Q1 Teamname
0 Points

CryptoCreeks

Q2 Commands
5 Points

List the commands used in the game to reach the ciphertext.

enter/go->enter/go->enter/go->enter/go->enter/go->give->read

Q3 Analysis
50 Points

Give a detailed description of the cryptanalysis used to figure out the password. (Explain in less than 100 lines and 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.)

We entered a narrow passage where there was no panel just a door with a handle on it. We pushed the door to go to the next side using the command "go/enter". Here we see a room with a man sleeping there. Also along with this, we saw a door and decided to get past it using the command "go/enter". After this, after a small conversation with the man present there, he left the room and so did we. But then we again re-entered the room using the command go/enter. Next, here we saw the door again and used the "go/enter" command to go through it. Here we once again saw a door and since there was no handle or panel on that, so we pushed through it using the command "go/enter" command. Here, after having some conversation with the man present there, we felt he was about to do something wrong and so we tried to flee away from there using the command "go", but after that, we died. So we re-entered again and instead of entering the "go" command at this phase, we used the command "give" to give the wand to the mystery man. He fled away after taking the wand. We saw a door and a panel next to it. We used the "read" command to see what is present on the panel and we saw the following hash values there :

61686060000000000601808066E964E16F6FEEE266E964E10E078E8266E964E1
08060E02000000000E078E8266E964E1696E6E620000000061686060000000000

There we also got to know that we will get our password using these hash, which are atmost 16 characters long and which is created from the toy version of SHA3, which consists of three-step mapping: Theta, Pi, and Chi.

BREAKING SHA3 TOY VERSION::

We tried to break the cryptosystem firstly for the first half of the hash(i.e. the first 64 characters of the hash) and then for the second half(i.e. the last 64 characters of the hash).

REVERSING THE LAST WHILE LOOP::

We were provided with the following hexadecimal mapping::
hexa = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F']

We iterated over all the characters of the input hash and for each character, we found its corresponding index using the above mapping, and then we found the corresponding binary value, after which we used each bit of this binary value to fill up the state array.

$$state[k/(64*5)][(k/64) \mod 5][k \mod 64 + j] = b_j$$

By this, we will get the first 256 bits for the state array and the rest of the values will be 0.

REVERSING Chi

The following is the mapping::

Chi_inverse=
{'00000':'00000','00101':'00001','01010':'00010','01011':'00011','10100':'00100','10001':'00101','10
110':'00110','10111':'00111','01001':'01000','01100':'01001','00011':'01010','00010':'01011','01101':'0110
0','01000':'01101','01111':'01110','01110':'01111','10010':'10000','10101':'10001','11000':'10010','11011':'10
011','00110':'10100','00001':'10101','00100':'10110','00111':'10111','11010':'11000','11101':'11001','10000':'
11010','10011':'11011','11110':'11100','11001':'11101','11100':'11110','11111':'11111', }

We formulated this mapping as follows ::

$$a[i] = b[i] \oplus (b[(i+1) \mod 5] \oplus 1) \,\&\, (b[(i+2) \mod 5] \oplus ((b[(i+3) \mod 5] \oplus 1) \,\&\, b[(i+4) \mod 5]))$$

We generated the values of the tempstate array from the original state array using the above-mentioned mapping. After this, we copied the tempstate array values to the state array.

REVERSING Pi

We then simply reversed the Pi equation given in the toy version :
tempstate[i][j][k] = state[j][((2 * i) + (3 * j)) % 5][k]

and found the value of the tempstate array. We then again copied the obtained values to the state array.

REVERSING Theta

For reversing Theta we analyzed some of the equations such as

state[i][j][k] = state[i][j][k] ^ column_parity[(i+4)%5][k] ^ column_parity[(i+1)%5][k]   -------------(1)

Let's refer the state[i][j][k] on RHS as old_state and that on LHS as new_state. So , our equation will be

new_state[i][j][k] = old_state[i][j][k] ^ column_parity[(i+4)%5][k] ^ column_parity[(i+1)%5][k] -------------(2)

The term column_parity[(i+4)%5][k] ^ column_parity[(i+1)%5][k] would be same for all the values of "j" , if i and k are fixed. So , if we xor all the given values for one particular "j" , we will get

new_state[i][0][k]^new_state[i][1][k]^new_state[i][2][k]^new_state[i][3][k]^new_state[i][4][k] = old_state[i][0][k] ^ column_parity[(i+4)%5][k] ^ column_parity[(i+1)%5][k] ^old_state[i][1][k] ^ column_parity[(i+4)%5][k] ^ column_parity[(i+1)%5][k] ^ old_state[i][2][k] ^ column_parity[(i+4)%5][k] ^ column_parity[(i+1)%5][k] ^ old_state[i][3][k] ^ column_parity[(i+4)%5][k] ^ column_parity[(i+1)%5][k] ^ old_state[i][4][k] ^ column_parity[(i+4)%5][k] ^ column_parity[(i+1)%5][k]   -------------(3)

=> new_state[i][0][k]^new_state[i][1][k]^new_state[i][2][k]^new_state[i][3][k]^new_state[i][4][k] = old_state[i][0][k] ^ column_parity[(i+4)%5][k] ^ column_parity[(i+1)%5][k] ^old_state[i][1][k] ^ old_state[i][2][k] ^ old_state[i][3][k] ^ old_state[i][4][k]   -------------(4)

Now , we know that the column parity of old_state was calculated as(as given in the toy version code) ::

column_parity[i][k] = old_state[i][0][k]^old_state[i][1][k] ^ old_state[i][2][k] ^ old_state[i][3][k] ^ old_state[i][4][k]   -------------(5)

Now, using equation (5) we can re-write equation (4) as

new_state[i][0][k]^new_state[i][1][k]^new_state[i][2][k]^new_state[i][3][k]^new_state[i][4][k] = column_parity[(i+4)%5][k] ^ column_parity[(i+1)%5][k] ^column_parity[i][k]   ------------(6)

Also the LHS is the column parity of the new_state , and hence we can replace the LHS by the new column parity and hence our equation (6) becomes

new_column_parity[i][k] = column_parity[i][k] ^ column_parity[(i+1)%5][k] ^ column_parity[(i+4)%5][k]   ------------(7)

For i = i+2

new_column_parity[(i+2)%5][k] = column_parity[i+2][k] ^ column_parity[(i+3)%5][k] ^ column_parity[(i+1)%5][k]   ------------(8)

For i=i+3

new_column_parity[(i+3)%5][k] = column_parity[i+3][k] ^ column_parity[(i+4)%5][k] ^ column_parity[(i+2)%5][k]   ------------(9)

Now xoring the eqns (8) and (9) , we get ::

new_column_parity[(i+2)%5][k] ^ new_column_parity[(i+3)%5][k]= column_parity[(i+4)%5] ^ column_parity[(i+1)%5][k]   ---------------------(10)

Now, using eqn (10) , we can re-write eqn(2)

new_state[i][j][k] = old_state[i][j][k] ^ new_column_parity[(i+2)%5][k] ^ new_column_parity[(i+3)%5][k]   --------------(11)

old_state[i][j][k] = new_state[i][j][k] ^ new_column_parity[(i+2)%5][k] ^ new_column_parity[(i+3)%5][k] --------------(12)

Now, we can find the column_parity of the existing new_state and from eqn (12) , we can find all the values of the old_state array.

We get the state value after reversing theta from the above calculation. From this we will obtain the message array by simply reversing the formula given in the toy version of SHA3::

message[k] = state[k//(64*5)][(k//64) % 5][k%64]

We then took 8 bits of this message and converted it to ASCII value and then only printed out the alphanumeric value and got the following for the 1st part of the hash::

`fybxfybxoowtfybxfybxfybxfybxfybxfybx`

and the following for the 2nd part of the hash::

`oowtfybxhaiggdiggdnippfybxoowtfybx`

We observed that there are some set of four characters that were repeating, again and again, such as:: "oowt", "fybx", "nipp", "iggd"

We tried all the combinations of these 4 pairs but none of them worked out to be the correct password and hence we tried different combinations by choosing any 3 of them and finally got the correct password as "nippfybxiggd"

The final password obtained is:: $\boxed{nippfybxiggd}$

No files uploaded

Q4 Password
25 Points

What was the final command used to clear this level?

```
nippfybxiggd
```

Q5 Codes
0 Points

It is mandatory that you upload the codes used in the cryptanalysis. If you fail to do so, you will be given 0 marks for the entire assignment.

Crypto_Assignment7.ipynb Download

Assignment 7

● GRADED

GROUP
SHRUTI WASNIK
TANISHA RASTOGI

View or edit group

TOTAL POINTS

80 / 80 pts

QUESTION 1

Teamname **0** / 0 pts

QUESTION 2

Commands **5** / 5 pts

QUESTION 3

Analysis **50** / 50 pts

QUESTION 4

Password R **25** / 25 pts

QUESTION 5

Codes **0** / 0 pts