Q1 Teamname
0 Points

CryptoCreeks

Q2 Commands
15 Points

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

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.)

After entering into the small chamber, we noticed that there were several exits present and only one of those exits did not have a door to it . Going close to it , we saw that this was "exit2" and hence we used the command "exit2" to exit from this chamber. Next we entered a chamber that had five exits out of which only "exit5" was closed. Hence ,we tried some commands from "exit1" to "exit4" .When we were trying the various set of exit commands ,

we came across several sets of hexadecimal numbers . So , we followed such a sequence of exits , which lead to unique occurence of the sets of the hexadecimal numbers. We stored the sets of hexadecimal numbers also which we encountered on this path. The suitable set of combinations we got which helped us reach the question statement is (exit2) -> (exit4) -> (exit3) -> (exit1)-> (exit4)->(exit4)-> (exit2)-> (exit2) -> (exit1) . After trying exit1, no such set of hexadecimal numbers appeared this time. So we tried "read" command and hence we were able to get the screen with the required question statement .

n=843644437357250348644025545338262791747038934397633433438632603427566
786092168950937792630288092465059556475721766826694452700088164817717014175547688712850204424030016492544050583034399062292019095993486695656
975343316520195164095148002658873885392833810539374334969944421464196820
27649079704982600857517093

CryptoCreeks : This door has RSA encryption with exponent 5 and the password is :
2370178774682911039678909490731983030553818037642728322629590658530188954399653341053938177968436688097089627901880710053017665162508698865521
0858554133345906272561027798171440923147960165094891980452757852685707020289384698322665347609905744582248157246932007978339129630067022987966
7069554825988698800151693

The RSA decryption algorithm is as follows ::

$$Plaintext = (Ciphertext)^d mod N$$

Now , it seemed impossible to perform the decryption since the value of d was unknown and although N was available with us , it was extremely large and hence impossible to use it or find its factors. Hence , we knew we need to see something else now . We then saw the value of public exponent e, provided to us. By first looking at the exponent , we got the first hint that since the value of the exponent is small , hence a low-exponent attack can be used in this case. Now, this can be solved using two algorithms, firstly using the algorithm

describe in the following paper , "Finding Small Roots of Univariate Modular Equations" by Nicholas Howgrave , and secondly the Coppersmith algorithm.

Next , we thought of checking whether the value of $(Ciphertext)^{1/e}$ was integral or not . This is because if the value is integral , then no padding would be used else we need to apply the concept of padding also. We computed its value and found that it was not an integral value and hence we concluded that padding would be used here.

The original encryption formula ::

$$Ciphertext = (Plaintext)^e \, modN$$ --
(1)

Encryption formula with padding of the form $2^{len} * padding$ ::

$$(Plaintext + (2^{len} * padding))^e = (Ciphertext) modN$$ ------------------------------------
(2)

The algorithm described by Nicholas is used for finding roots, $|x_0| < N^{1/k}$, and thus, the length of $Plaintext$ can be atmost around 200bits. So the value of the variable len , will be in the range of (1, 200). So we iterated over the possible values of $len.$
 After this we tried to find a smaller polynomial with root being $Plaintext$ and which could be solved in Z.
The paper we referred mentioned that the value of h $\geq$ 2 , so we chose a random value of h, and a value of X such that $|x_0| \leq X$. The value of k = degree of the polynomial (e) = 5 . Next , we tried to construct a lattice of polynomials, since the number of bits in N were large enough to be not possible to factorise it. We defined the lower triangular matrix m , with dimensions $hk * hk$. The value of m_{ij} can be defined as ::

$$m_{ij} = e_{ij} * X^{j-1}$$ --(3)

where e_{ij} is the coefficient of x^{j-1} in the expression

$$q_{uv}(x) = N^{h-1-v} * x^u * (p(x))^v \ \text{---(4)}$$

where v = floor((i-1)/k) and u = ((i-1)-kv)

We then also analyzed that

$$q_{uv}(x_0) = 0 mod(N^{h-1}) \ , \text{for all } u, v \geq 0. \ \text{-------------------------------------(5)}$$

Then we used the LLL algorithm to find the roots of the polynomial as mentioned in the paper. Next , we tried different values of len and $padding$ to get the appropriate roots .for this we tried different types of strings for the padding such as ::

1) CryptoCreeks : This door has RSA encryption with exponent 5 and the password is :
2) This door has RSA encryption with exponent 5 and the password is :
3) CryptoCreeks : This door has RSA encryption with exponent 5 and the password is
and many more ....

So , we got reminded of the sets of hexidecimal values which we had stored when we were trying different exit commands. The set of strings we got is as follows ::

t=["59 6f 75 20 73 65 65 20", "61 20 47 6f 6c 64 2d 42","75 67 20 69 6e 20 6f 6e","65 20 63 6f 72 6e 65 72","2e 20 49 74 20 69 73 20","74 68 65 20 6b 65 79 20","74 6f 20 61 20 74 72 65","61 73 75 72 65 20 66 6f","75 6e 64 20 62 79"]

As we can see, these values are in hexadecimal format and so we converted these into integers and then to binary and then to ASCII values and from the ASCII values , we decoded the strings corresponding to them.
The sentence corresponding to each value of the set we got is as follows ::

A) 59 6f 75 20 73 65 65 20 - "You see "
B) 61 20 47 6f 6c 64 2d 42 - "a Gold-B"
C) 75 67 20 69 6e 20 6f 6e - "ug in on"
D) 65 20 63 6f 72 6e 65 72 - "e corner"
E) 2e 20 49 74 20 69 73 20 - ". It is "
F) 74 68 65 20 6b 65 79 20 - "the key "
G) 74 6f 20 61 20 74 72 65 - "to a tre"
H) 61 73 75 72 65 20 66 6f - "asure fo"
I) 75 6e 64 20 62 79   - "und by"

On concatenating the sentences present above , we got the following sentence :::

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

We then tried giving this string as input to the code but on conversion of roots from integer to binary to ascii characters , taking 8 bits at a time , we got a password but this was incorrect and hence we tried again by appending a space to the end of our sentence,

`"You see a Gold-Bug in one corner. It is the key to a treasure found by "`
which gave appropriate roots , but on converting them to binary , it gave 63 length binary value , so we appended a "0" to the start and then on conversion of which to ASCII , gave the correct password which helped us in clearing the level.

The value of len = 64
The obtained integral root :: 4773930458381642785
The binary obtained on conversion of integral roots to binary ::
100001001000000011010000111010101100010010000010110110000100001

This did not gave appropriate password and hence the **correct binary** with the appended zero ::

0100001001000000011010000111010101100010010000010110110000100001

The resultant on converting using ASCII character codes is

PASSWORD:: B@hubA1!

We also tried a 2nd approach using the Coppersmith's Algorithm to find the value of the unknowns present in the above (2) equation.

Coppersmith's Algorithm :::

Let N be an integer and f be a polynomial of degree e=5. Given N and f, one can recover in polynomial

time all x_0 such that $f(x_0) = 0 \, mod N$ and $x_0 < N^{1/\delta}$

Our problem can be shown as below ::

$$f(Plaintext) = (Plaintext + padding)^e \, mod N$$

For solving this polynomial we used the coppersmith.sage code available on
https://github.com/mimoo/RSA-and-LLL-attacks/

In this code , we started with a custom padding , which we further converted to padding_binary . Next , although the value of the plaintext is unknown , we have already assumed the value of the $x_0 < N^{1/e}$ and hence we can say that the length of our Plaintext , i.e. the required password will be of atmost 200 bits.

Thus , our final equation of the required polynomial to be fed as input to the algorithm will be :::

$$polynomial = (Plaintext + (padding\_binary << len))^e \, mod N - Ciphertext$$

Now , the roots of this algorithm can be found out by using the Coppersmith's Algorithm and the concept of Lattice reduction. The root which we will obtain from this as a result , will be our required password. We used the same sentence for padding, i.e.,

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

and got the root. After getting the root, we repeated the same procedure as the above algorithm.

So, the final password we got from both the algorithms was:

PASSWORD:: `B@hubAl!`

REFERENCES:::
1. For 1st algorithm :::
"Finding Small Roots of Univariate Modular Equations" by Nicholas Howgrave."
2. For coppersmith algorithm ::
https://github.com/mimoo/RSA-and-LLL-attacks/

📄 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.zip	⬇ Download
1	Binary file hidden. You can download it using the button above.

Assignment 6

GROUP

SHRUTI WASNIK

TANISHA RASTOGI

✏ 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 |