CS641A (Modern Cryptology)

Chapter-4 Report

Instructor: Prof. Manindra Agrawal

Group Name: CMEN

Names :	Pratyush	170500
	Pravar Deep Singh	160508
	Tushar Garg	160749

Date of submission of report : 29/02/2020

In Chapter 4, as we enter we see the following message:

The rumbling sound is very loud here. It is coming from your right side. A cold blast of air hits you sending shivers up your spine. You look in that direction. There is a large opening on the right from where the sound and the air is coming from. There is a fair amount of light also coming from that direction (you realize that you have not lighted a matchstick and still you can see). There is another door, with a panel nearby, to your left which is closed. The chamber is rocky and cold. Another blast of air hits you from your right and you shiver again.

Now we exhaustively tried all the commands which we got by typing the keyword "**list**" which we got from the mail, and we get the following message by using the command "**go**":

Turning to your right, you go through the opening. You come on a rocky pathway that curves slightly to the right. On both sides of the pathway, the mountain goes up steeply, but you can see the blue sky above you. You seem to have come out of the caves! As you move ahead, the rumbling sound gets louder and louder. You see that the pathway ends at a distance. Hurrying up, you come to the end of it and find that there is a huge lake in front of you! The lake is enclosed by mountain walls. On one side of the lake, there is a big waterfall that is pouring water in the lake, and creating a lot of noise in the process. Where is all the water going, you wonder? You stand at the edge of

the lake, transfixed by its beauty.

Since the message said there is a lake, we dived. Then we resurfaced , dived again and pulled the wand (we died a few times trying to figure these out). Then we tried various commands in various places in chapter 4 but didn't work. So we went back to chapter 3 where a ghost got stuck with us using the wand. Then we came back to the level 4 and we typed "**read**" so the ghost opened the following page:

You come up to the closed door and look at the glass panel..... there is nothing written on it!!

As you wonder what is happening here, you hear the spirit whispering in your ears...

"This is a magical screen. You can whisper something close to the screen and the corresponding coded text would appear on it after a while. So go ahead and try to break the code! The code used for this is a 4-round DES, so it should be easy for you!! Er wait ... maybe it is a 6-round DES ... sorry, my memory has blurred after so many years. But I am sure you can break even 6-round DES easily. A 10-round DES is a different matter, but this one surely is not 10-round ...(long pause) ... at least that is what I remember. One thing that I surely remember is that you can see the coded password by whispering 'password'. There was something funny about how the text appears, two letters for one byte or something like that. I do not recall more than that. I am sure you can figure it out though ..."

Encryption method :

It has been told in the mail that the given encryption is three-round DES.

3 Round DES:-

The three-round DES first has an initial permutation (IP) which is applied at the input and then there is a Feistel structure of 3 iterations as given in fig1 and then an initial permutation inverse (IPinv) is applied to the output of the Feistel structure.

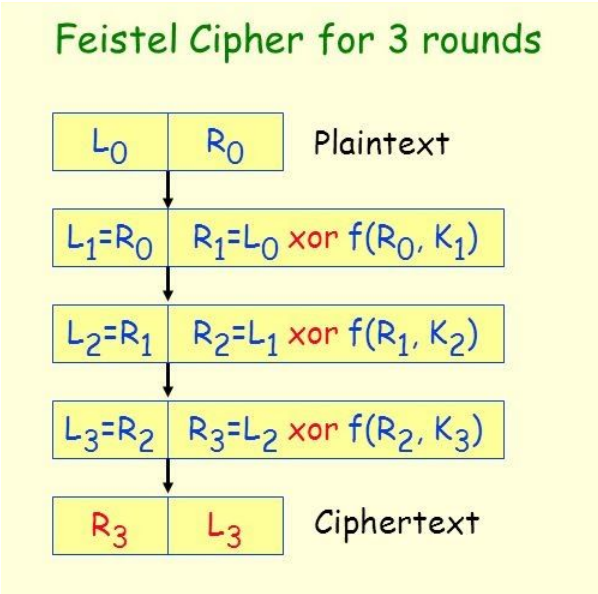

Fig1: Feistel structure for 3 round DES

Here, we can see that if we apply the known-plaintext attack then we know L_0R_0 and L_3R_3 so that we, in turn, know R_2 and L_1 so the only variable we do not know is R_1. Now as done in class we choose two plaintexts L_0R_0 and L'_0R_0 with the same right halves so that we know the xor of the output of the 3rd round permutation.

We know,

$$R_1 = f(R_0, key_1) \oplus L_0 \quad \text{and} \quad R'_1 = f(R_0, key_1) \oplus L'_0$$

$$\Rightarrow \quad R_1 \oplus R'_1 = L_0 \oplus L'_0$$

So the output just before the permutation of the third round will be

$$f(R_2, key_3) = f(L_3, key_3)$$

Now we know

$$R_3 = L_2 \oplus f(L_3, key_3) \text{ and } R'_3 = L'_2 \oplus f(L'_3, key_3)$$

$$\Rightarrow \quad R_3 = R_1 \oplus f(L_3, key_3) \text{ and } R'_3 = R'_1 \oplus f(L'_3, key_3)$$

$$\Rightarrow \quad R_3 \oplus R'_3 = (R_1 \oplus R'_1) \oplus (f(L'_3, key_3) \oplus f(L_3, key_3))$$

$$\Rightarrow \quad f(L'_3, key_3) \oplus f(L_3, key_3) = (R_1 \oplus R'_1) \oplus (R_3 \oplus R'_3)$$

$$\Rightarrow \quad f(L'_3, key_3) \oplus f(L_3, key_3) = (L_0 \oplus L'_0) \oplus (R_3 \oplus R'_3) \qquad \ldots.(1)$$

Hence, we can calculate the xor of the output of the 3rd round permutation.

Breaking the 3 Round DES:-

Firstly in the message, it was given that "two letters for one byte or something like that" so to find some pattern we started to put random input on the site and we observed a few things as follows:-

1. If the input string is of less than or equal to 16 characters then the output string was of 16 characters. So, we assumed there might be some kind of padding done to input to make it 64-bit long as the input to the DES is a 64-bit text and there will be no padding on the input consisting of 16 characters.

2. Now If the input plaintext was given a longer string (17 to 32 characters) then the number of characters in the output became 32. So we thought that the output may represent decryption of two blocks of the input.

3. To confirm the size of block length, we tried the input, given in table 1, on the site and observed that the ciphertext corresponding to 17 'a's is the same as ciphertext corresponding to 16 'a's appended with the ciphertext of 1 'a'. So it was apparent from this that the encryption scheme does the encryption by encryption 64-bit at a time starting from the start of the string i.e. taking 16 characters at a time.

Input Length	plaintext	ciphertext	Output Length
1	a	ritfonspqfptlpkl	16
16	aaaaaaaaaaaaaaaa	ljhomunkthhnipmr	16
17	aaaaaaaaaaaaaaaaa	ljhomunkthhnipmrritfonspqfptlpkl	32

Table 1: Plaintexts and their corresponding ciphertexts.

4. Now we generated 1,000 input text randomly and obtained their corresponding cipher text and did a frequency analysis. What we found was that those 1,000 ciphertext only consisted of letters from the alphabets ‘f’ to ‘u’.

Based on above observations we concluded the following:

Since it was given that two letters correspond to one byte, it means we have a single letter represented by four bits, but four bits can represent at most 16 different letters which is confirmed as only letters from ‘f’ to ‘u’ appear in the output ciphertext. Now that we required a mapping from letters ‘f’ to ‘u’ to corresponding 4 bits, we started with the simplest mapping as given in table 2.

f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Table 2: number in decimal corresponding to the characters

Now, to find the keys we need to generate binary inputs. Also since the website takes input in form of characters we needed to convert our binary input to characters using mapping of table 2. So we generated the input pairs of 64 bits with the same right halves i.e. same last 32 bits. Since we wanted the right halves of the input to be the same after initial permutation so we applied initial

permutation inverse on the randomly generated inputs and then converted them to the desired input format (characters format) by using table 2.

Now we used a script ‘query.py’ to obtain the ciphertext of the input from the site and wrote that to a file. And then we used another script ‘process.py’ on the obtained ciphertext from the website to first convert it to binary and then used initial permutation on this binary string to cancel the initial permutation inverse and then we exchanged the right and left halves.

Now for a pair of input (L_0R_0, L'_0R_0) we found the corresponding values of (L_3R_3, L'_3R') by all the processing described above. For the given input pair the xor of the outputs after the permutation in the third round is $(R_3 \oplus R'_3) \oplus (L_0 \oplus L'_0)$ using equation 1. Now we applied inverse of round permutation on this xor to get the xor value after s-box of third round.

Now to find the xor of input to s-box in the third round we used $L_3(=R_2)$ and $L'_3(=R'_2)$. We first expanded $L_3(R_2)$ and $L'_3(R'_2)$. Now consider the key of round 3 to be made of 8, 6-bit blocks i.e. $key_3 = K_1K_2K_3K_4K_5K_6K_7K_8$ where each K_i is represented by 6 bits. Consider a fixed K_1. Now we take the first 6 bits of expanded $L_3(R_2)$ and $L'_3(R'_2)$,then take xors with K_1, then find the corresponding outputs from s1 and then xor the values. Now this xor value should match with the first 4 bits of $(R_3 \oplus R'_3) \oplus (L_0 \oplus L'_0)$. We tried all the possible values for each K_1 and got the set of values of K_1 for which match happens. Similarly we can find a set of values possible for each of K_is. We used 20 input pairs and found that only single values of K_is is left possible in the intersection of possible values for these 20 input pairs. Therefore we can uniquely determine the value of key_3.

Now as we knew the key of round three so we simulated round 3 exactly and obtained the output after 3rd round permutation hence we knew L_2. So now we did the similar procedure for the second round but not with xor but actual values. So using those same 20 input and output pairs we got the key of the round 2 as well. Similarly we got the key of round 1 as well and we saved it in file ‘keys.txt’.

Key_3	010000 000110 000101 110110 011100 101111 001101 101101
Key_2	000000 001101 110001 000100 110111 110011 001111 001001
Key_1	001100 000011 110000 110100 101001 011011 011111 010110

Now to decrypt we just used the key values and decrypted the password “mugulrhnrstmohmjhnlfmhusnrnfskgn” to be “liipoorlrlmlhpgqfmlnglphpoptgjth”.

Explanation of codes:-

1. The file constants.py is the same as the file uploaded on moodle with the extra inverse of initial permutation and the inverse of the round permutation named ‘ipinv’ and ‘perm’ respectively.
2. The file function.py contains various functions to calculate
    a. The xor of two binary strings.
    b. The output of the expansion box.
    c. The output of the initial permutation and its inverse.
    d. The output of the inverse of round permutation.
    e. The encryption used to represent the input.
3. The file query.py randomly generates ‘t’ number of input pairs with the same right halves and then it applies the ‘ipinv’ on these 64-bit strings so that after initial permutation we get the right halves of both the string to be the same. Now the file calls the site and gets us the encryption of those input pairs. The plaintext and ciphertext pairs are stored in the file random_io.txt.
4. The file random_io.txt contains input and output pairs from 3 round DES. The odd line and consecutive even line represents a pair whose right halves are the same after initial permutation.
5. The file process.py read the file random_io.txt and convert all the input and output pairs to their binary form just before entering the Feistel structure i.e. input after initial permutation and output before ‘ipinv’ and saves it in the file ‘bin_io.txt’.
6. The file ‘find_key.py’ reads the input and output pairs from the file ‘bin_io.txt’ in pairs i.e. two lines at a time and reduces the space of possible keys. The reduction of keys space is done recursively i.e. first for 3rd round then for 2nd and finally for 4th round and the final keys are saved in the same order they were found in the file ‘keys.txt’.
7. The file ‘decrypt.py’ decrypts the ciphertext using the keys from the file ‘keys.txt’.

Reference:-

1. *https://slideplayer.com/slide/6196919/*