Assignment 6

Graded

Group

ANANYAE KUMAR BHARTARI
ABHAYA PRATAP SINGH
NAVEEN KUMAR MATHUR
View or edit group

Total Points

95 / 100 pts

Question 1

Teamname

0 / 0 pts

- ✔ **+ 0 pts** Correct
- **+ 0 pts** Incorrect

Question 2

Commands

15 / 15 pts

- ✔ **+ 15 pts** Correct
- **+ 0 pts** Incorrect

Question 3

Analysis

55 / 60 pts

- ✔ **+ 15 pts** Figuring out the known part/padding of the message, i.e., the hexadecimal strings.
- ✔ **+ 20 pts** Briefly description the coppersmith attack (15 marks) and mentioning how the length of the unknown part was handled (5 marks).
- ✔ **+ 5 pts** Extracting the final password, i.e., converting the root into ASCII.
- ✔ **+ 20 pts** Properly implementing the above in the code.
- **+ 0 pts** Wrong answer or NA.
- **− 5 pts** not mentioned how the length of the unknown part was handled

Question 4

Password

25 / 25 pts

- ✔ **+ 5 pts** B@hubAl!.
- ✔ **+ 20 pts** Completing before deadline.
- **+ 0 pts** Incorrect

Question 5

Codes

0 / 0 pts

✔ **+ 0 pts** Correct

Q1 Teamname

0 Points

NAA

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.)

When we entered level 6, there was written about different exits numbered from 1 to 5. So we first enter the command “exit1”, after that we see some hexadecimal numbers and when we again type some command from exit1-exit4, we got some more numbers and finally at the end typing command “read” we reach to the panel where N and encrypted password is given.

Commands used in sequence:
exit1
exit2
exit4
exit3
exit1
exit4
exit4
exit2
exit2
exit1
read

We got
N=84364443735725034864402554533826279174703893439763343343863260342756678609216895093779263028809246505955647572176682669445270008816481771701417554768871285020442403001649254405058303439906229201909599348669565697534331652019516409514800265887388539283381053937433496994442146419682027649079704982600857517093
and
Encrypted Password (C) =
2370178774682911039678909490731983030553818037642728322629590658530188954399653341053938177968436688097089627901880710053017665162508698865521085855413334590627256102779817144092314796016509489198045275785268570702028938469832266534760990574458224815724693200797833912963006702298796670695548259886980015169

and
Exponent = 5

In RSA:

Encryption (C) = (M^e) mod N
Decryption (M) = (C^d) mod N

As N is very large we cannot factorize it but exponent(5) is small so we can use Coppersmith's Algorithm(low-exponent attack).
This algorithm requires a polynomial as an input. Thus we need to formulate the same. For this, we first need to check if any padding is added to the message. This can be done by checking if C^(1/e) is an integer or not.
Let p be the padding, so the equation becomes : (p+M)^e = C mod N.
We convert padding p into binary form p_bin and the polynomial becomes: ((p_bin<<length_M)+M)^e - C. Root of this polynomial is the password and can be calculated using Coppersmith's Algorithm and LLL(Lattice reduction).
Reference of code is given in the end.
When we type different commands like exit1-exit4, we get some hexadecimal numbers, so we convert them into characters using Ascii values and combine them which makes "You see a Gold-Bug in one corner. It is the key to a treasure found by" (without quotes) and think that it can be used as padding, so we use it as padding and find the binary string of 70 bits long.

Original Binary
100000010000100100000000110100001110101011000100100000010110110000100001

This is 70 bits long which is not a factor of 8(as the ASCII value of character is 8 bit long). So we try to reduce it to 64 bits by removing 6 digits from right and then left. When we remove it from the right, it does not give any readable result. So we reduce 6 digits from the left. and we got the binary string of 64 bits long.
Final Binary
010000100100000000110100001110101011000100100000010110110000100001

So we divide it into sets of 8 bits and find the corresponding character using Ascii value and we found the password.

Password = B@hubAl!

Coppersmith's Theorem:
Let N be an integer and f be a polynomial of degree D .
From given N,D we can recover in polynomial time all Xo such that f(Xo)=0 mod N and Xo<N/D.
So, our problem becomes f(M) = ((p + M)^e) mod N.

REFERENCE FOR CODE : 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.

Assignment6.sage

Download

```
def coppersmith_howgrave_univariate(pol, modulus, beta, mm, tt, XX):
    dd = pol.degree()
    nn = dd * mm + tt

    polZ = pol.change_ring(ZZ)
    x = polZ.parent().gen()

    # compute polynomials
    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)

    # construct lattice B
    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 =
```

```
8436444373572503486440255453382627917470389343976334334386326034275667860
C = 2370178774682911039678909490731983030553818037642728322629590658530188954
# RSA known parameters
ZmodN = Zmod(N);


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')
        pol = ((int(p_binary_str, 2) << length_M) + M) ^ e - C
        dd = pol.degree()

        # Tweak those
        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("Root is :", ' {0:b}'.format(roots[0]))
            return

    print('No solution found\n')


if __name__ == "__main__":
    print("The padding p using the hexadecimal numbers is : ", end="")

    a = [["59", "6f", "75", "20", "73", "65", "65", "20"], ["61", "20", "47", "6f", "6c", "64", "2d", "42"],
         # numbers found when we type commands exit1-exit4
         ["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"]]
    for i in range(8):
        for j in range(8):
```

```
            bytes_object = bytes.fromhex(a[i][j])
            ascii_string = bytes_object.decode("ASCII")
            print(ascii_string, end="")
    b = ["75", "6e", "64", "20", "62", "79"]
    for i in range(6):
        bytes_object = bytes.fromhex(b[i])
        ascii_string = bytes_object.decode("ASCII")
        print(ascii_string, end="")
    print()

    break_RSA("You see a Gold-Bug in one corner. It is the key to a treasure found by", 300)

    b = "0100000100100000011010000111010101100010010000010110110000100001"
    print("Password : ", end="")
    for i in range(0, 64, 8):
        x = b[i:i + 8]
        y = int(x, 2)
        print(chr(y), end="")
    print()
```