CSCI 400 Cryptography Lab 2

Prof. Faheem Abdur-Razzaaq

September 13, 2019

(Priya Thapa, Emranul Hakim, Lakpa S. Sherpa, Kenneth Asamoah,)

Topic: Secret-Key Encryption

2.1 Task1: FrequencyAnalysisAgainstMonoalphabeticSubstitutionCipher: (Priya and Lakpa)

1.Generating Key using Python:

```
[09/11/19]seed@VM:~/Desktop$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for mo
re information.
>>> import random
>>> s ="abcdefghijklmnopqrstuvwxyz"
>>> list = random.sample(s, len(s))
>>> ''.join(list)
'plorgdhiynmqkvwjctezfusbax'
>>>
```

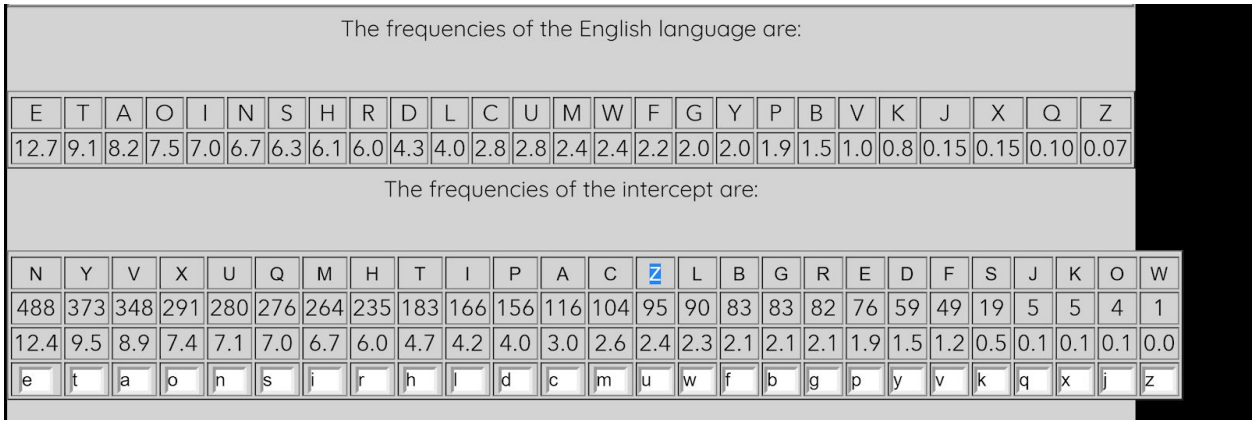
Key generated:

plorgdhiynmqkvwjctezfusbax

Encrypted using the same key:

```
[1]+  Stopped                 python
[09/11/19]seed@VM:~/Desktop$ tr 'abcdefghijklmnopqrstuv
wxyz''plorgdhiynmqkvwjctezfusbax' \ <words.txt> out.txt
```

Result:



The frequencies of the English language are:

E	T	A	O	I	N	S	H	R	D	L	C	U	M	W	F	G	Y	P	B	V	K	J	X	Q	Z
12.7	9.1	8.2	7.5	7.0	6.7	6.3	6.1	6.0	4.3	4.0	2.8	2.8	2.4	2.4	2.2	2.0	2.0	1.9	1.5	1.0	0.8	0.15	0.15	0.10	0.07

The frequencies of the intercept are:

N	Y	V	X	U	Q	M	H	T	I	P	A	C	Z	L	B	G	R	E	D	F	S	J	K	O	W
488	373	348	291	280	276	264	235	183	166	156	116	104	95	90	83	83	82	76	59	49	19	5	5	4	1
12.4	9.5	8.9	7.4	7.1	7.0	6.7	6.0	4.7	4.2	4.0	3.0	2.6	2.4	2.3	2.1	2.1	2.1	1.9	1.5	1.2	0.5	0.1	0.1	0.1	0.0
e	t	a	o	n	s	i	r	h	l	d	c	m	u	w	f	b	g	p	y	v	k	q	x	j	z

The common trigraphs:

The most common trigraphs in the english language are:

THE,AND,THA,ENT,ION,TIO,FOR,NDE,HAS,NCE,TIS,OFT,MEN

The most common trigraphs in the message are:

YTN,VUP,MUR,YNH,XZY,MXU,NQY,GNQ,YTV,VII,BXH,LVQ,NUY

Changing Patterns:

```
the XQcarQ tZrn  Xn QZndaD LhMch QeeCQ aGXZt rMRht aBter thMQ
IXnR QtranRe
aLardQ trME the GaRRer BeeIQ IMSe a nXnaRenarMan tXX

the aLardQ race LaQ GXXSended GD the deCMQe XB harFeD
LeMnQteMn at MtQ XZtQet
and the aEEarent MCEIXQMXn XB hMQ BMIC cXCEanD at the end and
Mt LaQ QhaEed GD
the eCerRence XB CetXX tMCeQ ZE GIacSRXLn EXIMtMcQ arCcandD
actMFMQC and
```

2.2 Task2: Encryption using Different Ciphers and Modes

Priya

In order to get the cipher type, I used this commands:

$ openssl enc -ciphertype -e -in plain.txt -out cipher.bin \ -K 00112233445566778889aabbccddeeff \ -iv 0102030405060708

```
Cipher Types
-aes-128-cbc            -aes-128-ccm
aes-128-cfb
-aes-128-cfb1           -aes-128-cfb8
aes-128-ctr
-aes-128-ecb            -aes-128-gcm
aes-128-ofb
-aes-128-xts            -aes-192-cbc
aes-192-ccm
-aes-192-cfb            -aes-192-cfb1
aes-192-cfb8
-aes-192-ctr            -aes-192-ecb
aes-192-gcm
-aes-192-ofb            -aes-256-cbc
aes-256-ccm
-aes-256-cfb            -aes-256-cfb1
aes-256-cfb8
-aes-256-ctr            -aes-256-ecb
aes-256-gcm
-aes-256-ofb            -aes-256-xts
```

```
-aes-256-ofb            -aes-256-xts
aes128
-aes192                 -aes256
bf
-bf-cbc                 -bf-cfb
bf-ecb
-bf-ofb                 -blowfish
camellia-128-cbc
-camellia-128-cfb       -camellia-128-cfb1
camellia-128-cfb8
-camellia-128-ecb       -camellia-128-ofb
camellia-192-cbc
-camellia-192-cfb       -camellia-192-cfb1
camellia-192-cfb8
-camellia-192-ecb       -camellia-192-ofb
camellia-256-cbc
-camellia-256-cfb       -camellia-256-cfb1
camellia-256-cfb8
-camellia-256-ecb       -camellia-256-ofb
camellia128
-camellia192            -camellia256
cast
```

Cipher type to encrypt and decrypt a message:

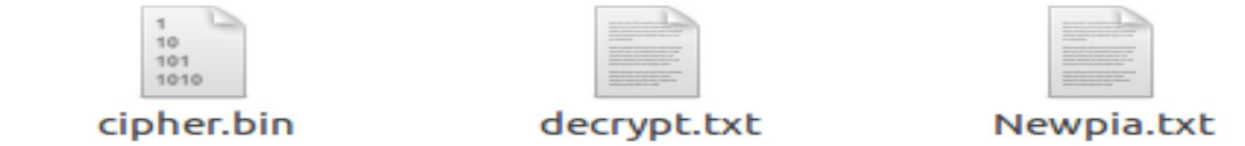
1. **-aes-128-cbc**

   $ openssl enc -aes-128-cbc -e -in Newpia.txt -out cipher.bin \ -K 00112233445566778889aabbccddeeff \ -iv 0102030405060708

```
[09/09/19]seed@VM:~/Desktop$ openssl enc -aes-128-cbc -
e -in Newpia.txt -out cipher.bin -K 0011223344556778888
9aabbccddeeff  -iv 0102030405060708
```

$ openssl enc -aes-128-cbc -d -in cipher.bin -out Newpia.txt \ -K 00112233445566778889aabbccddeeff \ -iv 0102030405060708

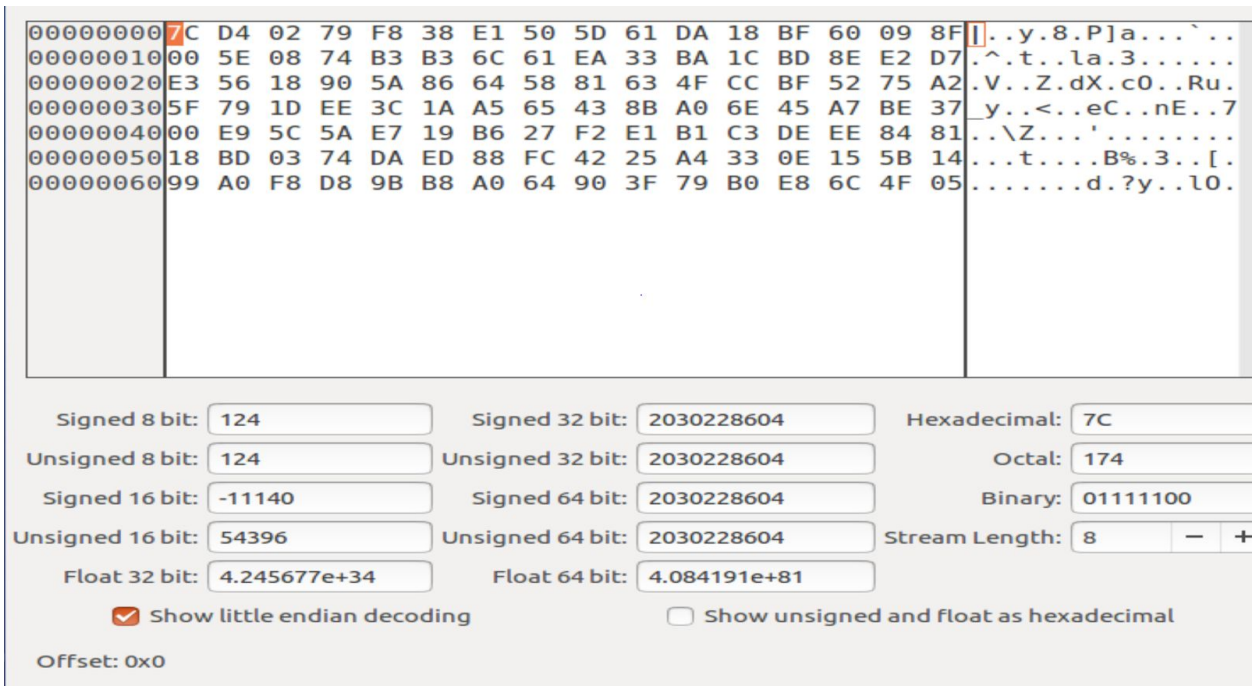
```
[09/09/19]seed@VM:~/Desktop$ openssl enc -aes-128-cbc -
d -in cipher.bin -out decrypt.txt -K 0011223344556677889
89aabbccddeeff  -iv 0102030405060708
```



Opened encrypt file with GHex:

GHex is a simple binary editor. It lets users view and edit a binary file in both hex and ascii with a multiple level undo/redo mechanism.

Features include find and replace functions, conversion between binary, octal, decimal and hexadecimal values, and use of an alternative, user-configurable MDI concept that lets users edit multiple documents with multiple views of each.



2. **-bf-ofb**

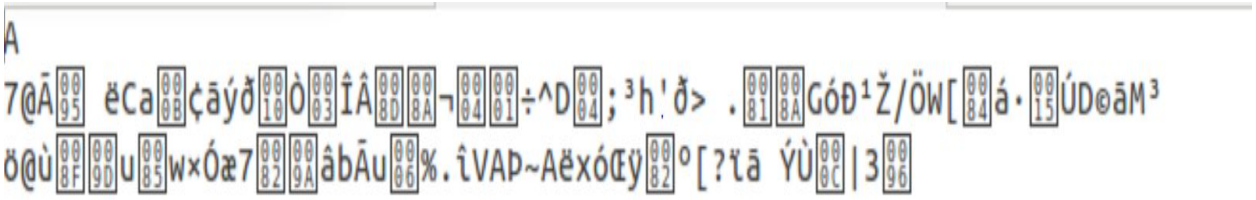
Encrypt:

```
[09/09/19]seed@VM:~/Desktop$ openssl enc -bf-ofb -e -in
 Newpia.txt -out Secondcipher.txt -K 0011223344556677888
9aabbccddeeff  -iv 0102030405060708
```

Decrypt:

```
[09/09/19]seed@VM:~/Desktop$ openssl enc -bf-ofb -e -in
 Secondcipher.txt -out Decrypt2.txt -K 0011223344556677
8889aabbccddeeff  -iv 0102030405060708
```

Opened encrypt file with Gedit:



3. **-des-ede3-cfb**

Encrypt:

```
[09/09/19]seed@VM:~/Desktop$ openssl enc -des-ede3-cfb
 -e -in Newpia.txt -out Thirdcipher.txt -K 001122334455
66778889aabbccddeeff  -iv 0102030405060708
```

Decrypt:

```
[09/09/19]seed@VM:~/Desktop$ openssl enc -des-ede3-cfb
 -d -in Thirdcipher.txt -out Decrypt3.txt -K 0011223344
5566778889aabbccddeeff  -iv 0102030405060708
```

Opened encrypt file with Gedit:

Opened encrypt file with Sublime Text:

```
6d8c 4b8f ebd4 f471 0205 7f34 e758 57fa
425e b0fe 6706 e3e2 b75d 8548 4e02 4fab
7e4e 1056 e630 f9bc fed4 33a0 cd94 b252
e001 e972 0e4b a178 62b0 4d57 800f ef6c
3267 2a19 87f7 c5e3 8f38 5300 dc7e 4fe9
0fc5 e111 951c e12f d0fe 21ce 82e3 6747
2eef 42af 2b39 9852 1b
```

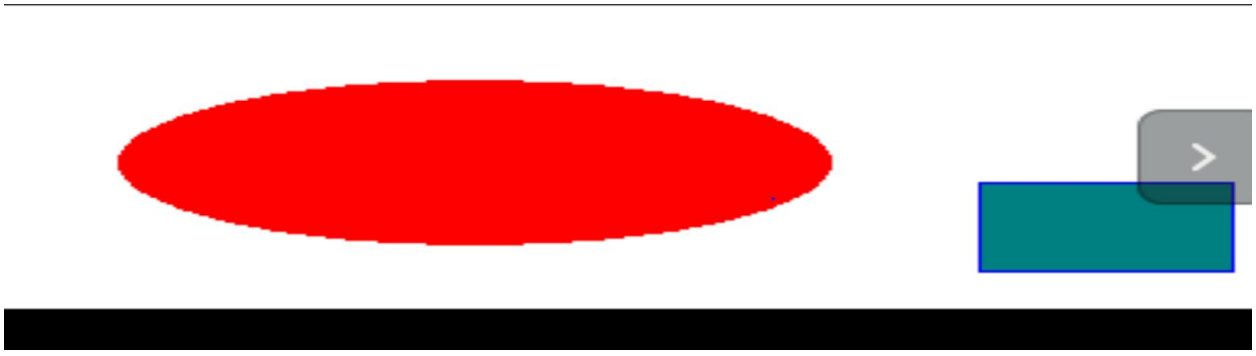
Result:

All the plaintext were retrieved from the cipher text correctly.

2.3 Task3: EncryptionMode–ECBvs. CBC

Encrypt the file using the ECB (Electronic Code Book) and CBC (Cipher Block Chaining) modes:

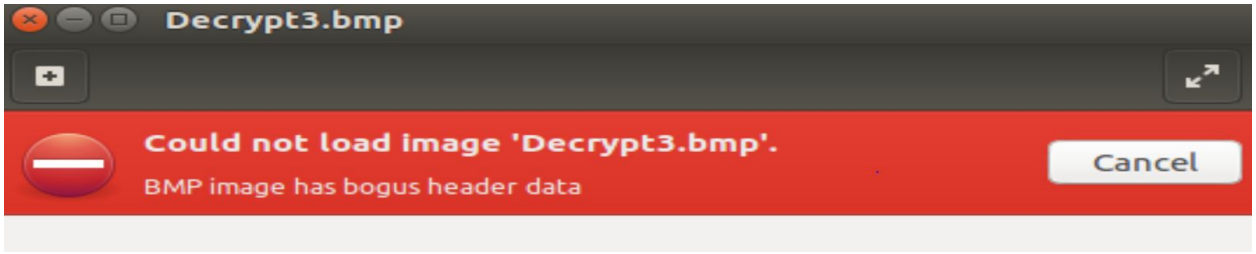
Original Pic:



Now, the encrypted pictures cannot be viewed because the header information is "damaged" by the encryption process.

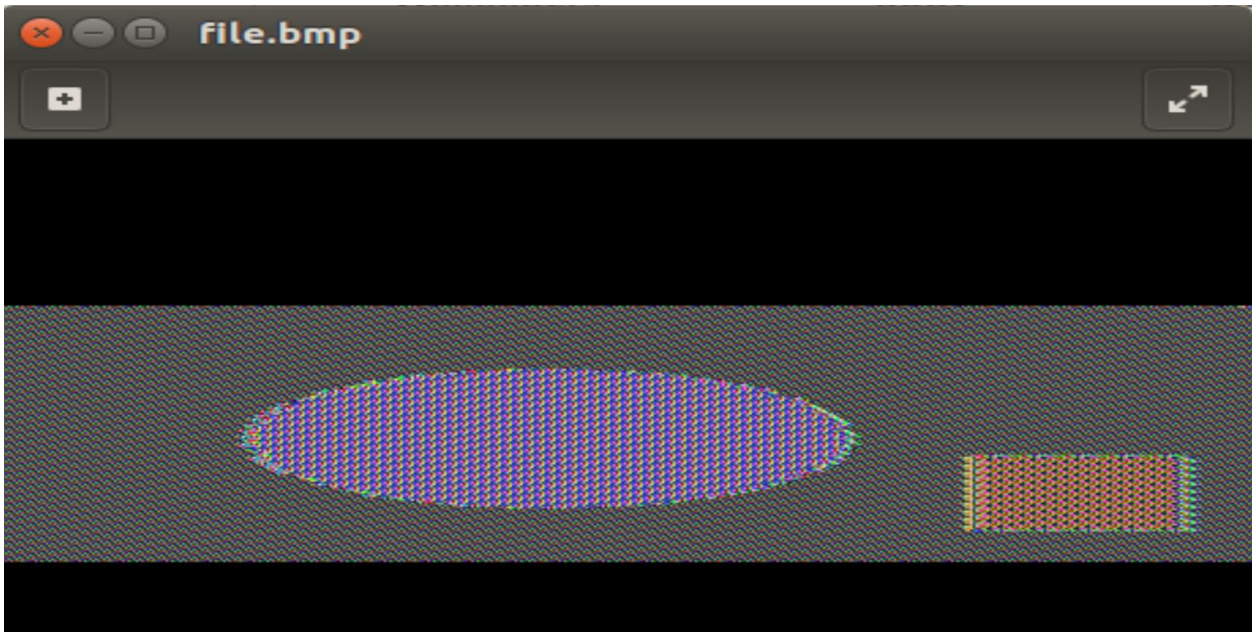
Using ECB:

```
[09/10/19]seed@VM:~/Desktop$ cd Crypto\ lab\ secret\ encryption/
[09/10/19]seed@VM:~/.../Crypto lab secret encryption$ ls
Crypto lab secret encryption  __MACOSX
[09/10/19]seed@VM:~/.../Crypto lab secret encryption$ cd Crypto\
ab\ secret\ encryption/
[09/10/19]seed@VM:~/.../Crypto lab secret encryption$ openssl enc
-aes-128-ecb -e -in pic_original.bmp -out cipher.bmp -k 0011 -iv
102
warning: iv not use by this cipher
[09/10/19]seed@VM:~/.../Crypto lab secret encryption$ openssl enc
-aes-128-ecb -e -in pic_original.bmp -out cipher.bmp -k 0011
[09/10/19]seed@VM:~/.../Crypto lab secret encryption$ ls
cipher.bmp              lowercase.txt          pic_original.bmp
ciphertext.txt          nnew_ciphertext.txt    plaintxt.txt
Crypto_Encryption.pdf   out_1.txt              words.txt
encrypt_1.txt           out.txt
[09/10/19]seed@VM:~/.../Crypto lab secret encryption$
```



Using the commands to combine header and data into a new file:

```
[09/10/19]seed@VM:~/Pictures$ head -c 54 pic_original.b
mp > header
[09/10/19]seed@VM:~/Pictures$ tail -c +55 Decrypt3.bmp
> body
[09/10/19]seed@VM:~/Pictures$ cat header body > file.bm
p
[09/10/19]seed@VM:~/Pictures$
```



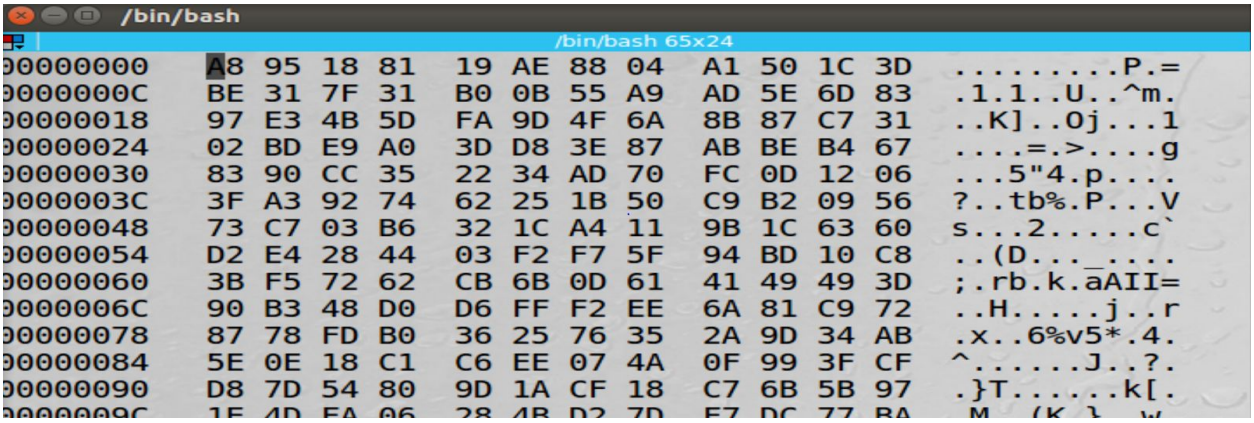
Using CBC:

```
[09/10/19]seed@VM:~/Pictures$ openssl enc -aes-128-cbc  -e -in pic
_original.bmp -out Decrypt4.bmp -K 0011
iv undefined
```

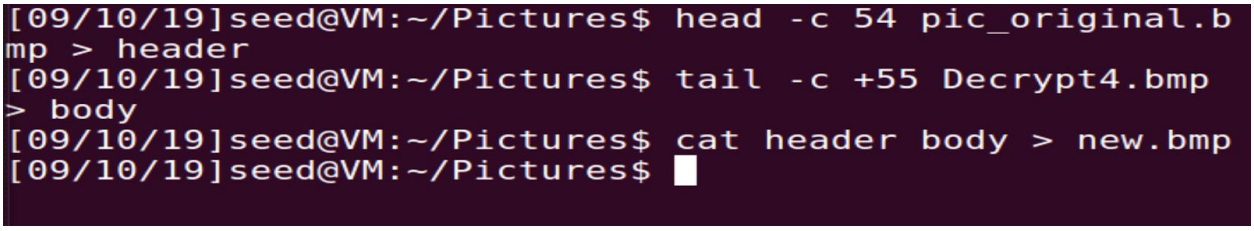
IV undefined**

```
[09/10/19]seed@VM:~/Pictures$ openssl enc -aes-128-cbc  -e -in pic
_original.bmp -out Decrypt4.bmp -K 0011 -iv 0102
[09/10/19]seed@VM:~/Pictures$
```

Header of Encrypted data changing through Hexedit:

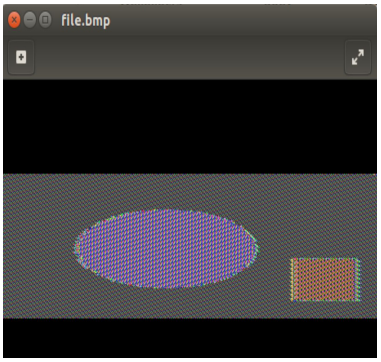


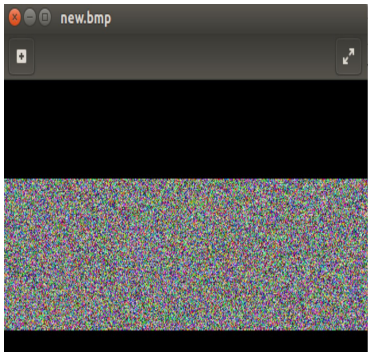
Using the commands to combine header and data into a new file:



CBC encrypted mode data:

Result:





Unencrypted Data **ECB MODE** **CBC MODE**

Encrypted Data **Encrypted Data**

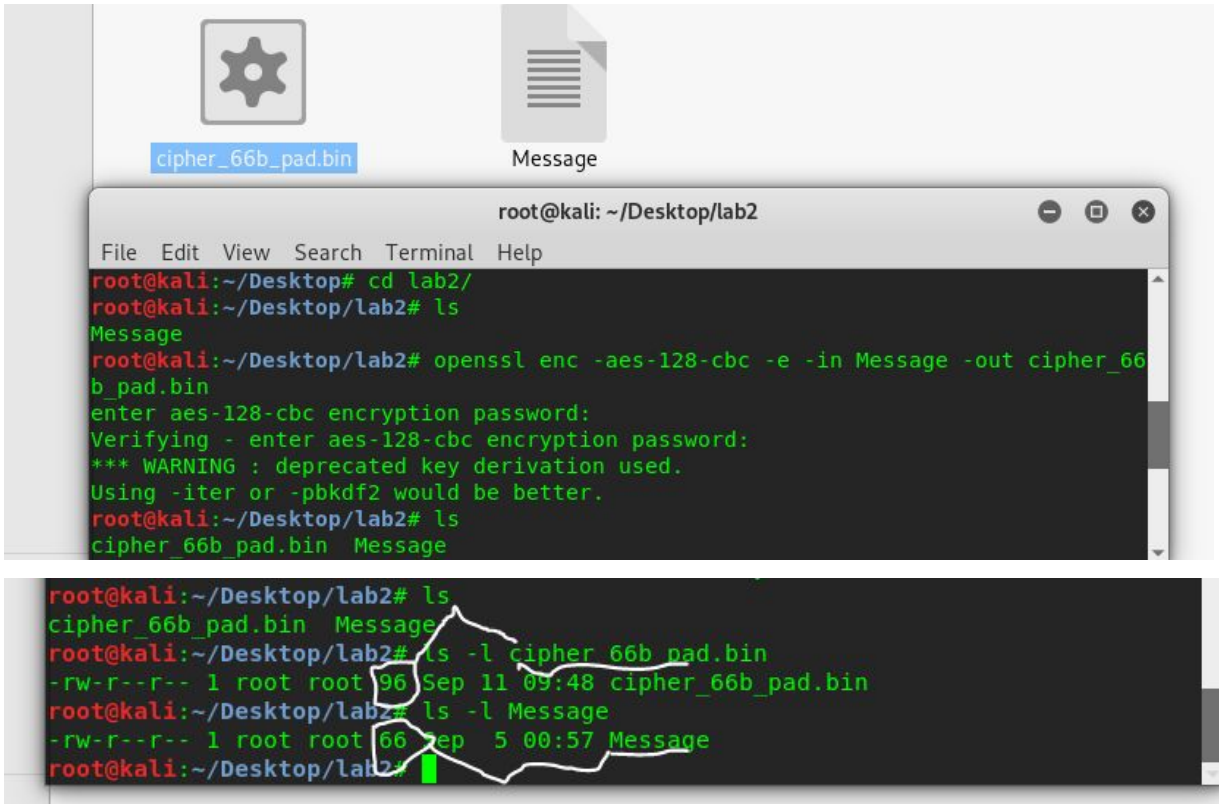
The CBC encrypted data mode is more complex than the ECB. The property of plain text is shown in the ciphertext. Hemce, by analyzing the pattern of ECB mode encrypted files, we can read the original file easily.

Task-4: Padding

Emranul

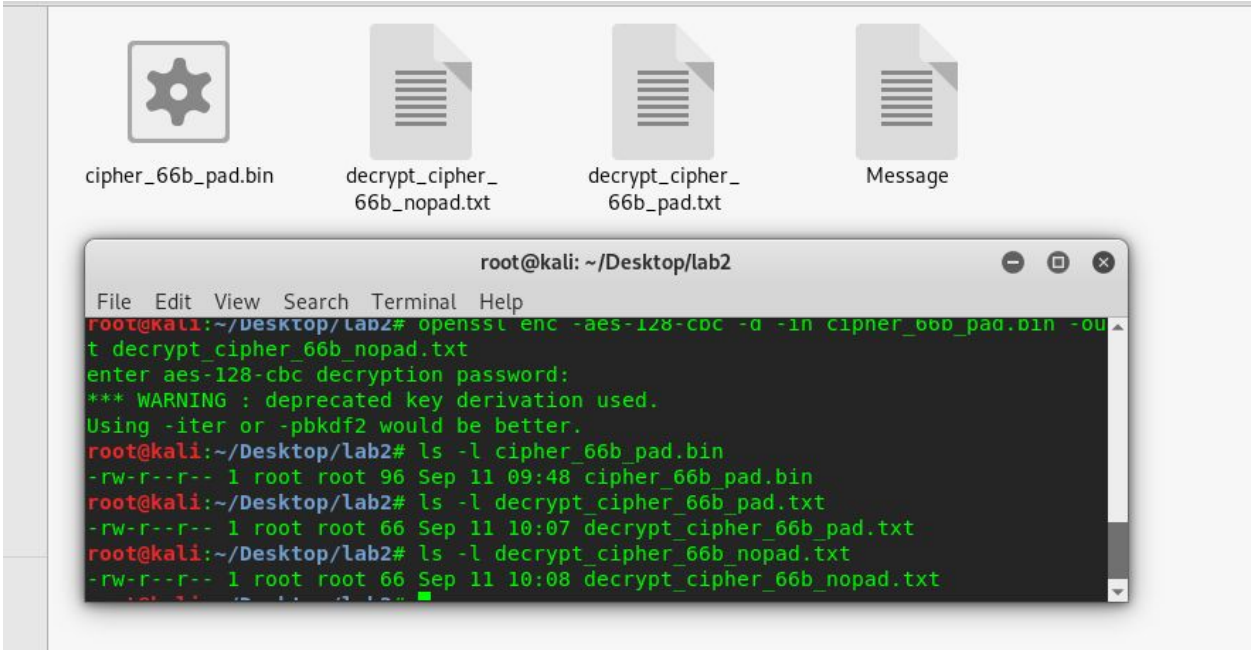
Step 1:Encrypt in CBC mode:

First we take the simple text file name Message whose size is 66 byte
When we encrypt in CBC mode as cipher_66b_pad.bin the file size changed to 96 Bytes. Which shows that 30 bytes padding is added in encrypted file.



Step 2:

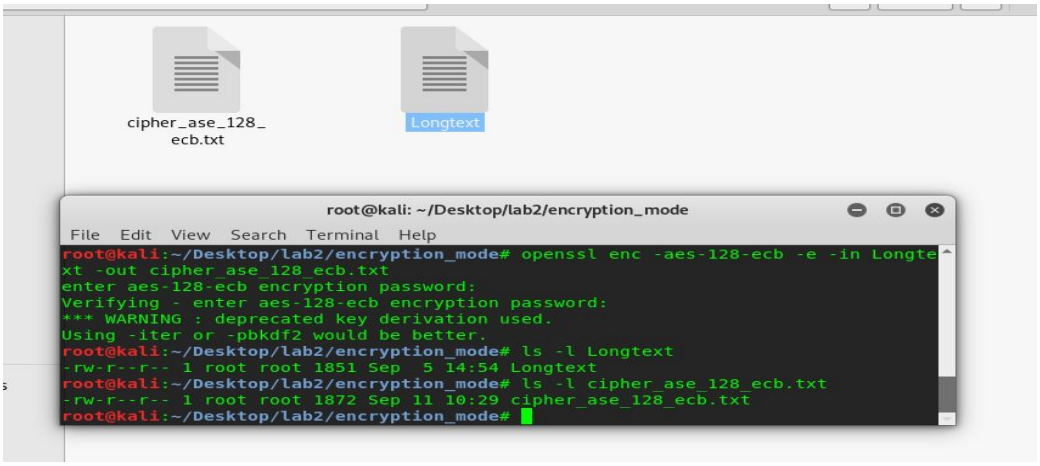
The cipher text has been decrypted twice using two different options to see the padding size difference which should be 30bytes and 30bytes from the cipher and decrypted files.



Step3:Encryption Mode and padding

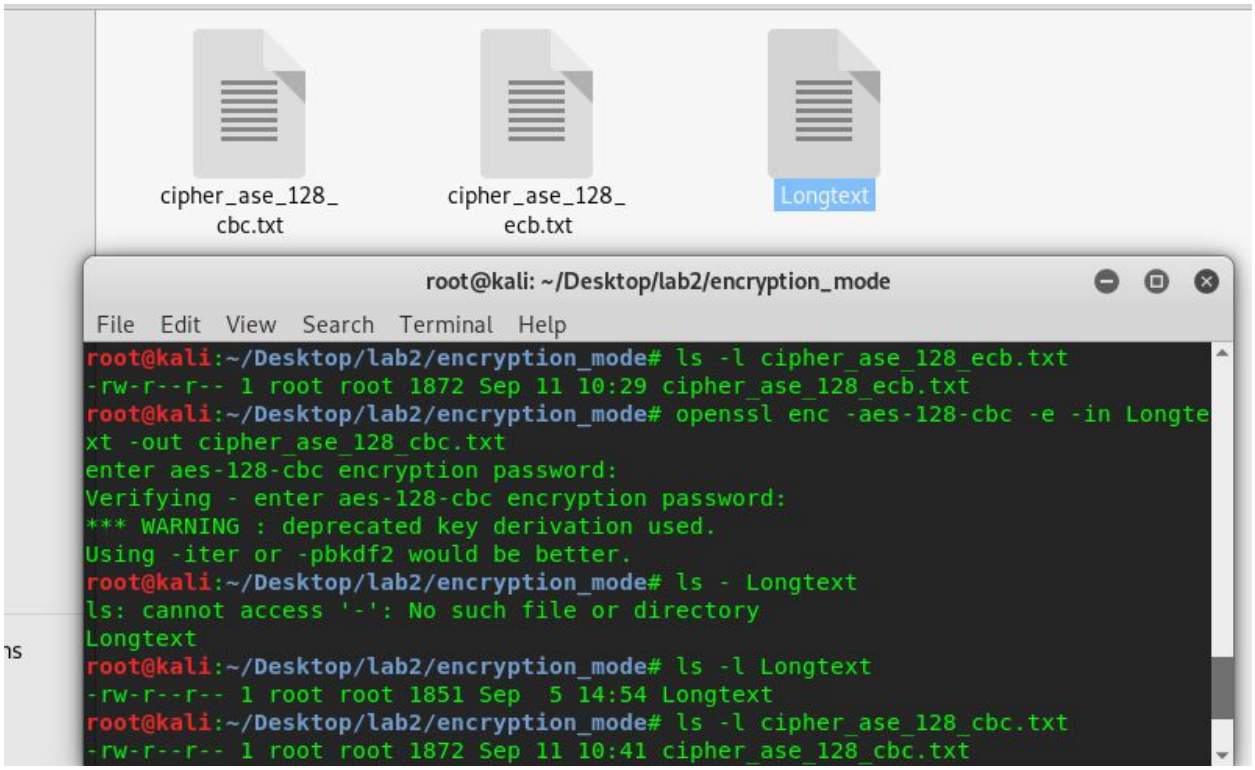
In order to know which mode required the padding i have taken a simple plaintext file of size 1851 bytes and the file is encrypted in different mode showing the file size. Which show us the padding size added to the original file in different mode.

EBC MODE:

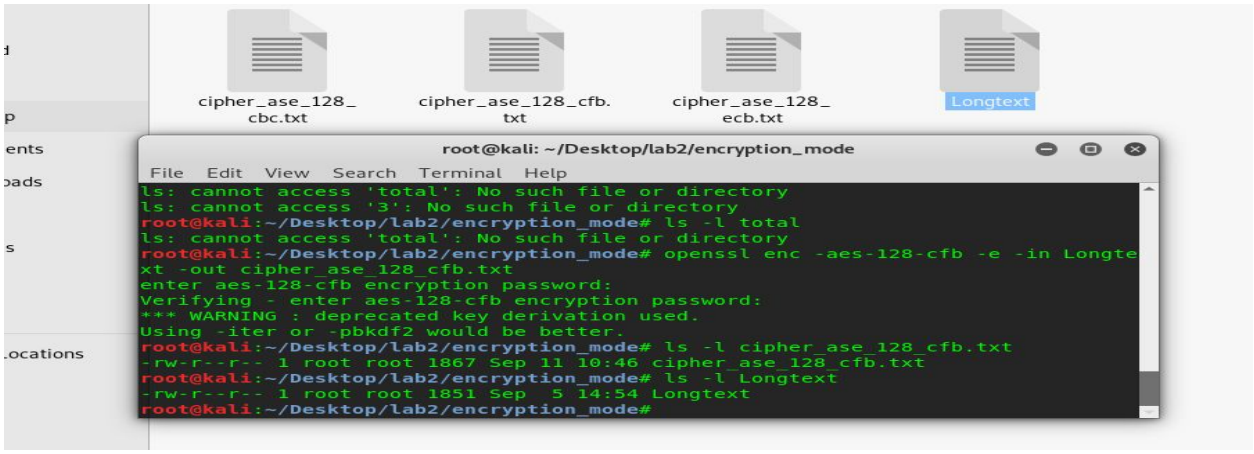


In this mode 1872bytes-1851bytes =21 bytes padding has been added on the encrypted file**.**

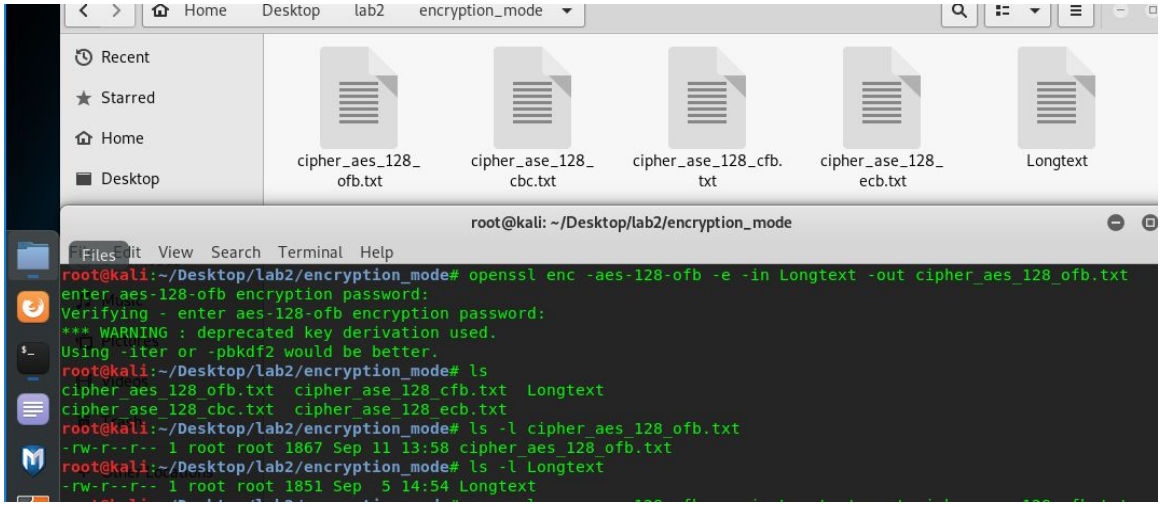
CBC MODE:



CFB MODE:



OFB Mode:



Task5: ErrorPropagation–CorruptedCipherText
(Priya and Lakpa)

ECB MODE:

Original text file size: 1220 Bytes ; Newpia.txt

Gwalior was the winter capital of the state of Madhya Bharat which later became a part of the larger state of Madhya Pradesh. Before Indian independence on 15 August 1947, Gwalior remained a princely state of the British Raj with Scindias as the local rulers. High rocky hills surround the city from all sides, on the north it just forms the border of the Ganga- Yamuna Drainage Basin. The city however is situated in the valley between the hills. Gwalior's metropolitan area includes Lashkar Gwalior (Lashkar Subcity), Morar Gwalior (Morar Subcity), Thatipur and the city centre. Gwalior was one of the major sites of rebellion

```
[09/11/19]seed@VM:~/Pictures$  openssl enc -aes-128-ecb -e -in New
pia.txt  -out cipher.txt -K 00112233445566778889aabbccddeeff
[09/11/19]seed@VM:~/Pictures$  openssl enc -aes-128-ecb -e -in New
pia.txt  -out ECBDecrypt.txt -K 00112233445566778889aabbccddeeff
[09/11/19]seed@VM:~/Pictures$ openssl enc -aes-128-ecb -e -in Newp
ia.txt  -out ECBDecrypt.txt -K 0011
[09/12/19]seed@VM:~/Pictures$ openssl enc -aes-128-ecb -d -in ECBD
ecrypt.txt  -out CorruptedEBC.txt -K 0011
[09/12/19]seed@VM:~/Pictures$ openssl enc -aes-128-ecb -d -in ECBD
ecrypt.txt  -out CorruptedEBC.txt -K 0011
```

Original file 55th bytes:

```
00000000 47 77 61 6C 69 6F 72 20 77 61 73 20 74 68 65 20 Gwalior was the
00000010 77 69 6E 74 65 72 20 63 61 70 69 74 61 6C 20 6F winter capital o
00000020 66 20 74 68 65 20 73 74 61 74 65 20 6F 66 20 4D f the state of M
00000030 61 64 68 79 61 20 42 68 61 72 61 74 20 77 68 69 adhya Bharat whi
```

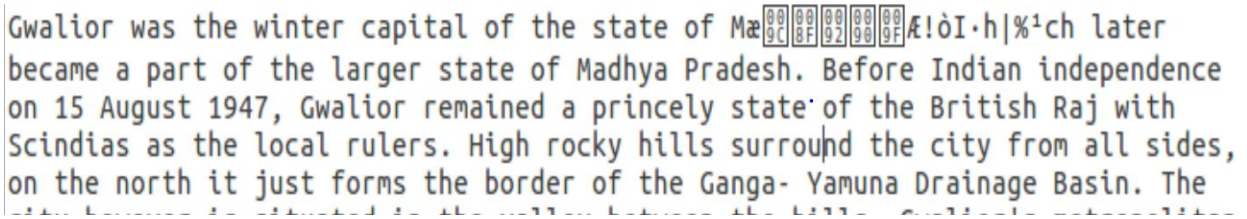
55th Bytes of encrypted file:

```
00000000 64 53 12 10 4E 58 2B 76 BD 15 69 D4 C6 C9 7F 76 dS..NX+v..i....v
00000010 D9 23 41 09 C8 10 9C A3 C1 75 9A E4 9C 82 E0 75 .#A......u.....u
00000020 62 48 31 68 B1 86 99 0E 01 E3 16 84 ED 0A 19 8F bH1h............
00000030 E5 2A 97 0B 76 9C 30 5A C9 AD BE 06 12 82 E4 CD .*..v.0Z........
```

Corrupting the file:

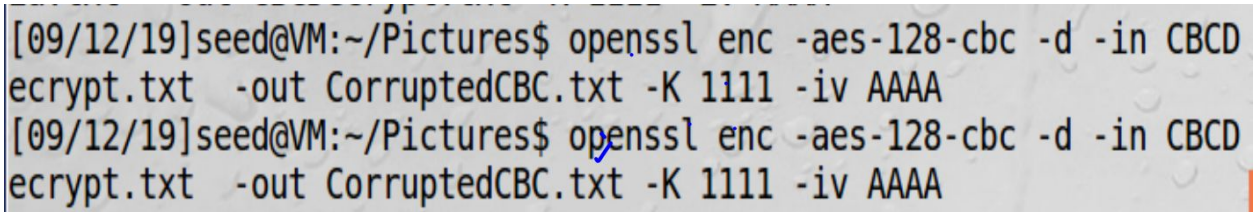
```
00000000 64 53 12 10 4E 58 2B 76 BD 15 69 D4 C6 C9 7F 76 dS..NX+v..i....v
00000010 D9 23 41 09 C8 10 9C A3 C1 75 9A E4 9C 82 E0 75 .#A......u.....u
00000020 62 48 31 68 B1 86 99 0E 01 E3 16 84 ED 0A 19 8F bH1h............
00000030 E5 2A 97 0B 76 9C FA 5A C9 AD BE 06 12 82 E4 CD .*..v..Z........
00000040 C1 9A 7A 34 CA C7 34 A1 02 70 69 D0 3E CF 59 BF ..z4..4..pi.>.Y.
```

Decrypting the Corrupted file:

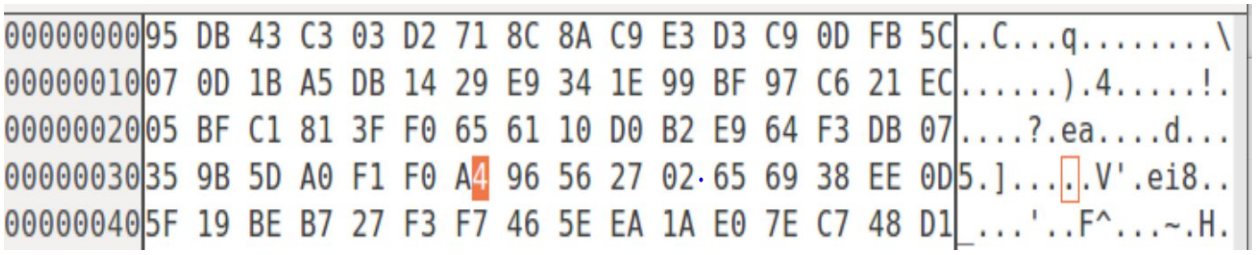


CBC MODE:

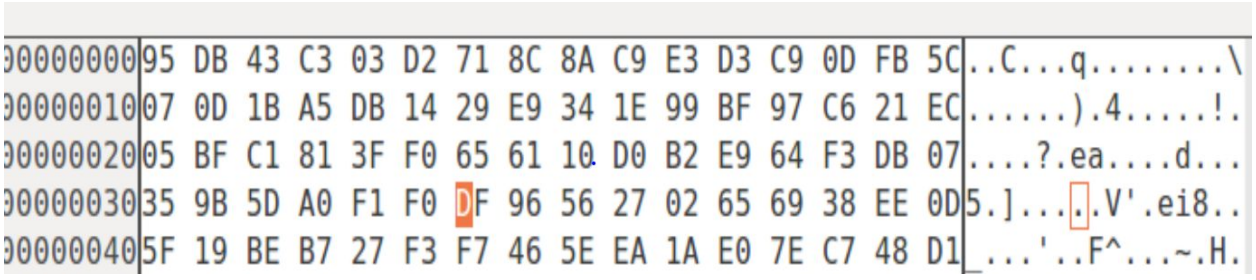
I used the same original file Newpia.txt.



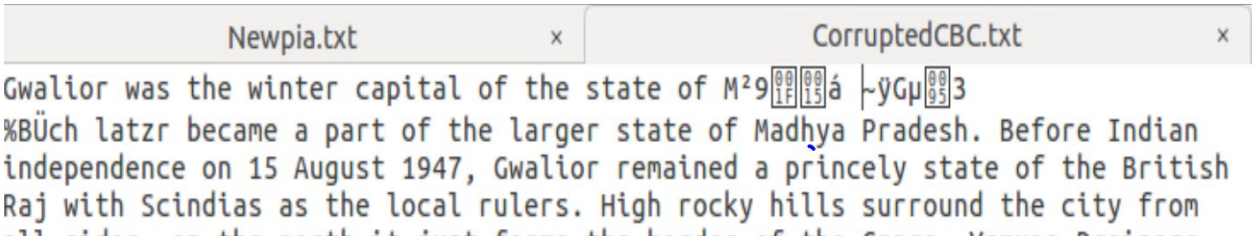
Encrypted file:



Corrupted File: Changed A4 to DF



Decrypting Corrupted file:



OFB MODE:

I used the same original file Newpia.txt.

```
[09/12/19]seed@VM:~/Pictures$ openssl enc -aes-128-ofb -e -in Newp
ia.txt  -out OFBDecrypt.txt -K 1001 -iv 0102AAAA
[09/12/19]seed@VM:~/Pictures$ openssl enc -aes-128-ofb -d -in OFBD
ecrypt.txt  -out CorruptedOFB.txt -K 1001 -iv 0102AAAA
```

Encrypted file:

```
00000000 92 62 30 C4 D3 8F B7 10 49 ED 4F 34 78 2D 01 CE |.b0.....I.O4x-..|
00000010 38 EF 6F 12 04 44 0D 04 12 75 7F B2 AD A9 81 EB |8.o..D...u......|
00000020 E7 8A CA E8 7E 52 D6 A2 6A B8 15 53 3C 8F 0E 13 |....~R..j..S<...|
00000030 94 8B 83 4B 25 9A B2 DF 1D 70 A8 C0 0E C3 A4 9A |...K%....p......|
```

Corrupted File: Changed B2 to AC

```
00000000 92 62 30 C4 D3 8F B7 10 49 ED 4F 34 78 2D 01 CE |.b0.....I.O4x-..|
00000010 38 EF 6F 12 04 44 0D 04 12 75 7F B2 AD A9 81 EB |8.o..D...u......|
00000020 E7 8A CA E8 7E 52 D6 A2 6A B8 15 53 3C 8F 0E 13 |....~R..j..S<...|
00000030 94 8B 83 4B 25 9A AC DF 1D 70 A8 C0 0E C3 A4 9A |...K%....p......|
00000040 AB 36 11 7C 82 6C F1 57 1B D2 AF D6 4A D0 8B 22 |.6.|.l.W....J.."|
```

Decrypting Corrupted file:

```
Gwalior was the winter capital of the state of Madhya Bharat which later became
a part of the larger state of Madhya Pradesh. Before Indian independence on 15
August 1947, Gwalior remained a princely state of the British Raj with Scindias
as the local rulers. High rocky hills surround the city from all sides, on the
north it just forms the border of the Ganga- Yamuna Drainage Basin. The city
```

Result:

From the details, we can conclude that OFB mode is the best way to retrieve the most of the data correctly because if one single 55th byte is corrupted then in the plain text only that corrupted byte cannot be retrieved.
In CBC, the effects seems to have in two blocks exactly and the data cannot be fully retrieved. The first block is used to build the second block.
In EBC,the ciphertext is decrypted one block at a time.The same ciphertext is used for the same block. Still some properties of cipher text is reflected in plain recovered text and hence we can analyze the cipher text and guess it.

Task6: InitialVector(IV)
Kenneth

The goal for this task was to use a different IV simply because when the plaintext is encrypted the ciphertext looks identical. It is important to avoid identical ciphertexts because this allows for information leaks.

```
[09/11/19]seed@VM:~/Desktop$ xxd -g 1 same_iv1.txt
00000000: 93 91 16 f6 ec f1 4f 98 45 2a 8c 7e 3d 13 00 c8  ......O.E*.~=...
00000010: 95 90 d2 41 53 0d 9c 2a 6c 67 81 e3 9a 1a 96 c5  ...AS..*lg......
[09/11/19]seed@VM:~/Desktop$ xxd -g 1 same_iv2.txt
00000000: 93 91 16 f6 ec f1 4f 98 45 2a 8c 7e 3d 13 00 c8  ......O.E*.~=...
00000010: 95 90 d2 41 53 0d 9c 2a 6c 67 81 e3 9a 1a 96 c5  ...AS..*lg......
[09/11/19]seed@VM:~/Desktop$
```

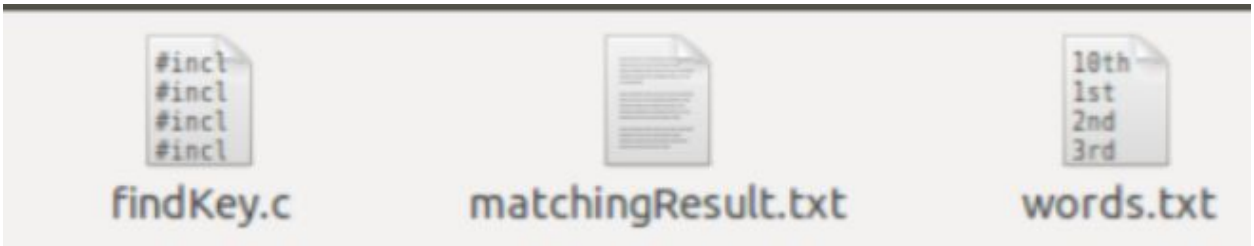
```
[09/11/19]seed@VM:~/Desktop$ xxd -g 1 same_iv1.txt
00000000: 93 91 16 f6 ec f1 4f 98 45 2a 8c 7e 3d 13 00 c8  ......O.E*.~=...
00000010: 95 90 d2 41 53 0d 9c 2a 6c 67 81 e3 9a 1a 96 c5  ...AS..*lg......
[09/11/19]seed@VM:~/Desktop$ xxd -g 1 different_iv.txt
00000000: 9d 74 71 f7 38 8d cf ce c4 b7 2e 9a 5a 97 4c 2a  .tq.8.......Z.L*
00000010: 2e aa 88 8d 7a 63 52 8a 53 9b 9e b9 7c ce 22 05  ....zcR.S...|.".
[09/11/19]seed@VM:~/Desktop$
```

Task7: ProgrammingusingtheCryptoLibrary

Kenneth

The goal of this task is to find the right key that was used to encrypt the given plain text, “This is a top secret.”.Given (refers to the lab description above):

- Cipher was used is AES-128-CBC
- Numbers in the IV are all zeros
- Key that used an English word shorter than 16 characters
- Cipher text that resulted from the encryption from the plain text is8d20e5056a8d24d0462ce74e4904c1b513e10d1df4a2ef2ad4540fae1ca0aaf9Main condition:
  IF (ChipherText = Encryption(Plain text, Key))
  Key is match;
  ELSE
  KEY is no match;



I have three files:

- findKey.c is a c file that contains c code using the OpenSSL crypto library API EVP
- matchingResult.txt is a text file contains result from matching
- words.txt is a text file contains a English word list; it is given by the lab at

Here is the code in fileKey.c; the images are ordered by the ordered of the code. I also uploaded code files to sakai lab1_crypto folder. There are comments in the codes that help understand well.

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

```c
void pad(char *s, int length);
int print_result(unsigned char *buf, char *s, int len, FILE *outFile, char *match);
int strcicmp(char const *a, char const *b);
```

```c
int main()
{
  unsigned char match[] = "MATCH";
  unsigned char noMATCH[] = "NO MATCH";
  int i;
  char words[16], t; // each word in the dictionary
  FILE *key, *outFile;
  unsigned char outbuf[1024 + EVP_MAX_BLOCK_LENGTH];
  unsigned char iv[16] = {0};   // given in the description of task 5
  int outlen, tmplen;
  int num;

  EVP_CIPHER_CTX ctx;
  EVP_CIPHER_CTX_init(&ctx);
  char inText[] = "This is a top secret.";      // given in the description of task 5
  char cipherTextGiven[] = "8d20e5056a8d24d0462ce74e4904c1b513e10d1df4a2ef2ad4540fae1ca0aaf9"; // given in the description of task 5
  key = fopen("words.txt", "r"); // given from the http://www.cis.syr.edu/~wedu/seed/Labs_12.04/Crypto/Crypto_Encryption/
  if( remove("matchingResult.txt") == -1 ) {    // matchingResult.txt is the file contains the result of each key
    perror("Error deleting file");
  }
  outFile = fopen("matchingResult.txt", "a+");
  if( key < 0 || outFile < 0 )
    {
      perror ("Cannot open file");
      exit(1);
    }

  char pbuffer[1024];
```

```
  //int count;
  while ( fgets(words, 16, key) ) // get each word from dictionary that suppose to be a key to encrypt
    {
      i=strlen(words);
      words[i-1]='\0';  // in the text editor it automatically add null or end of file so we need to remove that
      i=strlen(words);
      if (i < 16){ // 16 becuase we use AES-128
        // Since the word has less than 16 characters (i.e. 128 bits), space characters (hexadecimal value 0x 20)
        // are appended to the end of the word to form a key of 128 bits
        pad(words, (16));
      }
      EVP_EncryptInit_ex(&ctx, EVP_aes_128_cbc(), NULL, words, iv);
      if(!EVP_EncryptUpdate(&ctx, outbuf, &outlen, inText, strlen(inText)))
        {
          EVP_CIPHER_CTX_cleanup(&ctx);
          return 0;
        }
      if(!EVP_EncryptFinal_ex(&ctx, outbuf + outlen, &tmplen))

        {
          EVP_CIPHER_CTX_cleanup(&ctx);
          return 0;
        }
      outlen += tmplen;

      int i;
      char* buf_str = (char*) malloc (2*outlen + 1);
      char* buf_ptr = buf_str;
      for (i = 0; i < outlen; i++)
      {
        buf_ptr += sprintf(buf_ptr, "%02X", outbuf[i]);
      }
      *(buf_ptr + 1) = '\0';
      if (strcicmp(cipherTextGiven, buf_str) == 0)
        print_result(outbuf, words, outlen, outFile, match);
      else

          print_result(outbuf, words, outlen, outFile, noMATCH);
      }
  fclose(key);
  fclose(outFile);
  return 1;
}
```

```c
// print result to out put file matchResult.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);
}
```

```c
// 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 space */
    l++;
  }
  s[l]= '\0'; /* strings need to be terminated in a null */
}
```

```c
//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;
    }
}
```

In the code above, I read word line by line from the words.txt and do the encryption then using the main condition to test if the key is match. Then I save the results to the file matchResult.txt.

RSA Public-Key Encryption and Signature Lab

3.1 Task 1: Deriving the PrivateKey

Emran, Priya, Lakpa

We know,
Private key is derived by:
$d \equiv e^{-1} \pmod{\lambda(n)}$;
Lambda n= (p-1) (Q-1)

```
// Initialize a, b, n ,e
BN_hex2bn (&p,"F7E75FDC469067FFDC4E847C51F452DF");
BN_hex2bn (&q,"E85CED54AF57E53E092113E62F436F4F");
BN_dec2bn (&p1,"1");
BN_dec2bn (&neg1,"-1");

BN_hex2bn (&e,"0D88C3");

// res = a*b
BN_sub(res1, p, p1);
BN_sub(res2, q, q1);
BN_mul(n, res1, res2, ctx);
//printBN("a * b =", n);
// res = aàb mod n
BN_mod_inverse (d, e, n, ctx);

//BN_mod_exp (d, e, neg1, n, ctx);

printBN("private key=", d);
return 0;
}
```

```
[09/13/19]seed@VM:~/Desktop$ a.out
private key= 460C4FA0AAA5E848E2B26E2DC2A7A878BFC7738407
FE7469303A7EA2BDB9B851
```

Result:

Private key d was calculated as p,q, and e were given.
As a result the private key is:
0x3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB

3.2 Task 2: Encrypting a Message:

```
BN_hex2bn (&m,"4120746F7020736563726574421");
BN_hex2bn(&a, "F7E75FDC469067FFDC4E847C51F452DF");
BN_hex2bn(&b, "E85CED54AF57E53E092113E62F436F4F");

BN_hex2bn (&e,"0D88C3");

// res = a*b

BN_mul(n, a, b, ctx);
printBN("A*B= ", n);

BN_mod_exp(c,m,e, n, ctx);
printBN("cipher= ", c);
```

```
[09/13/19]seed@VM:~/Downloads$ ls
a.out   Newpia.txt   sample.c
[09/13/19]seed@VM:~/Downloads$ a.out
A*B=   E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD
0AA1968DBB143D1
cipher=   90A81343DFE08415EDF79337CDE00457BAB56AFFA1B0CE
5647BF9025665B396A
```

Decrypting a Message:

```
// Initialize a, b, n ,e
BN_hex2bn
(&d,"460C4FA0AAA5E848E2B26E2DC2A7A878BFC7738407FE7469303A7EA2BDB9B851");
BN_hex2bn(&c,"90A81343DFE08415EDF79337CDE00457BAB56AFFA1B0CE5647BF9025665B396A")
BN_hex2bn(&a, "F7E75FDC469067FFDC4E847C51F452DF");
BN_hex2bn(&b, "E85CED54AF57E53E092113E62F436F4F");


//BN_hex2bn (&e,"0D88C3");

// res = a*b

BN_mul(n, a, b, ctx);
//printBN("A*B= ", n);

BN_mod_exp (z,c,d, n, ctx);
printBN("Decrypt= ", z);

return 0;
}
```

C ▾ Tab Width: 8 ▾ Ln 35, Col 80 ▾ INS

```
[09/13/19]seed@VM:~/Downloads$ a.out
Decrypt=  76597BF60845F4640B6BEB1ACF8C677E8F000E340DB54
125032AC275FCA197CB
```

Encoding back to ASCII:

```
[09/13/19]seed@VM:~/Downloads$ python -c 'print("412074
6f702073656372657421".decode("hex"))'
A top secret!
```

Result:

The goal for this task was to use a command in order to get a hex string: python-c' print.
As a result the hex string is: 4120746F702073656372657421

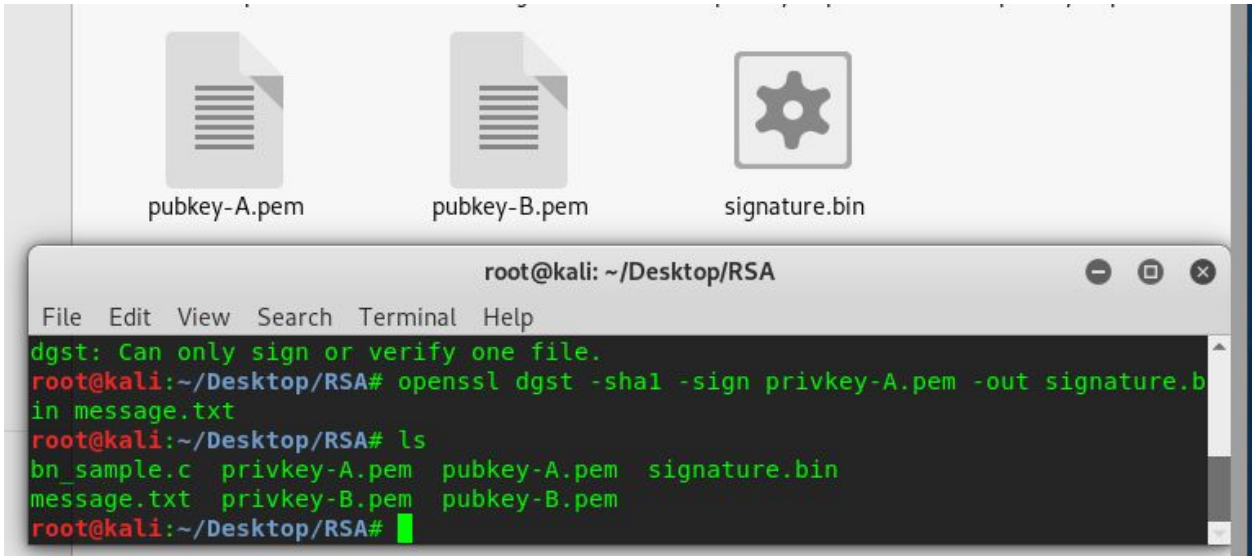
We derived a private key, encrypted plain text and then decrypted back too.

Attempts:

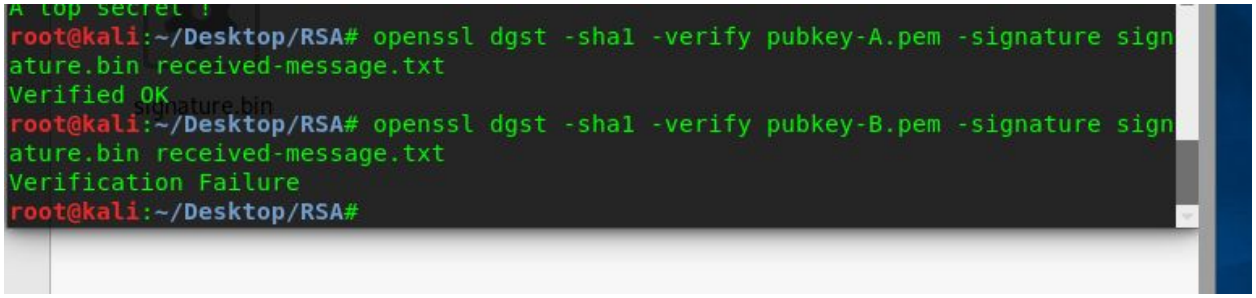
```
[09/13/19]seed@VM:~/Downloads$ python -c 'print("76597B
F60845F4640B6BEB1ACF8C677E8F000E340DB54125032AC275FCA19
7CB".decode("hex"))'
vY{E?d
?A%??*?u????~?4
```

```
[09/12/19]seed@VM:~/.../RSA$ cat privkey-A.pem | less
[09/12/19]seed@VM:~/.../RSA$ cat pubkey-A.pem | less
[09/12/19]seed@VM:~/.../RSA$ nano message.txt
[09/12/19]seed@VM:~/.../RSA$ cat message.txt
A top secret.
[09/12/19]seed@VM:~/.../RSA$ openssl dgst -sha1 message.txt
SHA1(message.txt)= e546b9a71be05583deaff5164cb77925ad4d35b9
[09/12/19]seed@VM:~/.../RSA$
```

3.4 Task 4: Signing a Message:



3.5 Task 5:Verifying a Signature:


A top secret !
root@kali:~/Desktop/RSA# openssl dgst -sha1 -verify pubkey-A.pem -signature sign
ature.bin received-message.txt
Verified OK
root@kali:~/Desktop/RSA# openssl dgst -sha1 -verify pubkey-B.pem -signature sign
ature.bin received-message.txt
Verification Failure
root@kali:~/Desktop/RSA#