Name: Henry Salgado

ID: 80509684

CS 4351/5352: Computer Security

Assignment 4

Problem 1: Caesar Cipher

```
GNU nano 4.8                          caesar_decrypt.py                          Modified
def caesar_decrypt(text, shift):

    alphabet = 'abcdefghijklmnopqrstuvwxyz'
    shifted_alphabet = alphabet[-shift:] + alphabet[:-shift]  # Rotate end to start
    table = str.maketrans(alphabet, shifted_alphabet)
    return text.translate(table)

# Message
text = "Tvsxigx csyv hexe amxl gevi erh geyxmsr. Epaecw ywi wxvsrk erh yrmuyi tewwasvhw jsv iegl eggsyrx, vikypevpc ythexi, erh oiit csyv hizmgiw erh wsjxaevi yt >

# Most common letters in English to compare against
common_letters = 'etaoin'

# Trying different shifts to find the most plausible one
best_match = 0
for shift in range(26):
    decoded = caesar_decrypt(text, shift)
    match_count = sum(letter in decoded for letter in common_letters)
    if match_count > best_match:
        best_match = match_count
        best_shift = shift
        best_result = decoded

# Printing
print(f"Best shift: {best_shift}")
print(f"Decrypted message: {best_result}")
```

Running the file:

```
seed@seedvm:/home/henrysalgado94/src-cloud/Files/System/Desktop/PracticeFiles/hw4$ nano caesar_decrypt.py
seed@seedvm:/home/henrysalgado94/src-cloud/Files/System/Desktop/PracticeFiles/hw4$ python3 caesar_decrypt.py
Best shift: 4
Decrypted message: Trotect your data with care and caution. Elways use strong and unique passwords for each account, regularly update, and keep your devices and so
ftware up to date
seed@seedvm:/home/henrysalgado94/src-cloud/Files/System/Desktop/PracticeFiles/hw4$
```

Observations: For the most part, the message is decrypted by my code, finding the best shift to be 4 and the message to be: *"Protect your data with care and caution. Always use strong and unique passwords for each account, regularly update, and keep your devices and software up to date"*

For part 2, I keep the code logic, but change the indexing

```
GNU nano 4.8                          caesar_encrypt.py
def caesar_encrypt(text, shift):

    alphabet = 'abcdefghijklmnopqrstuvwxyz'
    # I changed the indexing so that the shift is foward
    shifted_alphabet = alphabet[shift:] + alphabet[:shift]
    table = str.maketrans(alphabet, shifted_alphabet)
    return text.lower().translate(table)

#message
plaintext = "Programming is like cooking. Even if you follow the recipe perfectly, there's always a chance you'll end up with a big mess."

encrypted_message = caesar_encrypt(plaintext, 7)


print(f"Message: {encrypted_message}")
```

```
henrysalgado94@seedvm:~/src-cloud/Files/System/Desktop/PracticeFiles/hw4$ python3 caesar_e
ncrypt.py
Message: wyvnyhttpun pz sprl jvvrpun. lclu pm fvb mvssvd aol yljpwl wlymljasf, aolyl'z hsd
hfz h johujl fvb'ss luk bw dpao h ipn tlzz.
henrysalgado94@seedvm:~/src-cloud/Files/System/Desktop/PracticeFiles/hw4$
```

Task 2: Frequent Analysis

```
  GNU nano 4.8                                              frequency_analysis.py
import collections

ciphertext = (
    "lrvmnir bpr sumvbwvr jx bpr lmiwv yjeryrkbi jx qmbm wi bpr xjvni mkd ymibrut jx"
    " irhx wi bpr riirkvr jx ymbinlmtmipw utn qmumbr dj w ipmhh but bj rhnvwdmbr bpr"
    " yjeryrkbi jx bpr qmbm mvvjudwko bj yt wkbrusurbmbwjk lmird jk xjubt trmui jx"
    " ibndt wb wi kjb mk rmit bmiq bj rashmwk rmvp yjeryrkb mkd wbi iwokwxwvmkvr mkd"
    " ijyr ynib urymwk nkrashmwkrd bj ower m vjyshrbr rashmkmbwjk jkr cjnhd pmer bj"
    " lr fnmhwxwrd mkd wkiswurd bj invp mk rabrkb bpmb pr vjnhd urmvp bpr ibmbr jx"
    " rkhwopbrkrd ywkd vmsmlhr jx urvjokwgwko ijnkdhrii ijnkd mkd ipmsrhrii ipmsr w"
    " dj kjb drry ytirhx bpr xwkmh mnbpjuwbt lnb yt rasruwrkvr cwbp qmbm pmi hrxb kj"
    " djnlb bpmb bpr xjhhjcwko wi bpr sujsru msshwvmbwjk mkd wkbrusurbmbwjk w jxxru"
    " yt bprjuwri wk bpr pjsr bpmb bpr riirkvr jx jqwkmcmk qmumbr cwhh urymwk wkbmvb"
)

#Cleaning
ciphertext = ciphertext.replace(" ", "").lower()

letter_counts = collections.Counter(ciphertext)
total_letters = len(ciphertext)

for letter, count in letter_counts.items():
    relative_frequency = count / total_letters * 100
    print(f"{letter}: {relative_frequency:.2f}%")
```

After compiling and running

```
seed@seedvm:/home/henrysalgado94/src-cloud/Files/System/Desktop/PracticeFiles/hw4$ nano frequency_analysis.py
seed@seedvm:/home/henrysalgado94/src-cloud/Files/System/Desktop/PracticeFiles/hw4$ python3 frequency_analysis.py
l: 1.24%
r: 13.00%
v: 3.41%
m: 9.60%
n: 2.63%
i: 6.35%
b: 10.53%
p: 4.64%
s: 2.63%
u: 3.72%
w: 7.28%
j: 7.43%
x: 3.10%
y: 2.94%
e: 0.77%
k: 7.59%
q: 1.08%
d: 3.56%
t: 2.01%
h: 3.56%
o: 1.08%
a: 0.77%
c: 0.77%
f: 0.15%
g: 0.15%
seed@seedvm:/home/henrysalgado94/src-cloud/Files/System/Desktop/PracticeFiles/hw4$
```

According to Google,

- E: Appears most frequently, around 12-13% of the time in typical text.
- T: Second most common, around 9% frequency.
- A, O, I, N: Follow closely behind, all with roughly 7-8% frequencies.
    - Therefore, I will first try r = e

Observation 1: “bpr” appears in the text several times together, and since r = e, I will assume that “bpr” is “the” and if so then b = t and p = h and r = e

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

	t														h		e								

Observation 2: Another common letter is “a” and I have a suspicion that it might match with “m” (9.60%). I see that the text has alot of “mkd” and “mk”. In that case, “mkd” could = “and” and “mk” = “an”

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
m	t		d							n					h		e								

Observation 3: Another common letters in English are “i,o, n” and in my frequencies, I can see that “j” and “w” are around 8%. When searching my text, I see that there are alot of “jx” and “wi”. I am thinking these could be transition words such as “is” or “to” or “of”. I will try “wi” as “is” and “jx” as “of”

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
m	t		d					s	o	n					h		e					i	f		

Observation 4: There is a sequence “RMIT” = EAS_. I belive that letter t = y. In that case, the word “RMIT” = “EASY”. There are also two other letter words “YT” and if t = y, then Y = m. Therefore, “YT” = “MY”.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
m	t		d					s	o	n					h		e		y			i	f	m	

Observation 5: The last word of the paragraph is “wkbmvb” which would translate to “inta_t”. From the list of english, it could be the word “intact”. If that is the case then v = c.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y
m	t		d					s	o	n					h		e		y		c	i	f	m

Observation 6: Second to last word is “urymwk” which would translate to “_emain”. If the last word was “intact” then a possible previous word could be “remain”. In that case u = r

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y
m	t		d					s	o	n					h		e		y	r	c	i	f	m

Observation 7: The third to last word is “cwhh”. This could translate to _i_ _. It is an insteristing word since it has two repeated HHs. In english, there are not that many repeated letters, and they are often “LL , SS , TT and etc. Considering that the we found “remain intact”, I will this word could be “will”. In that case, c = w and h = L

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y

m	t	w	d				l	s	o	n					h	k	e		y	r	c	i	f	m

Observation 9: The fourth to last word is “qmumbr”, this could be _arate. I had to think about this one a lot. Since the word should have consonants, one of the words could fit would “KARATE”. In that case, q = K

Observation 10: I will go back to the top of the paragraph and began translating from there.

- The first word is “lrvmnir” = _eca_se -> Because. L = b and n = u

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y
m	t	w	d				l	s	o	n	b		u		h	k	e		y	r	c	i	f	m

Observation 11: Second word is bpr = the, third word is “sumvbwvr” = _ractice -> Practice. Therefore, s = p.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y
m	t	w	d				l	s	o	n	b		u		h	k	e	p	y	r	c	i	f	m

Observation 12: The fifth word in the paragraph is “lmiwv” = basic and the sixth word is “yjeryrkbi” = mo_ement -> movement. Therefore, e = v

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y
m	t	w	d	v			l	s	o	n	b		u		h	k	e	p	y	r	c	i	f	m

With these we can determine the sentence as *“Because the practice of the basic movements of kata is the focus and mastery of self is the essence of matsubayashi ryu karate do I shall try to elucidate the movements of the kata according to my interpretation based on forty years of study it is not an easy task to explain each movement and its significance and some must remain unexplained to give a complete explanation one would have to be qualified and inspired to such an extent that he could reach the state of enlightened mind capable of recognizing soundless sound and shapeless shape i do not deem myself the final authority but my experience with kata has left no doubt that the following is the proper application and interpretation i offer my theories in the hope that the essence of okinawan karate will remain intact.*

Question 3: DES (DAT Encryption Standard)

- **Block Cipher:** DES operates on fixed-size data blocks (64 bits in this case). It encrypts each block independently using a secret key.
- **Key Length: The key length in DES is 56 bits, but it starts with a 64-bit key.** Certain bits are discarded for parity.
- **Number of Rounds:** The encryption process in DES consists of 16 rounds of mathematical operations on the data block and the key.

Encryption Process:

1. The permuted block is split into a left half (L0) and a right half (R0).
2. For 16 rounds, each block goes through a function that includes expansion, substitution, and permutation operations. The round function is applied to the right half of the block and the result is XORed with the left half.
3. The output of the XOR is then used as the new right half, and the previous right half becomes the new left half for the next round.
4. After the 16 rounds are completed, the halves are swapped and the combined block goes through a final permutation, which is the inverse of the initial permutation.

For Decryption of DES:

- The ciphertext is processed with the same initial and final permutations as encryption.
- The 16 rounds are applied in reverse order, using the subkeys in the reverse order as well.

Question 4: AES (Advanced Encryption Standard)

Block Cipher Size: 128 bits

Key Lengths: AES supports multiple key lengths: 128 bits, 192 bits, and 256 bits.

Number of Rounds: The number of rounds in AES depends on the key length. There are 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys.

Encryption Process:

1. Initial Permutation: 128-bit data block is rearranged using a specific substitution table.
2. SubBytes: Each byte (8 bits) in the data block is substituted using a non-linear substitution table.
3. ShiftRows: The rows of the data block are shifted by a certain number of positions to disrupt data organization.
4. MixColumns: The columns of the data block are mixed using a specific mathematical operation to further enhance diffusion.

5. AddRoundKey: The data block is XORed with a round key derived from the main key. This is where the key gets incorporated into the encryption process.
6. Rounds 2 to (N-1): Steps 2-5 are repeated for a specified number of rounds (depending on the key length). In each round, a different round key is used.
7. Final Round: The final round consists of steps 2, 3, and 4, but without the MixColumns step.
8. Final Permutation: The data block is subjected to a final permutation using another substitution table.

Note: My Google VM stopped working so I had to go back and download the VM on my machine.

Question 5:

First, I will create a plain text file that will serve as input for my encryption

```
seed@VM: ~/.../hw4
[03/10/24]seed@VM:~/.../hw4$ nano plain.txt
[03/10/24]seed@VM:~/.../hw4$
```

```
seed@VM: ~/.../hw4
  GNU nano 4.8                    plain.txt                    Modified
Hello I am Henry Salgado
```

I will try three different ciphers 1) AES-256-CBC 2) DES-EDE3-CBC 3) RC4

```
[03/10/24]seed@VM:~/.../hw4$ openssl enc -aes-256-cbc -e -in plain.txt -out cipher-aes-256-cbc.bin -K 00112233445566778899aabbccddee
f00112233445566778899aabbccddeeff -iv 01020304050607080102030405060708
[03/10/24]seed@VM:~/.../hw4$ openssl enc -des-ede3-cbc -e -in plain.txt -out cipher-des-ede3-cbc.bin \
> -K 00112233445566778899aabbccddeeff0011223344556677 \
> -iv 0102030405060708
[03/10/24]seed@VM:~/.../hw4$ openssl enc -rc4 -e -in plain.txt -out cipher-rc4.bin \
> -K 00112233445566778899aabbccddeeff
[03/10/24]seed@VM:~/.../hw4$ ls
cipher-aes-256-cbc.bin  cipher-des-ede3-cbc.bin  cipher-rc4.bin  plain.txt
[03/10/24]seed@VM:~/.../hw4$
```

Outputs

```
[03/10/24]seed@VM:~/.../hw4$ hexdump -C cipher-aes-256-cbc.bin | less
```

```
00000000  ab 5e b1 60 5c bf 7e de  2d a0 8b 12 99 e0 bd 93  |.^.`\.~.-.......|
00000010  04 05 be f0 1d 28 44 45  d2 3e 99 7a af 2b fb b2  |.....(DE.>.z.+..|
00000020
(END)
```

hexdump -C cipher-des-ede3-cbc.bin | less

```
00000000  76 f5 3e 33 30 72 b0 ce  e8 93 6a 3c 6f 0c 8e 09  |v.>30r....j<o...|
00000010  77 cc 33 3d 83 b4 a2 38  84 a4 40 41 5d d1 c8 0a  |w.3=...8..@A]...|
00000020
(END)
```

hexdump -C cipher-rc4.bin | less

```
00000000  cd 3e c0 3b fd 07 28 a5  99 34 99 73 f0 2b 0f ed  |.>.;..(..4.s.+..|
00000010  29 96 f2 3c c6 76 ef 41  6a                       |)..<.v.Aj|
00000019
(END)
```

Observations:

- No clear patterns in hex output
- Both AES-256-CBC and DES-EDE3-CBC encrypted files are of the same size (32 bytes). This uniformity in size might be because these are both block ciphers.
- The encrypted file using RC4 is slightly smaller (25 bytes) than those encrypted with the block ciphers. RC4 is a stream cipher, meaning it encrypts data one bit or byte at a time and does not require padding to align with a block size.

```
[03/20/24]seed@VM:~/.../hw4$ ls -l
total 16
-rw-rw-r-- 1 seed seed 32 Mar 10 11:53 cipher-aes-256-cbc.bin
-rw-rw-r-- 1 seed seed 32 Mar 10 11:53 cipher-des-ede3-cbc.bin
-rw-rw-r-- 1 seed seed 25 Mar 10 11:53 cipher-rc4.bin
-rw-rw-r-- 1 seed seed 25 Mar 10 11:41 plain.txt
```

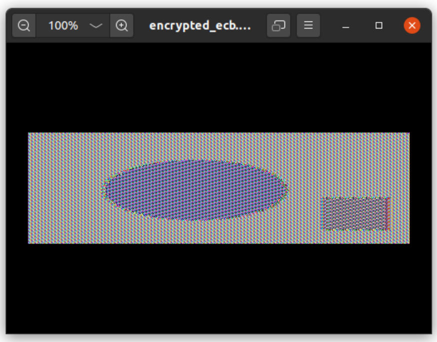
Question 6:

Instead of using the provided commands for combining header and body, I will put commands in a python code called image_encrypter.py. This will generate both cbc and ecb images for me.

```
[03/20/24]seed@VM:~/.../hw4$ python3 image_encrypter.py
[03/20/24]seed@VM:~/.../hw4$ ls
body                    cipher-des-ede3-cbc.bin  encrypted_cbc.bmp  header              pic_original.bmp
cipher-aes-256-cbc.bin  cipher-rc4.bin           encrypted_ecb.bmp  image_encrypter.py  plain.txt
```

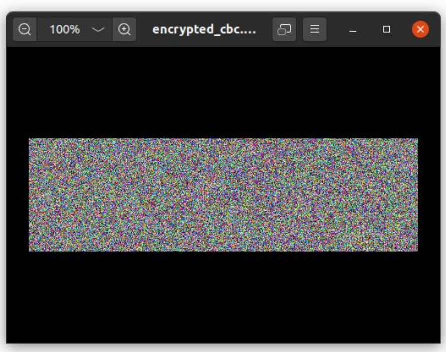
Then, I will view the images using eog

```
[03/20/24]seed@VM:~/.../hw4$ eog encrypted_ecb.bmp
```



Now, cbc

```
[03/20/24]seed@VM:~/.../hw4$ eog encrypted_cbc.bmp
```



Observations:

ECB Mode: The encrypted image displays some patterns from the original image. This is because identical blocks of plaintext result in identical blocks of ciphertext. This can leak information about the image structure.

CBC Mode: The encrypted image appears as random noise.

Question 7:

First, I put some text from a book in a “sample.txt” file. I checked to see how many bytes, and then truncated to 1000.

```
[03/20/24]seed@VM:~/.../hw4$ nano sample.txt
[03/20/24]seed@VM:~/.../hw4$ ls -l sample.txt
-rw-rw-r-- 1 seed seed 4291 Mar 20 15:29 sample.txt
[03/20/24]seed@VM:~/.../hw4$ truncate -s 1000 sample.txt
[03/20/24]seed@VM:~/.../hw4$ ls -l sample.txt
-rw-rw-r-- 1 seed seed 1000 Mar 20 15:29 sample.txt
[03/20/24]seed@VM:~/.../hw4$ 
```

I will create a python script that encrypts my sample.txt file using ecb, cbc, cfb, ofb

```
[03/20/24]seed@VM:~/.../q7$ ls
q7encrypter.py  sample.txt
[03/20/24]seed@VM:~/.../q7$ python3 q7encrypter.py
[03/20/24]seed@VM:~/.../q7$ ls
ciphertext_cbc.bin  ciphertext_cfb.bin  ciphertext_ecb.bin  ciphertext_ofb.bin  q7encrypter.py  sample.txt
[03/20/24]seed@VM:~/.../q7$ 
```

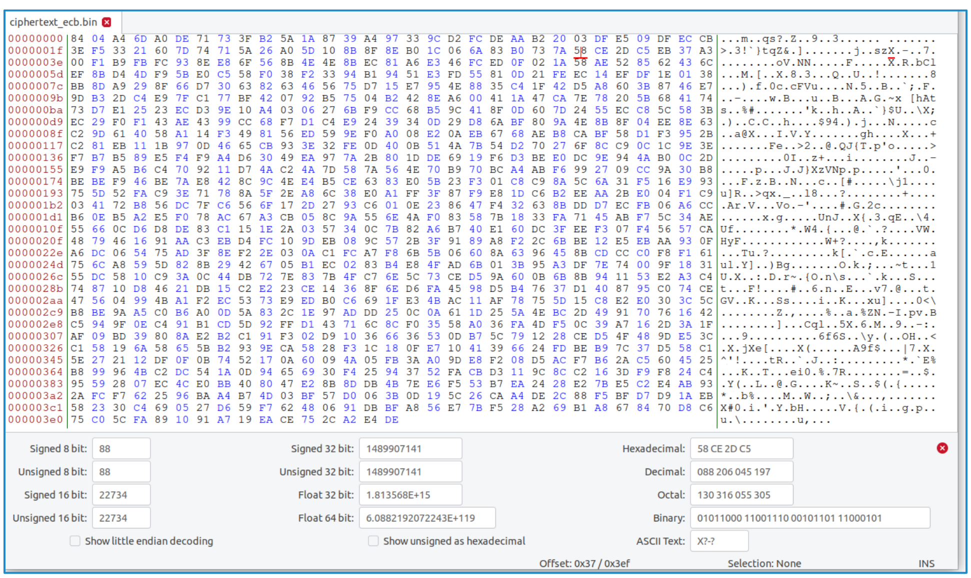
I will open each of the files and corrupt the 55th byte (offset = 0x37)

bless ciphertext_ecb.bin

bless ciphertext_cbc.bin

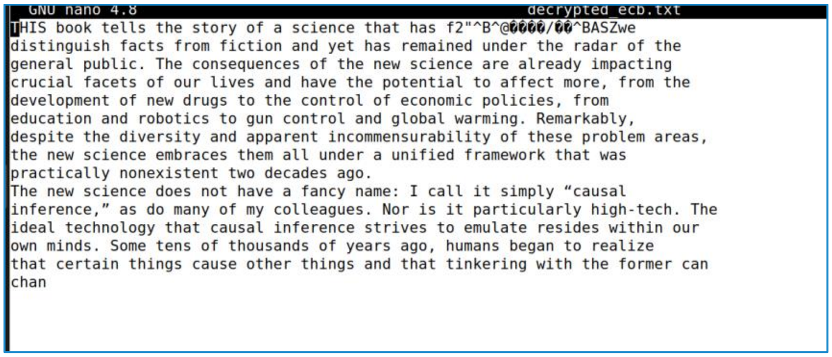
bless ciphertext_cfb.bin

bless ciphertext_ofb.bin



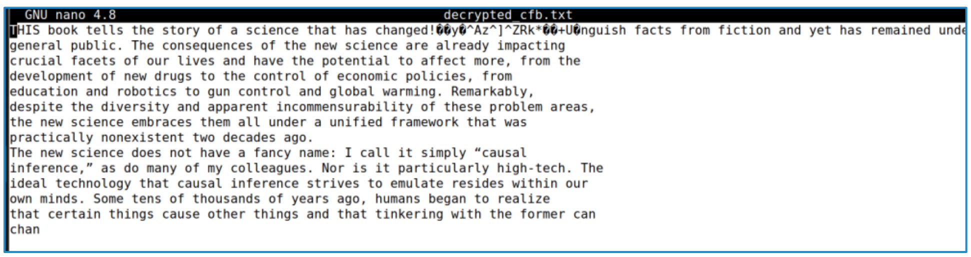
For example, in ecb, I corrupted by changing value from 58 to 57

Now, after corrupting all files, I will run de the decrypter and observe the text files. We will start with ECB.



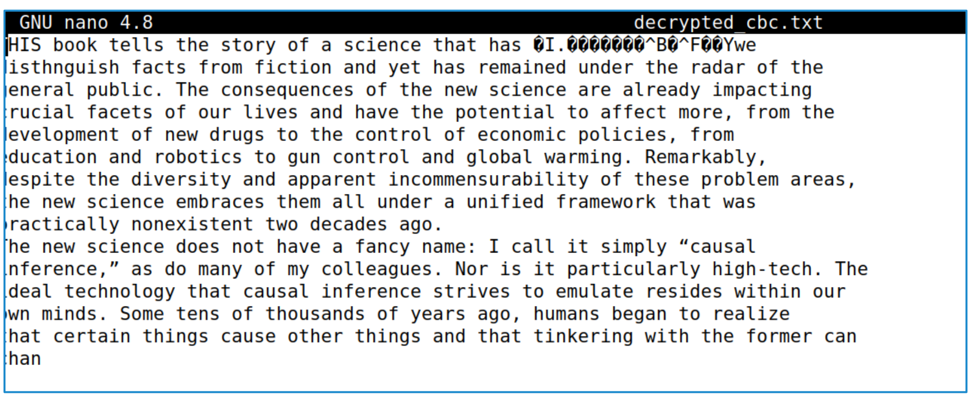
```
  GNU nano 4.8                              decrypted_ecb.txt
THIS book tells the story of a science that has f2"^B^@����/��^BASZwe
distinguish facts from fiction and yet has remained under the radar of the
general public. The consequences of the new science are already impacting
crucial facets of our lives and have the potential to affect more, from the
development of new drugs to the control of economic policies, from
education and robotics to gun control and global warming. Remarkably,
despite the diversity and apparent incommensurability of these problem areas,
the new science embraces them all under a unified framework that was
practically nonexistent two decades ago.
The new science does not have a fancy name: I call it simply “causal
inference,” as do many of my colleagues. Nor is it particularly high-tech. The
ideal technology that causal inference strives to emulate resides within our
own minds. Some tens of thousands of years ago, humans began to realize
that certain things cause other things and that tinkering with the former can
chan
```

CFB



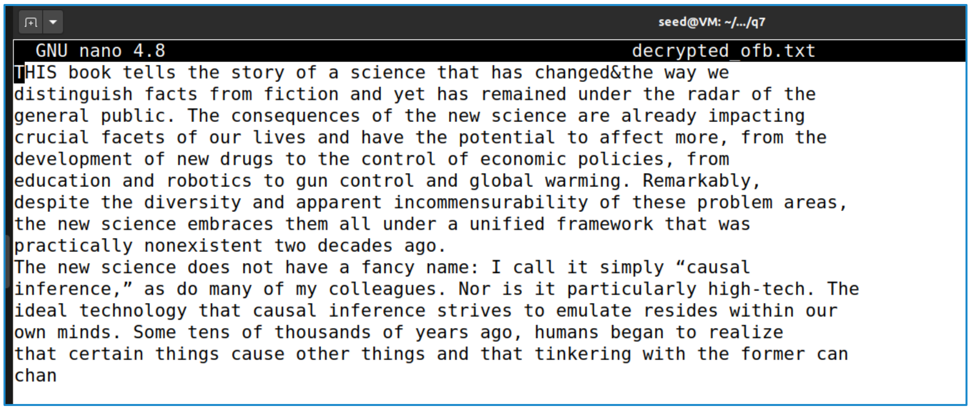
```
  GNU nano 4.8                              decrypted_cfb.txt
THIS book tells the story of a science that has changed!��y�^Az^]^ZRk*��+U�nguish facts from fiction and yet has remained unde
general public. The consequences of the new science are already impacting
crucial facets of our lives and have the potential to affect more, from the
development of new drugs to the control of economic policies, from
education and robotics to gun control and global warming. Remarkably,
despite the diversity and apparent incommensurability of these problem areas,
the new science embraces them all under a unified framework that was
practically nonexistent two decades ago.
The new science does not have a fancy name: I call it simply “causal
inference,” as do many of my colleagues. Nor is it particularly high-tech. The
ideal technology that causal inference strives to emulate resides within our
own minds. Some tens of thousands of years ago, humans began to realize
that certain things cause other things and that tinkering with the former can
chan
```

CBC



```
 GNU nano 4.8                               decrypted_cbc.txt
HIS book tells the story of a science that has �I.�������^B�^F��Ywe
isthnguish facts from fiction and yet has remained under the radar of the
eneral public. The consequences of the new science are already impacting
rucial facets of our lives and have the potential to affect more, from the
evelopment of new drugs to the control of economic policies, from
ducation and robotics to gun control and global warming. Remarkably,
espite the diversity and apparent incommensurability of these problem areas,
he new science embraces them all under a unified framework that was
ractically nonexistent two decades ago.
he new science does not have a fancy name: I call it simply “causal
nference,” as do many of my colleagues. Nor is it particularly high-tech. The
deal technology that causal inference strives to emulate resides within our
wn minds. Some tens of thousands of years ago, humans began to realize
hat certain things cause other things and that tinkering with the former can
han
```

OFB



```
seed@VM: ~/.../q7
  GNU nano 4.8                                        decrypted_ofb.txt
THIS book tells the story of a science that has changed&the way we
distinguish facts from fiction and yet has remained under the radar of the
general public. The consequences of the new science are already impacting
crucial facets of our lives and have the potential to affect more, from the
development of new drugs to the control of economic policies, from
education and robotics to gun control and global warming. Remarkably,
despite the diversity and apparent incommensurability of these problem areas,
the new science embraces them all under a unified framework that was
practically nonexistent two decades ago.
The new science does not have a fancy name: I call it simply “causal
inference,” as do many of my colleagues. Nor is it particularly high-tech. The
ideal technology that causal inference strives to emulate resides within our
own minds. Some tens of thousands of years ago, humans began to realize
that certain things cause other things and that tinkering with the former can
chan
```

Observations:

ECB (Electronic Codebook) mode:

- In my analysis of ECB mode, I found that each block is encrypted independently, which means that the corruption will only affect the block containing the 55th byte. Upon decrypting the file, I observed that the first 6 blocks (48 bytes) remained intact, while the 7th block (16 bytes) was corrupted. The remaining blocks were decrypted correctly.

CBC (Cipher Block Chaining) mode:

- For CBC mode, I discovered that each block's encryption depends on the previous block's ciphertext. When I introduced corruption in the 55th byte, it caused the 7th block to be decrypted incorrectly, and this error propagated to the 8th block as well. As a result, the decrypted file had the first 6 blocks (48 bytes) intact, but the 7th and 8th blocks (32 bytes) were corrupted. The remaining blocks were decrypted correctly.

Note: Very similar results for CFB

OFB (Output Feedback) modes:

- For OFB modes, I learned that the encryption of each block depends on the previous block's keystream. When I introduced corruption in the 55th byte, it caused a single byte error in the decrypted file at the same position. Consequently, the decrypted file had a single byte corrupted at the 55th position, while the rest of the file was decrypted correctly.