Q1 Team Name
0 Points

ANV

Q2 Commands
10 Points

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

Commands Used :-

- **exit1**
- **exit3**
- **exit4**
- **exit4**
- **exit1**
- **exit3**
- **exit4**
- **exit1**
- **exit3**
- **exit2**
- **read**

After executing the above commands in sequence, we reached the cip

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.)

■**Analysis :**

First, we pressed the command **exit1**. Then, some hexadecimal numbers were displayed on the screen. We converted those numbers to characters by assuming that they were ascii. Then, we got this following mapping:

a Gold 20 61 20 47 6f 6c 64

You see 59 6f 75 20 73 65 65

-Bug in 2d 42 75 67 20 69 6e

one co 20 6f 6e 65 20 63 6f

rner. I 72 6e 65 72 2e 20 49

t is th 74 20 69 73 20 74 68

e key t 65 20 6b 65 79 20 74

o a tre 6f 20 61 20 74 72 65

asure f 61 73 75 72 65 20 66

ound by 6f 75 6e 64 20 62 79

The message would be like:

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

We then pressed read. The following appeared:

$\mathbf{n = 84364443735725034864402554533826279174703893439763343343}$

ANV: This door has RSA encryption with exponent 5 and the passwo

We realized that the cryptosystem used here is **RSA with exponent 5**.

To decrypt this, we thought of using **CopperSmith Attack**

Coppersmith's attack describes a class of cryptographic attacks on the public-key cryptosystem RSA based on the Coppersmith method.
This Coppersmith's attack is generally used when the public exponent e is small.

Let $\mathbf{N}$ be an integer and $f \in \mathbb{Z}[x] f \in \mathbb{Z}$ [x] be a monic polynomial of degree d over the integers. Set $X = N^{\frac{1}{d}-\epsilon}$ for $\dfrac{1}{d} > \epsilon > 0$. Then, given $\langle N, f \rangle$, attacker (Eve) can efficiently find all integers $x_0 < X$ satisfying $f(x_0) \equiv 0 \pmod{N}$. The running time is dominated by the time it takes to run the LLL algorithm on a lattice of dimension $O(w) with w = \min\left\{\frac{1}{\epsilon}, \log_2 N\right\}$.
This theorem states the existence of an algorithm that can efficiently find all roots of $\mathbf{f \text{ modulo } N}$ that are smaller than $X = N^{\frac{1}{d}}$. As X gets smaller, the algorithm's runtime decreases. This theorem's strength is the ability to find all small roots of polynomials modulo a composite N.

Coppersmith attack uses **LLL reduction**.

In normal RSA, we have a ciphertext c, modulus N and a public exponent e. We find m such that m^e = c mod N.

In relaxed model of RSA, we have c = (m + x)^e, we know a part of the message m, but we do not know x. Coppersmith says that, if you are looking for N^1/e of the message, it is then a small root and you should be able to find it pretty quickly.
let our polynomial be f(x) = (m + x)^e - c which has a root we want to find modulo N

This m is padding. We did not have any idea of the padding at start. Then, we saw the ciphertext and saw this

ANV: This door has RSA encryption with exponent 5 and the passwo

We assumed that the padding might be similar to this. We manually tried the following paddings:

1:- ANV: This door has RSA encryption with exponent 5 and the pass

2:- ANV: This door has RSA encryption with exponent 5 and the pass

3:- ANV: This door has RSA encryption with exponent 5 and the pass

4:- ANV: This door has RSA encryption with exponent 5 and the pass

5:- This door has RSA encryption with exponent 5 and the password i

We found out that the correct padding was the following:

ANV: "This door has RSA encryption with exponent 5 and the passw

without quotes

We used **Sage Library** for implementing the coppersmith attack. We used some of code for the coppersmith attack implementation from this github link:

https://github.com/mimoo/RSA-and-LLL-attacks

We converted each character in the padding to ASCII and converted the total thing into binary padding and converted this to integer.
For finding the length of the unknown part, we tried all the lengths starting from 0 to 500.
For each of this length 'le' from 0 to 500, we left shift the padding by 'le' and add x to it.
That is pol = ((padding<<le) + x)^e - C
The unknown length we got was 88.
We got the solution to be the following:
10000001000011001110000101100101010000001101110110111101001

The length of this was 86. So, we ignored the first 6 bits and then converted each 8 bits to integer and considered these as ascii values and mapped them to characters to get the following password: **C8YP7oLo6Y**
We realized that the password looks like the word Cryptology.
The name of the sage script is decode.sage
The final message would be like:

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

■Resources used:
1:- https://github.com/mimoo/RSA-and-LLL-attacks
2:- https://www.cryptologie.net/article/222/implementation-of-cop

3:- https://en.wikipedia.org/wiki/Coppersmith%27s_attack

No files uploaded

Q4 Password

10 Points

What was the final command used to clear this level?

The password used to clear this level is :- **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.

▾ Makefile Download

```
run:
	sage decode.sage

```

▾ Readme.txt Download

```
TO run the attack, do the following:

	press "make" in the terminal in the directory of all the files without quotes.

It will take around 20 seconds to complete

```

▾ d.py Download

```
s=["20 61 20 47 6f 6c 64","59 6f 75 20 73 65 65"," 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"]
for j in s:
        r=""
        for i in j.split():
                r+=chr(int(i,16))
        print(r,j)
```

decode.sage   Download

```
def coppersmith_howgrave_univariate(pol, modulus, beta, mm, tt, XX):

    dd = pol.degree()
    nn = dd * mm + tt

    if not 0 < beta <= 1:
        raise ValueError("beta should belong in (0, 1]")

    if not pol.is_monic():
        raise ArithmeticError("Polynomial must be monic.")

    polZ = pol.change_ring(ZZ)
    x = polZ.parent().gen()
    gg = []
    for ii in range(mm):
        for jj in range(dd):
            gg.append((x * XX)**jj * modulus**(mm - ii) * polZ(x * XX)**ii)
    for ii in range(tt):
        gg.append((x * XX)**ii * polZ(x * XX)**mm)

    BB = Matrix(ZZ, nn)

    for ii in range(nn):
        for jj in range(ii+1):
            BB[ii, jj] = gg[ii][jj]

```

```
    # LLL
    BB = BB.LLL()

    # transform shortest vector in polynomial
    new_pol = 0
    for ii in range(nn):
        new_pol += x**ii * BB[0, ii] / XX**ii

    # factor polynomial
    potential_roots = new_pol.roots()

    # test roots
    roots = []
    for root in potential_roots:
        if root[0].is_integer():
            result = polZ(ZZ(root[0]))
            if gcd(modulus, result) >= modulus^beta:
                roots.append(ZZ(root[0]))


    return roots
e = 5
N =
843644437357250348644025545338262791747038934397633433438632603427566786092168950937792
C =
259250464085150344189639062896456753787636324055158863112997919061591661748817605562071

ZmodN = Zmod(N)

def decrypt(padding):
    z=""
    for i in padding:
        i=bin(ord(i))[2:]

        z+='0'*(8-len(i))+i
    padding = int(z,2)
```

```
    for le in range(0, 501):


        P.<x> = PolynomialRing(ZmodN)
        pol = ((padding<<le) + x)^e - C
        dd = pol.degree()
        beta = 1
        epsilon = beta / 7
        mm = ceil(beta**2 / (dd * epsilon))
        tt = floor(dd * mm * ((1/beta) - 1))
        XX = ceil(N**((beta**2/dd) - epsilon))

        roots = coppersmith_howgrave_univariate(pol, N, beta, mm, tt, XX)
        if roots!=[]:
            print("length of unknown part is: ",le)
            password=bin(roots[0])[2:]
            print ("A solution has been found out. It is : "+password)
            final_password=""
            for i in range(len(password)%8,len(password),8):
                final_password+=chr(int(password[i:i+8],2))
            print("\nDecrypted password is: ",final_password)
            return

    print('There is no solution for this padding')

padding="ANV: This door has RSA encryption with exponent 5 and the password is"
decrypt(padding)
```

Assignment 6

● GRADED

GROUP

Dibbu Amar Raja
Vikas
Idamakanti Venkata Nagarjun Reddy

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