Q1 Team Name
0 Points

Group Name

team_7

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->dive->dive->back->pull->back->back->go->wave->back->back->thrnxxtyz->read->the_magic_of_wand->c->read->password

Q3 Cryptosystem
10 Points

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

6-Round Data Encryption Standard (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.)

First, use the above command to reach the ciphertext.
It was given that two letters were for one byte and DES has a block size of 64 bits or 8 bytes so 16 letters represented 1 block size. Because there were only 16 letters in the output which we found after doing frequency analysis which were from ‘f’ to ‘u’ so we thought that input also consists of these letters only as only 16 letters could be represented by four bits and each alphabet was mapped with a number from 0 to 15.
The differential iterative characteristic of 6-round DES ‘405c0000 04000000’ was used to break the code which gives the differential ‘00540000 04000000’ after 4 rounds with probability 0.000381 for which we need approximately 100,000 pairs of plaintext-cipher text pairs.
Used 2 programs random_strings.py which gives random 64-bit strings output (in random_strings.txt) and matching_pairs.py which outputs (in matching_pairs.txt) only those texts (in ASCII) whose bitwise XOR is equal to '0000902010005000' in hex which is what we get from inverse initial permutation of ‘405c0000 04000000’. Note that conversion of them from bit-strings to ASCII is done as following alphabet mapping: {0: 'f', 1: 'g', 2: 'h', 3: 'i', 4: 'j', 5: 'k', 6: 'l', 7: 'm', 8: 'n', 9: 'o', 10: 'p', 11: 'q', 12: 'r', 13: 's', 14: 't', 15: 'u' }
The plan was to then write a python script which feeds input strings from matching_pairs.txt to the SSH (note that after entering password we could type whatever we want as plaintext to get ciphertext knowing plaintext has 8 letters input from 'f' to 'u') and store the corresponding output in cipher_pairs.txt. We were unable to write it in time.
We then use cipher_to_bin.py to convert those cipher_pair.txt from alphabets to 64-bit strings stored in cipher_to_bin.txt. Since there is going to be a final permutation after 6th round, the ciphertexts we get in cipher_to_bin.txt need to be inversed final permutated in order to get it in L6R6 format. Hence we use reversefinalperm.py which takes input as cipher_to_bin.txt as input and gives reversefinalperm.txt as output. Note that reverse final permutation is nothing but initial permutation itself applied on ciphertexts. This gives us L6R6.

Now we run xor.py. This simulates input XOR and output XOR in the last round. It takes reversefinalperm.txt as input. In the last round of des, let gamma be output value after expansion. We know gamma precisely as we know R5 which is L6. So 32 bits of L6 are expanded and stored in gamma.txt. Let alpha be the input XOR after XORing with key and before Sbox. We don't know the precise inputs (say a1 and a2) which give us alpha but we know that gamma XOR as output will be equal to alpha (c1 ^ c2 = a1 ^ a2) as we use the same key so it becomes 0 when XORed with itself. This pairwise XOR is saved in alphaxor.txt. Then let Beta be output XOR after Sbox and before round permutation. We can get beta by XORing R6 with L5. We know R6 precisely but L5 has a probabilistic nature (as L5 is R4 which is '04000000' in hex with probability 0.000381) so we need to code it with this probability. So basically our overall XOR scheme will look like: (L5^R6)^(L5'^R6'). This has to be then reverse round permutated in order to get betaxor. This output is saved in betaxor.txt.

This is how much we were able to code before we ran out of time. Our plan further ahead was as follows:

Make a new program (say Sbox.py) which scans through all the alpha XORs from alphaxor.txt and tries to find the corresponding values of a1^a2. This is done by permuting 6 bits of a1 and then calculating value of a2 by taking XOR of a1 and (a1 ^ a2) from alphaxor.txt. With these corresponding values of a1 and a2, we try to narrow it down to only those a1 and a2 such that S(a1)^S(a2) = (b1^b2) from betaxor.txt.

After this, We would have to calculate k6 by taking the XOR of the candidate a1 with corresponding c1 (from gamma.txt) and kept a counter for it. The real key will repeat itself more frequently as compare to others. Hence we could have gotten 6 blocks values out of 8 blocks but we could not get the key values from S-boxes S3 and S4. So now we have knowledge of 36 bits out of 56 bits. After this we apply brute force on the rest 20 bits of the key and run 6 round brute force DES using each of the permutated key and check whether a given input is encrypted in the same way or not. If it is, then that is the key.

Once the key was found, all we had to do was decrypt the

ciphertext password using the key and convert the resultant plaintext into ASCII.
We've uploaded all the programs which we were able to to write.

Q5 Password
5 Points

What was the password used to clear this level?

Q6 Code
0 Points

Please add your code here. It is MANDATORY.

CS641-Assignment 4.zip Download

```
1  Large file hidden. You can download it using the button above.
```

Assignment 4

Graded

Group

CHANDEKAR VIDISH VIJAY
RASHMI BHIKAJI WAGHMARE
HARSH SAROHA
View or edit group

Total Points

0 / 100 pts

Question 1

Team Name **0** / 0 pts

Question 2

Commands **0** / 5 pts

Question 3

Cryptosystem **0** / 10 pts

Question 4

Analysis **0** / 80 pts

Question 5

Password **0** / 5 pts

Question 6

Code **0** / 0 pts