Q1 Teamname

0 Points

Cipherberg

Q2 Commands

5 Points

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

enter, enter, go, go, go, give, read, c, ovtsjicuhshz

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.)

On the panel, we're given the hash value of our password created using a toy version of SHA3 consisting of only 3 steps - Theta, Pi and Chi. We're also provided with the code used for the encryption. We observe that, to get our password, we have to basically reverse these three operations.

Reversing Chi operation:: We perform the following steps:
1. Take a random input in array str[].
2. For rounds 0-23, print the state[] array before applying the chi operation and after applying the chi operation. For each round, this gives a 3D array consisting of five 2D arrays $A_{i,j}, 0 \leq i < 5$, $0 \leq j < 64$ before applying the chi operation and a 3D array

consisting of five 2D arrays $B_{i,j}, 0 \leq i < 5, 0 \leq j < 64$ after applying the chi operation, each of dimensions 5 X 64.
3. Find the chi-mapping by mapping the columns of each such 2D array $A_{i,j}$ to the columns of the 2D array $B_{i,j}$.
This gives us the following chi mapping:
chi_mapping = {'00000': '00000', '10000': '10010', '01001': '01100', '10100': '00110', '00001': '00101', '00100': '10100', '11110': '11100', '01101': '01000', '10101': '00001', '11001': '11101', '11000': '11010', '11101': '11001', '11011': '10011', '01100': '01101', '01011': '00010', '00010': '01010', '11111': '11111', '01110': '01111', '10110': '00100', '11100': '11110', '10011': '11011', '10001': '10101', '01010': '00011', '01000': '01001', '00111': '10111', '01111': '01110', '00101': '10001', '10010': '11000', '00011': '01011', '10111': '00111', '11010': '10000', '11000': '11010', '00111': '10111', '00110': '10110'}

We then inverse this chi-mapping to get the inverse chi-mapping:
chi_inverse_mapping= {'00000': '00000', '10010': '10000', '01100': '01001', '00110': '10100', '00101': '00001', '10100': '00100', '11100': '11110', '01000': '01101', '00001': '10101', '11101': '11001', '11010': '11000', '11001': '11101', '10011': '11011', '01101': '01100', '00010': '01011', '01010': '00010', '11111': '11111', '01111': '01110', '00100': '10110', '11110': '11100', '11011': '10011', '10101': '10001', '00011': '01010', '01001': '01000', '10111': '00111', '01110': '01111', '10001': '00101', '11000': '10010', '01011': '00011', '00111': '10111', '10000': '11010', '10110': '00110'}

Reversing Pi operation:: The Pi operation involves a simple statement in a triple nested for loop. We simply reverse this operation by swapping the LHS and RHS of the statement: state[j][((2 * i) + (3 * j)) % 5][k] = tempstate[i][j][k];

Reversing Theta operation:: To reverse the theta operation we computed column parity using state[] array obtained after applying reverse Pi operation.

During encryption, the following equation is used:
$state[i][j][k] \wedge = column\_parity[(i+4)\%5][k] \wedge column\_parity[(i+1)\%5][k]$ ----- (1)
In this equation, column_parity is computed on state[] array before applying $theta$ operation, which is unknown to us at the time of decryption. After analysis, we found the following relation between column parity of state[] array before applying $theta$ operation and

after applying $theta$ operation.

$new\_column\_parity[(i+2)\%5][k] \wedge$
$new\_column\_parity[(i+3)\%5][k] =$
$old\_column\_parity[(i+4)\%5][k] \wedge$
$old\_column\_parity[(i+1)\%5][k]$ ------(2)

where old_column_parity and new_column_parity are computed using state[] array before and after applying the $theta$ operation respectively.

Using equation (2), we can rewrite equation (1) as:

$state[i][j][k] \wedge = new\_column\_parity[(i+2)\%5][k] \wedge$
$new\_column\_parity[(i+3)\%5][k]$ ------(3)

Using equation (3), we reversed the $theta$ operation.

Performing $Reverse\ Chi,\ Reverse\ Pi,\ Reverse\ Theta$ operation for 24 rounds gave us our state[] array. Using this state[] array, we reverse the loop given in the code which derives the state_array from the message:

```
for(k = 0; k < r; ++k)
        state[k/(64*5)][(k/64) % 5][k%64] = message[k];
```

This can be reversed by swapping the LHS and RHS, and hence we can derive the message[] array from the state[] array. Taking 8 bits of this message[] array at a time and finding the $ASCII$ values gave a string of characters containing our password.

Finding Password:

From the obtained 128 bits of the hash value from panel, lets call it $h_1 \text{and } h_2$, where $h_1 \text{and } h_2$ are 64 bits each, we perform 2 passes:

1. First 64 bits taken first, i.e. hash value $h_1 h_2$ is taken. In the $ASCII$ values obtained from the message[] array we took 2 repeating sets (of 4 character each) - "ovts" and "jicu".
2. Last 64 bits taken first, first 64 bits taken last, i.e. $h_2 h_1$ is taken as hash value. In the $ASCII$ value obtained from the message[] array of this pass we got one more set (of 4 characters) of repeating characters- "hshz"

From the obtained 3 sets of these 4 characters, we try all 3! permutations of these pairs to get our password - ovtsjicuhshz.

No files uploaded

Q4 Password

25 Points

What was the final command used to clear this level?

ovtsjicuhshz

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.

decrypt SHA3.ipynb Download

In [273]:

```
rounds=24
hash_val='00080101000000000E0A830965696CEA61EC61E5

message=[]
for r in range(576):
    message.append(0)
```

In [274]:

```
state=[]
temp_state=[]
column_parity=[]
for i in range(5):
    state.append([])
    temp_state.append([])
    column_parity.append([])
for i in range(5):
    for j in range(5):
        state[i].append([])
        temp_state[i].append([])
for i in range(5):
    for j in range(5):
        for k in range(64):
            state[i][j].append(0)
            temp_state[i][j].append(0)
for i in range(5):

    for k in range(64):
        column_parity[i].append(0)
```

In [275]:

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

In [276]:

```
chi_inverse_mapping= {'00000': '00000',
                      '10010': '10000',
                      '01100': '01001',
                      '00110': '10100',
                      '00101': '00001',
                      '10100': '00100',
                      '11100': '11110',
                      '01000': '01101',
                      '00001': '10101',
                      '11101': '11001',
                      '11010': '11000',
                      '11001': '11101',
                      '10011': '11011',
                      '01101': '01100',
                      '00010': '01011',
                      '01010': '00010',
                      '11111': '11111',
                      '01111': '01110',
                      '00100': '10110',
                      '11110': '11100',

                      '11011': '10011',
                      '10101': '10001',
```

```
                    '00011': '01010',
                    '01001': '01000',
                    '10111': '00111',
                    '01110': '01111',
                    '10001': '00101',
                    '11000': '10010',
                    '01011': '00011',
                    '00111': '10111',
                    '10000': '11010',
                    '10110': '00110'}
```

In [277]:

```
hexToBinary= {'0':'0000', '1':'0001',
'2':'0010','3':'0011', '4': '0100', '5':
'0101', '6':'0110', '7': '0111', '8': '1000',
'9': '1001', 'A': '1010', 'B': '1011', 'C':
'1100', 'D': '1101', 'E': '1110', 'F':
'1111'}
```

In [278]:

```
k=0
for ele in hash_val:
    bin_val= hexToBinary[ele]
    for j in reversed(range(0,4)):
        state[k//(64*5)][(k//64)%5][k%64 +
j]=int(bin_val[3-j])
    k+=4
```

In [279]:

```
current_round=0
while(current_round<rounds):
    temp_state=[]
    for i in range(5):
        temp_state.append([])
    for i in range(5):
        for j in range(5):
            temp_state[i].append([])
    for i in range(5):
        for j in range(5):
            for k in range(64):
                temp_state[i][j].append(0)

#     inverse_chi operation
    for i in range(5):
        for k in range(64):
            s_after_chi=''
            for j in range(5):
                s_after_chi+= str(state[i][j]
[k])
            s_before_chi =
chi_inverse_mapping[s_after_chi]
            for j in range(5):
                state[i][j][k]=
int(s_before_chi[j])

#     inverse pi operation
    for i in range(5):
        for j in range(5):
            for k in range(64):

                temp_state[i][j][k]= state[j]
[((2 * i) + (3 * j)) % 5][k]
```

```
    state=temp_state.copy()

#     inverse theta operation
    for i in range(5):
        for k in range(64):
            column_parity[i][k]=0
            for j in range(5):
                column_parity[i][k]^=
temp_state[i][j][k]

    for i in range(5):
        for j in range(5):
            for k in range(64):
                state[i][j][k] =
temp_state[i][j][k] ^ column_parity[(i+2)%5]
[k] ^ column_parity[(i+3)%5][k]
#                 state[i][j][k] =
temp_state[i][j][k]

    current_round+=1
```

In [280]:

```
for k in range(576):
    message[k]= state[k//(64*5)][(k//64) % 5]
[int(k%64)]
```

In [281]:

```
string=''
for i in range(0,576,8):
    s=''.join(str(x) for x in message[i:i+8])
#     print(s)
    string +=''.join(chr(int(s, 2)))
```

In [282]:

```
print(string)
```

```
hr`r    hshz    ow|{jicuovtsjicu ���            hs
```

In [ ]:

Assignment 7

GRADED

1 DAY, 21 HOURS LATE

GROUP

SAMBHRANT MAURYA

DEEKSHA ARORA

View or edit group

TOTAL POINTS

50 / 80 pts

QUESTION 1

Teamname **0** / 0 pts

QUESTION 2

Commands **5** / 5 pts

QUESTION 3

Analysis **40** / 50 pts

QUESTION 4

Password **5** / 25 pts

QUESTION 5

Codes **0** / 0 pts