

## Q1 Teamname

0 Points

NULL

## 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.)

### 1. Finding the padding

While we were navigating through the cave to reach the ciphertext, we noticed there were numbers written on the screen at the end of each message. After observing them closely, we noticed that they were ASCII values of printable characters. Then, we used them to navigate through the cave to reach ciphertext by constructing a meaningful sentence along the way. By the time we reached the ciphertext the formed sentence was "You see a Gold-Bug in one corner. It is the key to a treasure found by" Since the sentence is incomplete, we guessed that the missing part of the sentence might be the password and this incomplete sentence might be the padding used to make sure that  $M^e$  is sufficiently larger than  $N$ . We did further analysis using these assumptions and were able to find the password.

### 2.Approaching the Problem

The message on the screen stated that the password was encrypted using

RSA encryption with values of exponent,  $N$  and encrypted text  $C$  given below:

$$N = 84364443735725034864402554533826279174703893439763345$$

$$C = 23701787746829110396789094907319830305538180376427283$$

$$e = 5$$

$$M^e \bmod N = C$$

where  $M = \text{padding} + \text{password}$  (+ refers to string concatenation here). Since there is no straightforward way to extract decryption exponent  $d$  from the information above without factoring  $N$ , we decided to use Lattice-Based cryptanalysis which was discussed in Lecture 13.

Firstly we expected the password to be at most 10-15 characters, Let  $L$  be the number of bits in the password. Let  $A = (\text{padding} \ll L)$  which is padding shifted by  $L$  bits to the left. So we basically need to solve the following modular equation to obtain the password.

$$(x + A)^e - C = 0 \bmod N$$

In general, there is no efficient way of solving the above equation without knowing the factorization of  $N$ . But Since we expect the password to be quite small compared to  $N$ , we can exploit that to find the password. We used Coppersmith's theorem which states that for any polynomial  $f(x)$  of degree  $n$  whose coefficients are integers, all the roots which are  $< N^{\frac{1}{n}}$  of  $f(x) = 0 \bmod N$  can be found efficiently and gives an algorithm to do so. The polynomial in our case is  $f(x) = (x + A)^e - C$  and since we expect the password to be around 100 bits long which is less than  $N^{\frac{1}{e}}$  (here  $N$  is around 1000 bits long), we can use the coppersmith's theorem to find the password.

## 2. Constructing the Basis

The core of the algorithm is to find another polynomial  $h(x)$  such that the password is also the root of  $h(x) = 0$  over integers (not modulo  $N$ ) and then use the regular methods (binary search in our case) for finding roots of polynomials over integers. To do that, we construct a set of polynomials for which password is also a root modulo  $N$ , and then since password would also be a root of their integer linear combination, we treat these polynomials as the basis of lattice and find the vector given by LLL algorithm and then prove that this polynomial can serve as our  $h(x)$ .

Let  $K$  be an upper bound on the password. We construct the following polynomials given by  $g_{u,v}(Kx) = N^{1-v} * f(Kx)^v * (Kx)^u$  for  $u \in \{0, \dots, e-1\}$  and  $v \in \{0, 1\}$ . It is easy to see that password is the root of each of these polynomials modulo  $N$ . Now we find the volume of the lattice formed by these polynomials as a basis (which is the determinant of the matrix formed by taking these basis vectors as row vectors). By construction, the matrix formed by putting the coefficients of these vectors as rows is lower triangular and so the determinant of the matrix is the product of diagonal elements which is nothing but the product of the coefficient of highest power in each of these polynomials which is given by

$$volume(L) = \prod_{u,v} K^{ev+u} * N^{1-v} = K^{45} N^5$$

Let  $v$  be the vector obtained by LLL reduction, we know that

$$|v| \leq 2^{\frac{D-1}{2}} * \sqrt{D} * volume(L)^{1/D}$$

where in our case  $D = 10$ .

so we have  $|v| \leq 2^{4.5} * \sqrt{10} * K^{4.5} * N^{0.5}$ . If we take  $K = 2^{100}$  we can see that  $|v| < \frac{N}{10}$ . So this basically means that if we treat the coordinates of  $v$  as coefficients of a polynomial  $h(Kx)$ , we have  $\sum_i |h_i| * K^i < N$ . Since we assumed password to be less than  $K$ , we have  $\sum_i |h_i| * (password)^i < N$ . Since password is root of  $h(x) = 0 \pmod{N}$  by construction, we now have password is root of  $h(x) = 0$  without the modulus.

We implemented the above algorithm in python using "fpylll" library and observed that  $h(x)$  is a polynomial of degree 9. we tried different lengths of passwords like 8,9,10 chars and found that 9 yielded the password. Since  $h(x)$  is an odd degree polynomial, we used binary search to find the root of

$h(x)$  which can be seen in the code section below.

The password came out to be " B@hubAI!" and the complete sentence was "You see a Gold-Bug in one corner. It is the key to a treasure found by B@hubAI!".

References:

<https://web.eecs.umich.edu/~cpeikert/lic13/lec04.pdf>

 No files uploaded

## Q4 Password

25 Points

What was the final command used to clear this level?

B@hubAI!

## 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.

▼ solver.py

 Download

```
1  #!/usr/bin/python3
2  from fpylll import *
3  from math import sqrt
4
5  N =
8436444373572503486440255453382627917470389343976334334386326034275
6  C =
2370178774682911039678909490731983030553818037642728322629590658530
7  e = 5
8  padding =
0x596f7520736565206120476f6c642d42756720696e206f6e6520636f726e65722
9  L = 72
10 K = 2**100
11 m = 1
12
13 assert(2**(4.5) * sqrt(10) * K**(4.5) * N**(0.5) < N/10)
14
15 # Helper Functions to deal with polynomials
16
```

```

17 def value(poly, x):
18     return sum([x**i*poly[i] for i in range(len(poly))])
19
20
21 def add(poly1, poly2):
22     length = max(len(poly1), len(poly2))
23     res = [0] * length
24     for i in range(length):
25         if i < len(poly1):
26             res[i] += poly1[i]
27         if i < len(poly2):
28             res[i] += poly2[i]
29     return res
30
31
32 def mult(poly1, poly2):
33     length = len(poly1) + len(poly2) - 1
34     res = [0] * length
35     for i in range(len(poly1)):
36         for j in range(len(poly2)):
37             res[i + j] += poly1[i] * poly2[j]
38     return res
39
40
41 def power(poly, k):
42     res = [1]
43     while k > 0:
44         if k % 2 == 1:
45             res = mult(res, poly)
46             poly = mult(poly, poly)
47             k >>= 1
48     return res
49
50
51 # constructing the basis and doing LLL reduction
52 padding <=<= L
53
54 f = add(power([padding, 1], e), [N - C])
55 f = [x % N for x in f]
56 for i in range(len(f)):
57     f[i] = K**i * f[i]
58
59 A = IntegerMatrix(e*(m+1), e*(m+1))
60 i = 0
61 for u in range(e):
62     for v in range(m + 1):
63         g = mult([N ** (m - v)], power(f, v))
64         foo = [0] * (u + 1)
65         foo[-1] = K ** u
66         g = mult(g, foo)
67
68         while len(g) < e * (m + 1):
69             g.append(0)

```

```

69         for j in range(e*(m+1)):
70             A[i,j] = g[j]
71         i += 1
72
73
74 LLL.reduction(A)
75
76 # shortest vector given by LLL
77 v = [0]*len(A[0])
78 for i in range(len(v)):
79     v[i] = A[0][i] // (K**i)
80
81
82 # using binary search to find the root
83 lo = -(1 << L)
84 hi = (1 << L)
85
86 assert(value(v, lo)*value(v, hi) < 0)
87
88 ITERS = 1000
89
90 for _ in range(ITERS):
91     mid = (lo + hi) // 2
92     mid_val = value(v, mid)
93     if mid_val == 0:
94         break
95     if mid_val*value(v, hi) > 0:
96         hi = mid
97     else:
98         lo = mid
99
100 root = (lo + hi)//2
101 padding += root
102 password = ""
103
104 while root > 0:
105     password += chr(root % 256)
106     root >>= 8
107
108 password = password[::-1]
109
110 print('Password is', password)
111
112
113 fulltext = ""
114 while padding > 0:
115     fulltext += chr(padding % 256)
116     padding >>= 8
117
118 print(fulltext[::-1])
119

```

## Assignment 6


● **UNGRADED**

### GROUP

AJAY PRAJAPATI

A5 - SURYADEVARA SAI KRISHNA

A11 - GARIMELLA MOHAN RAGHU

 [View or edit group](#)

### TOTAL POINTS

- / **100 pts**

### QUESTION 1

[Teamname](#) 0 pts

### QUESTION 2

[Commands](#) 15 pts

### QUESTION 3

[Analysis](#) 60 pts

### QUESTION 4

[Password](#) 25 pts

### QUESTION 5

[Codes](#) 0 pts