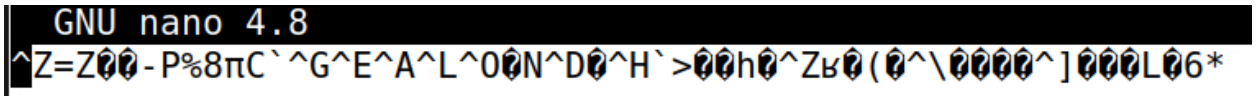
```
[09/27/24]seed@VM:~/.../Files$ openssl enc --bf-cbc -d -in cipher.bin -out cipherdecrypt.bin -K 00112233445566778889aabbccdd
6778889aabbccddeeff -iv 01020304050607080102030405060708
```



3. aes-128-cfb

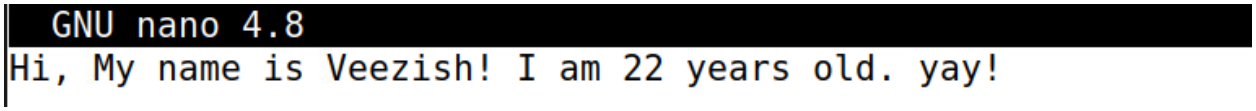
a. Encryption

```
[09/27/24]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -e -in plain.txt -out cipher.bin -K 00112233445566778889aabbccddeeff00112233445566778
889aabbccddeeff -iv 01020304050607080102030405060708
```



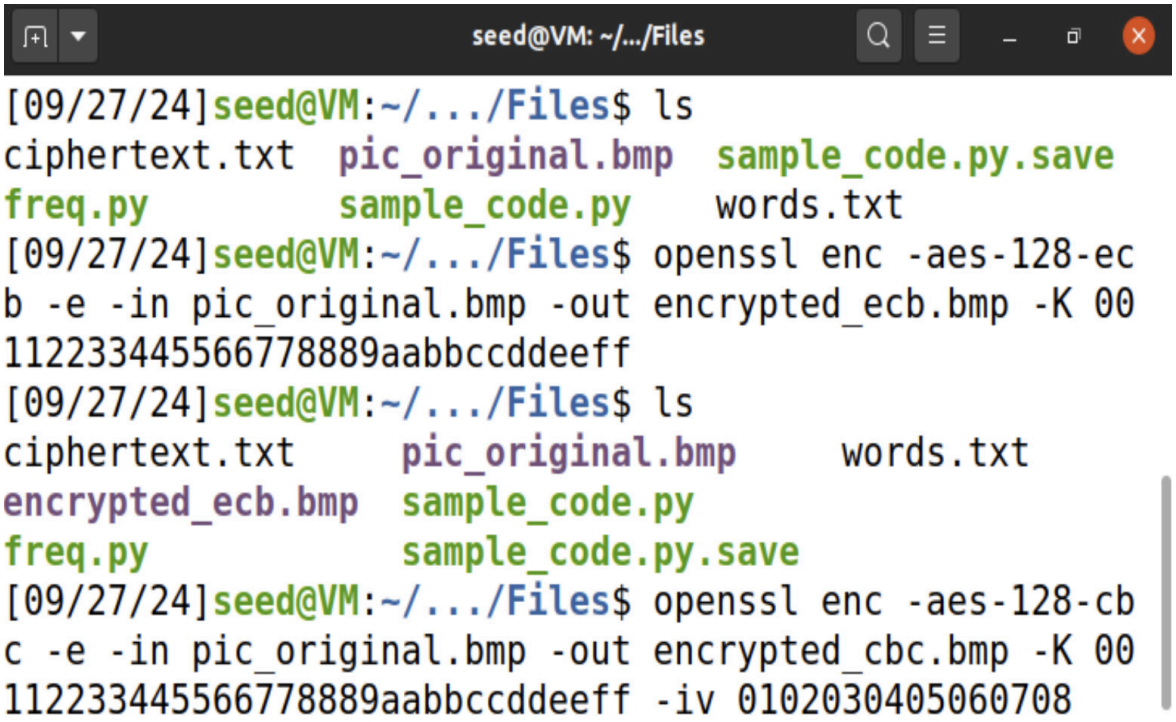
b. Decryption

```
[09/27/24]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -d -in cipher.bin -out cipherdecrypt.bin -K 00112233445566778889aabbccddeeff001122334
45566778889aabbccddeeff -iv 01020304050607080102030405060708
```

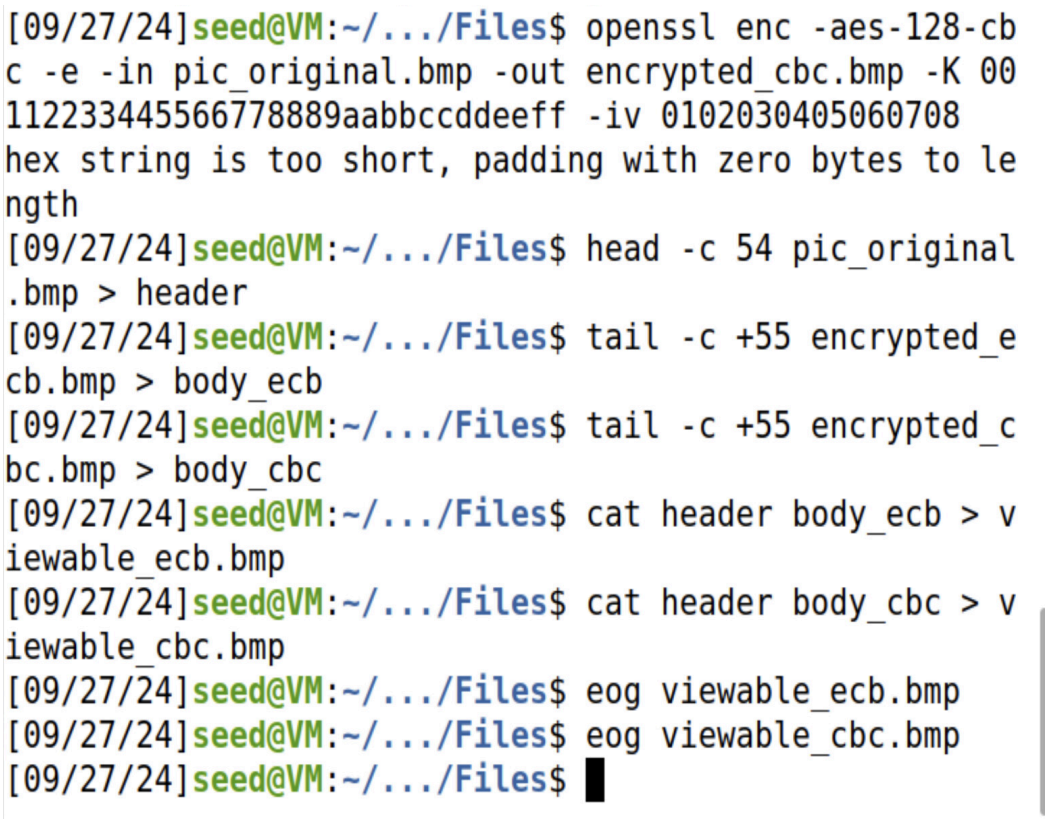


Task 3:

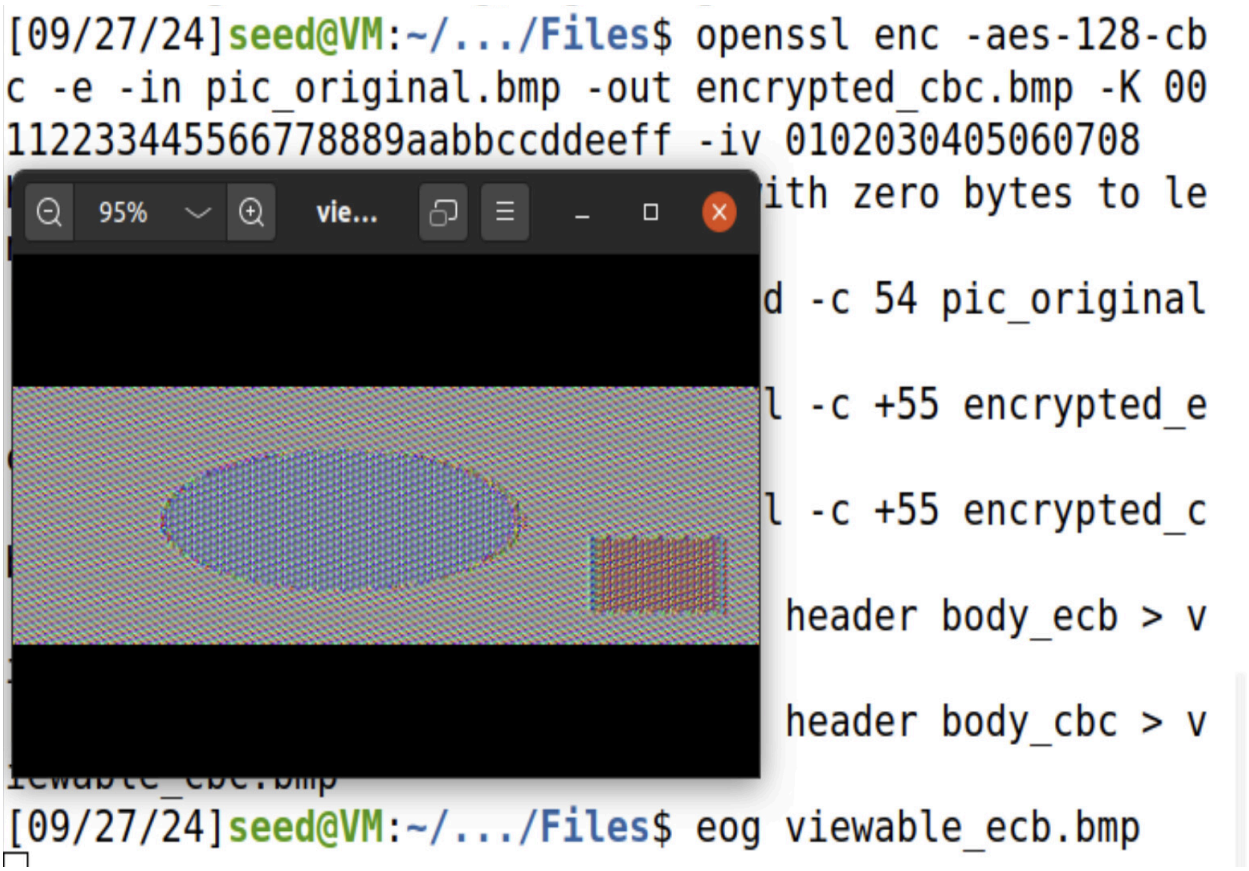
1. Enter Directory



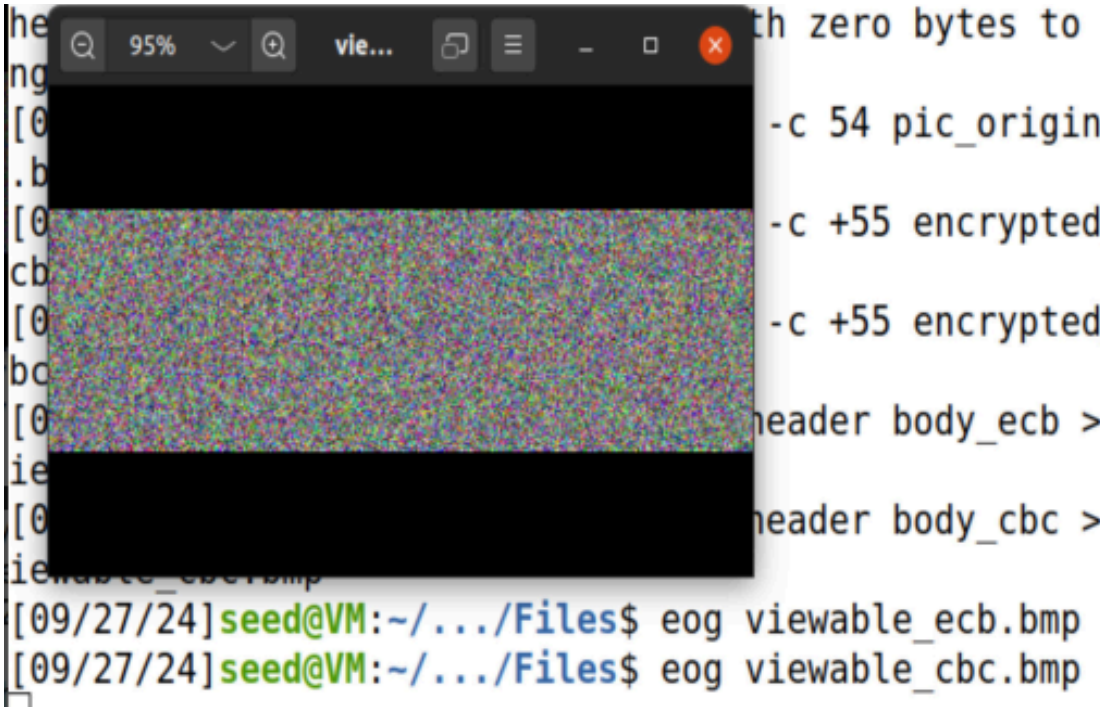
a. CBC (Ciphertext Block Chain) encryption - given bmp picture (Code)



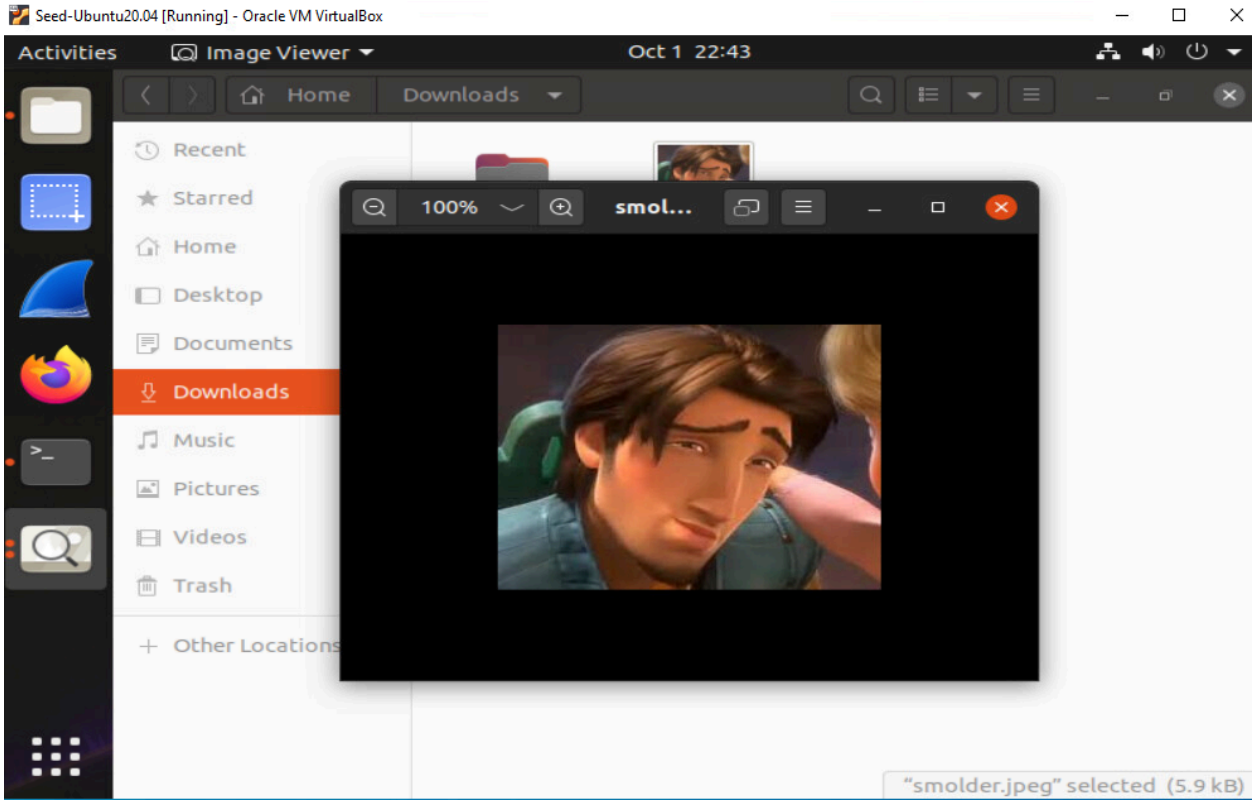
b. ECB (Electronic Code Block) encryption - bmp picture (Output)



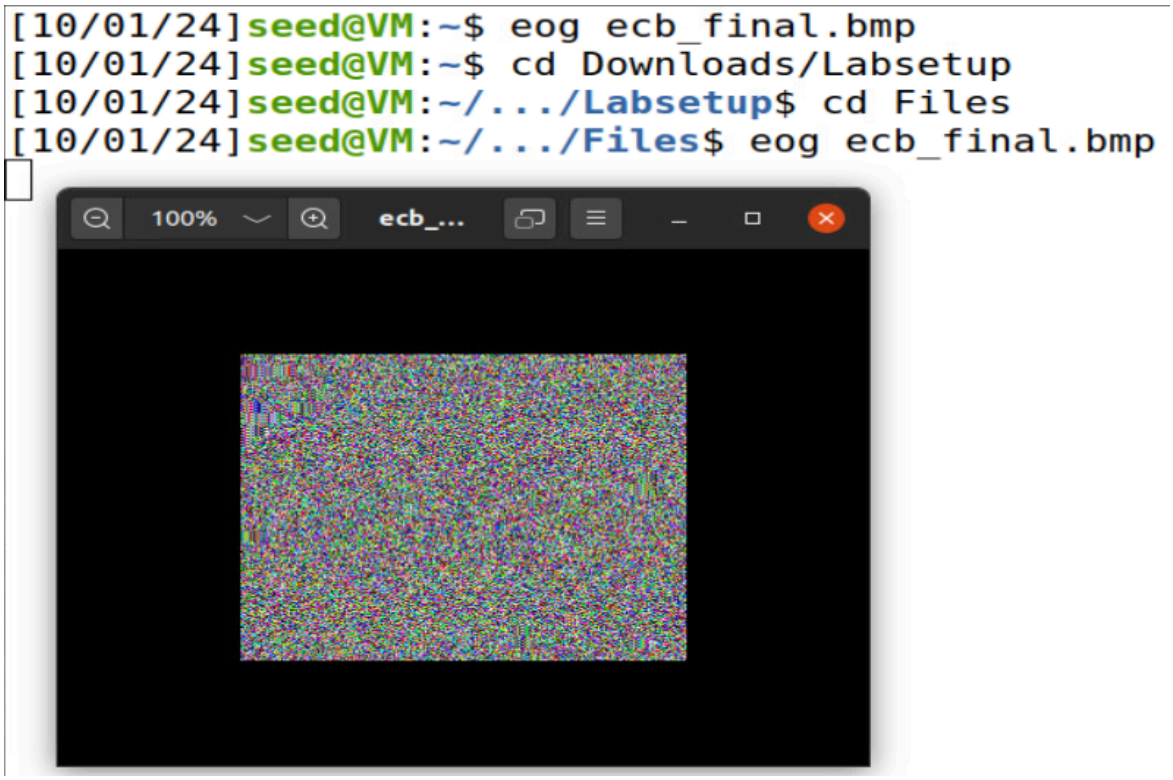
c. CBC encryption - given bmp picture (Output)



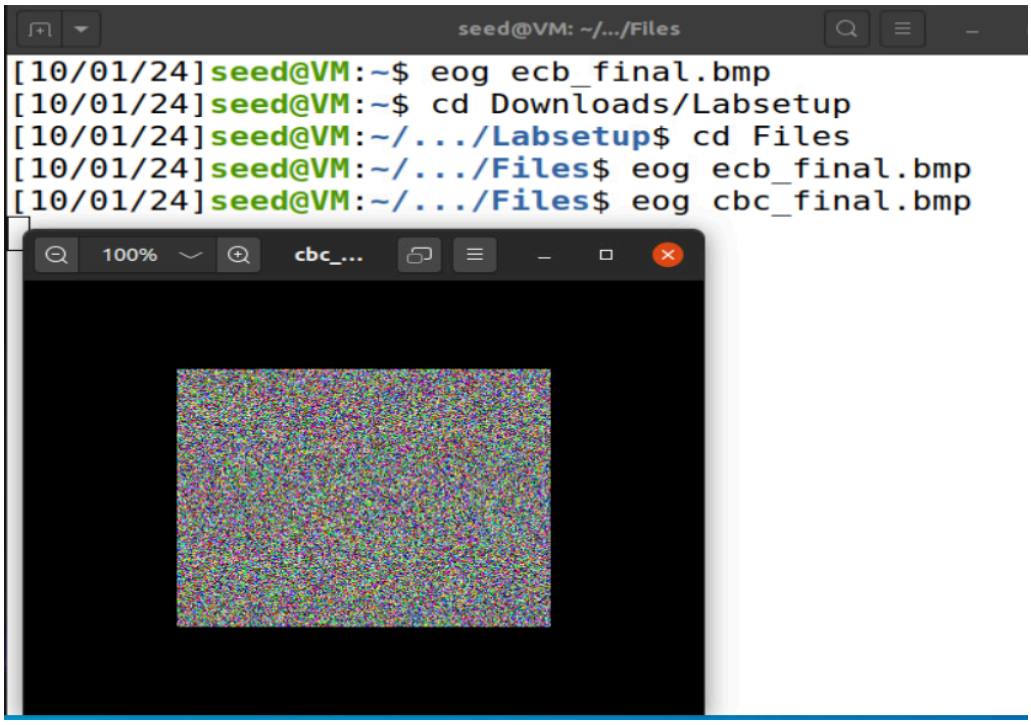
d. Original Chosen bmp picture:



e. ECB encryption - chosen bmp picture (Output)



f. CBC encryption - given bmp picture (Output)



Task 4:

Summary of Padding Requirements:

Mode	Padding Required	Explanation
ECB	Yes	Operates on fixed-size blocks, so padding is needed to fill the last block if the plaintext isn't a multiple of the block size.
CBC	Yes	Operates on fixed-size blocks; padding is necessary to complete the last block.
CFB	No	Operates as a stream cipher; can handle arbitrary-length plaintext without padding.
OFB	No	Operates as a stream cipher; padding is not required due to byte-level processing.

Why Some Modes Don’t Need Padding0:

Modes like **CFB** and **OFB** treat the plaintext as a stream and encrypt it byte-by-byte (or bit-by-bit), rather than in fixed-size blocks. This allows them to process plaintext of any length without needing to fill out the data to match a block size. The block cipher in these modes is only used to generate keystreams, not to directly encrypt blocks of data.

```
[09/29/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in f1.txt -out f1_enc.txt
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[09/29/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in f2.txt -out f2_enc.txt
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[09/29/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in f3.txt -out f3_enc.txt
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[09/29/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -in f1_enc.txt -out f1_dec_nopad.txt -nopad
enter aes-128-cbc decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[09/29/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -in f2_enc.txt -out f2_dec_nopad.txt -nopad
enter aes-128-cbc decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[09/29/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -in f3_enc.txt -out f3_dec_nopad.txt -nopad
enter aes-128-cbc decryption password:
*** WARNING : deprecated key derivation used.
```

Password used: Ll

```
[09/29/24]seed@VM:~/.../Files$ hexdump -C f1_dec_nopad.txt
00000000  c4 68 92 23 ac a8 30 0d  3d 07 10 14 4f 27 41 35  |.h.#..0.=...O'A5|
00000010
[09/29/24]seed@VM:~/.../Files$ hexdump -C f2_dec_nopad.txt
00000000  31 32 33 34 35 36 37 38  39 30 06 06 06 06 06 06  |1234567890......|
00000010
[09/29/24]seed@VM:~/.../Files$ hexdump -C f3_dec_nopad.txt
00000000  31 32 33 34 35 36 37 38  39 30 41 42 43 44 45 46  |1234567890ABCDEF|
00000010  10 10 10 10 10 10 10 10  10 10 10 10 10 10 10 10  |................|
00000020
```

Conclusion:

- The size of the encrypted files is influenced by the padding, with AES adding padding to make the total size a multiple of the block size (16 bytes).

- By decrypting without removing the padding and inspecting the hexadecimal content, we can see the PKCS#7 padding bytes added during encryption.

Task 5

Expected Results:.

- **CBC**: Two blocks (the corrupted block and the next one) will be lost, but the rest of the file will be recoverable.
- **ECB**: All data except the corrupted block will be recoverable
- **CFB**: Only a few bytes starting from the corrupted byte will be corrupted, but the rest of the file will be recoverable.
- **OFB**: Only the corrupted byte will be lost, and the rest of the file will be recoverable.

Justification for Expected Results:

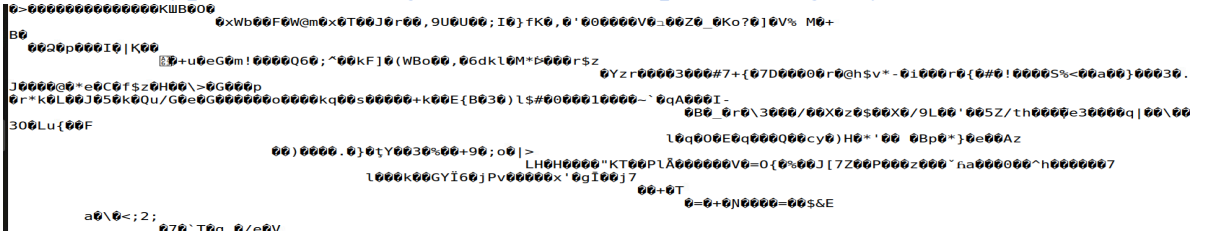
- ECB encrypts blocks independently, so corruption only affects the corrupted block.
- CBC relies on the previous block for decryption, so corruption affects both the corrupted block and the next block.
- CFB is a stream mode, and corruption propagates for a short time, affecting only a few bytes.
- OFB is also a stream mode, but corruption doesn't propagate beyond the corrupted byte because the key stream is independent of the ciphertext.

Doing the Task:

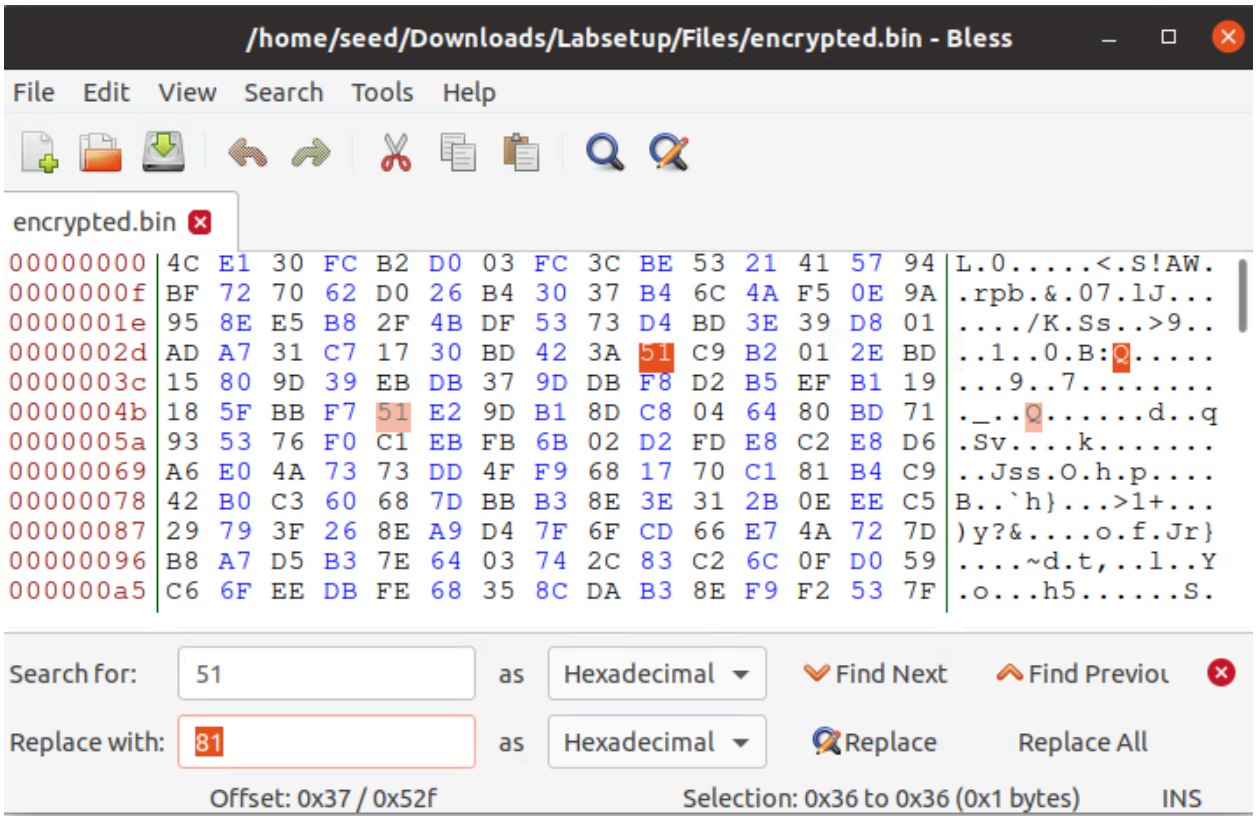
- Create a file that's >1000 bytes:

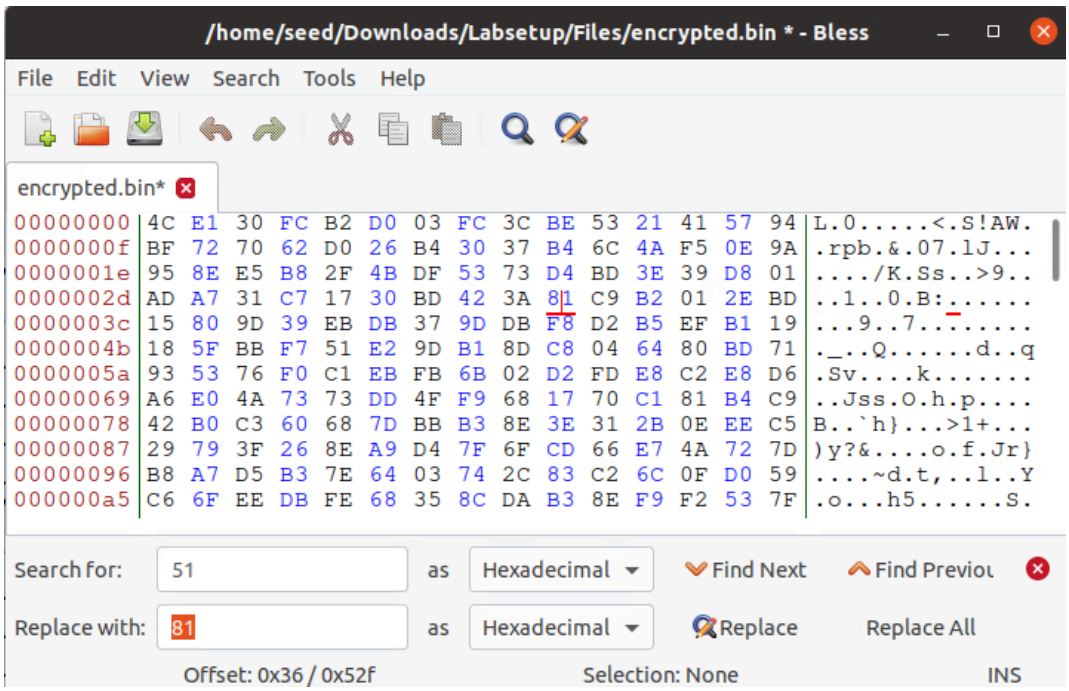
```
[09/30/24]seed@VM:~/.../Files$ ls -l plaintext.txt
-rw-rw-r-- 1 seed seed 1327 Sep 30 13:59 plaintext.txt
```

- Encrypt the file using the AES-128 cipher, using key and iv:



- 55th byte in the encrypted file got corrupted using the bless hex editor:





- Decrypt the corrupted ciphertext file using the correct key and IV.
    a. CBC:

```
[09/30/24]seed@VM:~/.../Files$ openssl enc -d -aes-128-cbc -in encrypted.bin -out decrypted.txt -K $(cat key.bin) -iv $(cat iv.bin)
[09/30/24]seed@VM:~/.../Files$ cat decrypted.txt
This is a sample file for encryption testing.
}{@n@\5@@ Ecu@l sampoe text.
```

    b. ECB:

```
[09/30/24]seed@VM:~/.../Files$ openssl enc -d -aes-128-ecb -in encrypted_ecb.bin -out decrypted_ecb.txt -K $(cat key_ecb.bin)
[09/30/24]seed@VM:~/.../Files$ cat decrypted_ecb.txt
This is a sample file for encryption testing.
T@@@P<@@@{@@l sample text.
```

    c. CFB:

```
[09/30/24]seed@VM:~/.../Files$ openssl enc -d -aes-128-cfb -in encrypted_cfb.bin -out decrypted_cfb.txt -K $(cat key_cfb.bin) -iv $(cat iv_cf
b.bin)
[09/30/24]seed@VM:~/.../Files$ cat decrypted_cfb.txt
This is a sample file for encryption testing.
This isaadditiona@@7@@hR,
6@J@/his is additional sample text.
```

    d. OFB:

```
[09/30/24]seed@VM:~/.../Files$ openssl enc -d -aes-128-ofb -in encrypted_ofb.bin -out decrypted_ofb.txt -K $(cat key_ofb.bin) -iv $(cat iv_of
b.bin)
[09/30/24]seed@VM:~/.../Files$ cat decypted_ofb.txt
cat: decypted_ofb.txt: No such file or directory
[09/30/24]seed@VM:~/.../Files$ cat decrypted_ofb.txt
This is a sample file for encryption testing.
This is@additional sample text.
```

As per the screenshots above, (original statement is “This is additional sample text”), the expected results match the known behavior, with CBC being the most affected and OFB being the least.

Task 6

6.1 - Using the same IV (Initialisation Vectors)

1. Using Different IVs:

```
[09/30/24]seed@VM:~/.../Files$ cat encrypted_iv1.bin
@y@}'{@Ow@n@a,@Cd@q@[09/30/24]seed@VM:~/.../Files$ cat encrypted_iv2.bin
(:@f@@@,@1r@\@:@@Fp}@q@@@@k@[09/30/24]seed@VM:~/.../Files$
```

2. Using the same IV:

```
[09/30/24]seed@VM:~/.../Files$ cat encrypted_same_iv1.bin
@y@}'{@Ow@n@a,@Cd@q@[09/30/24]seed@VM:~/.../Files$ cat encrypted_same_iv2.bin
@y@}'{@Ow@n@a,@Cd@q@[09/30/24]seed@VM:~/.../Files$
```

- Using the same IV will result in the same encryption, while different random IVs will change how it's encrypted. If the same IV is used throughout, it’ll allow hackers to infer patterns and eventually be able to get into these systems, making the uniqueness of IVs essential.

6.2 - Decrypting an unknown plain text with other plaintext-cipher, knowing that IVs are the same:

1. Running python code for decryption (OFB Decryption, works by XORing the plaintext and cipher to get the encryption key, if another text were to be encrypted with the same IV the hacker would already have the key and be able to decrypt it easily, code below was edited to get the plain text from cipher C2, using P1 and C1 as reference, CBC works the same way but it changes each block of code):

```
#!/usr/bin/python3

# XOR two bytearrays
def xor(first, second):
   return bytearray(x^y for x,y in zip(first, second))

MSG   = "This is a known message!"
HEX_1 = "a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159"
HEX_2 = "bf73bcd3509299d566c35b5d450337e1bb175f903fafc159"

# Convert ascii string to bytearray
D1 = bytes(MSG, 'utf-8')

# Convert hex string to bytearray
D2 = bytearray.fromhex(HEX_1)
D3 = bytearray.fromhex(HEX_2)

r1 = xor(D1, D2)
r2 = xor(r1, D3)
print(r1.hex())
print(r2.hex())
```

Output:

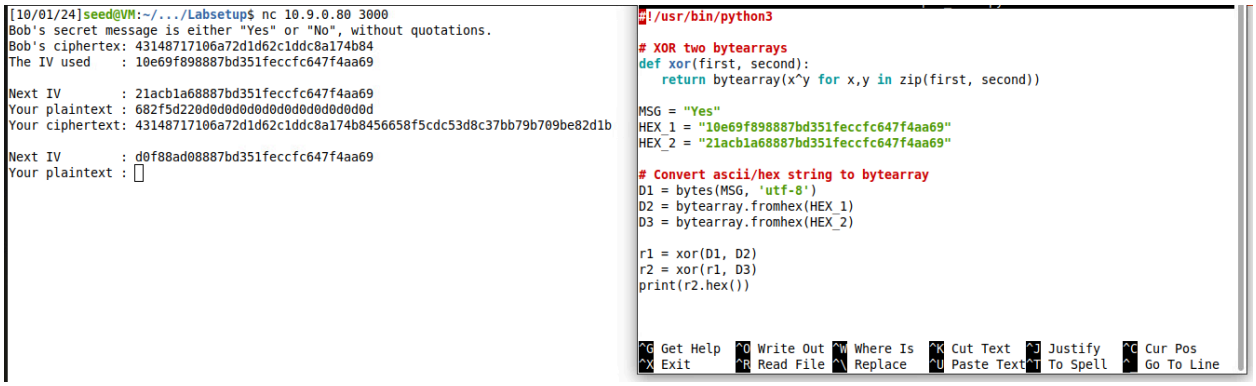
```
[09/30/24]seed@VM:~/.../Files$ python3 sample_code.py
f001d8b622a8b99907b6353e2d2356c1d67e2ce356c3a478
4f726465723a204c61756e63682061206d697373696c6521
[09/30/24]seed@VM:~/.../Files$ echo "4f726465723a204c61756e63682061206d697373696c6521" |xxd -r -p
Order: Launch a missile![09/30/24]seed@VM:~/.../Files$
```

2. If we replace OFB in this experiment with CFB (Cipher Feedback), CFB encrypts on a block basis and makes changes after each, reusing the same IV would allow the attacker to decrypt only **the first block** of **P2**, but not the entire message.

6.3 - Using Predictable IVs

Using trends and the code in 6.2, assume Bob's Message is "Yes." Change the message in the code to yes, and replace the hex with the 2 IV codes. Print out

the XOR results and put them in plain text. If the ciphertext is similar then yes is the message, otherwise, your answer is “No”:



The code above shows that when the “Yes” result is put in plaintext, the ciphertext matches Bob’s, meaning that his message was yes with respect to the previous IV and the next IV that will be used from the plaintext from the code.

Task 7

1. C Code to find the key that is used for the encryption.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <openssl/evp.h>
#include <openssl/err.h>
#include <openssl/conf.h>

void handleErrors(void)
{
   ERR_print_errors_fp(stderr); // Print any errors to stderr
   abort(); // Abort the program on error
}

// Function to encrypt plaintext using AES-128-CBC
int encrypt(unsigned char *plaintext, int plaintext_length, unsigned char *key,
         unsigned char *iv, unsigned char *ciphertext)
{
   EVP_CIPHER_CTX *ctx; // Context for encryption
   int length; // Length of the output ciphertext
   int ciphertext_length; // Total length of the encrypted data
```

```
    // Create and initialize the encryption context
    if (!(ctx = EVP_CIPHER_CTX_new()))
        handleErrors();

    // Initialize the encryption operation
    if (1 != EVP_EncryptInit_ex(ctx, EVP_aes_128_cbc(), NULL, key, iv))
        handleErrors();

    // Provide the plaintext to be encrypted and obtain the encrypted output
    if (1 != EVP_EncryptUpdate(ctx, ciphertext, &length, plaintext, plaintext_length))
        handleErrors();
    ciphertext_length = length;

    // Finalize the encryption; additional ciphertext bytes may be written
    if (1 != EVP_EncryptFinal_ex(ctx, ciphertext + length, &length))
        handleErrors();
    ciphertext_length += length; // Update total ciphertext length

    // Clean up the context to free resources
    EVP_CIPHER_CTX_free(ctx);
    return ciphertext_length; // Return the length of the ciphertext
}

// Function to pad the string to ensure it's 16 bytes long
void padString(unsigned char* buffer) {
    int size = 16; // AES block size in bytes
    int diff = size - strlen(buffer); // Calculate padding needed

    if (diff <= 0) // No padding needed if the buffer is already the correct size
        return;

    // Fill the buffer with '#' characters to achieve the required length
    memset(buffer + strlen(buffer), '#', diff);
}

int main(void)
{
    // Initialization Vector (IV) for AES-128-CBC mode
    unsigned char iv[] = { 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x99,
                           0x88, 0x77, 0x66, 0x55, 0x44, 0x33, 0x22, 0x11 };
```

```
// Correct ciphertext to verify against
unsigned char correct_ciphertext[32] = {
    0x76, 0x4a, 0xa2, 0x6b, 0x55, 0xa4, 0xda, 0x65,
    0x4d, 0xf6, 0xb1, 0x9e, 0x4b, 0xce, 0x00, 0xf4,
    0xed, 0x05, 0xe0, 0x93, 0x46, 0xfb, 0x0e, 0x76,
    0x25, 0x83, 0xcb, 0x7d, 0xa2, 0xac, 0x93, 0xa2
};

// The plaintext message to be encrypted
unsigned char *plaintext = (unsigned char *)"This is a top secret.";

// Buffer for the resulting ciphertext
unsigned char ciphertext[128];
int ciphertext_length; // Length of the generated ciphertext

// Buffer for reading words from the wordlist
const size_t BUF_SIZE = 16;
unsigned char buffer[BUF_SIZE];

// Open the wordlist file for reading
FILE *in = fopen("words.txt", "r");
if (in == NULL) {
    fprintf(stderr, "Could not open words.txt\n");
    return 1; // Exit with error if file cannot be opened
}

// Read words from the wordlist and test each as a potential key
while (fgets(buffer, BUF_SIZE, in)) {
    buffer[strcspn(buffer, "\n")] = 0; // Remove newline character from the string
    padString(buffer); // Pad the key to 16 bytes for AES compatibility

    // Encrypt the plaintext using the current key candidate
    ciphertext_length = encrypt(plaintext, strlen((char *)plaintext), buffer, iv, ciphertext);

    // Compare the computed ciphertext with the expected ciphertext
    if (memcmp(ciphertext, correct_ciphertext, ciphertext_length) == 0) {
        printf("Found Key: %s\n", buffer); // Print the found key
        memset(ciphertext, 0, sizeof(ciphertext)); // Clear the ciphertext buffer for security
    }
```

```
    }

    fclose(in); // Close the wordlist file
    return 0; // Successful execution
}
```

Output:

```
[09/30/24]seed@VM:~/.../Files$ gcc -o decrypt decrypt.c -lcrypto
[09/30/24]seed@VM:~/.../Files$ ./decrypt
Found Key: Syracuse########
```