Q1 Teamname

0 Points

cryptoEngineers

Q2 Commands

5 Points

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

1. go
2.go
3.go
4.go
5.go
6.give
7.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.)

After reaching to Panel, and listening the word whispered by the spirit, we got to know the following things:

1. The given value is the hash of our Password.
2. This hash is created by using toy version of SHA3, it uses only three step mapping
   i.e. Theta, pi, chi.
3. The Password is no more than 16 characters.

Given,
hash=64616261000000008002840A6968616EE46BEFEB6968616E800A8D8A6968616E000809800000000800A8D8A6968616E64696BE1000000006461626100000000

We started going reverse to the given code i.e. sha3.txt provided in resource section. Firstly, we converted this hash into binary bits. As given hash is of 128

characters, from this we got 512 binary bits. Remaining bit i.e. other than 512 bits are default assumed as 0.

Chi Reverse :

From observing the given toy version code of SHA3, We see that in chi part of code state depends on tempstate. And this Dependency is on index j only. Because given equation is written as,
state[i,j,k]=tempstate[i,j,k] ⊕ (~tempstate[i,(j+1)%5,k] & tempstate[i,(j+2)%5,k])
this above equation clearly make us understand that state[i,j,k] depends on tempstate[i,j,k] where index j only varies.
After noticing this, we start mapping the 5 input bits of tempstate to 5 output bits of state and formed a Map(hardcoded). With the help this map we performed chi inverse. We give 5 state bit as input and map returns corresponding 5 new_state(same as tempstate bit of given sha3.txt) bit as output. Finally, new_state bits assigned to state bits.

Pi Reverse :

This Pi reverse is just reversing the original given equation.

state[j][((2 * i) + (3 * j)) % 5][k] = tempstate[i][j][k]; (eq. from given code $sha3.txt$)
is converted to
newState[i][j][k] = state[j][(2*i+3*j)%5][k];(eq. from our code $SHA3Decrypt.cpp$)

Theta Reverse :

As given in toy version code of SHA3, what θ mapping performs is that it takes state array and returns updated state array. It performs following steps :
1. It computes
column_parity[i][k] = state[i,0,k]⊕state[i,1,k]⊕state[i,2,k]⊕state[i,3,k]⊕state[i,4,k] .
2. Next, it computes new state array,
new_state[i,j,k] = state[i,j,k] ⊕ column_parity[(i+1)%5,k] ⊕ column_parity[(i+4)%5,k] ,
where 0<=i<5, 0<=j<5 and 0<=k<64.

Now, we need to reverse the θ function. What we will have is new state array and we need to find state array.
Let, S[] be old state array, NS[] be updated (new) state array, C[] be column parity derived from (old) state array, NC[] be column parity derived from new state array .
Lets calculate the above arrays for fix value of k.
From the above mentioned first step of calculating column parity,
C[0,k] = S[0,0,k]⊕S[0,1,k]⊕S[0,2,k]⊕S[0,3,k]⊕S[0,4,k] . .....eq(1.1)
C[1,k] = S[1,0,k]⊕S[1,1,k]⊕S[1,2,k]⊕S[1,3,k]⊕S[1,4,k] . .....eq(1.2)
C[2,k] = S[2,0,k]⊕S[2,1,k]⊕S[2,2,k]⊕S[2,3,k]⊕S[2,4,k] . .....eq(1.3)
C[3,k] = S[3,0,k]⊕S[3,1,k]⊕S[3,2,k]⊕S[3,3,k]⊕S[3,4,k] . .....eq(1.4)
C[4,k] = S[4,0,k]⊕S[4,1,k]⊕S[4,2,k]⊕S[4,3,k]⊕S[4,4,k] . .....eq(1.5)

Likewise, given updated state array NS, column parity NC[] can be calculated as
NC[i,k] = NS[i,0,k] ⊕ NS[i,1,k] ⊕ NS[i,2,k] ⊕ NS[i,3,k] ⊕ NS[i,4,k] for 0<=i<5.
....eq(2)

From the 2nd step of theta function, for i=0 new state array can be calculated as,
NS[0,0,k] = S[0,0,k] ⊕ C[4,k] ⊕ C[1,k] ...eq(3.1)
NS[0,1,k] = S[0,1,k] ⊕ C[4,k] ⊕ C[1,k] ...eq(3.2)
NS[0,2,k] = S[0,2,k] ⊕ C[4,k] ⊕ C[1,k] ...eq(3.3)
NS[0,3,k] = S[0,3,k] ⊕ C[4,k] ⊕ C[1,k] ...eq(3.4)
NS[0,4,k] = S[0,4,k] ⊕ C[4,k] ⊕ C[1,k] ...eq(3.5)

Now, taking XORs of LHS and RHS of above equations,
NS[0,0,k]⊕NS[0,1,k]⊕NS[0,2,k]⊕NS[0,3,k]⊕NS[0,4,k] = S[0,0,k]⊕S[0,1,k]⊕S[0,2,k]⊕S[0,3,k]⊕S[0,4,k]⊕C[4,k]⊕C[1,k]

NC[0,k] = C[0,k] ⊕ C[4,k] ⊕ C[1,k] ...eq(4)
Likewise, other column parity NC[] can be calculated. Lets take NC[2,k] and NC[3,k].
NC[2,k] = C[2,k] ⊕ C[1,k] ⊕ C[3,k]
NC[3,k] = C[3,k] ⊕ C[4,k] ⊕ C[2,k]

Taking XORs of LHS and RHS of above equations,
NC[2,k] ⊕ NC[3,k] = C[1,k] ⊕ C[4,k] ...eq(5)

Now, from eq(3.1),
$NS[0,0,k] = S[0,0,k] \oplus C[4,k] \oplus C[1,k]$
It can also be written as
$NS[0,0,k] \oplus C[4,k] \oplus C[1,k] = S[0,0,k]$
from eq(5), substituting the values,
$NS[0,0,k] \oplus NC[2,k] \oplus NC[3,k] = S[0,0,k]$ eq(6)

So, from eq(6), we can say that if we have new state array, then we can calculate its column parity and from these 2 entites we can get (old) state array, i.e. we can reverse theta function.

After performing this reverse we got state bits and this state bit is converted into ASCII. To check the decryption code, we generated hash of known plain message. This hash is then given as input in our code. Output shown is not exactly same what we had given as plain message, but we got the first few characters of plain message. Now we flipped the hashed message i.e. made first 64 characters as last 64 characters and last 64 characters as first 64 characters. On running the decryption code on this flipped hash, we got remaining characters of plain message. This same observation is used for given hash in problem, firstly we run on original hash and got some characters of password and then run on flipped hash and got remaining characters of password. There are two hash value in our code, one is uncommented and other is commented. Uncommented is original

hash of 128 characters. And 2nd i.e. commented is flipped hash of 128 characters. This way we got our final Password.

password = rlvmiahgbimx

Programs :

SHA3Decrypt.cpp : This program has implementation of all reverse i.e. chi reverse, pi reverse and theta reverse. And gives the 128 character string, which has password.

No files uploaded

Q4 Password

25 Points

What was the final command used to clear this level?

rlvmiahgbimx