Assignment 3. DES

Crypto basics 1

Tables:

S_3

Row 0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
Row 1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
Row 2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
Row 3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4

Row 0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
Row 1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
Row 2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
Row 3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S3(001101) = *0110* (table column: 0110 = 6, row: 01 = 1) and S3(001001)=*0011* (column: 0100=4, row: 01=1)
Two inputs to an S-box differ in the two middle bits - true
Outputs differ in at least two bits -true

S4(001101)= 0000 (table column: 0110 =6, row: 01=1) and S4(110010)=0001 (table column: 1001 = 9 , row: 10=2)
Two inputs to an S-box differ in the two middle bits -true
Outputs differ in at least two bits -false, only at one

Crypto basics 2

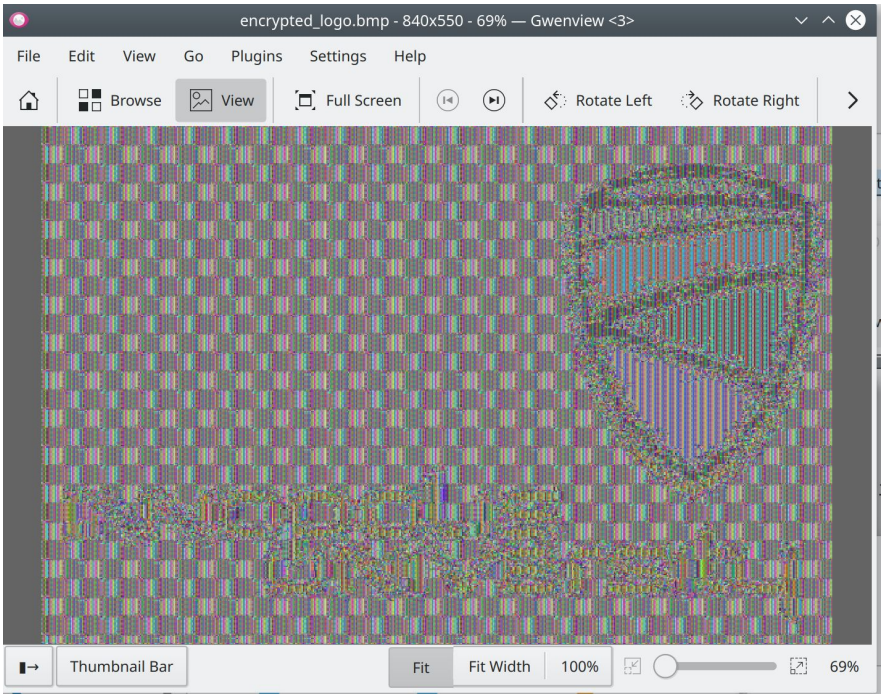
- What kind of hex-editing do you think I had to do to show you a picture?

  Because of the aim was to show the effect of the encryption (scrambled image), the probable steps were to encrypt only the "image data" and keep the file structure unchanged (add headers after encryption). Or encrypt the whole image (with headers) and then add headers again.

All in all, we should leave some headers to make it possible to recognize data as an image.

- Can you encrypt the image with openssl using AES instead of DES to create a similar result where the image is encrypted, but still reveals much of its original structure?

  Yes, it is possible to encrypt the image with openssl using AES that way. Proof:

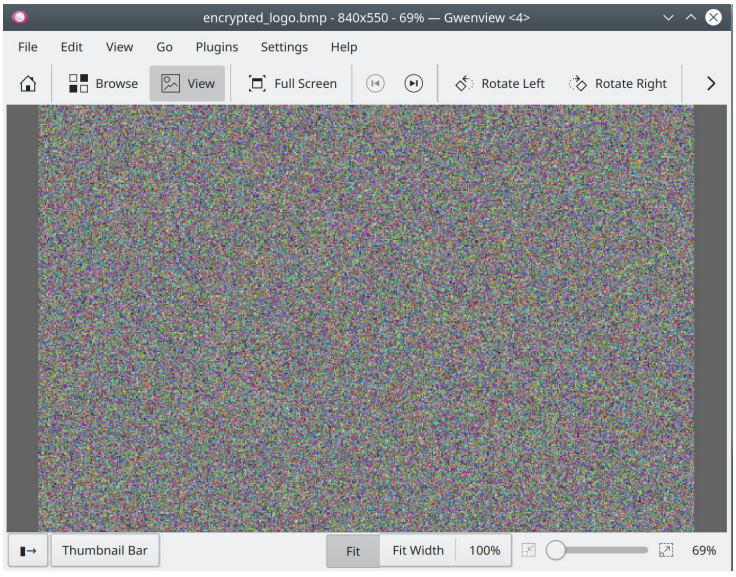


Commands:

```
openssl enc -aes-128-ecb -in logo.bmp -out encrypted_logo.bmp
```

Explanation:

-in and -out are just the paths to original and encrypted files.
-aes-128-ecb, used ecb because the aim was to show that the encryption with lack of diffusion, as an example encryption with -aes-128-cbc gives:



This command was used to return headers:

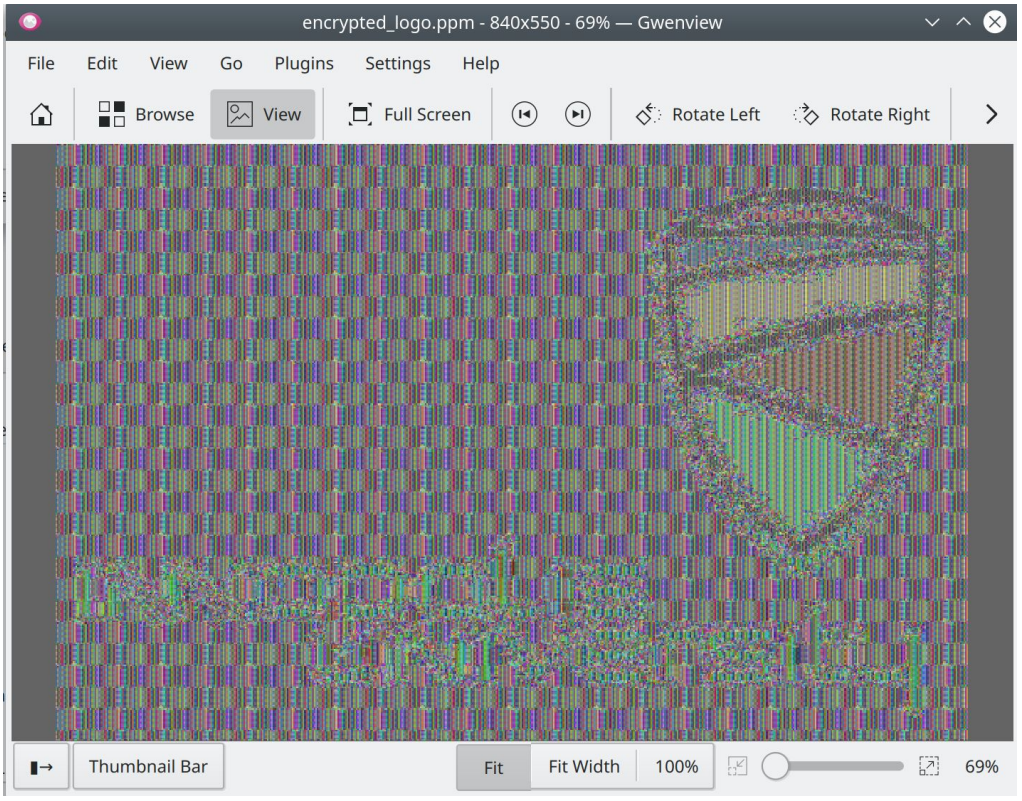
```
dd if=logo.bmp of=encrypted_logo.bmp bs=1 count=54 conv=notrunc
```

- Explain the basic idea behind this trick.

  ECB encrypt each block separately, such that the same plaintext block is encrypted into the same ciphertext block. This approach is not really good at hiding patterns.

- Can this also be done with an image which is not a bitmap?

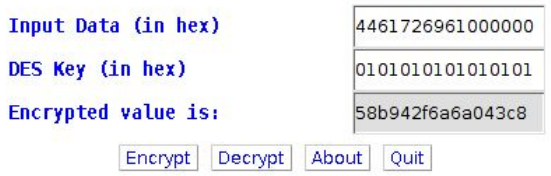
  Yes. Proof:



Commands:

```
head -n 3 logo.ppm > header.txt
tail -n +4 logo.ppm > body.bin
openssl enc -aes-128-ecb -nosalt -pass pass:"badpass" -in body.bin -out body.ecb.bin
cat header.txt body.ecb.bin > encrypted_logo.ppm
```

Crypto basics 3



- The internal state of the device at the 8th round:

```
Rnd8   f(R7=0b360e46, SK8=00 00 00 00 00 00 00 00 ) = 8f5e7d99
```

- Inspect the key schedule phase for the given key and explain how the sub keys are generated for each of the 16 steps

1) DES uses a 56 bit key but the initial key consists of 64 bits. So before the DES process even starts, every 8th bit of the key is deleted to get the key with length of 56 bits.

That is our key 0101010101010101 in a binary format:
0000000100000001000000010000000100000001000000010000000100000001
That is it after the first step in hex:
00 00 00 00 00 00 00 00

2) 56 bit key is divided into two halves, each of 28 bits. These halves are circularly shifted left by one or two positions (according to the schedule presented below).
(And because all the elements in our 56 bits key are zero, nothing changes)

(d) Schedule of Left Shifts

Round Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits Rotated	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

3) 48 of the 56 bit are selected using a table given below:

14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

(And because all the elements in our 56 bits key are zero, nothing changes)

Those things are replied for every round.

- Comment on the behavior of DES when using the given key.

Every key is 00 00 00 00 00 00 00 00, because of the process explained above. All 16 rounds of DES encryption using these keys will be identical and encrypting twice produces the original plaintext: