(Team details can be found at the end of the assignment)

Cipher Method: *6 Round DES*

Passcode: *puikjiiytvi000000*

Justification:

i. On the first screen, there was a panel nearby so we used ***read*** but nothing was written on that panel. So, we used ***enter***.
ii. On the next screen, we were at the edge of the lake so we used ***jump***. We again came out. So, we ***jump*** again. There we see magic wand and tried ***pull*** but went out of breath and died.
iii. So, we entered the above commands again and the instead of ***pull*** we used ***back*** and then used ***dive*** again. Now, we use ***pull*** and got the wand.
iv. Now, we went back to the first screen using series of ***back***. And tried ***read*** but it was still empty. After a while we figured out that it has something to do with the chapter name which is THE SPIRIT and there was a spirit in level 3.
v. So, we went back to level 3 and used ***enter*** and reached second screen where we used ***wave*** and freed the spirit. Then we use the same commands as in chapter 3 to clear it (***thrnxxtzy -> read -> ttd_qinmc_li***). Then from the first screen of 4th chapter we used ***read*** and we got the question read by spirit.
vi. It said this is either 4,6 or 10 rounds DES. As 10 rounds is very difficult to break and 4 rounds would be easy to break. So, we started with 6 round thinking if this won't work then we can use the similar analysis for 4 round DES.
vii. 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.
viii. After trying enough inputs, we found that 16 letters in the output are from 'f' to 'u'. Also, we considered 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.

ix. We used differential iterative characteristic of 6-round DES ‘405c0000 04000000’ to break the code, which gives the differential ‘00540000 04000000’ after 4 rounds with probability 0.000381. For this we need approximately around one lakh pairs of plaintext & cipher text.

x. We used ***DES_InputGenerate.ipynb*** to generate one lakh random inputs and also to get input pairs whose XOR is equal to ‘0000902010005000’ which we get from inverse initial permutation of ‘405c0000 04000000’ and it is stored in ***inputs.txt***. Now we used ***script.sh*** to generate ciphertext corresponding to inputs we just generated and stored them in ***output.txt***. (Server was crashing thus we could generate lesser pairs compared to input [Around 500 lesser] but we made sure that pairs aren’t affected)

xi. Now we use ***DES_Differential_Crypt.ipynb*** where first we reverse final permutation of block R6 to get the output of the sixth round of the DES (let’s say it as xor_outs). We then find out the output differentials (i.e. the XOR of alternate pairs of xor_outs). The xor_outs’s left half is equal to the right half output of round 5. So, we expand that to get the input differential of the S-Boxes (the differential doesn’t get changed after key XORing) that are stored in XS_inp.txt. Now, we inverse permuted the right half of the xor_outs to get the S-Box’s output differential and stored them in XS_out.txt. From every two entries in xor_outs (Out1, Out2) we extract the left half of Out1, expand it and store it to XExp_out.txt. So, XExp_out.txt stores the intermediate value in sixth round just after Expansion box and before key XORing.

xii. In ***DES_SBox_Phase.ipynb***, for every correct possible corresponding input-output differential pairs (stored in XS_inp.txt and XS_out.txt), we can extract the 6-bit key set corresponding to that S-Box, using XExp_out.txt. This is done in the following manner: For every entry in XS_inp.txt, we generate input pairs for each S-Box so that the XOR of the two inputs is equal to that iteration’s XS_inp.txt value (let them be s_inp1 and s_inp1 XOR XS_inp[i]). And if the output pair XOR is equal to the that iteration’s XS_out.txt value, we update the counter of XExp_out.txt

XOR s_inp1 (that is the corresponding key bits for that S-Box). After the complete iteration over XS_inp.txt, we then find out the 6-bit key sets for each S-Box that are highly probable. The stats of the 6-bit key values is given for each S-Box.

S-Box	1	2	3	4	5	6	7	8
Maximum Frequency	14844	12051	6972	6852	12405	12376	12214	11800
Average Frequency	7013	6928	6247	6227	7071	6962	6917	7035

xiii. Now, we get 6 blocks values out of 8 blocks as their frequency is way above mean. But we cannot get key from S-Boxes S3 and S4. So, now we have knowledge of 36 bits out of 56 bits.

xiv. We then use the permutation using key scheduling and map it in the latter half of ***DES_SBox_Phase.ipynb*** as follows:

[-1, -1, 1, -1, -1, 0, -1, -1, -1, 0, 1, -1, 1, -1, 0, 0, -1, -1, -1, 1, 0, -1, -1, 0, 0, -1, 0, -1, 1, 1, 1, 0, 0, 0, 1, -1, 1, 0, 1, 0, 1, 1, 0, 1, -1, 1, 0, -1, 0, 1, 0, 1, -1, 1, 0, 1]

Next, we apply brute force on the remaining 20 bits of the key (Here represented by -1) using ***Key_Maker.py*** and stored these possible keys in ***Keys.txt***. Then we referred this site: (http://desimplementation.blogspot.com/2015/09/data-encryption-standard-algorithm-data.html) for DES implementation (6 Rounds) and modified it for bruteforcing and checking whether a given input is encrypted as the corresponding output or not. If we found one, then that is the final key.

Input: ffffffffffffffff {0,0,0,0,0,0,0,0}
Output: omlhlphkfgjlirri {151, 98, 106, 37, 1, 70, 60, 195}
Key:
01110000001110000011011001011110001110101101110101010101

xv. Then we run Decryption function from DES implementation for 6-Rounds for 16 letters (8 Bytes) at a time and decrypt the password using the key.

Encrypted Password: krpsqqonftrjkktkklfqiuipgqstqpsj
{92, 173, 187, 152, 14, 196, 85, 229, 86, 11, 63, 58, 27, 222, 186, 212}

Decrypted Password: {112, 117, 105, 107, 106, 105, 121, 116, 118, 105, 48, 48, 48, 48, 48, 48}

xvi. We converted it back to characters using the initial mappings from 'f' to 'u' which became: **mfmklolqlplomomjmlloififififif** which didn't work. After a while we figured out that these can be ASCII codes as the numbers lie below 127. So, after mapping them we found the password to be (Where 0s are just for padding):

puikjiytvi000000

Team Details:

- Ashish Pal (18111010)
- Darshit Vakil (18111013)
- Mayank Rawat (18111040)