Q1 Teamname

0 Points

Stuxnet

Q2 Commands

15 Points

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

exit1, exit2, exit4, exit3, exit1, exit4, exit4, exit2, exit2, exit1, read

Q3 Analysis

60 Points

Give a detailed description of the cryptanalysis used to figure out the password. (Explain in less than 150 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 first screen exit1 exit2 exit3 exit4 were open, on entering either exit1 exit3 exit4 on the first screen we got a same hex code and on entering exit2 we got a different hex code. At first we were not able to decide what to do with those Hex codes but later figured out that they may represent encoded text forms. So we converted hex values to ascii to get the text form. On the first screen while entering exit1/exit3/exit4 we got this hex string "59 6f 75 20 73 65 65 20" which when converted to ascii gives "You see" and on entering exit2 in the first screen we got this hex string "61 20 47 6f 6c 64 2d 42" which when converted to ascii gave "a Gold-B". After doing exit1 on the first screen we saw that only exit2 is only working in the second screen which produces the same output of "61 20 47 6f 6c 64 2d 42" which when converted to ascii gave "a Gold-B" that we were getting in the first screen. Since "You see a Gold-B" seems like starting of a suitable sentence we concluded that the first command to be used must be either exit1/exit3/exit4 followed by exit2 in the second screen. Then in the next screens only one of the 4 exits from exit1 to exit4 was working and others were taking us to the previous screen. We have listed the hex value encountered on different screens on giving different exits respectively

>exit1- 59 6f 75 20 73 65 65 20 - You see
>exit2- 61 20 47 6f 6c 64 2d 42 - a Gold-B
>exit4- 75 67 20 69 6e 20 6f 6e - ug in on
>exit3- 65 20 63 6f 72 6e 65 72 - e corner
>exit1- 2e 20 49 74 20 69 73 20 - . It is
>exit4- 74 68 65 20 6b 65 79 20 - the key
>exit4- 74 6f 20 61 20 74 72 65 -   to a tre
>exit2- 61 73 75 72 65 20 66 6f -   asure fo
>exit2- 75 6e 64 20 62 79   -   und by

>exit1- We got no hex here, and we tried exit1,exit2,exit3,exit4 but all were taking us to the previous steps. So we tried "read" and we got this

You see the following written on the panel:

n =
8436444373572503486440255453382627917470389343976334334386326034275667860921689509377926302880924650595564757217668266944527000881648177170141755476887128502044240300164925440505830343990622920190959934866956569753433165201951640951480026588738853928338105393743349699444214641968202764907970498260085751709

Stuxnet: This door has RSA encryption with exponent 5 and the password is:

2370178774682911039678909490731983030553818037642728
322629590658530188954399653341053938177968436688097
0896279018807100530176651625086988655210858554133345
9062725610277981714409231479601650948919804527578526
85707020289384698322665347609905744582248157246932
007978339129630067022987966706955482598869800151693

We know that RSA works as follows

Encryption: $c = m^e \; mod \; n$
Decryption: $m = c^d \; mod \; n$

At first, we have thought of few ways to decrypt the password but they were not possible due to the various constraints. For example, we have thought of at first trying to find factors of n, which we realized is impossible as the size of n is very large. We also thought of finding 'd', but since n is very large it was not possible to factor n and hence phi(n) could not have been found. And as we require phi(n) to find d hence it is not possible to find 'd'. It is given that the public exponent is 5, which is very small, so we tried to exploit this fact and deployed the low exponent attack aka Coppersmith Algorithm. Now, this coppersmith algorithm takes polynomial as input thus we need to formulate the same. To do this, we must first check whether any padding is added to the message or not, which can be done by checking if $c^{1/e}$ is an integer or not. After computing, we found out that padding is added. Say 'p' is the padding, so the final equation can be written as $(p + m)^e = c \; mod \; n$. In the above equation, e,c,n are given and we have also found the padding by converting the hex values to ascii which when joined turns out to be-->

"You see a Gold-Bug in one corner. It is the key to a treasure found by "

---------$Coppersmith's \;\; Theorem$----------

Let N be an integer and f be a polynomial of degree δ. Given N and f one can recover polynomial-time all x_0 such that $f(x_0) = 0$ mod N and $x_0 < N^{1/\delta}$ So we can form the problem as follows $f(M) = (p + M)^e$ mod N. We have referred the code from [1] and have modified it in the following way.

1. Converted the padding 'p' to its binary form 'p_bin'
2. We don't know the length of the password M, but from our assumption, $x_0 < N^{(1/e)} = 10^6$ thus M cannot be longer than 200 bits
3. So the final polynomial equation becomes : $((p_bin << length_m) + m)^e - c$
4. Root of the above polynomial is the required password and can be calculated using Coppersmith's algorithm and Lattice reduction

We found the root to be 100001001000000011010000111010101100010010000010110110000100001

The length of the root is 63, but to convert it to ASCII we need to divide it in 8 bit chunks so we padded it with a 0 at the MSB. So the eight 8 bit chunks were as following--> 01000010 01000000 01101000 01110101 01100010 01000001 01101100 00100001 which when converted to ASCII gave the password as 'B@hubAl!'

NOTE : We have used SageMath framework for our code as it contains all the required libraries.

References--> https://github.com/mimoo/RSA-and-LLL-attacks/blob/master/coppersmith.sage [1] ,https://web.eecs.umich.edu/~cpeikert/lic13/lec04.pdf [2], https://www.math.arizona.edu/~ura-reports/022/McCallum_group/DyerFinal.pdf [3]

No files uploaded

Q4 Password

25 Points

What was the final command used to clear this level?

B@hubAl!

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_assign_6.py Download

```
#!/usr/bin/env python
# coding: utf-8

# In[5]:


e = 5
N = 843644437357250348644025545338262791747038934397633433438632603
C = 237017877468291103967890949073198303055381803764272832262959065

# RSA known parameters
ZmodN = Zmod(N);


def coppersmith_howgrave_univariate(pol, modulus, beta, mm, tt, XX):
    # Defining variables
    gg = []
    new_pol = 0
    roots = []

    dd = pol.degree()

    polZ = pol.change_ring(ZZ)
    x = polZ.parent().gen()
    a = x * XX

    # Here, polynomial computation takes place

    for ii in range(mm):
        for jj in range(dd):
```

```
            temp1 = (a) ** jj
            temp2 = mm - ii
            temp3 = polZ(a) ** ii
            gg.append(temp1 * modulus ** (temp2) * temp3)
    for ii in range(tt):
        result = (a) ** ii * polZ(a) ** mm
        gg.append(result)

    # constructing a lattice B

    BB = Matrix(ZZ, dd * mm + tt)

    for ii in range(dd * mm + tt):
        for jj in range(ii + 1):
            temp = gg[ii][jj]
            BB[ii, jj] = temp

    # LLL
    BB = BB.LLL()

    # transform shortest vector in polynomial

    for ii in range(dd * mm + tt):
        a = x ** ii
        b = BB[0, ii]
        c = XX ** ii
        new_pol += a * b / c

    # Now we will dinf the factor polynomial
    potential_roots = new_pol.roots()

    # test roots

    for root in potential_roots:
        if root[0].is_integer():
            a = ZZ(root[0])
            result = polZ(a)
            if gcd(modulus, result) < modulus ^ beta:
                pass
            else:
                roots.append(a)

    return roots


def break_RSA(p_str, max_length_M):
    global e, C, ZmodN

    p_binary_str = ''.join(['{0:08b}'.format(ord(x)) for x in
```

```
p_str])

    for length_M in range(0, max_length_M + 1, 4):  # size of the root

        # Problem to equation (default)
        P.< M > = PolynomialRing(ZmodN)  # , implementation='NTL')
        temp = (int(p_binary_str, 2) << length_M)
        pol = (temp + M) ^ e - C
        dd = pol.degree()

        # Tweak those
        beta = 1
        epsilon = beta / 7
        a = dd * epsilon
        mm = ceil(beta ** 2 / a)
        b = dd * mm
        c = 1 / beta
        tt = floor(b * (c - 1))
        XX = ceil(N ** ((beta ** 2 / dd) - epsilon))

        roots = coppersmith_howgrave_univariate(pol, N, beta, mm, tt, XX)

        if not roots:
            pass
        else:
            print("Root is :", ' {0:b}'.format(roots[0]))
            return


if __name__ == "__main__":
    break_RSA("You see a Gold-Bug in one corner. It is the key to a treasure found by ", 200)


# In[ ]:





# In[ ]:
```

```
# In[ ]:





# In[ ]:
```

Assignment 6

● GRADED

GROUP

MAYANK BANSAL
HIRAK MONDAL
YASH SARASWAT

View or edit group

TOTAL POINTS

100 / 100 pts

QUESTION 1

Teamname **0** / 0 pts

QUESTION 2

Commands **15** / 15 pts

QUESTION 3

Analysis **60** / 60 pts

QUESTION 4

Password **25** / 25 pts

QUESTION 5

Codes **0** / 0 pts