Computer Network Security

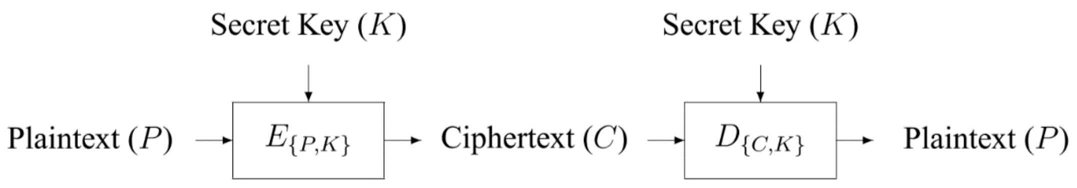
Project I: Implementation of Data Encryption Algorithm.

Student Names:

Jigar Makwana (01711370)
Sreevathsav Dr (01705880)

Date: 03/7/2018

INTRODUCTION:



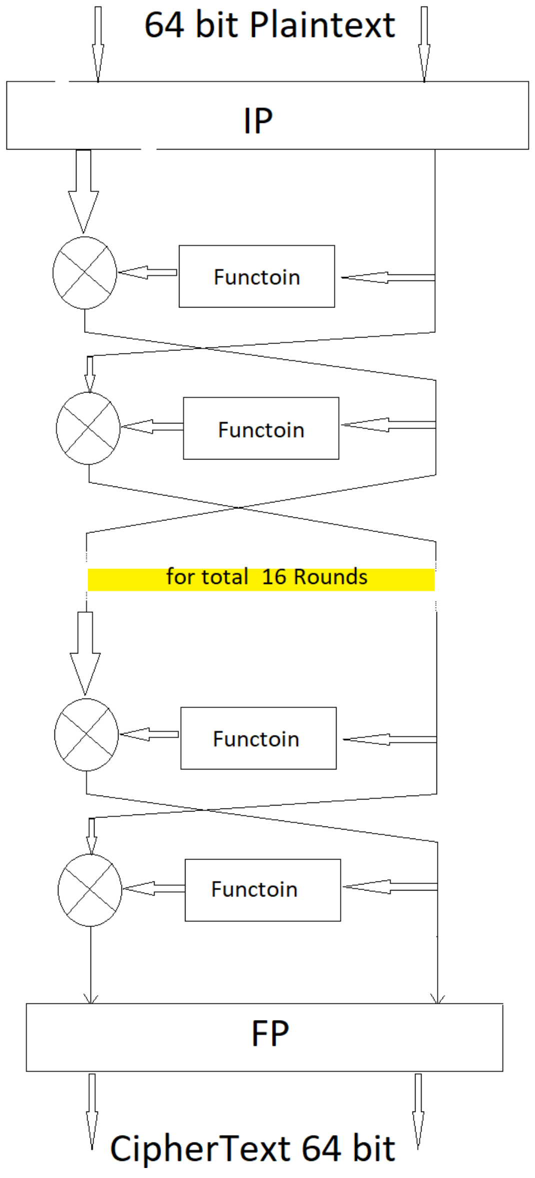
Typical Encryption

DES

DES is a 64-bit block cipher algorithm. It encrypts data 64 bits at a time. It follows Feistel Structure. It consists of several (16) rounds where each round contains bit-shuffling, non-linear substitutions (S-boxes) and exclusive OR operations.

The way the plaintext is accepted, and the key arrangement used for encryption and decryption, both determine the type of cipher it is. DES is therefore a symmetric, 64-bit block cipher as it uses the same key for both encryption and decryption and only operates on 64-bit blocks of data at a time. The key size used is 56 bits, however a 64-bit (or eight-byte) key is the input. The least significant bit of each byte is either used for parity (odd) or set arbitrarily and does not increase the security in any way. All blocks are numbered from left to right which makes the eight bit of each byte the parity bit.

Decryption is the inverse of encryption, but the keys are used in the reverse order. DES uses 64-bit key, out of which 8-bits are used for parity check therefore limiting the length of the key used to 56-bits.



Data Permutation:

DES performs permutation on data twice during encryption that initial and final permutation in the beginning and end of the algorithm, which makes the implementation of the DES algorithm software more efficient.

Initial Permutation (IP)	Final Permutation (IP-1)
58 50 42 34 26 18 10 2	40 8 48 16 56 24 64 32
60 52 44 36 28 20 12 4	39 7 47 15 55 23 63 31
62 54 46 38 30 22 14 6	38 6 46 14 54 22 62 30
64 56 48 40 32 24 16 8	37 5 45 13 53 21 61 29
57 49 41 33 25 17 9 1	36 4 44 12 52 20 60 28
59 51 43 35 27 19 11 3	35 3 43 11 51 19 59 27
61 53 45 37 29 21 13 5	34 2 42 10 50 18 58 26
63 55 47 39 31 23 15 7	33 1 41 9 49 17 57 25

The numbers in the above table specify the bit numbers of the input to the permutation. The order of the numbers in the table specify output bit position. The input of 64-bits is divided into 8 octets, the bits of the first octet is spread into 8th bits of each octet, in general the bits of ith octet gets spread into (9-i)th bits of all octet. This permutation has no security significance.

Expansion:

Expansion table for turns 32-bit data blocks into 48 bits so that its can be XORed with 48 bit key.

expansion_table

31, 0, 1, 2, 3, 4,
3, 4, 5, 6, 7, 8,
7, 8, 9, 10, 11, 12,
11, 12, 13, 14, 15, 16,
15, 16, 17, 18, 19, 20,
19, 20, 21, 22, 23, 24,
23, 24, 25, 26, 27, 28,
27, 28, 29, 30, 31, 0

S-box Permutation:

32-bit permutation function used on the output of the S-boxes

```
p = 15 6 19 20 28 11
    27 16 0 14 22 25
    4 17 30 9 1 7
    23 13 31 26 2 8
   18 12 29 5 21 10
    3 24
```

Generating Keys:

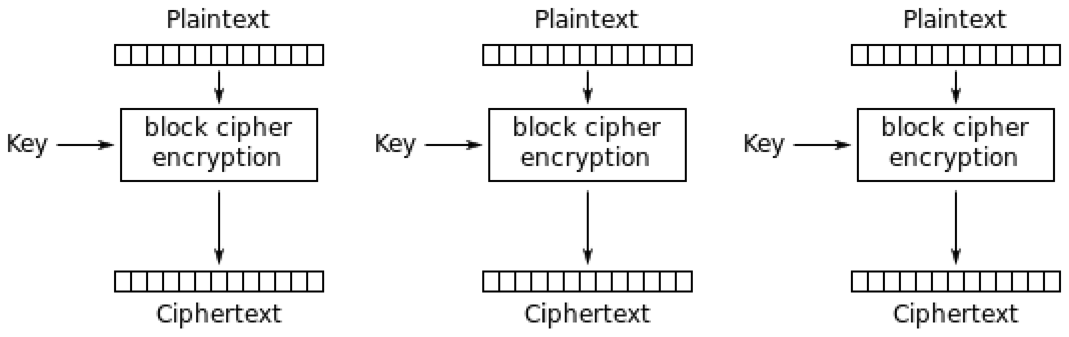
Keys are generated for each of the sixteen rounds, key in each round is 48-bits in length. The key is originally 64-bit in length, but 8 of the bits are parity bits which include 8,16,26,32,40,48,56,64 bit. After the initial permutation of removing the 8 parity bits the new length of key is 56 bits which is further divided into two halves namely c0 and d0. The sixteen keys are K1, K2…K16. The table below gives the removal of 8-bits as well as permutation of c0 and d0.

C0	D0
57 49 41 33 25 17 9	63 55 47 39 31 23 15
1 58 50 42 34 26 18	7 62 54 46 38 30 22
10 2 59 51 43 35 27	14 6 61 53 45 37 29
19 11 3 60 52 44 36	21 13 5 28 20 12 4

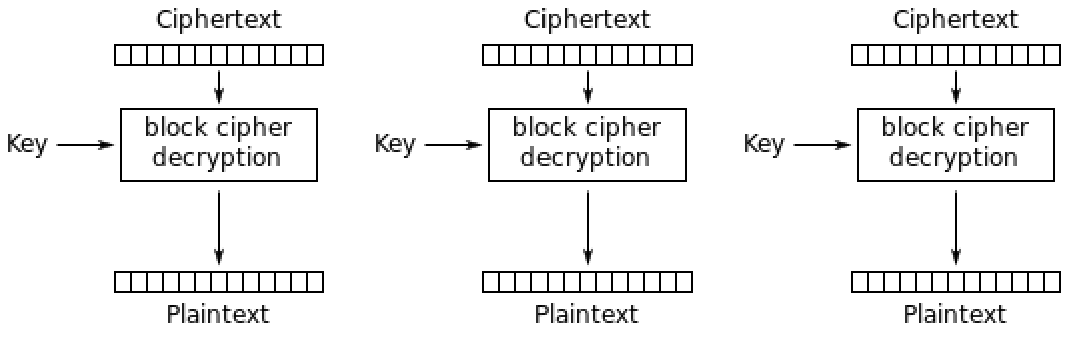
The permutated output gives Ci and Di in each round which gives the left half and right half of each of the key which is of 24-bit neglecting 9,18,22,25,35,38,43,54. Ci and Di are concatenated to give a 48-bit output. The obtained permutated bits are then further shuffled by performing shifts in each round of key generation. The bits undergo a single bit left shift in 1,2,9 and 16 rounds of key generation and 2-bit left shift in all other rounds. Therefore, Ki is 48-bit long.

Electronic Code Book (ECB):

In this mode, the message is divided into blocks of 64-bits each and the last block is padded to make it 64-bit. Each of the block is encrypted using the secret key, each block undergoes sixteen rounds of DES algorithm.



Electronic Codebook (ECB) mode encryption

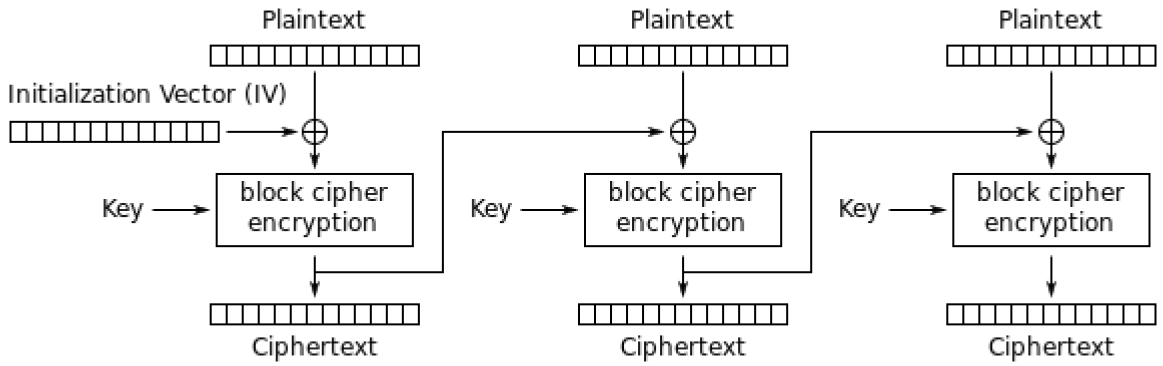


Electronic Codebook (ECB) mode decryption

During decryption, the ciphertext is again divided into 64-bit block and then it performs the similar action as encryption but uses the keys generated in reverse order.

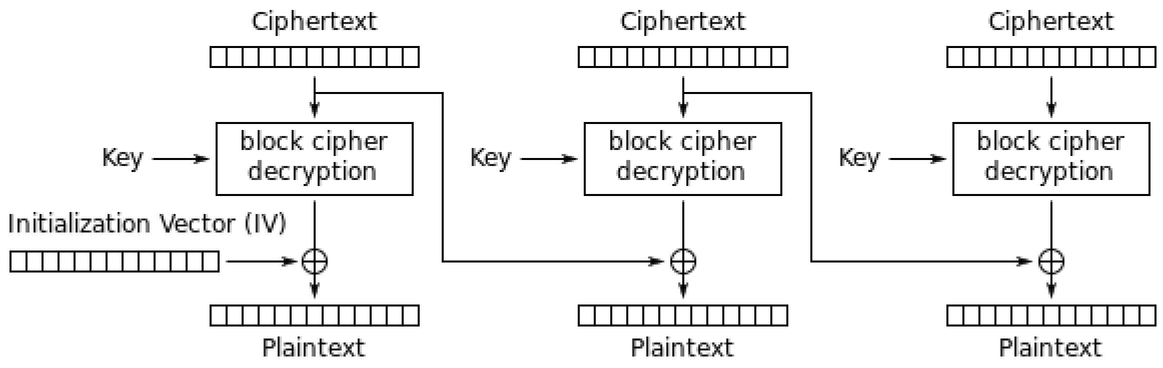
Cipher Block Chaining (CBC):

In this mode, a random number Ri is generated as Initialization Vector (IV) for each plaintext block Mi to be encrypted. In encryption initially, the message block is exored with the IV and then it is encrypted using DES algorithm, after encryption the obtained cipher text of each block is xored with the succeeding message block before applying DES algorithm. This is continued until all the blocks of the data are encrypted.



Cipher Block Chaining (CBC) mode encryption

Decryption in this mode is like encryption, here the cipher text is first subjected to DES decryption algorithm and then is XORed with Initialization Vector to obtain the plain text. Henceforth the first ciphertext block is used to XOR the second decrypted block to obtain the second message block and this is continued until the entire message is decrypted to obtain the plain text.



Cipher Block Chaining (CBC) mode decryption

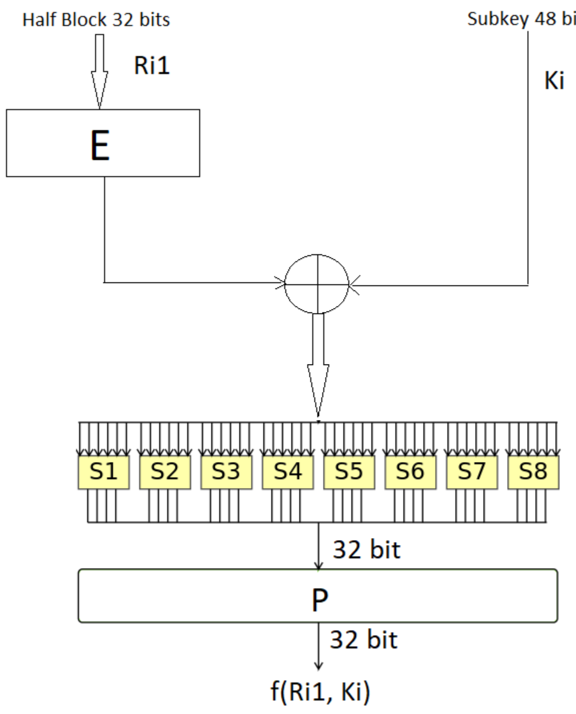
Algorithm:

Initial Permutation: The input block is first transposed under an initial permutation. The IP table is a public table. It is read from left-to-right, top-to-bottom.
16 Rounds: After the initial permutation, 16 rounds are performed on the permuted block. This involves a combination of substitutions and transpositions. The block is identified as two blocks of 32 bits each, Li and Ri. Then:

Li = Ri-1, and

Ri = Li-1 xor f(Ri-1, Ki).

(Where Ki is a series of 48-bit keys generated from K.)



Function f() and S-boxes: Initially, the first block say, Ri-1 is expanded to 48-bits using the bit-selection table. The bit-selection table is used in essentially the same way as the initial permutation table except that a few bits are chosen more that once as we have to expand from 32-bits to 48-bits. After expanding to 48 bits, the block is broken up into eight 6-bit blocks after calculating the exclusive OR of the expanded blocks and the key. Next, each 6-bit block is fed into the selection fuctions which return 4-bits. These bits are concatenated together to give 32 bits. A point to be noted is that in the last (i.e., 16th) round, an interchange does not take place.

That is:

R16 = L15 xor f(R15, K16) and it remains on the left side.

L16 = R15 and it remains on the right side.

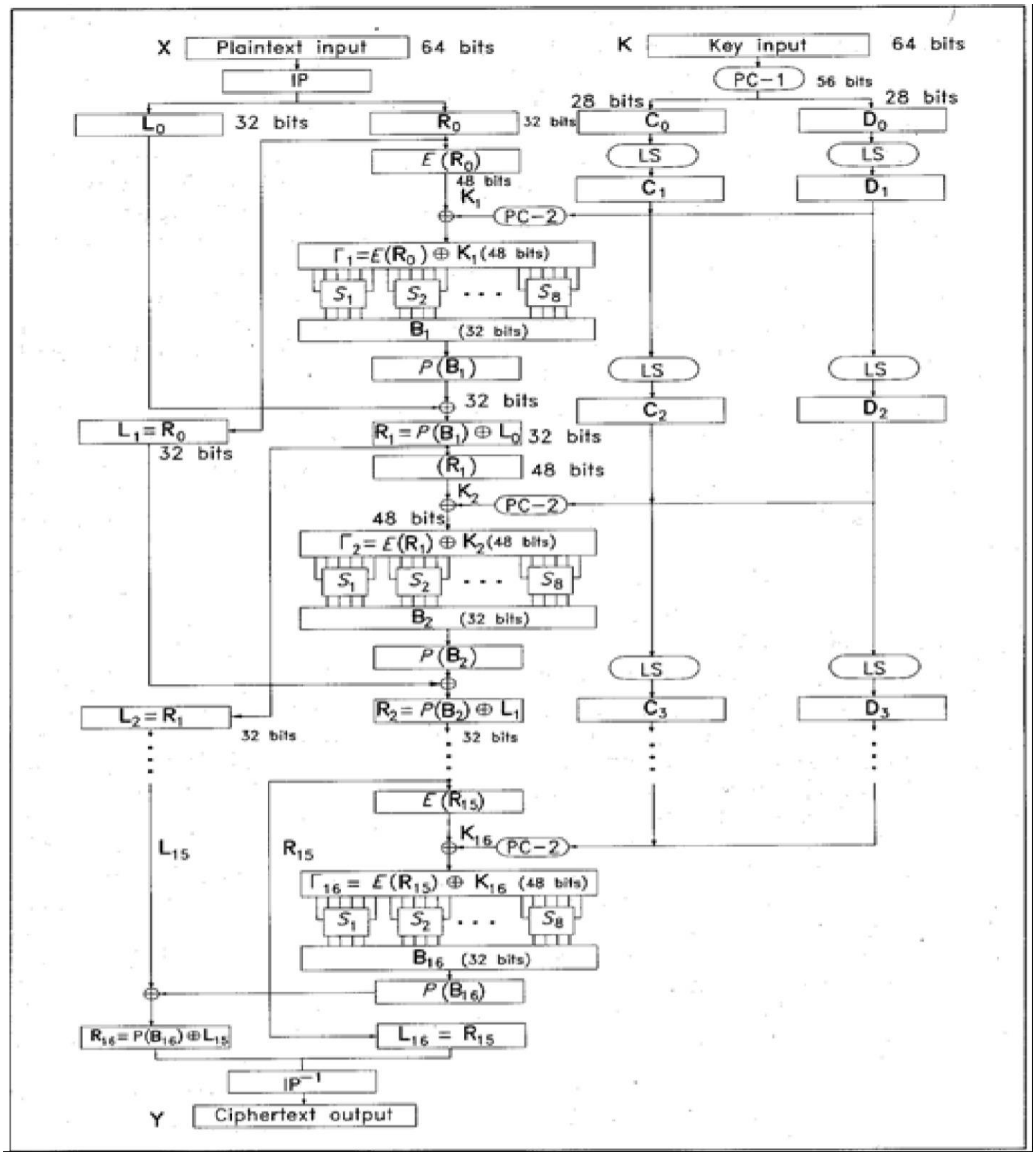
This is so because decryption takes place with the same algorithm with the keys being fed in, in the reverse order.

Key calculation: Each of the 16 iterations mentioned above uses a different 48-bit key derived from the initial 56-bit key K. K is input as a 64-bit block, with 8 parity bits in positions 8, 16, ..., 64. The parity bits are discarded using the permutation. It is then split into two halves of 28-bits each, say Ci and Di. The blocks Ci and Di are each successively shifted left to derive each key Ki.

Deciphering: Deciphering is performed using the same algorithm except that K16 is used in the first iteration, K15 is the second and so on. This is so because the final permutation is the inverse of the initial permutation. Note that initial and final permutation do not enhance the security of the DES cryptosystem, however to adhere to the standard the permutations cannot be omitted.

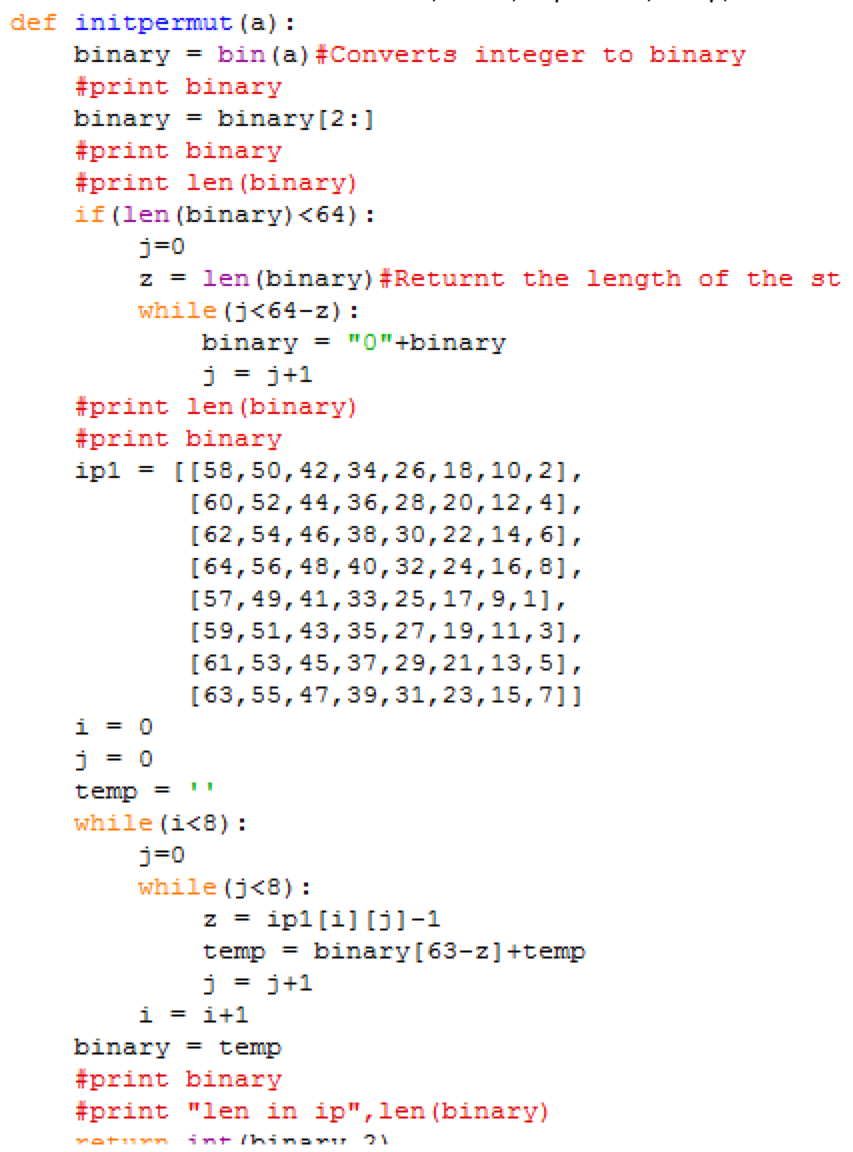
IMPLEMENTATION:

Program is written in **Python** language using Atom text editor and tested on **windows 7**. Code structure follows diagram below.



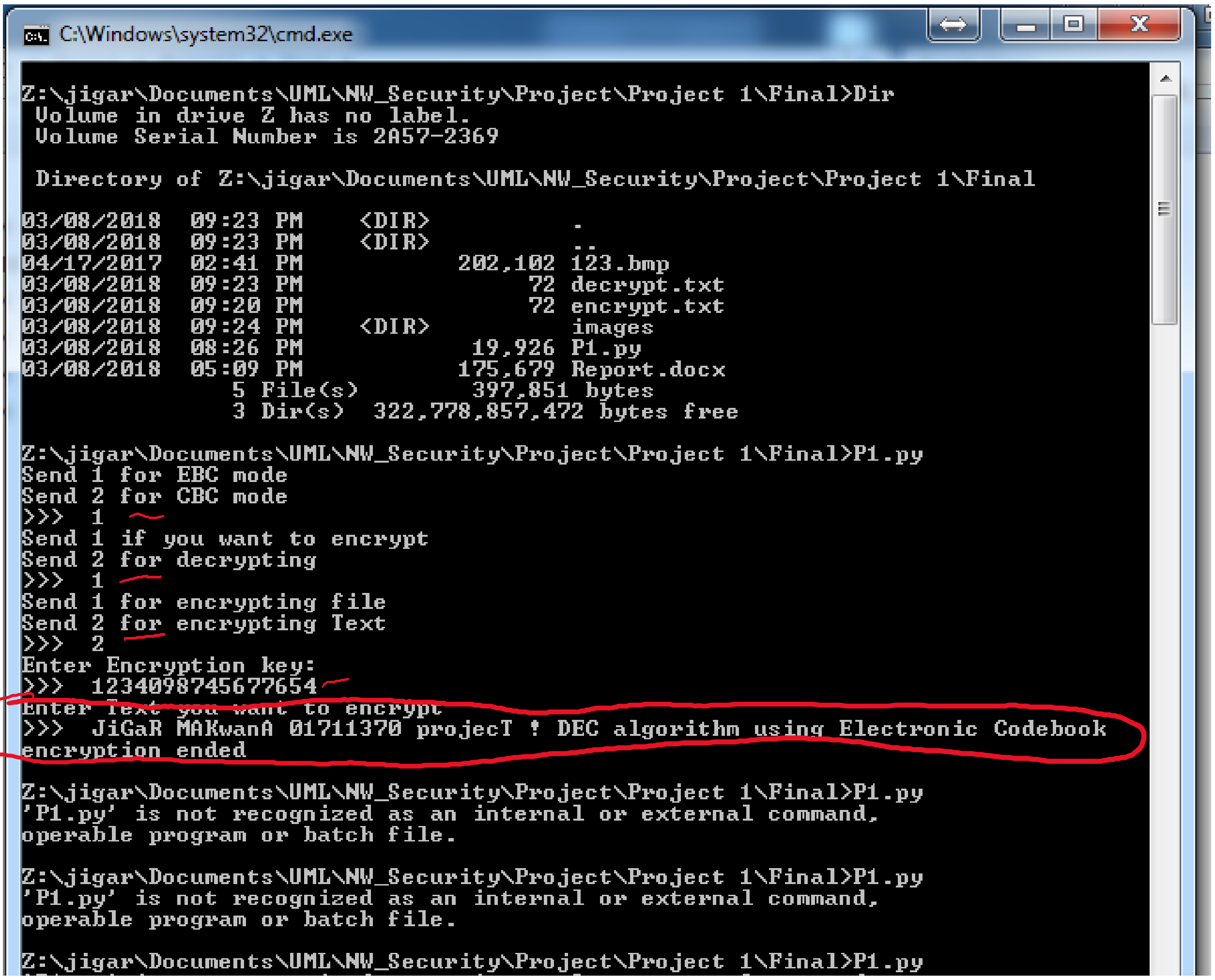
Two modes are used 1) Electronic Codebook and 2) Cipher Block Chaining.
Normal encryption and decryption of TEXT using command line works in both mode. However File encryption-decryption is having some issues sometimes.

Different functions like Permutation, S-box, expansion, swap, shift are used.



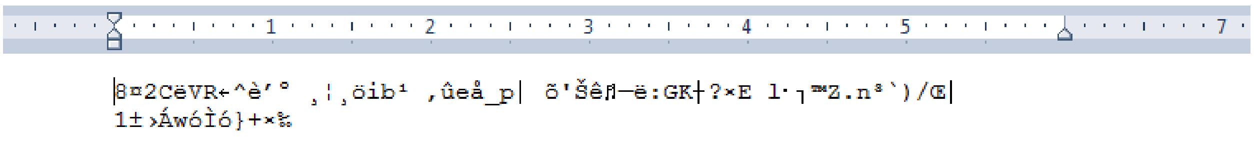
```
def initpermut(a):
    binary = bin(a)#Converts integer to binary
    #print binary
    binary = binary[2:]
    #print binary
    #print len(binary)
    if(len(binary)<64):
        j=0
        z = len(binary)#Returnt the length of the st
        while(j<64-z):
            binary = "0"+binary
            j = j+1
    #print len(binary)
    #print binary
    ip1 = [[58,50,42,34,26,18,10,2],
           [60,52,44,36,28,20,12,4],
           [62,54,46,38,30,22,14,6],
           [64,56,48,40,32,24,16,8],
           [57,49,41,33,25,17,9,1],
           [59,51,43,35,27,19,11,3],
           [61,53,45,37,29,21,13,5],
           [63,55,47,39,31,23,15,7]]
    i = 0
    j = 0
    temp = ''
    while(i<8):
        j=0
        while(j<8):
            z = ip1[i][j]-1
            temp = binary[63-z]+temp
            j = j+1
        i = i+1
    binary = temp
    #print binary
    #print "len in ip",len(binary)
    return int(binary,2)
```

Electronic Codebook mode is used in below figure.
When you run the code it asks you what you want all you need to remember is the key for ECB mode.



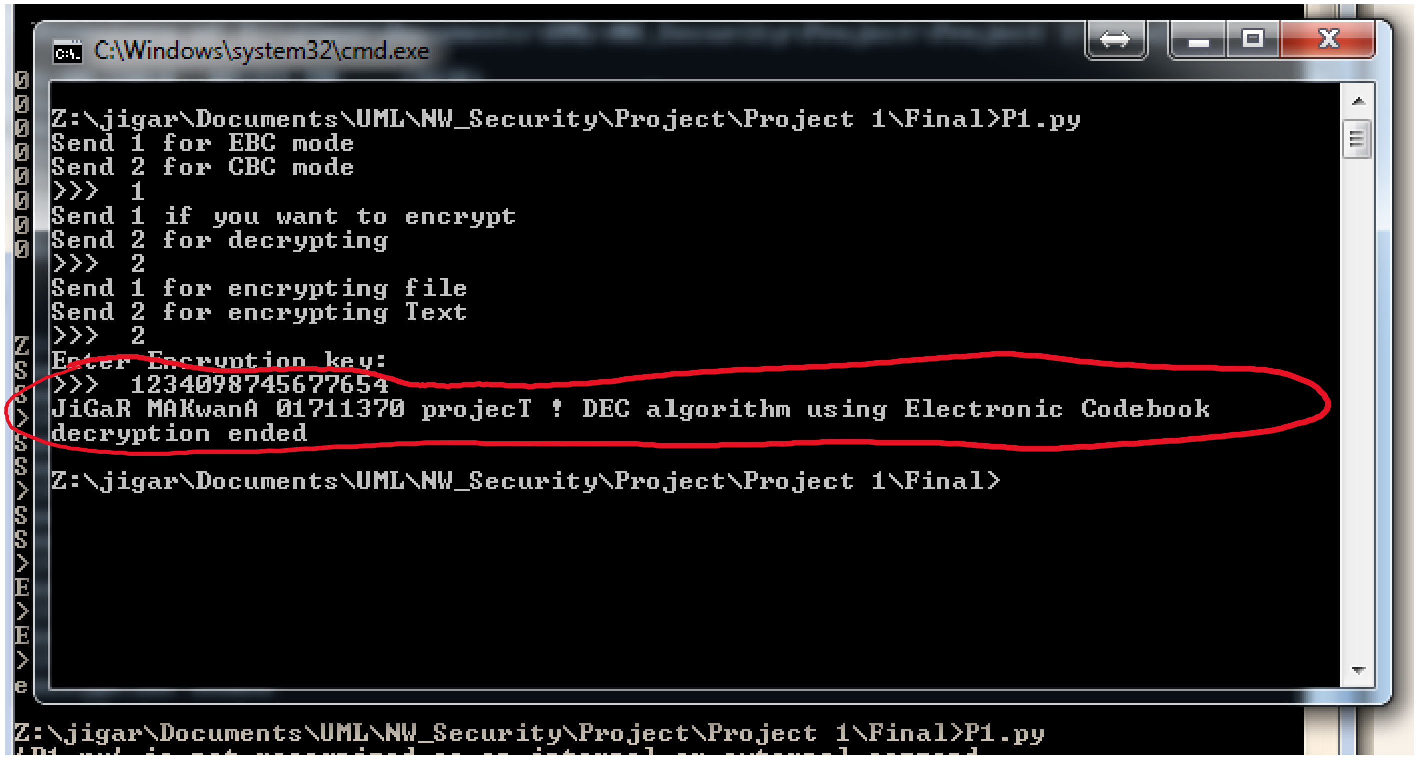
```
Z:\jigar\Documents\UML\NW_Security\Project\Project 1\Final>Dir
 Volume in drive Z has no label.
 Volume Serial Number is 2A57-2369

 Directory of Z:\jigar\Documents\UML\NW_Security\Project\Project 1\Final

03/08/2018  09:23 PM    <DIR>          .
03/08/2018  09:23 PM    <DIR>          ..
04/17/2017  02:41 PM           202,102 123.bmp
03/08/2018  09:23 PM                72 decrypt.txt
03/08/2018  09:20 PM                72 encrypt.txt
03/08/2018  09:24 PM    <DIR>          images
03/08/2018  08:26 PM            19,926 P1.py
03/08/2018  05:09 PM           175,679 Report.docx
               5 File(s)        397,851 bytes
               3 Dir(s)  322,778,857,472 bytes free

Z:\jigar\Documents\UML\NW_Security\Project\Project 1\Final>P1.py
Send 1 for EBC mode
Send 2 for CBC mode
>>>  1
Send 1 if you want to encrypt
Send 2 for decrypting
>>>  1
Send 1 for encrypting file
Send 2 for encrypting Text
>>>  2
Enter Encryption key:
>>>  1234098745677654
Enter Text you want to encrypt
>>>  JiGaR MAKwanA 01711370 projecT ! DEC algorithm using Electronic Codebook
encryption ended

Z:\jigar\Documents\UML\NW_Security\Project\Project 1\Final>P1.py
'P1.py' is not recognized as an internal or external command,
operable program or batch file.

Z:\jigar\Documents\UML\NW_Security\Project\Project 1\Final>P1.py
'P1.py' is not recognized as an internal or external command,
operable program or batch file.

Z:\jigar\Documents\UML\NW_Security\Project\Project 1\Final>P1.py
```

Below figure shows the encrypted text saved in encrypt.txt file which will be in same directory you ran P1.py code.



```
8¤2CëVR←^è′°  ,¦,öib¹ ,ûeå_p|  õ'Šê♫—ë:GK┼?×E l·┐™Z.nª`)/Œ
1±›ÁwóÌó}+×‰
```

Now for decryption use the same code but chose decryption and use the same key. You will see decrypted TEXT on screen and there will be other file in which it will be saved name decrypt.txt.



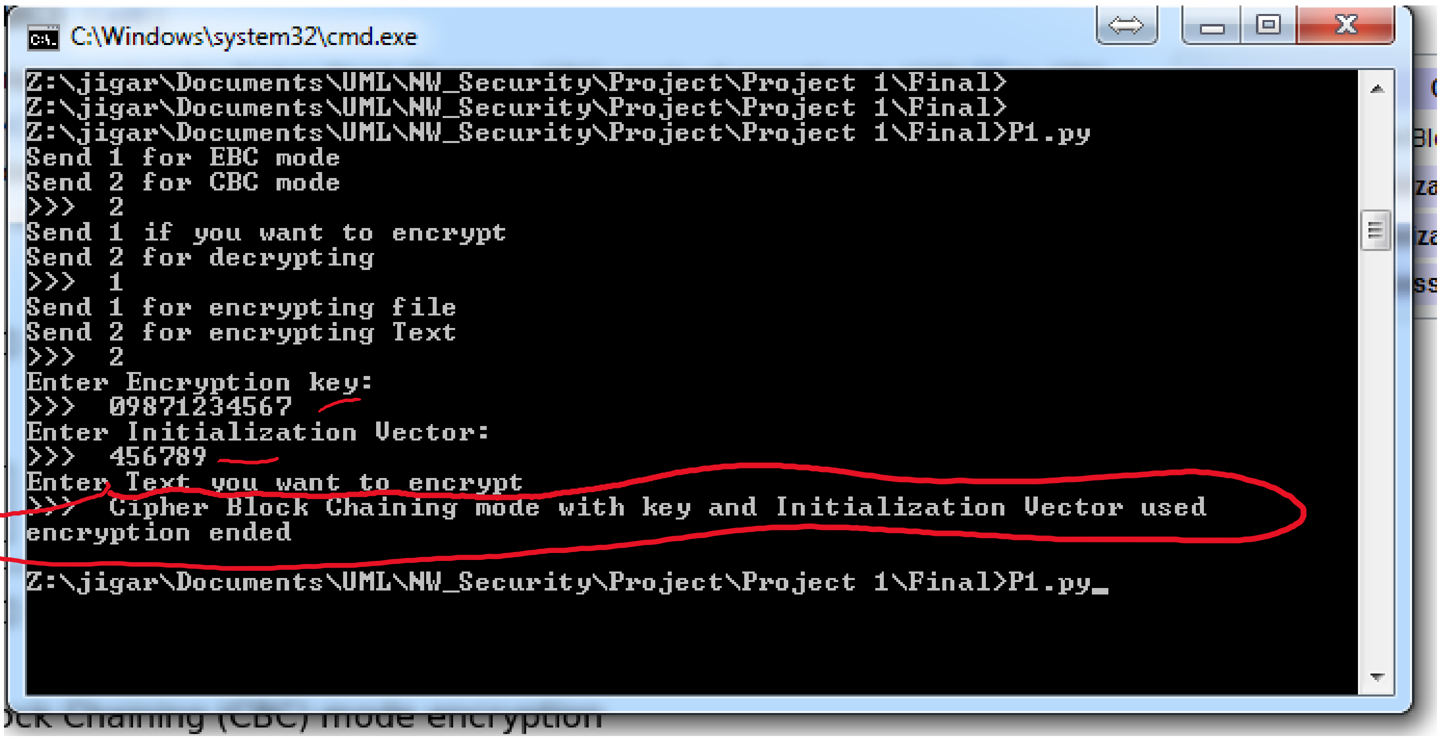
C:\Windows\system32\cmd.exe

```
Z:\jigar\Documents\UML\NW_Security\Project\Project 1\Final>P1.py
Send 1 for EBC mode
Send 2 for CBC mode
>>>  1
Send 1 if you want to encrypt
Send 2 for decrypting
>>>  2
Send 1 for encrypting file
Send 2 for encrypting Text
>>>  2
Enter Encryption key:
>>>  1234098745677654
JiGaR MAKwanA 01711370 projecT ! DEC algorithm using Electronic Codebook
decryption ended

Z:\jigar\Documents\UML\NW_Security\Project\Project 1\Final>
```

```
Z:\jigar\Documents\UML\NW_Security\Project\Project 1\Final>P1.py
```

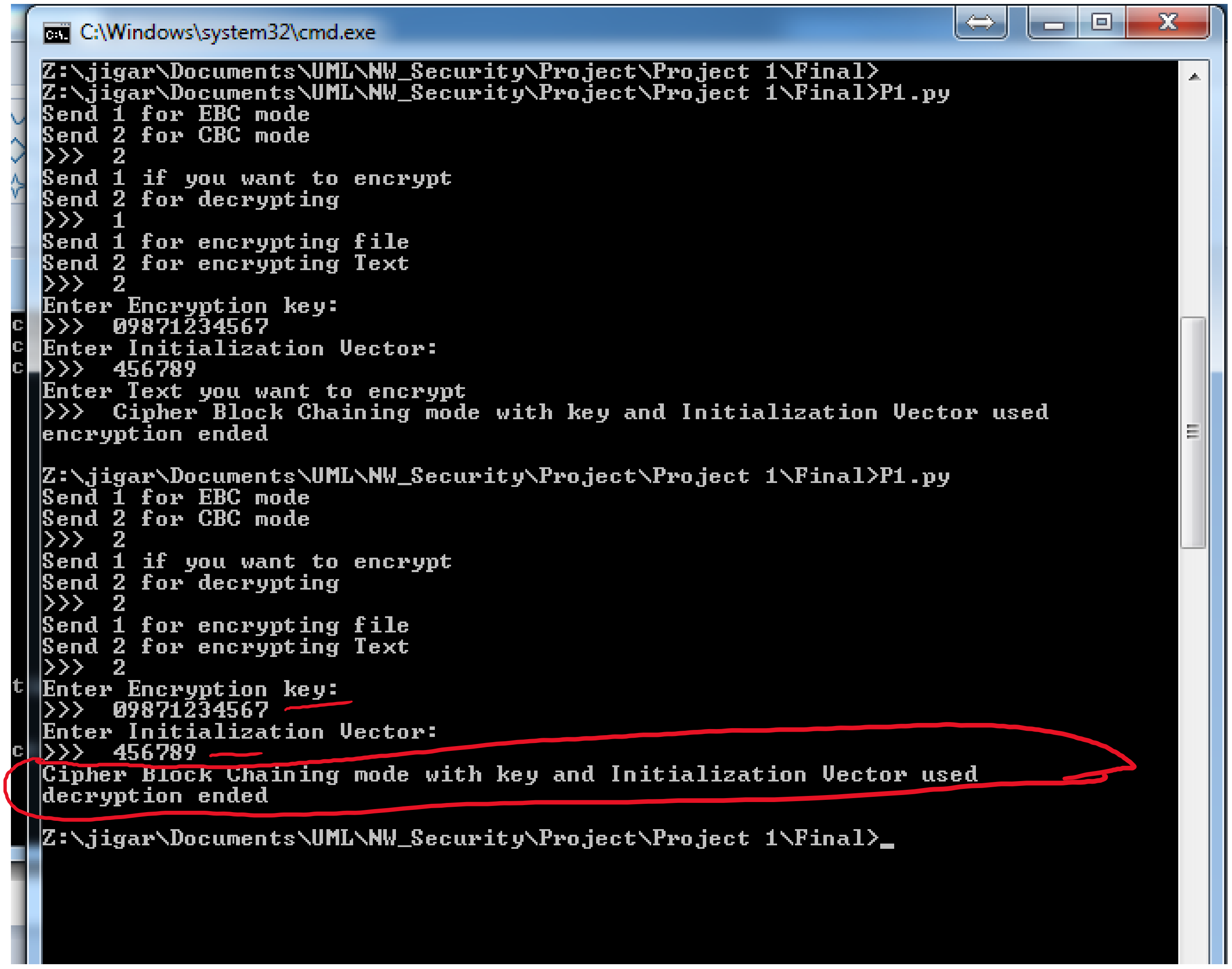
.

This is result of second mode we used Cipher Block Chaining.
Below figure show the encryption in which one key and Initialization Vector is asked for the encryption.
And after encryption is done message is send that it is been encrypted.



```
Z:\jigar\Documents\UML\NW_Security\Project\Project 1\Final>
Z:\jigar\Documents\UML\NW_Security\Project\Project 1\Final>
Z:\jigar\Documents\UML\NW_Security\Project\Project 1\Final>P1.py
Send 1 for EBC mode
Send 2 for CBC mode
>>>  2
Send 1 if you want to encrypt
Send 2 for decrypting
>>>  1
Send 1 for encrypting file
Send 2 for encrypting Text
>>>  2
Enter Encryption key:
>>>  09871234567
Enter Initialization Vector:
>>>  456789
Enter Text you want to encrypt
>>>  Cipher Block Chaining mode with key and Initialization Vector used
encryption ended

Z:\jigar\Documents\UML\NW_Security\Project\Project 1\Final>P1.py_
```

Below figure shows the decryption with CBC mode using same key and IV


C:\Windows\system32\cmd.exe

```
Z:\jigar\Documents\UML\NW_Security\Project\Project 1\Final>
Z:\jigar\Documents\UML\NW_Security\Project\Project 1\Final>P1.py
Send 1 for EBC mode
Send 2 for CBC mode
>>>  2
Send 1 if you want to encrypt
Send 2 for decrypting
>>>  1
Send 1 for encrypting file
Send 2 for encrypting Text
>>>  2
Enter Encryption key:
>>>  09871234567
Enter Initialization Vector:
>>>  456789
Enter Text you want to encrypt
>>>  Cipher Block Chaining mode with key and Initialization Vector used
encryption ended

Z:\jigar\Documents\UML\NW_Security\Project\Project 1\Final>P1.py
Send 1 for EBC mode
Send 2 for CBC mode
>>>  2
Send 1 if you want to encrypt
Send 2 for decrypting
>>>  2
Send 1 for encrypting file
Send 2 for encrypting Text
>>>  2
Enter Encryption key:
>>>  09871234567
Enter Initialization Vector:
>>>  456789
Cipher Block Chaining mode with key and Initialization Vector used
decryption ended

Z:\jigar\Documents\UML\NW_Security\Project\Project 1\Final>_
```

Conclusion:

These results conclude that we have been able to successfully implement Data Encryption Algorithm using Cipher Block Chaining mode and Electronic Codebook mode. The Code can handle any size texts. However, it needs improvement to handle files, sometimes it corrupts the file, but simple text never goes wrong. Simple text is encrypted and decrypted in less than a second.

References

[1] https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation
[2] http://page.math.tu-berlin.de/~kant/teaching/hess/krypto-ws2006/des.htm
[3] stackoverflow.com
[4] https://www.tutorialspoint.com/python
[5] Network security; Private communication in Public world by Charlie Kaufman
[6] https://docs.python.org