SYMMETRIC ENCRYPTION LAB

UFCFVN-30-M - Computer & Network Security

A Research Coursework

Prepared by

Kiran Kumar Mohanraj - 20054954

MSc Cyber Security

University of the West of England, Bristol

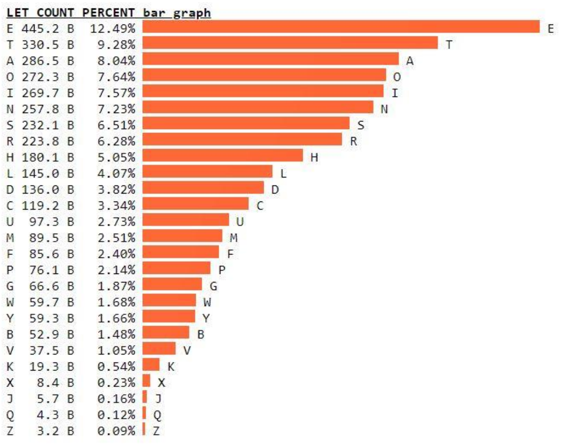
TABLE OF CONTENTS

1 LAB TASK

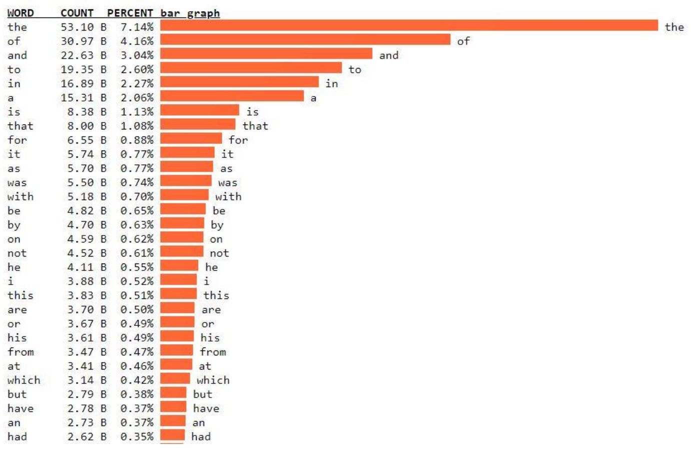
1.1 Task 1: Cryptanalysis of Substitution Cipher

Task 1: To recover the plaintext from the ciphertext (Tas1CT.txt) using Frequency Analysis.

Before approaching the problem, we need to know about the frequency distribution of letters and words in English text. Figure 1 provides the frequency order of letters while Figure 2 provides frequency distribution of the top 30 words from Google books data considering 97,565 distinct words(Norvig, no date).



LET	COUNT	PERCENT
E	445.2 B	12.49%
T	330.5 B	9.28%
A	286.5 B	8.04%
O	272.3 B	7.64%
I	269.7 B	7.57%
N	257.8 B	7.23%
S	232.1 B	6.51%
R	223.8 B	6.28%
H	180.1 B	5.05%
L	145.0 B	4.07%
D	136.0 B	3.82%
C	119.2 B	3.34%
U	97.3 B	2.73%
M	89.5 B	2.51%
F	85.6 B	2.40%
P	76.1 B	2.14%
G	66.6 B	1.87%
W	59.7 B	1.68%
Y	59.3 B	1.66%
B	52.9 B	1.48%
V	37.5 B	1.05%
K	19.3 B	0.54%
X	8.4 B	0.23%
J	5.7 B	0.16%
Q	4.3 B	0.12%
Z	3.2 B	0.09%

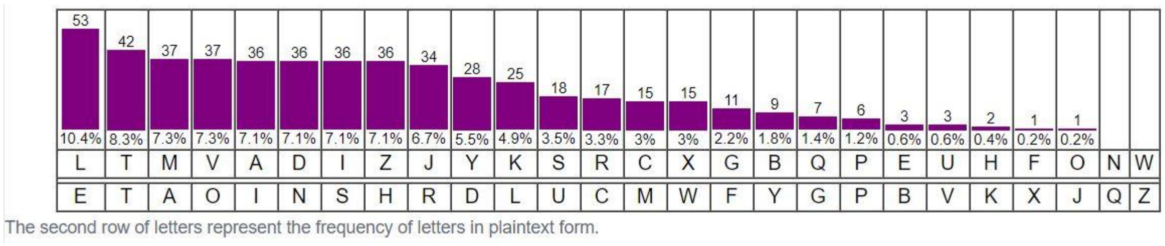


WORD	COUNT	PERCENT
the	53.10 B	7.14%
of	30.97 B	4.16%
and	22.63 B	3.04%
to	19.35 B	2.60%
in	16.89 B	2.27%
a	15.31 B	2.06%
is	8.38 B	1.13%
that	8.00 B	1.08%
for	6.55 B	0.88%
it	5.74 B	0.77%
as	5.70 B	0.77%
was	5.50 B	0.74%
with	5.18 B	0.70%
be	4.82 B	0.65%
by	4.70 B	0.63%
on	4.59 B	0.62%
not	4.52 B	0.61%
he	4.11 B	0.55%
i	3.88 B	0.52%
this	3.83 B	0.51%
are	3.70 B	0.50%
or	3.67 B	0.49%
his	3.61 B	0.49%
from	3.47 B	0.47%
at	3.41 B	0.46%
which	3.14 B	0.42%
but	2.79 B	0.38%
have	2.78 B	0.37%
an	2.73 B	0.37%
had	2.62 B	0.35%

Figure 1: Frequency distribution of letters (Norvig, no date)

Figure 2: Frequency distribution of words (Norvig, no date)

To decrypt the cipher, we use frequency analysis on the provided ciphertext since the brute-force approach is not very much effective on substitution cipher. Figure 3 shows the frequency distribution of letters in the ciphertext and plaintext(*Frequency Analysis*, no date).

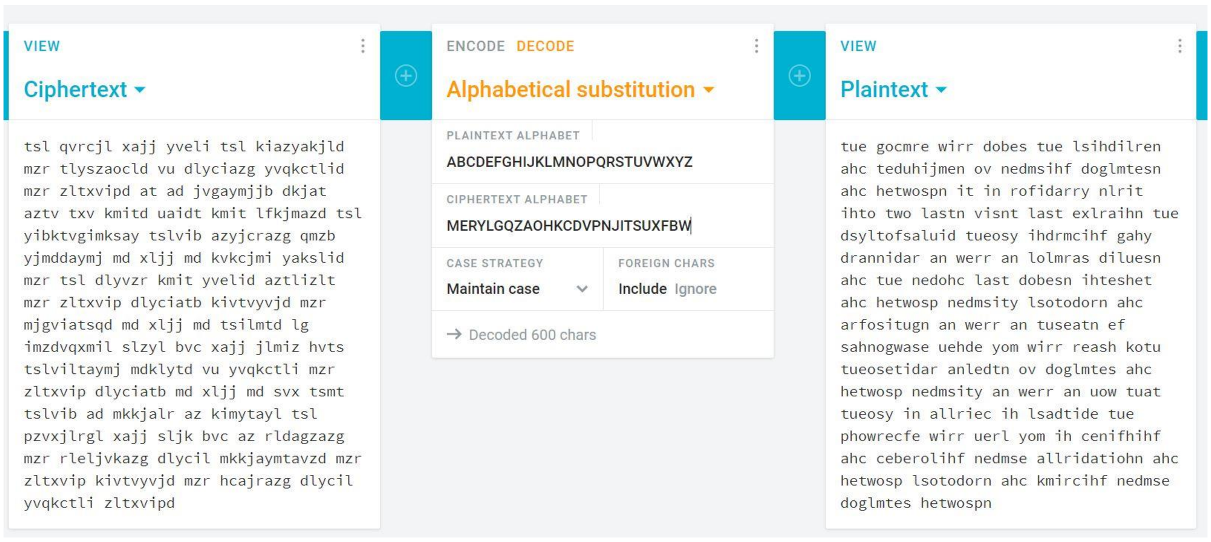


53	42	37	37	36	36	36	36	34	28	25	18	17	15	15	11	9	7	6	3	3	2	1	1		
10.4%	8.3%	7.3%	7.3%	7.1%	7.1%	7.1%	7.1%	6.7%	5.5%	4.9%	3.5%	3.3%	3%	3%	2.2%	1.8%	1.4%	1.2%	0.6%	0.6%	0.4%	0.2%	0.2%		
L	T	M	V	A	D	I	Z	J	Y	K	S	R	C	X	G	B	Q	P	E	U	H	F	O	N	W
E	T	A	O	I	N	S	H	R	D	L	U	C	M	W	F	Y	G	P	B	V	K	X	J	Q	Z

The second row of letters represent the frequency of letters in plaintext form.

Figure 3: Frequency distribution of the given ciphertext with the English letter frequency (*Frequency Analysis*, no date)

Upon substituting the key generated from the above-shown distribution, we get results as shown in Figure 4 (Cryptii, no date).



```
tsl qvrcjl xajj yveli tsl kiazyakjld
mzr tlyszaocld vu dlyciazg yvqkctlid
mzr zltxvipd at ad jvgaymjjb dkjat
aztv txv kmitd uaidt kmit lfkjmazd tsl
yibktvgimksay tslvib azyjcrazg qmzb
yjmddaymj md xljj md kvkcjmi yakslid
mzr tsl dlyvzr kmit yvelid aztlizlt
mzr zltxvip dlyciatb kivtvyvjd mzr
mjgviatsqd md xljj md tsilmtd lg
imzdvqxmil slzyl bvc xajj jlmiz hvts
tslviltaymj mdklytd vu yvqkctli mzr
zltxvip dlyciatb md xljj md svx tsmt
tslvib ad mkkjalr az kimytayl tsl
pzvxjlrgl xajj sljk bvc az rldagzazg
mzr rleljvkazg dlycil mkkjaymtavzd mzr
zltxvip kivtvyvjd mzr hcajrazg dlycil
yvqkctli zltxvipd
```

PLAINTEXT ALPHABET
ABCDEFGHIJKLMNOPQRSTUVWXYZ

CIPHERTEXT ALPHABET
MERYLGQZAOHKCDVPNJITSUXFBW

CASE STRATEGY
Maintain case

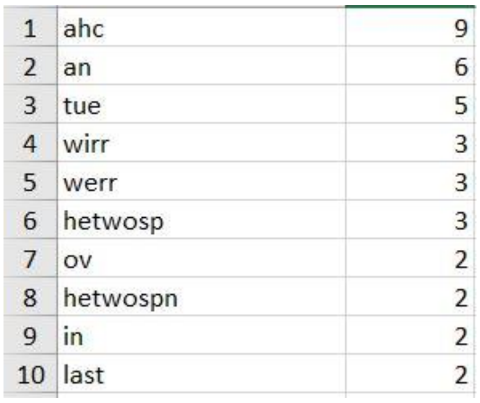
FOREIGN CHARS
Include Ignore

→ Decoded 600 chars

```
tue gocmre wirr dobes tue lsihdilren
ahc teduhijmen ov nedmsihf doglmtesn
ahc hetwospn it in rofidarry nlrit
ihto two lastn visnt last exlraihn tue
dsyltofsaluid tueosy ihdrmcihf gahy
drannidar an werr an lolmras diluesn
ahc tue nedohc last dobesn ihteshet
ahc hetwosp nedmsity lsotodorn ahc
arfositugn an werr an tuseatn ef
sahnogwase uehde yom wirr reash kotu
tueosetidar anledtn ov doglmtes ahc
hetwosp nedmsity an werr an uow tuat
tueosy in allriec ih lsadtide tue
phowrecfe wirr uerl yom ih cenifhihf
ahc ceberolihf nedmse allridatiohn ahc
hetwosp lsotodorn ahc kmircihf nedmse
doglmtes hetwospn
```

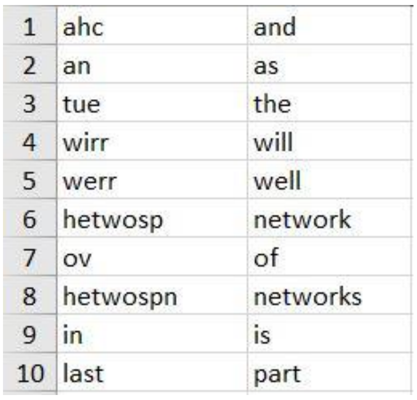
Figure 4: Alphabetical substitution of the key generated from Frequency Analysis (Cryptii, no date)

Since the above process does not provide the plain text completely, we proceed to word frequency analysis. From Figure 5 (contains the top 10 high-frequency words in Figure 4 plain text) shown below, we compare the cipher word frequency to the frequency distribution of words in English text as shown in Figure 2. Figure 6 shows the changes made to the words from Figure 5 by comparing its word count to Figure 2.



1	ahc	9
2	an	6
3	tue	5
4	wirr	3
5	werr	3
6	hetwosp	3
7	ov	2
8	hetwospn	2
9	in	2
10	last	2

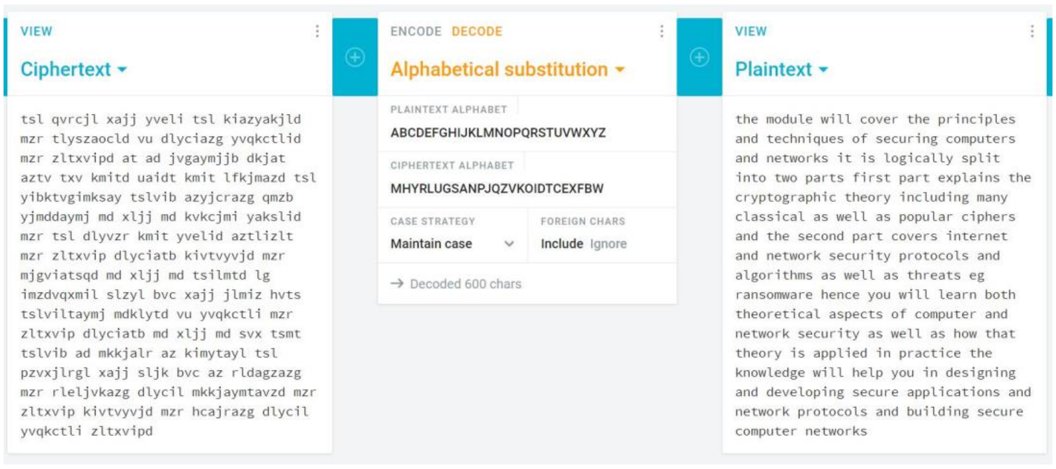
Figure 5: Word Frequency of plaintext from Figure 4



1	ahc	and
2	an	as
3	tue	the
4	wirr	will
5	werr	well
6	hetwosp	network
7	ov	of
8	hetwospn	networks
9	in	is
10	last	part

Figure 6: Decrypted words by comparing Figure 2

Upon substituting the words, we get the desired key and decrypted plain text. Figure 7 shows the recovered key and the decrypted plaintext through alphabetical substitution.



```
tsl qvrcjl xajj yveli tsl kiazyakjld
mzr tlyszaocld vu dlyciazg yvqkctlid
mzr zltxvipd at ad jvgaymjjb dkjat
aztv txv kmitd uaidt kmit lfkjmazd tsl
yibktvgimksay tslvib azyjcrazg qmzb
yjmddaymj md xljj md kvkcjmi yakslid
mzr tsl dlyvzr kmit yvelid aztlizlt
mzr zltxvip dlyciatb kivtvyvjd mzr
mjgviatsqd md xljj md tsilmtd lg
imzdvqxmil slzyl bvc xajj jlmiz hvts
tslviltaymj mdklytd vu yvqkctli mzr
zltxvip dlyciatb md xljj md svx tsmt
tslvib ad mkkjalr az kimytayl tsl
pzvxjlrgl xajj sljk bvc az rldagzazg
mzr rleljvkazg dlycil mkkjaymtavzd mzr
zltxvip kivtvyvjd mzr hcajrazg dlycil
yvqkctli zltxvipd
```

PLAINTEXT ALPHABET
ABCDEFGHIJKLMNOPQRSTUVWXYZ

CIPHERTEXT ALPHABET
MHYRLUGSANPJQZVKOIDTCEXFBW

CASE STRATEGY: Maintain case

FOREIGN CHARS: Include Ignore

→ Decoded 600 chars

```
the module will cover the principles
and techniques of securing computers
and networks it is logically split
into two parts first part explains the
cryptographic theory including many
classical as well as popular ciphers
and the second part covers internet
and network security protocols and
algorithms as well as threats eg
ransomware hence you will learn both
theoretical aspects of computer and
network security as well as how that
theory is applied in practice the
knowledge will help you in designing
and developing secure applications and
network protocols and building secure
computer networks
```

Figure 7: Alphabetical substitution of the recovered key generated from Frequency Analysis of words (Cryptii, no date)

Recovered Key:

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	H	Y	R	L	U	G	S	A	N	P	J	Q	Z	V	K	O	I	D	T	C	E	X	F	B	W

Plain Text:

“ *the module will cover the principles and techniques of securing computers and networks it is logically split into two parts first part explains the cryptographic theory including many classical as well as popular ciphers and the second part covers internet and network security protocols and algorithms as well as threats eg ransomware hence you will learn both theoretical aspects of computer and network security as well as how that theory is applied in practice the knowledge will help you in designing and developing secure applications and network protocols and building secure computer networks* ”

1.2 Task 2: Using OpenSSL Command-Line

Tase 2: To provide 2 examples of encrypting/decrypting with 2 different block ciphers.

AES Encryption:

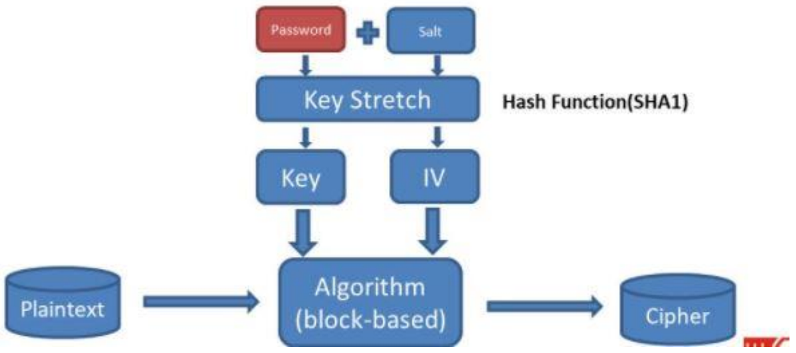
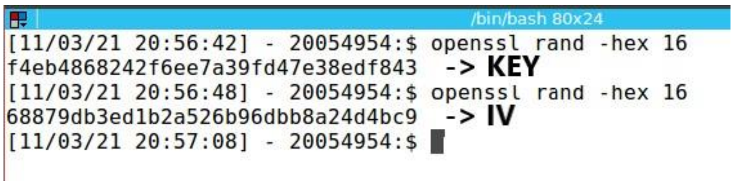


Figure 8: AES Design (Lecture slides)

Figure 8 depicts the process behind AES encryption. There are three types of AES encryption: AES-128, AES-192, AES-256, where 128, 192, 256 represent the length of the considered for encryption/decryption in bits. Since AES is based on symmetric key encryption, it requires a 128/192/256 bit Key. For Key generation, a password of variable length is used to produce a fixed-length key by using the Hash function. According to the mode (CBC/ECB) provided, the plaintext is encrypted(Crawford, 2019).

Example 1:

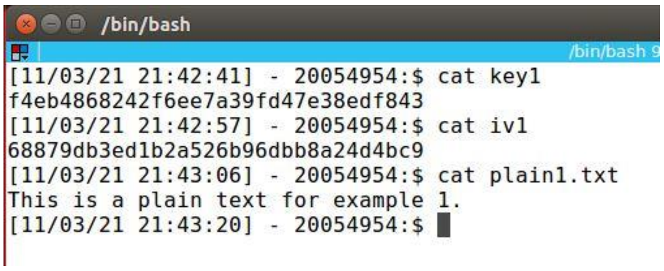
For this example, we use 128-bit AES encryption (-aes-128-cbc) in CBC mode. Instead of using Password and Salt, we can directly provide Key and Initialization Vector (IV). For AES-128 encryption, we need Key and IV of 128bits (16 bytes), if the required length is not provided zeros are padded. Here, we use the OpenSSL rand command to generate 16 bytes random value as shown in Figure 9.



```
[11/03/21 20:56:42] - 20054954:$ openssl rand -hex 16
f4eb4868242f6ee7a39fd47e38edf843  -> KEY
[11/03/21 20:56:48] - 20054954:$ openssl rand -hex 16
68879db3ed1b2a526b96dbb8a24d4bc9  -> IV
[11/03/21 20:57:08] - 20054954:$
```

Figure 9: Key and IV generation using Openssl rand command

Figure 10 provides information about the files used for encryption. Here, the plain1.txt file contains plain text (“This is a plain text for example 1.”), which is to be encrypted.

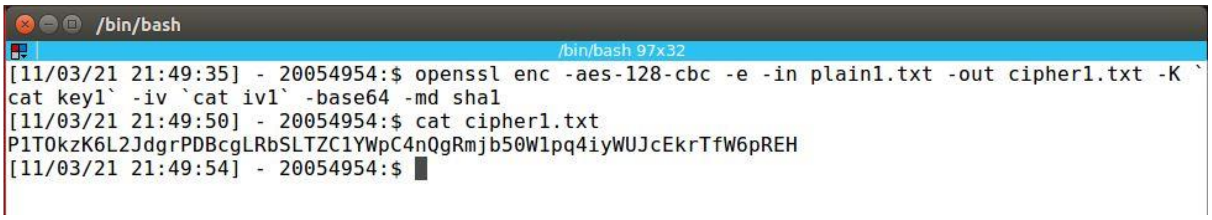


```
[11/03/21 21:42:41] - 20054954:$ cat key1
f4eb4868242f6ee7a39fd47e38edf843
[11/03/21 21:42:57] - 20054954:$ cat iv1
68879db3ed1b2a526b96dbb8a24d4bc9
[11/03/21 21:43:06] - 20054954:$ cat plain1.txt
This is a plain text for example 1.
[11/03/21 21:43:20] - 20054954:$
```

Figure 10: Contents of the files used for encryption

Encryption:

For encryption, we use the OpenSSL enc command which supports various ciphers. Figure 11 shows the command for encryption where -e represents encryption. The Base64 encoding (-base64) is used for plain text encryption so that we can process the ciphertext in a text editor. Since Key and IV are used directly, there is no need for the hash function. For explanatory purposes, we use the sha1 hash function by providing Message Digest command (-md). Rather than explicitly providing the Key and IV we can use command substitution as shown in Figure 11 below(OpenSSL, no date).

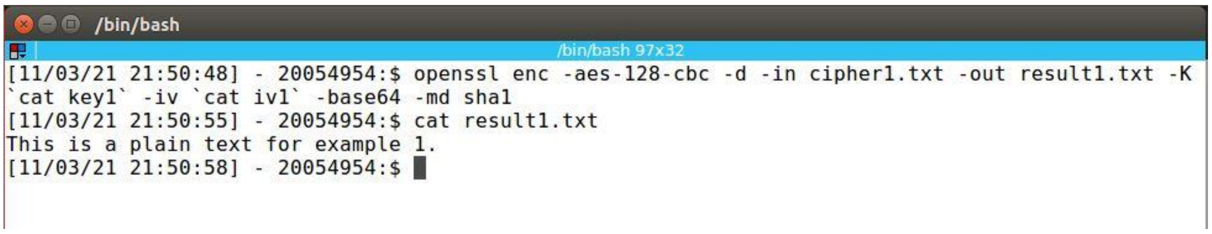


```
[11/03/21 21:49:35] - 20054954:$ openssl enc -aes-128-cbc -e -in plain1.txt -out cipher1.txt -K `
cat key1` -iv `cat iv1` -base64 -md sha1
[11/03/21 21:49:50] - 20054954:$ cat cipher1.txt
P1TOkzK6L2JdgrPDBcgLRbSLTZC1YWpC4nQgRmjb50W1pq4iyWUJcEkrTfW6pREH
[11/03/21 21:49:54] - 20054954:$
```

Figure 11: Encryption of plaintext (plain1.txt) to produce ciphertext (cipher1.txt) using Openssl enc command

Decryption:

Using the same Key and IV which we used for encryption, we decrypt the ciphertext in the cipher1.txt file and store the result in a new file result1.txt. Figure 12 provides the command for decryption and the decrypted plain text. The result1.txt contains the decrypted plaintext - “This is a plain text for example 1.”, which matches the contents of plain1.txt(OpenSSL, no date).



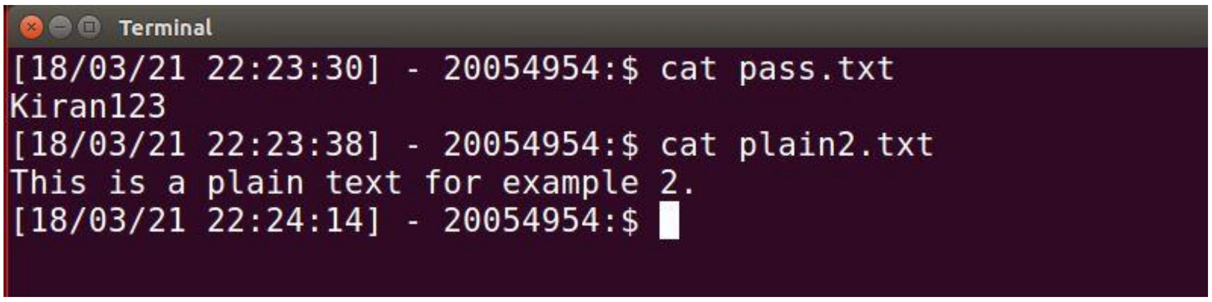
```
[11/03/21 21:50:48] - 20054954:$ openssl enc -aes-128-cbc -d -in cipher1.txt -out result1.txt -K
`cat key1` -iv `cat iv1` -base64 -md sha1
[11/03/21 21:50:55] - 20054954:$ cat result1.txt
This is a plain text for example 1.
[11/03/21 21:50:58] - 20054954:$
```

Figure 12: Decryption of ciphertext (cipher1.txt) with the same Key/IV pair using Openssl enc -d command

Example 2:

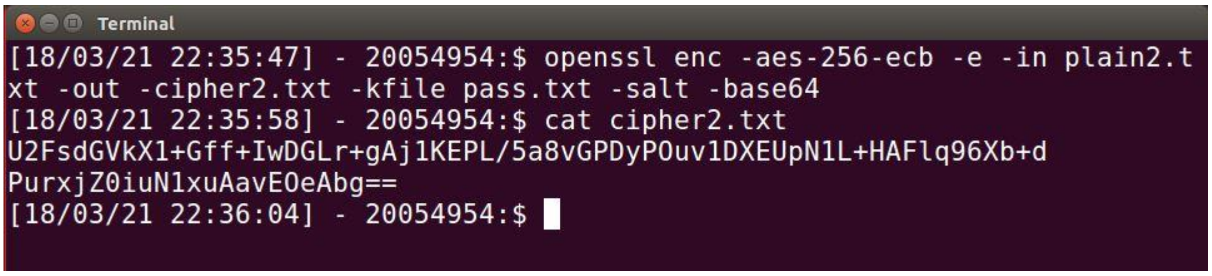
Encryption:

In this example, we use 256-bit AES encryption (-aes-256-ecb) in ECB mode. Figure 14 provides contents of the files used for encryption. Here, plain2.txt file contains plaintext ("This is a plain text for example 2.") and the password is stored in pass.txt. The salt is randomly generated when -s is provided. The -kfile option is used to derive the password from the first line of the file (pass.txt). Figure 15 demonstrates the AES-256 encryption using Openssl command and the resultant ciphertext (OpenSSL, no date).



```
[18/03/21 22:23:30] - 20054954:$ cat pass.txt
Kiran123
[18/03/21 22:23:38] - 20054954:$ cat plain2.txt
This is a plain text for example 2.
[18/03/21 22:24:14] - 20054954:$
```

Figure 14: Contents of the files used for encryption

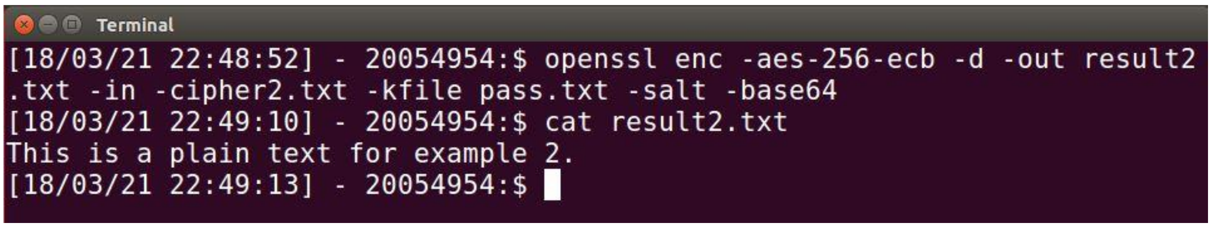


```
[18/03/21 22:35:47] - 20054954:$ openssl enc -aes-256-ecb -e -in plain2.t
xt -out -cipher2.txt -kfile pass.txt -salt -base64
[18/03/21 22:35:58] - 20054954:$ cat cipher2.txt
U2FsdGVkX1+Gff+IwDGLr+gAj1KEPL/5a8vGPDyPOuv1DXEUpN1L+HAFlq96Xb+d
PurxjZ0iuN1xuAavEOeAbg==
[18/03/21 22:36:04] - 20054954:$
```

Figure 15: Encryption of plaintext (plain2.txt) to produce ciphertext (cipher2.txt) using Openssl enc command

Decryption:

Similar to example 1, we use same password which we used for encryption and provide -d which is used for decryption. Figure 12 provides the command for decryption and the decrypted plaintext. The result2.txt contains the decrypted plaintext - "This is a plain text for example 1.", which matches the contents of plain2.txt (OpenSSL, no date).



```
[18/03/21 22:48:52] - 20054954:$ openssl enc -aes-256-ecb -d -out result2
.txt -in -cipher2.txt -kfile pass.txt -salt -base64
[18/03/21 22:49:10] - 20054954:$ cat result2.txt
This is a plain text for example 2.
[18/03/21 22:49:13] - 20054954:$
```

Figure 16: Decryption of ciphertext (cipher2.txt) with the same Password using Openssl enc -d command

1.3 Task 3: CBC mode vs. ECB Mode

Task 3: To compare the use of electronic codebook (ECB) and cipher block chaining (CBC) modes of encryption.

Setup:

In this Task, we use BMP image in which the first 54 characters are reserved for the header which identifies the image format. To preserve the header, we need to copy it to another file (header) and copy it back to the encrypted image file. Figure 18 shows the commands for coping the header from the BMP image (original.bmp) which is to be encrypted and the password required for AES-128 encryption.

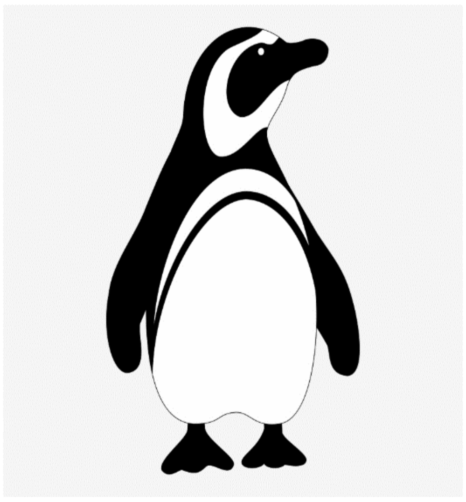
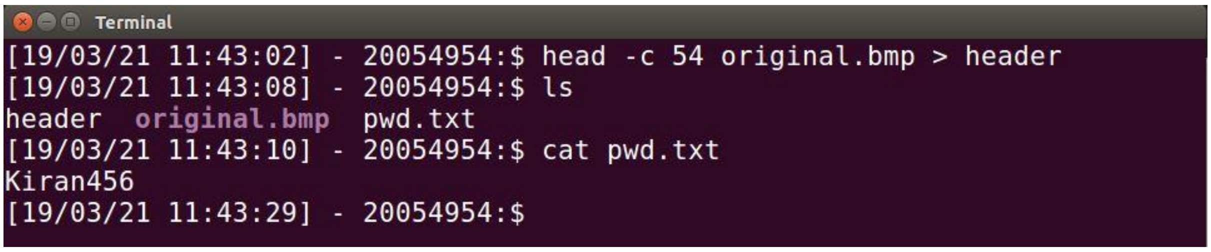


Figure 17: Original BMP image which is to be encrypted



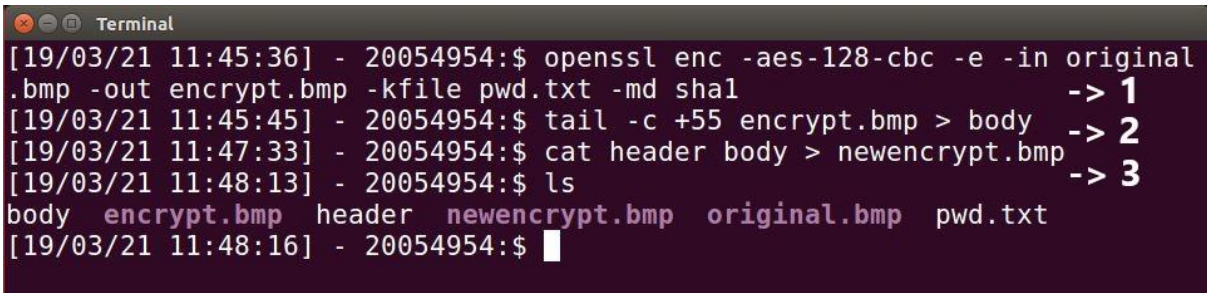
```
[19/03/21 11:43:02] - 20054954:$ head -c 54 original.bmp > header
[19/03/21 11:43:08] - 20054954:$ ls
header  original.bmp  pwd.txt
[19/03/21 11:43:10] - 20054954:$ cat pwd.txt
Kiran456
[19/03/21 11:43:29] - 20054954:$
```

Figure 18: Password for AES-128 encryption and command for coping header from image

Image Encryption:

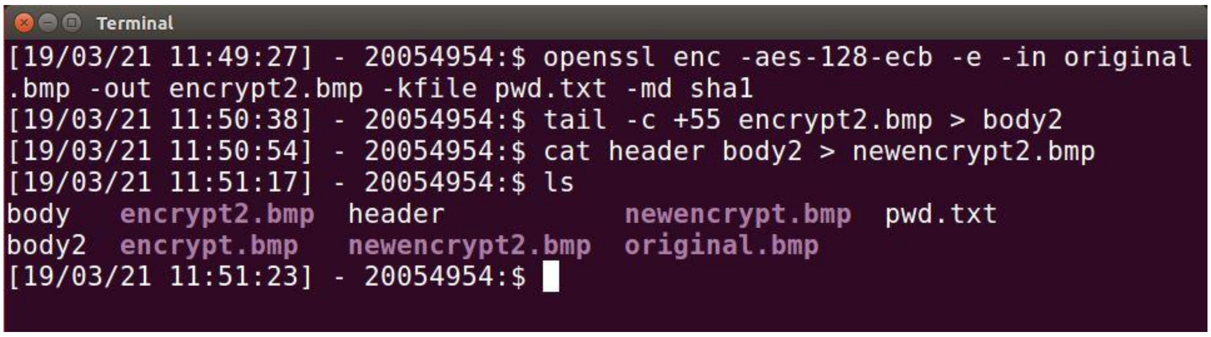
In this Task, we consider AES-128 encryption for comparing the use of electronic codebook (ECB) and cipher block chaining (CBC) modes of encryption. For CBC mode of encryption, we use OpenSSL enc command with -aes-128-cbc cipher as shown in command (1) (in Figure 19). The sha1 hash function is used for generating Key and Initialization Vector (IV) from the password provided. The encrypted image (encrypt.bmp) is not a valid image since it doesn't have a valid header. To overcome this, we combine the original header to the body of encrypted image (as shown in command (3)). The newencrypt.bmp contains the valid encrypted image.

For ECB mode of encryption, we follow the steps used for CBC encryption but replace the -aes-128-cbc to -aes-128-ecb. Figure 20 shows the command for AES-128 encryption in ECB mode.



```
[19/03/21 11:45:36] - 20054954:$ openssl enc -aes-128-cbc -e -in original
.bmp -out encrypt.bmp -kfile pwd.txt -md sha1                          -> 1
[19/03/21 11:45:45] - 20054954:$ tail -c +55 encrypt.bmp > body        -> 2
[19/03/21 11:47:33] - 20054954:$ cat header body > newencrypt.bmp      -> 3
[19/03/21 11:48:13] - 20054954:$ ls
body  encrypt.bmp  header  newencrypt.bmp  original.bmp  pwd.txt
[19/03/21 11:48:16] - 20054954:$
```

Figure 19: AES-128 in CBC mode of encryption



```
[19/03/21 11:49:27] - 20054954:$ openssl enc -aes-128-ecb -e -in original
.bmp -out encrypt2.bmp -kfile pwd.txt -md sha1
[19/03/21 11:50:38] - 20054954:$ tail -c +55 encrypt2.bmp > body2
[19/03/21 11:50:54] - 20054954:$ cat header body2 > newencrypt2.bmp
[19/03/21 11:51:17] - 20054954:$ ls
body   encrypt2.bmp  header           newencrypt.bmp  pwd.txt
body2  encrypt.bmp   newencrypt2.bmp  original.bmp
[19/03/21 11:51:23] - 20054954:$
```

Figure 20: AES-128 in ECB mode of encryption

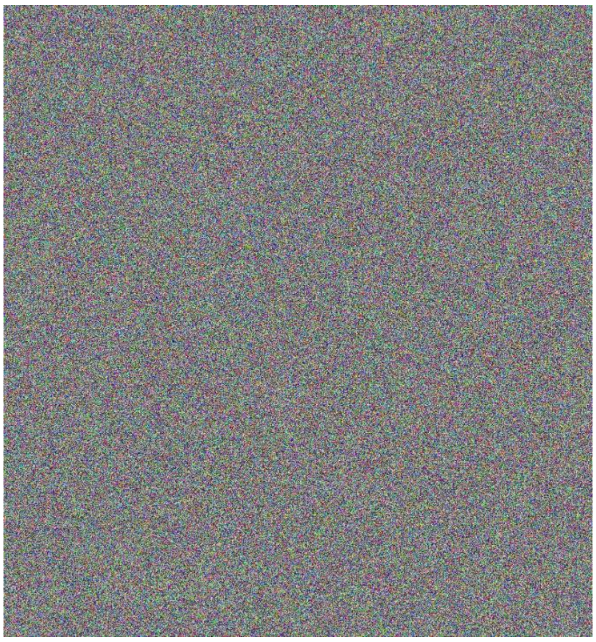


Figure 21: Encrypted image in CBC mode

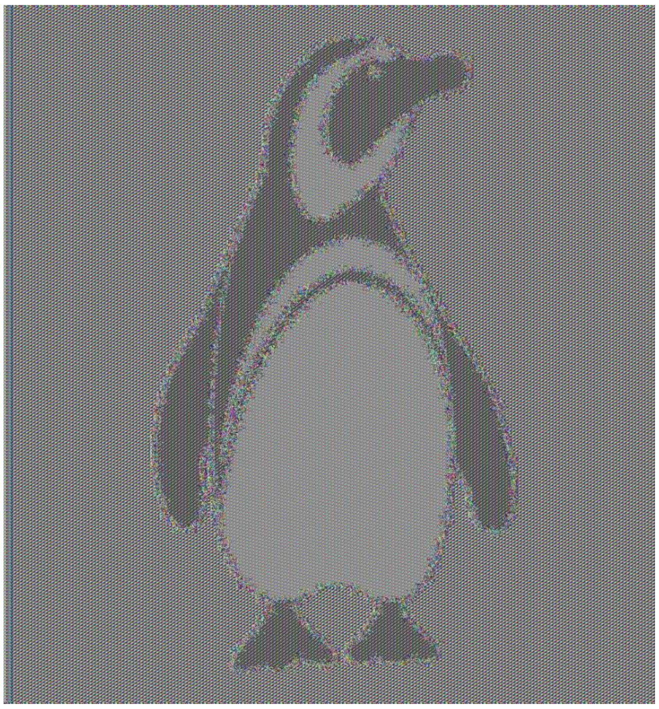


Figure 22: Encrypted image in ECB mode

Observations:

From Figure 21 and Figure 22, we can observe that CBC provides much stronger encryption than ECB. In ECB, the important contents of the original image are easily visible even after encryption. Since it's the basic form of block cipher encryption, it encrypts every 128-bit of data with the same Key and IV. Whereas in CBC, only the first block is XOR'd with IV and encrypted with a Key while other blocks depend on the previous cipher block which is XOR'd with IV and encrypted (as shown in Figure 23). Thus, providing an more complexity to the encrypted data (Shibli049, 2016).

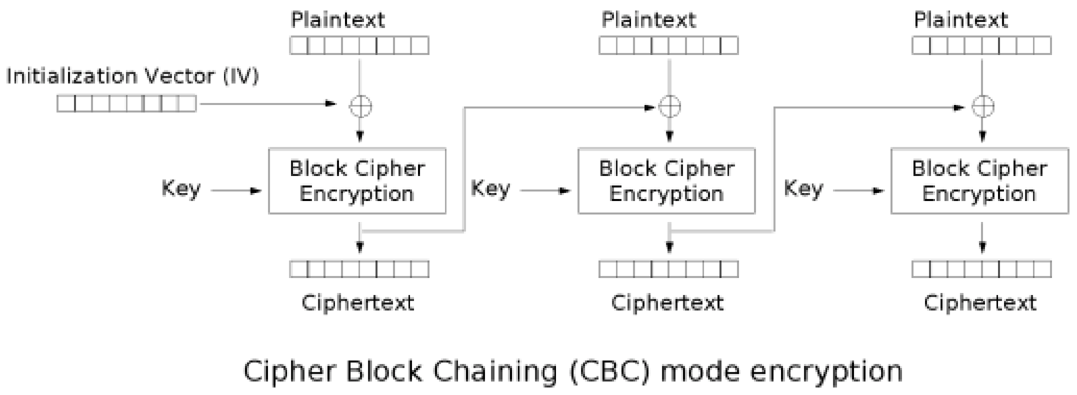
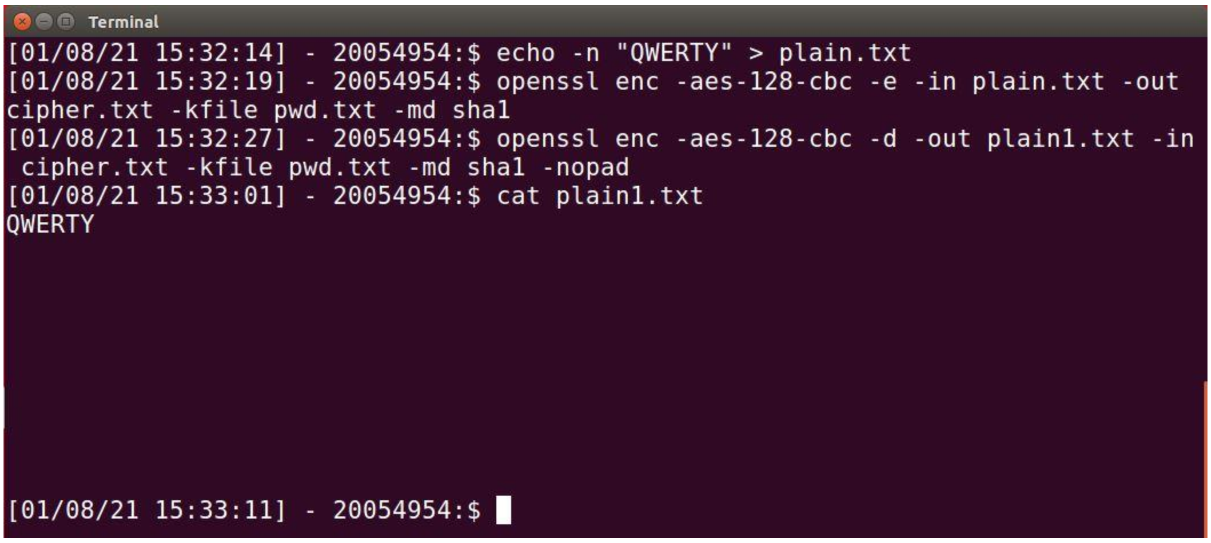


Figure 23: CBC mode of encryption (Source: Cbc_encryption.png (600×243) (wikimedia.org))

1.4 Task 4: Plaintext Padding

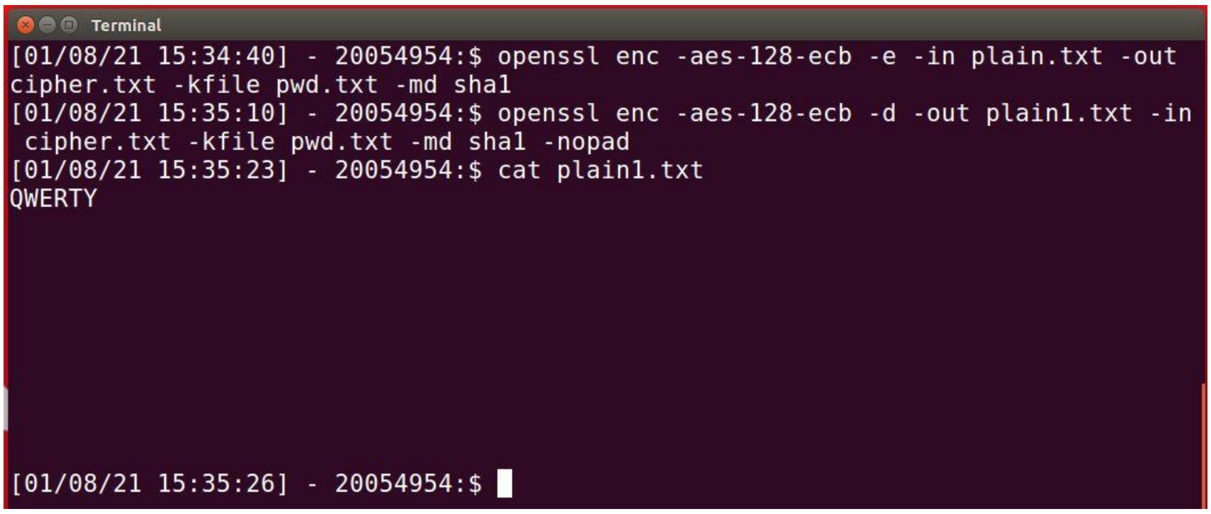
Task 4: To observe the use of padding in ECB, CBC, CFB and OFB modes of encryption.

The padding of plaintext is done only in ECB and CBC mode because the plaintext is divided into blocks of particular length (in this case 16 bytes) for encryption as shown in Figure 23. If the plaintext does not satisfy the block length, zeroes are padded. Figure 24 and Figure 25 shows that padding has been done in CBC and ECB mode respectively. Since the length of plain text is 6 bytes, zeros are padded. After decryption we can see that 10 newline is included in the plain text file. This is to fulfil the block length of size 16 bytes.



```
[01/08/21 15:32:14] - 20054954:$ echo -n "QWERTY" > plain.txt
[01/08/21 15:32:19] - 20054954:$ openssl enc -aes-128-cbc -e -in plain.txt -out
cipher.txt -kfile pwd.txt -md sha1
[01/08/21 15:32:27] - 20054954:$ openssl enc -aes-128-cbc -d -out plain1.txt -in
 cipher.txt -kfile pwd.txt -md sha1 -nopad
[01/08/21 15:33:01] - 20054954:$ cat plain1.txt
QWERTY









[01/08/21 15:33:11] - 20054954:$
```

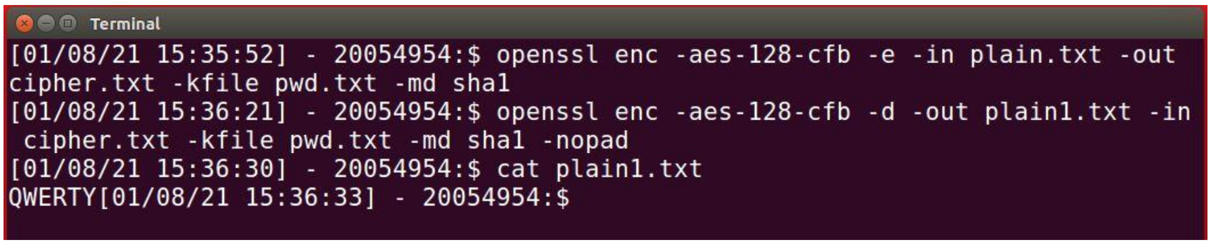
Figure 24: Demonstration of padding done in CBC mode



```
[01/08/21 15:34:40] - 20054954:$ openssl enc -aes-128-ecb -e -in plain.txt -out
cipher.txt -kfile pwd.txt -md sha1
[01/08/21 15:35:10] - 20054954:$ openssl enc -aes-128-ecb -d -out plain1.txt -in
 cipher.txt -kfile pwd.txt -md sha1 -nopad
[01/08/21 15:35:23] - 20054954:$ cat plain1.txt
QWERTY









[01/08/21 15:35:26] - 20054954:$
```

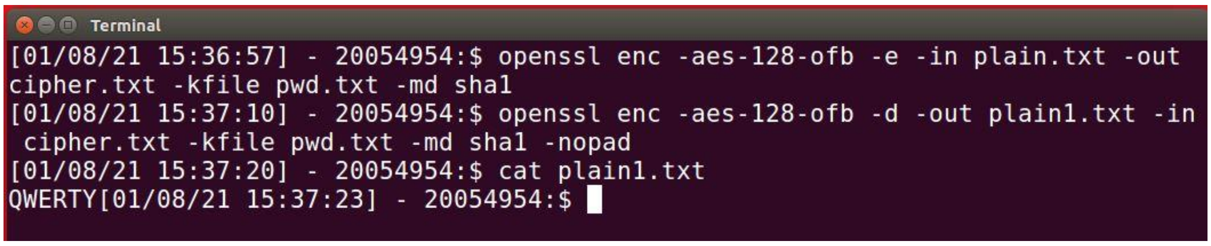
Figure 24: Demonstration of padding done in ECB mode

Whereas in the CFB and OFB mode, the padding is not required. This is because the block length of these modes is equal to the length of plaintext. Figure 25 and Figure 26 shows that no padding is done in CFB and OFB modes respectively.



```
[01/08/21 15:35:52] - 20054954:$ openssl enc -aes-128-cfb -e -in plain.txt -out
cipher.txt -kfile pwd.txt -md sha1
[01/08/21 15:36:21] - 20054954:$ openssl enc -aes-128-cfb -d -out plain1.txt -in
 cipher.txt -kfile pwd.txt -md sha1 -nopad
[01/08/21 15:36:30] - 20054954:$ cat plain1.txt
QWERTY[01/08/21 15:36:33] - 20054954:$
```

Figure 24: Demonstration of no padding in CFB mode



```
[01/08/21 15:36:57] - 20054954:$ openssl enc -aes-128-ofb -e -in plain.txt -out
cipher.txt -kfile pwd.txt -md sha1
[01/08/21 15:37:10] - 20054954:$ openssl enc -aes-128-ofb -d -out plain1.txt -in
 cipher.txt -kfile pwd.txt -md sha1 -nopad
[01/08/21 15:37:20] - 20054954:$ cat plain1.txt
QWERTY[01/08/21 15:37:23] - 20054954:$
```

Figure 24: Demonstration of no padding in OFB mode

1.5 Task 5: Incorrect use of IVs

The provided ciphertext and plaintext are:

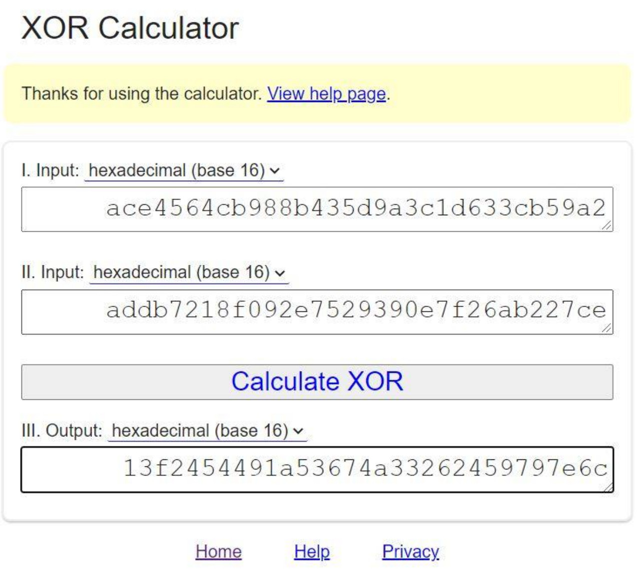
P1 (ASCII): UWE is G-ACE-CSE

C1 (Hex): ace4564cb988b435d9a3c1d633cb59a2

C2 (Hex): addb7218f092e7529390e7f26ab227ce

The property of XOR called Involution (A XOR B = C, A XOR C = B, B XOR C = A) can be used. When a two plaintext is encrypted with a same key and iv pair, a simple XOR between two ciphertext and then with the first plaintext will produce the second plaintext. Figure 25 shows the first of XORing the two ciphertext. Figure 26 shows the recovered plaintext from the first plaintext.

The recovered plaintext: “That is great:-)”


XOR Calculator

Thanks for using the calculator. View help page.

I. Input: hexadecimal (base 16)

ace4564cb988b435d9a3c1d633cb59a2

II. Input: hexadecimal (base 16)

addb7218f092e7529390e7f26ab227ce

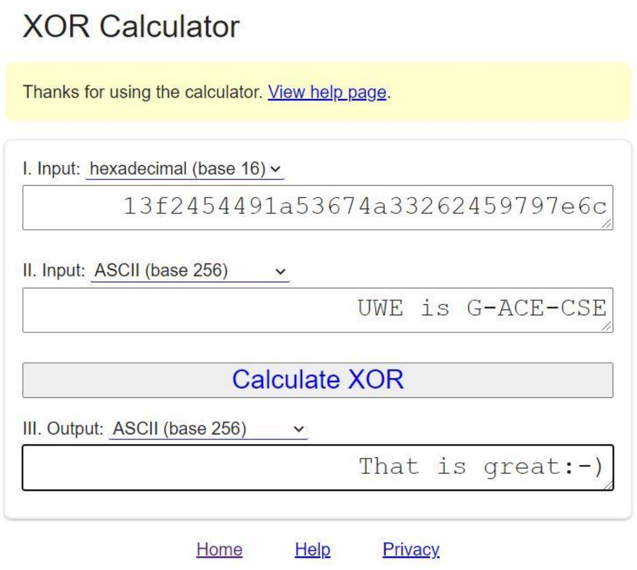
Calculate XOR

III. Output: hexadecimal (base 16)

13f2454491a53674a33262459797e6c

Home Help Privacy

Figure 25: XOR of two ciphertext


XOR Calculator

Thanks for using the calculator. View help page.

I. Input: hexadecimal (base 16)

13f2454491a53674a33262459797e6c

II. Input: ASCII (base 256)

UWE is G-ACE-CSE

Calculate XOR

III. Output: ASCII (base 256)

That is great:-)

Home Help Privacy

Figure 26: XOR of resultant hex value to plain text

1.6 Task 6: Brute-Force using the OpenSSL C Library

The provided plaintext and ciphertext for this task are:

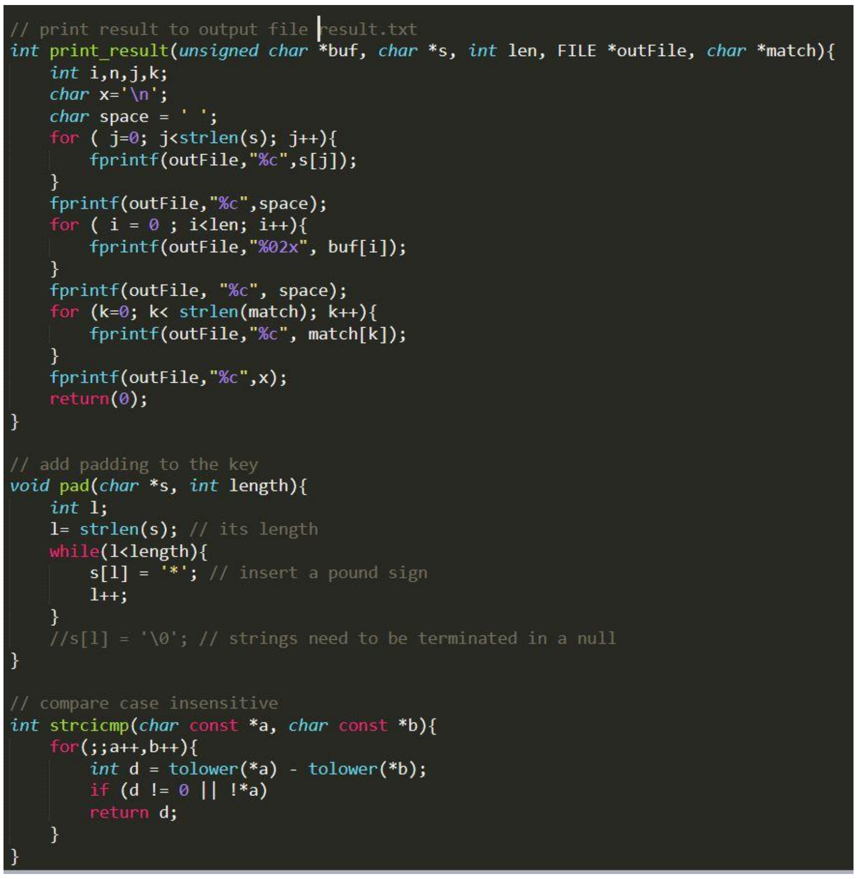
Plaintext (ASCII): Gold ACE-CSE!

Ciphertext (Hex): e2773a09c0e095da9eedebf34ed636da

IV (Hex): ffeeddccbbaa00112233445566778899

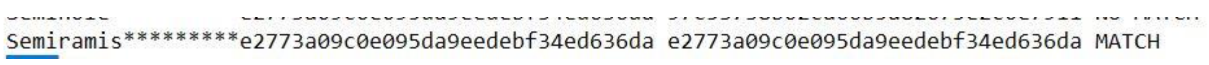
The provided wordlist.txt file is used for brute forcing the password. For brute-force, the provided Toy.c program is modified to get the desired output. Figure 27 shows the code snippet of the program. The print_result() is used to get the results of brute-forcing the encryption and print it in a file. The pad() function is used to add * to the password because password contains this character to get the required length. After execution, the results file was generated and in that we could see the matched password as shown in Figure 28.

The password is: "Semiramis*********"



```c
// print result to output file result.txt
int print_result(unsigned char *buf, char *s, int len, FILE *outFile, char *match){
    int i,n,j,k;
    char x='\n';
    char space = ' ';
    for ( j=0; j<strlen(s); j++){
        fprintf(outFile,"%c",s[j]);
    }
    fprintf(outFile,"%c",space);
    for ( i = 0 ; i<len; i++){
        fprintf(outFile,"%02x", buf[i]);
    }
    fprintf(outFile, "%c", space);
    for (k=0; k< strlen(match); k++){
        fprintf(outFile,"%c", match[k]);
    }
    fprintf(outFile,"%c",x);
    return(0);
}

// add padding to the key
void pad(char *s, int length){
    int l;
    l= strlen(s); // its length
    while(l<length){
        s[l] = '*'; // insert a pound sign
        l++;
    }
    //s[l] = '\0'; // strings need to be terminated in a null
}

// compare case insensitive
int strcicmp(char const *a, char const *b){
    for(;;a++,b++){
        int d = tolower(*a) - tolower(*b);
        if (d != 0 || !*a)
        return d;
    }
}
```

Figure 27 - Code snippet of bruteforce program



```
Semiramis*********e2773a09c0e095da9eedebf34ed636da e2773a09c0e095da9eedebf34ed636da MATCH
```

Figure 28 – Matched password

1.7 Task 7: Post-Quantum Ciphers

INTRODUCTION

With the rapid technological advancements in Information Technology, the need for secure data transmission makes cryptography one of the most critical fields. Cryptography consists of two types; one is asymmetric, which uses public key and private key for encryption and decryption, and another is symmetric, which involves a single private key (Mavroeidis *et al.*, 2018).

In the future, the technology which poses a significant threat to cryptography is Quantum computing. The processing power of quantum computers is more than a thousand times faster than current supercomputers (Novet, 2015). Presently, all the encryption methods are secure because brute force attacks are time-consuming. With the advent of quantum computing, it would take just a fraction of seconds to crack some of the existing encryption mechanisms. Thus, the time has arrived for developing new encryption methods and standards for the post-quantum era (Kirsch, 2015).

SYMMETRIC ENCRYPTION

In symmetric encryption (secret key encryption), the sender and receiver use a single private key for encryption and decryption. This private key should be kept confidential; only the sender and receiver show know it (Shaify and Meenakshi, 2014). The only method to crack the symmetric encryption is through brute-force attacks, but it would take about quadrillions of years to test every possibility (Kirsch, 2015). The well-known symmetric algorithms are DES (Data Encryption Standard), 3DES (Triple DES), AES (Advanced Encryption Standard).

QUANTUM ALGORITHM IN SYMMETRIC ENCRYPTION

Quantum computers, which are much more powerful than classical computers, can quickly break encryption methods by calculating all secret keys by interfering with the communication between sender and receiver. The two algorithms which affect the cryptographic systems are Shor's algorithm and Grover's algorithm. Shor's algorithm exposes the systems that depend on the difficulty in computing discrete logarithms or factorization. Since asymmetric encryption relies on large prime integer factorization or discrete logarithm problem, it is collapsed using Shor's algorithm (Mavroeidis *et al.*, 2018).

Grover's algorithm:

L. Grover created an algorithm that affected many cryptographic systems and provided the foundation for most of the positive applications used for quantum computing. Grover's algorithm searches for the roots (x) of a function f, where f(x) = 0 (Bernstein and Lange, 2017). While a classical computer finds an entry in an unordered database in *N/2* searches (*N* – total number of entries), this algorithm can find the same entry in $\sqrt{N}$ searches (Mavroeidis *et al.*, 2018). Since each $\sqrt{N}$ quantum evaluations depend on previous evaluation to finish, Shor's algorithm provides more speed-up than Grover's (Bernstein and Lange, 2017).

Grover's algorithm in symmetric encryption:

Quantum algorithms pose a minor threat to symmetric cryptography. Grover's algorithm is the only recognized threat that replaces the conventional brute-force algorithm. For example, the DES, which is one of the earliest symmetric algorithms with a 56-bit key, can be cracked using Grover's algorithm with just 185 searches (Mavroeidis *et al.*, 2018). Let us consider another example, the AES, which is more secure, efficient, and commonly used than DES and 3DES with key sizes 128-bits, 196-bits, and 256-bits. The 128-bit AES cipher has 2^{128} possible keys. A classical computer takes about 10.79 quadrillion years to test every possibility. A quantum computer using Grover's algorithm can search in $\sqrt{N}$ quantum evaluations, that is, $\sqrt{2^{128}} = 2^{64}$ searches, which takes about six months to test every possibility (Kirsch, 2015).

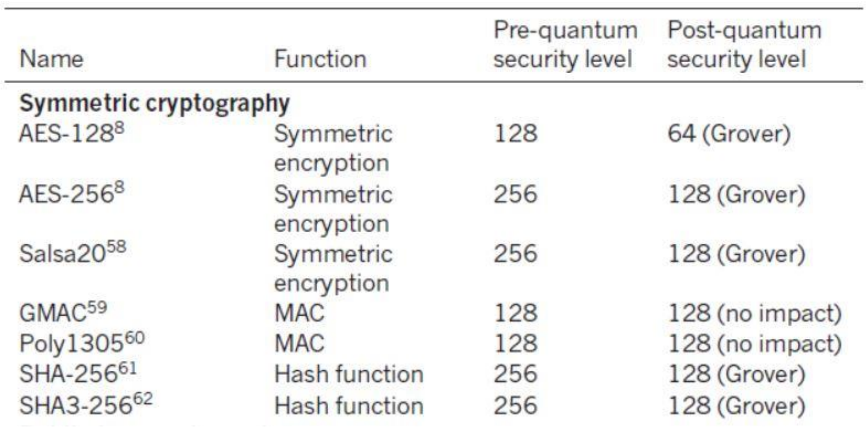
POST-QUANTUM CRYPTOGRAPHY

The ultimate aim of quantum-resistant cryptography is to develop encryption methods and standards secure against quantum and classical computing (Mavroeidis *et al.*, 2018). In July 2020, NIST (National Institute of Standard and Technology) published the Status Report of the second round for the post-quantum cryptography standardization process. It has selected seven finalists for the third round. By early 2022, NIST will report the final draft of standards (Alagic *et al.*, 2019).

The following cryptographic methods can be implemented in resisting quantum-based attacks.

Symmetric key cryptography:

Symmetric encryption methods are resistant against quantum attacks only when used with longer key lengths. The AES (Advanced Encryption Standard) is one of the most secure algorithms against quantum computations if used with key sizes of 192 bits or 256 bits (Mavroeidis *et al.*, 2018). Figure 34 provides the effects of Grover's algorithm on symmetric encryption.



Name	Function	Pre-quantum security level	Post-quantum security level
Symmetric cryptography			
AES-128[8]	Symmetric encryption	128	64 (Grover)
AES-256[8]	Symmetric encryption	256	128 (Grover)
Salsa20[58]	Symmetric encryption	256	128 (Grover)
GMAC[59]	MAC	128	128 (no impact)
Poly1305[60]	MAC	128	128 (no impact)
SHA-256[61]	Hash function	256	128 (Grover)
SHA3-256[62]	Hash function	256	128 (Grover)

Figure 34: Symmetric cryptography and their security levels (Bernstein and Lange, 2017)

Quantum Key Distribution:

Quantum Key Distribution (QKD) relies on quantum mechanics, and various protocols have been developed based on quantum properties. The two quantum properties that led to the QKD protocol development are explained below (Mavroeidis *et al.*, 2018).

Prepare-and-measure (P&M) protocol: This protocol uses the Heisenberg Uncertainty principle, which states that an object's quantum state changes when measured. This property manifests the eavesdropping attack on a channel through the changes in the quantum state. It can even provide the amount of information intercepted by the attacker (Mavroeidis *et al.*, 2018).

Entanglement-based (EB) protocol: It is a quantum phenomenon that links two or more objects, thus becoming a single object. When a pair of entangled objects are intercepted, the overall system gets altered. This reveals the attacker's presence and the amount of information intercepted (Mavroeidis *et al.*, 2018).

RECENT FINDINGS

In February 2021, Markus Pflitsch, founder and CEO of Terra Quantum, said Terra Quantum had found a vulnerability in the existing post-quantum cryptography. They have found a weakness in AES encryption methods on the message-digest algorithm MD5. This weakness can be exploited using a quantum annealer, which does not exist currently, containing 20,000 qubits. Terra Quantum has proposed a protocol named 'the superfast Boltzmann-Planck-protected secure information transmission' to overcome this vulnerability. Terra Quantum's CTOs, Professors Gordey Lesovik and Valerii Vinokur said: 'The critical component of the proposed protocol is the change of the security paradigm based on quantum irreversibility' (Terra Quantum, 2021).

CONCLUSION

Quantum computing will revolutionize medicine, communication, finance modeling, artificial intelligence, and much more. In contrast, one of the significant threats that quantum computing poses is collapsing the widely used encryption methods. The need for migrating from the present cryptography to post-quantum cryptography has arisen. Thus, implementing encryption schemes that are resistant to quantum algorithms such as Quantum Key Distribution (QKD) and AES-192/256 bit.

1.8 References

Alagic, G. *et al.* (2019) 'Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process', *Nistir 8309*, pp. 1–27. Available at: https://nvlpubs.nist.gov/nistpubs/ir/2020/NIST.IR.8309.pdf.

Bernstein, D. J. and Lange, T. (2017) ‘Post-quantum cryptography’, *Nature*, 549(7671), pp. 188–194. doi: 10.1038/nature23461.

Crawford, D. (2019) *How does AES encryption work?* Available at: https://proprivacy.com/guides/aes-encryption.

Cryptii (no date) *Alphabetical Substitution*. Available at: https://cryptii.com/pipes/alphabetical-substitution.

Frequency Analysis (no date). Available at: http://www.brianveitch.com/maze-runner/frequency-analysis/index.html.

Kirsch, Z. (2015) ‘Quantum Computing: The Risk to Existing Encryption Methods’. Available at: http://www.cs.tufts.edu/comp/116/archive/fall2015/zkirsch.pdf.

Mavroeidis, V. *et al.* (2018) ‘The impact of quantum computing on present cryptography’, *International Journal of Advanced Computer Science and Applications*, 9(3), pp. 405–414. doi: 10.14569/IJACSA.2018.090354.

Norvig, P. (no date) *English Letter Frequency Counts: Mayzner Revisited*. Available at: http://norvig.com/mayzner.html.

Novet, J. (2015) ‘Google says its quantum computer is more than 100 million times faster than a regular computer chip.’ Available at: http://venturebeat.com/2015/12/08/%0Agoogle-says-its-quantum-computer-is-more-than-100-million-times-faster-than-a-%0Aregular-computer-chip/.

OpenSSL (no date) ‘enc’. Available at: https://www.openssl.org/docs/man1.1.1/man1/enc.html.

Shaify, K. and Meenakshi, M. (2014) ‘Performance Evaluation of various Symmetric Encryption Algorithm’, *International Conference on Parallel, Distributed and Grid Computing*, pp. 105–109.

Shibli049 (2016) *Is there any difference between aes-128-cbc and aes-128 encryption?* Available at: https://stackoverflow.com/questions/33121619/is-there-any-difference-between-aes-128-cbc-and-aes-128-encryption.

Terra Quantum (2021) *Terra Quantum Makes Electronically Transmitted Communications Unbreakable After Revealing Weakness in ‘post-quantum Cryptography’*. Available at: https://www.businesswire.com/news/home/20210208005290/en/.