Q1 Teamname

0 Points

cryptoEngineers

Q2 Commands

5 Points

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

1. go (it takes us towards deep underground well)
2. wave (it makes you in slow motion)
3. dive (It takes you inside water and swimming through hole, you find yourself in a pool)
4. go (it takes us down the passage)
5. read (it takes us closer to screen to read the text)

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 reading magical screen, we got to know the following things :

1] Block of size 8 bytes as 8 x 1 vector over F_{128} constructed using degree 7 irreducible

polynomial $x^7 + x + 1$ over F_2.
2] 2 transformations :

a) Linear transformation given by invertible 8x8 key matrix A with elements from F_{128}.

b) Exponentiation given by 8x1 vector E whose elements are numbers between 1 and 126.

3] Coded password can be seen by whispering word "password" .

After trying several inputs, we observed that characters in Ciphertext were ranging from f to u, i.e. 16 characters. Hence, we can assume that 'f' maps to 0b0000, ..., 'u' maps to 0b1111. Since each byte of block can be a number between 0 to 127 as it is over F_{128}. So, we guessed this 128 pairs from ff to mu,
where

ff -----> 0b0000 0000 -----> 0x00 -----> 0
fg -----> 0b0000 0001 -----> 0x01 -----> 1
.
.
.
mu -----> 0b0111 1111 -----> 0x7f -----> 127

After feeding input plaintexts of certain patterns, we observed that on changing i^{th} byte of input, all bytes after i^{th} byte in output gets changed. It concludes that matrix A is lower triangular matrix.
For example, fffffgfffffffff -> ffffhmffggfpjskm
fffffffffffgffff -> ffffffffffipjthm

gfhjklihfuggjikl -> kqmoijmkgpgtlujt
gfhjklihfffghigj -> kqmoijmkgfjtjlhg

By knowing that matrix A is lower triangular, we targeted first the diagonal elements of A and all elements of E.
For this we generated input plaintexts of the form $C^{8-i}PC^{i-1}$ (1<=i<=8) where C=0 and P ranges from all possible non zero inputs over F_{128} (from 1 to 127) i.e. only one byte of input block as non zero and rest bytes as zero.

These plaintexts are generated using code $generatePlaintext.py$ and ciphertexts corresponding to plaintexts are generated using $script.sh$.

Output text after EAEAE transformation of plaintext is as follows :

$$c_i = (a_{ii} * (a_{ii} * p_i^{e_i})^{e_i})^{e_i}$$

where, c_i = i^{th} byte of output text block

p_i = i^{th} byte of plaintext block

a_{ii} = i^{th} diagonal element of A

e_i = i^{th} element of E

All operations (multiplication and exponentiation) are performed over F_{128} using irreducible polynomial $x^7 + x + 1$. Addition operation over F_{128} is bitwise XOR.
As a_{ii} ranges from 0 to 127 and e_i from 1 to 126. Taking all possible combinations of a_{ii} and e_i.
Computing c_i for every above combination over every p_i. Now checking c_i with corresponding i^{th} byte of Ciphertext. If equals, then a_{ii} is the possible i^{th} diagonal value and e_i is possible i^{th} element of E.
After iterating over all possible input plaintexts, we get 3 possible values of a_{ii} and e_i for every i^{th} byte.
And these are :

$a_{00} : \{84, 40, 49\}$ $e_0 : \{22, 37, 68\}$
$a_{11} : \{18, 17, 10\}$ $e_1 : \{39, 106, 109\}$
$a_{22} : \{23, 43, 9\}$ $e_2 : \{36, 42, 49\}$
$a_{33} : \{31, 11, 12\}$ $e_3 : \{9, 44, 74\}$
$a_{44} : \{103, 8, 112\}$ $e_4 : \{2, 38, 87\}$
$a_{55} : \{31, 11, 12\}$ $e_5 : \{9, 44, 74\}$
$a_{66} : \{16, 27, 28\}$ $e_6 : \{8, 25, 94\}$
$a_{77} : \{98, 38, 40\}$ $e_7 : \{5, 27, 95\}$

Now, these above values were used to find other non diagonal elements of A and also, to finalize above values from 3 possibilities. Taking non-diagonal elements of form $a_{i+1,i}$, we know that in Lower Triangular Matrix $a_{i+1,i}$ depends on $a_{i,i}$. So, to find element $a_{i+1,i}$, we

require $a_{i,i}$ and $a_{i+1,i+1}$. Also we have to check $(i+1)^{th}$ byte of ciphertext corresponding to plaintext having i^{th} byte as non-zero.
For all possible combinations $(a_{i,i}, e_i)$, $(a_{i+1,i+1}, e_{i+1})$ and $a_{i+1,i}$ $(0-127)$, we iterates over all input Plaintexts having i^{th} byte as non-zero and checking equality of c_{i+1} with $(i+1)^{th}$ byte of corresponding Ciphertexts.
If the equality holds over all input plaintexts, for particular combination, then, e_i, $a_{i,i}$, e_{i+1}, $a_{i+1,i+1}$, $a_{i+1,i}$ are the final values of A and E.

Similarly, we can find other non-diagonal elements of form $a_{i+2,i}$, $a_{i+3,i}$, . . . etc.
At last we get our final linear transformation matrix and exponentiation vector as
Linear Transformation Matrix,
A = [[84, 0, 0, 0, 0, 0, 0, 0],
[112, 70, 0, 0, 0, 0, 0, 0],
[26, 26, 43, 0, 0, 0, 0, 0],
[99, 27, 0, 12, 0, 0, 0, 0],
[100, 63, 1, 112, 112, 0, 0, 0],
[19, 45, 28, 51, 96, 11, 0, 0],
[11, 123, 0, 99, 28, 83, 27, 0],
[64, 8, 75, 27, 18, 72, 1, 38]]

and Exponentiation Vector,
E = [22, 106, 42, 74, 87, 44, 25, 27]

Decryption

Now, we have $password = "gsmpmhfkmphtlrfuhlmogsirfpgojshk"$. We divided it into two halves, 8 byte each(as our block size is of 8 byte).
Taking one half, for each byte, we iterate over every value of byte[0-127], performed EAEAE transformations and checked whether it is equal to corresponding byte of password or not.
If it is equal, then, add the value and move to next byte in that plaintext block.
Similarly, do for second Half.
At last, we concatenated the both plaintext halves, to, get final decrypted password.

Decrypted password = $vkspyqrxfg000000$ and after removing trailing zeroes, we get password **vkspyqrxfg**.

Programs :

generatePlaintext.py : It generates the plaintext in the pattern mentioned in above text and creates file $inputPlaintexts.txt$.

generateCiphertext.py : To generate ciphertexts corresponding to input plain texts.

script.sh : It first connects to server, then generates cipher texts corresponding to input plaintexts using a program $generateCiphertext.py$ which takes $inputPlaintexts.txt$ as input.

keyGeneration.py : It calculates linear transformation matrix A and exponentiation vector E.

generatePassword.py : It uses A and E calculated in $keyGeneration.py$ to find the decrypted password.

(pyfinite package is used for the operations over field. If not installed can give error.)

No files uploaded

Q4 Password

5 Points

What was the final command used to clear this level?

vkspyqrxfg

Q5 Codes

0 Points