Q1 Team Name

0 Points

Enigma

Q2 Commands

10 Points

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

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

Q3 Analysis

60 Points

Give a detailed description of the cryptanalysis used to figure out the password. (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 It said that exit 2 is open and we have come from exit 1 that means exit1 and exit2 are open, so we entered both exit2 and exit1 command on the screen to go ahead but that exit2 didn't lead us anywhere , only exit1 worked. After command got executed it showed some numbers looks like hexadecimal base.So after this point we've checked various exits to go ahead but only one exit gate works which lead us to different stage and all other exits bring us to some other previous stage in the cave.
So we followed same strategy at each step trying various exits commands to reach ahead. The exit commands we

followed were and their corresponding hexadecimals numbers appeared after entering that command-

exit1: 59 6f 75 20 73 65 65
exit3: 20 61 20 47 6f 6c 64
exit4: 2d 42 75 67 20 69 6e
exit4: 20 6f 6e 65 20 63 6f
exit1: 72 6e 65 72 2e 20 49
exit3: 74 20 69 73 20 74 68
exit4: 65 20 6b 65 79 20 74
exit1: 6f 20 61 20 74 72 65
exit 3: 61 73 75 72 65 20 66
exit2: 6f 75 6e 64 20 62 79

looking at all these hexadecimal numbers we tried to combine them all and change it to decimal to get an idea what it really means but the decimal number we not making much sense so we thought of converting the decimal numbers to their respective ascii values to get some English text so that we could understand what it is trying to say. It came to be –“ You see a Gold-Bug in one corner. It is the key to a treasure found by “. Now we have entered “read” to move further ahead then this showed on the screen-

n =
84364443735725034864402554533826279174703893439763343343863260342756678609216895093779263028809246505955647572176682669445270008816481771701417554768871285020442403001649254405058303439906229201909599348669565697534331652019516409514800265887388539283381053937433496994442146419682027649079704982600857517093

Enigma: This door has RSA encryption with exponent 5 and the password is
27134610716011038218289414701061334057796564566307454673763149698383784153702624439723336393385481090925713689618104277351967643197850119631689910

8123144135744030852204332692578964563691467998555118965989395102312935106117277206828680902662136323245081435841536004063264973751158316537631790101245331306134 21

RSA encryption is used with exponent(e) 5, the equation becomes

$$encryption = C = M^e mod N$$
$$Decryption = M = C^d mod N$$

this can be cracked by finding factors of n but here n is very large(976 bits) so finding factors is not trivial hence we cannot compute phi(N) hence cannot find d ,so we cannot use this method. So instead we could use simple RSA attacks as exponent is very low and padding is used , we went with Coppersmith's algorithm (stereotyped message attack)

So pre-requisite of coppersmith's algo is to check whether cipher text has padding with it or not so to check that we calculate $C^{1/e}$ where C is ciphertext and e is exponent which came out to be non-integer so it means that padding has been used with original message to generate cipher text, so to find padding used we tried text from above hexadecimal which we got but plain text password which came from it wasn't accepted so we went with pretext before the cipher password shown on the screen P = "Enigma: This door has RSA encryption with exponent 5 and the password is ". The Equation of becomes for RSA is-

$$(M + P)^e mod N = C$$

Coppersmith's Algorithm

Let n be an integer and f be a polynomial of degree d. Given n and f, all integers x_0 such that $f(x_0) = 0 mod n$ and $x_0 < n^{1/d}$ can recover in polynomial time where (1/d)>0, so we can impose our problem just like this.

Let's pose our problem as a polynomial function $f(x) =$

$(x+p)^e - CmodN$ where x is the polynomial ring of integers over mod N and roots of this function will be our password.
We wrote the code for solving above question in ===code.py--- . The steps performed by the code.py are translate paddings into the respective binary notation(binary_padding). The value of x(password) is unknown but we have assumed x<(1/e) which is approx. 200 bits. Therefore the final polynomial equation becomes ((binary_padding)<<$passwordlength$) $+ x)^e - C$ where password length iterate through 1 to 200 in increment of 4. We assumed x to be a multiple of 4 , and hence iterated 0 bits to 200 bits with an increment of 4 bits.
Now, finding roots of polynomial (f(x)) over Ring of Integer Modulo N using Coppersmith's
Algorithm is difficult. For that, Nicholas Howgrave-Graham proposed an
efficient technique for finding small roots of univariate modular polynomials.
Howgrave-graham theorem states that -
Let (g(x))be an univariate polynomial with n monomials.
Further, let m be a positive integer.
Suppose that

$g(x_0) = 0(modN^m)$ where $|x_0| <= X$
$||g(xX|| < \frac{N^m}{\sqrt{n}}$

Then g(x)=0 is true for integers
According to Howgrave, we need to find a polynomial that shares the same root as our
function f but modulo N^m.
Now, we referred to the steps from finding such polynomial to generating desired roots,
described in the alternative technique proposed by Howgrave-Graham in this paper [1].
This steps are as follows :
1. Creating polynomial f(x) over Ring of Integers modulo N.
2. Generating polynomial g(x)over Ring of Integers from f(x).

3. Constructing lattice L from coefficients of polynomial g(x) (basis vector).

For h=3 and natural number X, defining lattice L of (hk) x (hk) dimension. Each entry L(i,j) (where i and j are starting from 0) is given by $e_{i,j}X^j$, where $e_{i,j}$ is the coefficient of x^j of polynomial
$g(x) = N^{h-1-v}x^u(f(x))^v$ where v = ceil(i/e) and u = i - e*v

4. Performing LLL Reduction on to get shortest vector.
5. Converting shortest vector into polynomial.
6. Finding roots of the shortest vector polynomial.
7. This root will be our required password.

Using above algorithm we got the root in decimal , we converted it to binary which come out to be of 63 bits we have padded one extra 0 to left of it, now taking 8 bits at a time we are getting final decrypted password as: C8YP7oLo6Y.

References-
1.https://link.springer.com/content/pdf/10.1007/BFb0024458.pdf
2.https://github.com/mimoo/RSA-and-LLL-attacks

No files uploaded

Q4 Password

10 Points

What was the final command used to clear this level?

C8YP7oLo6Y

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 for the entire assignment.

assign_6.py Download

```
1
2   """ Run this code in 1)   >>
    https://sagecell.sagemath.org/     <<
3   # OR
4   # if not then in  CoCalc using jupyter notebook and
    change the kernel to SageMath 9.5
5
6   """
7
8
9   def calculate_m(pol):
10      term_1 = 1**2
11      term_2 = pol.degree()
12      term_3 = 1/7
13      ans = ceil(term_1/(term_2 * term_3))
14      return ans
15
16
17  def calculate_t(beta, x, pol):
18      term_1 = pol.degree()
19      term_2 = 1/beta
20      ans = floor(term_1 * (term_2 - 1)* x)
21      return ans
22
23
24  def calculate_x(pol, N):
25      term_1 = 1/7
26      term_2 = pol.degree()
27      term_3 = 1**2
28      ans = ceil(N**((term_3/term_2) - term_1))
29      return ans
30
31
32  def get_unicode(p):
33      hex_p = []
34      for x in p:
```

```
        hex_p.append('{0:08b}'.format(ord(x)))
    bin_p = ''.join(hex_p)
    return bin_p


def get_Polynomial(e, fx, M, C):
    term_1 = fx + M
    term_2 = term_1^e
    ans = term_2 - C
    return ans

def get_key_in_binary(x):
    pp = x[0]
    ans = bin(pp)
    ans = ans[2:]
    return ans


# ## The Copper Hograve algorithm

def Coppersmith_hograve_univariate(X, polynomial, t, modulus, m, beta):

    polynomials = [] # Polynomials
    polynomial = polynomial.change_ring(ZZ)

    x = polynomial.change_ring(ZZ).parent().gen()
    degree = polynomial.degree()

    for i in range(m):
        for j in range(degree):
            ppop = x*X
            polynomials = polynomials + [ppop**j * polynomial(ppop)**i * modulus**(m - i) ]

    for i in range(t):
        ppop = x*X
        polynomials = polynomials + [ppop**i * polynomial(ppop)**m]

    y = t + m* degree
    lattice_B = Matrix(ZZ, y)

    polynomial = 0
    for i in range(y):
        for j in range(i+1):
```

```
78              lattice_B[i, j] = polynomials[i][j]
79      lattice_B = lattice_B.LLL()
80
81      for i in range(y):
82          term_1 = x**i
83          term_2 = X**i
84          polynomial = polynomial + term_1 *
    lattice_B[0, i] / term_2
85
86      possible_roots = polynomial.roots()
87
88      roots = []
89      for root in possible_roots:
90          if root[0].is_integer():
91              result = polynomial(ZZ(root[0]))
92              if gcd(modulus, result) >=
    modulus^beta:
93                  roots = roots+[ZZ(root[0])]
94
95      return roots
96
97
98  def init():
99      unicode_p = get_unicode(p)
100     beta = 1
101     eps = beta/7
102     label = 0
103     return unicode_p,beta,eps,label
104
105
106 # ## RSA Algorithm for final answer
107
108
109 def run_RSA_Algo(N, e, p, C, len_M):
110     unicode_p,beta,eps,label=init()
111
112     for i in range(0, len_M+1, 4):
113         pop = 2**i
114         fx = (int(unicode_p, 2 )*pop)
115         P.<M> = PolynomialRing(Zmod(N))
116         poly = get_Polynomial(e, fx, M, C)
117         m = calculate_m(poly)
118         x = calculate_x(poly, N)
119         t = calculate_t(beta, m, poly )
120
121         roots = Coppersmith_hograve_univariate(x,
```

```
poly, t, N, m, beta)
        if roots:
            return(get_key_in_binary(roots))



# ## Parameters


n = 84364443735725034864402554533826279174703893439763344
e = 5
p = "Enigma: This door has RSA encryption with exponent 5 and the password is "
c = 27134610716011038218289414701061334057796564566307454
leng = 200


# ## Running the RSA Algorithm

roots = run_RSA_Algo(n, e, p, c, leng)
roots = '0' + roots


# ## changing binary root to its Ascii value to get the password


def text_from_bits(bits, encoding='utf-8', errors='surrogatepass'):
    n = int(bits, 2)
    return n.to_bytes((n.bit_length() + 7) // 8, 'big').decode(encoding, errors) or '\0'


print("Password for this assignment is : " ,text_from_bits(roots))
```

Assignment 6

GRADED

GROUP

Kajal Sethi
Gajender Sharma
Pranshu Sahijwani
View or edit group

TOTAL POINTS

80 / 80 pts

QUESTION 1

Team Name **0** / 0 pts

QUESTION 2

Commands **10** / 10 pts

QUESTION 3

Analysis **60** / 60 pts

QUESTION 4

Password **10** / 10 pts

QUESTION 5

Codes **0** / 0 pts