

**Q1 Team Name****0 Points**

Group Name

team\_13

**Q2 Commands****5 Points**

List all the commands in sequence used from the start screen of this level to the end of the level. (Use -> to separate the commands)

go -> jump -> jump -> back -> pull -> (Go to Level 3) -> enter  
-> wave -> (go to level 4) -> read -> password -> c ->  
rrrqicbrle

**Q3 Cryptosystem****10 Points**

What cryptosystem was used at this level? Please be precise.'

6-round DES

**Q4 Analysis****80 Points**

Knowing which cryptosystem has been used at this level, give a detailed description of the cryptanalysis used to figure out the password. (Use Latex wherever required. If your solution is not readable, you will lose marks. If necessary, the file upload option in this question must be used TO SHARE IMAGES ONLY.)

After "read" command, we reached the problem statement, which stated that it can be a 4-round, 6-round or 10-round DES. It emphasized more on 6-round DES and 4-round DES, but breaking a 4-round DES would've been too easy. So, we started by assuming it is a 6-round DES.

Clearly, we had access to chosen plaintext attack as the "magical screen".

Data Encryption Standard or DES is a block cipher with block size = 64 bits or 8 bytes and key size = 56 bits. It was given that two letters were for one byte. So, 16 letters represented 1 block size.

After trying inputs, it was clear that the 16 letters in the output screen are from 'f' to 'u'. Then the next analysis was that the input must also from 'f' to 'u' as we can represent these letters from 0 to 15 (i.e. 0000 to 1111 or 4-bit input).

To break r-round DES, we require an (r-2) round characteristic.

We used the differential iterative characteristic "405c0000 04000000" which gives "00540000 04000000" after 4-rounds with a probability of 0.000381.

Clearly,  $10^5$  or 1 lakh pairs of plaintext and ciphertext should be enough for breaking it.

Order of operations:

1. Generate random input
2. Make input pair with specific xor
3. Get output pairs from Server
4. 6-round DES Decryption

1. Generate random inputs

We generated  $10^5$  random inputs of 64 bits with each bit having  $P = 1/2$ . We used these as plaintext block. We stored these inputs.

2. Make input pair with specific xor

We found the inverse initial permutation of "405c0000 04000000" as "0000901010005000" in hex. Then for each

of the plaintext block, we take its xor with "0000901010005000" and generate its pair, and store it in a new file. The values are stored as alphabets where 2 letters are for one byte, hence 16 letters will form 64 bits. They were mapped as explained earlier: f->0000, g->0001, .... u ->1111.

### 3. Get output pairs from Server

The server was working as chosen plaintext attack screen. But we cannot enter  $10^5$  pair of input blocks manually. So we decided to automate it. We used "expect" command for shell script to automate the process of entering the plaintext and getting the ciphertext. We store the whole log of game in a logfile. After completing all  $10^5$  pairs of inputs, we cleaned the log by taking into account only the string which appears after "Slowly, a new text starts appearing on the screen. It reads ...". We store all the ciphertexts in a new file. Now we had both plaintext and ciphertext for  $10^5$  pairs of inputs. we convert the them obtained to 64-bits using the same map we explained i.e. f->0000, g->0001, .... u ->1111.

### 4. 6-round DES Decryption

This is explained below as titled Decryption

#### Decryption

As of this point, we had both plaintext and ciphertext for  $10^5$  pairs in binary form or 64-bit blocks. We can imagine the flow of 6-round DES as below blocks:

$$\begin{matrix} R_0 & R_1 & R_2 & R_3 & R_4 & R_5 & R_6 \\ L_0 & L_1 & L_2 & L_3 & L_4 & L_5 & L_6 \end{matrix}$$

Clearly,  $R_5 = L_6$  and we need output and input xor of Last round S-box. We also need the expanded value of  $R_5$ .

We assumed standard permutation, inverse permutation and bit selection table.

We already know the ciphertext corresponding to plaintexts, hence we calculated  $R_6L_6$  (reverse final permutation of ciphertexts).

Then, we find out the expansion of R5 using the standard bit selection table (let the expanded value be  $a_1, a_2$ ), let  $bxor = a_1 \oplus a_2$ . output xor is inverse permuted by standard inverse permutation function to get output xor of last S-box.

Then we select the left blocks (L6 or R5) of alternate outputs starting from first output (remember that we've generated input and output in pairs), expand it using bit selection table and store it in a file.

Then we generate  $b_1$  and  $b_2$  pairs using the given xor such that xor of there S-box outputs matches the resultant value, those satisfying the condition are selected.

Key 6 is calculated for all  $b_1$  values as xor of  $a_1$  and  $b_1$ . The frequency is maintained as well. Max frequency keys of 6-bits which correspond to the 8 S-boxes are generated. These are [45, 59, 57, 7, 46, 25, 1, 50]. But when we did repeat the same for different number of iterations, we came through different value for the S3 block. for e.g. [45, 59, 3, 7, 46, 25, 1, 50].

Hence we were sure about  $7 \times 6 = 42$  bits but not sure about the 6 bits from S3 and the remaining 8-bits that do not correspond to Key 6.

We wrote the key as [101101 111011 XXXXXX 000111 101110 011001 000001 110010]

'X' denotes that we do not know the bit value. The 8-bits not corresponding to Key 6 will also be denoted by 'X'.

Then we generate the permuted key using the standard permutation, applying it on the 48 bits mentioned above and the 8 bits not contained in Key 6.

The permuted key came out to be:

"X11XX1XX01011X100XX11X11100X1100101X00101000X01X0111X001"

There are total 14 bits unknown to us at this moment. So we generate all  $2^{14}$  possibilities of keys and store it in a

file. This file contains all the possibilities of keys.

We take a trivial input output pair where input = "ffffffffffffff" which correspond to input [0, 0, 0, 0, 0, 0, 0, 0, 0], the input plaintext generates the output ciphertext "srtpkqqpkpliotri" which correspond to [220, 234, 91, 186, 90, 99, 158, 195].

Now we brute force on all the possible keys and check the input output pair and finally get a key that satisfies it.

The final key we got was:

```
011011100101111001111011100011001011001010000011
01110001
```

The password we had initially, (giving password as a command after read) was:

"skoniutlksnnpsklplgognrtjpssilmu".

This has 32 letters, so we broke it into 2 halves and decrypted independently using the key generated above. The input had 64 bits each, [213, 152, 63, 230, 93, 136, 173, 86] and [166, 25, 24, 206, 74, 221, 54, 127] which on decryption gave [114, 114, 114, 113, 105, 99, 98, 114] and [108, 101, 48, 48, 48, 48, 48, 48]. The range gave idea of ascii values as every other number from 48 lies in the range of lowercase alphabet and 48 is 0.

We neglected the trailing zeroes and got the ascii characters of others as "rrrqicbrle".

We entered this on the prompt and solved this level.

## Q5 Password

5 Points

What was the password used to clear this level?

rrrqicbrle

Q6 Code

0 Points

Please add your code here. It is MANDATORY.

▼ src.zip

Download

1	Binary file hidden. You can download it using the button above.
---	---

Assignment 4


● Graded

Group

SOLANKI KEVALKUMAR BALVANTBHAI

SUKET RAJ

UTTAM KUMAR

 [View or edit group](#)

Total Points

100 / 100 pts

Question 1	
<a href="#">Team Name</a>	0 / 0 pts
Question 2	
<a href="#">Commands</a>	5 / 5 pts
Question 3	
<a href="#">Cryptosystem</a>	10 / 10 pts
Question 4	
<a href="#">Analysis</a>	<div>R</div> 80 / 80 pts
Question 5	
<a href="#">Password</a>	5 / 5 pts
Question 6	
<a href="#">Code</a>	0 / 0 pts