CS641A: Modern Cryptology

Assignment-6

Team name: codemeisters
Praveen(170504)
Vijit Malik(170791)
Hitesh Kumar(170305)

1. <u>GIVEN DATA FOR RSA :</u>

As mentioned in mail that it is RSA encryption and data were given as follow :
n=
84364443735725034864402554533826279174703893439763343343863260342756678609216895093779263028809246505955647572176682669445270008816481771701417554768871285020442403001649254405058303439906229201909599348669565697534331652019516409514800265887388539283381053937433496994442146419682027649079704982600857517093;
e=5;
Cipher
=5885119081935571454727589955844171566374613984724607561927074533865700705569837874063774277536176889970088885808705066261431830544306444889802650355675761034293849074136164369628505186726027856789699192735196455737497761964476363322989666851175243222252815921401317331985564535161939387143345555058174164329;

2. <u>BREAKING THE CIPHER:</u>

Then we analyzed the data and we saw that the public exponent e was very small. We search for how to solve these types of encryption. And we found that we can break low exponent rsa using the Coppersmith **Method.**

Then we found out that Coppersmith has two methods. One for some message known and other high bits known of q .

We have searched many things so we figure out that mostly half given password is given like “This password is...”

So we saw in the mail and found that the line given before the cipher "This door has RSA encryption with exponent 5 and the password is", is some kind of similar line as mentioned before. S0 we went to some known password method.

Using the first method if we know some bits of the message. You can find the rest of the message with this method.

The idea behind this method is this: -

The RSA model has a ciphertext c, a modulus N and a public exponent e. Find m such that

$$m^e = c \bmod N.$$

Now, we have the polynomial $c = (m + x)^e$, but we know some part of the message, m, but don't 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.

Our polynomial will be $f(x) = (m + x)^e - c$ which has a root we want to find *modulo N*.

Using this method of coppersmith we have a polynomial . *So now we have to find the root of this polynomial . To find the root of this polynomial we used the Howgrave-Graham algorithm . And we used this program :*

Imp :- Run the code with SageMath kernel

```
import time
debug = True

def matrix_overview(BB, bound):
    for ii in range(BB.dimensions()[0]):
        a = ('%02d ' % ii)
        for jj in range(BB.dimensions()[1]):
            a += '0' if BB[ii,jj] == 0 else 'X'
            a += ' '
        if BB[ii, ii] >= bound:
            a += '~'
#         print(a)

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

```
    # b|modulus, b >= modulus^beta , 0 < beta <= 1

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


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

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

    if debug:
        cond1 = RR(XX^(nn-1))
        cond2 = pow(modulus, beta*mm)
#         print ("* X^(n-1) < N^(beta*m) \n-> GOOD" if cond1 < cond2
else "* X^(n-1) >= N^(beta*m) \n-> NOT GOOD")

        cond2 = RR(modulus^(((2*beta*mm)/(nn-1)) -
((dd*mm*(mm+1))/(nn*(nn-1)))) / 2)
#         print ("* X <= M \n-> GOOD" if XX <= cond2 else "* X > M
\n-> NOT GOOD")

        detL = RR(modulus^(dd * mm * (mm + 1) / 2) * XX^(nn * (nn -
1) / 2))
#         print ("we can find a solution if 2^((n - 1)/4) *
det(L)^(1/n) < N^(beta*m) / sqrt(n)")
        cond1 = RR(2^((nn - 1)/4) * detL^(1/nn))
        cond2 = RR(modulus^(beta*mm) / sqrt(nn))
#         print ("* 2^((n - 1)/4) * det(L)^(1/n) < N^(beta*m) /
sqrt(n) \n-> SOLUTION WILL BE FOUND" if cond1 < cond2 else "* 2^((n -
1)/4) * det(L)^(1/n) >= N^(beta*m) / sqroot(n) \n-> NO SOLUTIONS
MIGHT BE FOUND (but we never know)")


    # change ring of pol and x
    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]

    # display basis matrix
    if debug:
        matrix_overview(BB, modulus^mm)

    # 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()
#     print ("potential roots:", potential_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



length_N = 1024       # size of the modulus
Known_bits = 512      # size of the known password
e = 5                 # value of public exponent

N=84364443735725034864402554533826279174703893439763343343863260342756
6786092168950937792630288092465059556475721766826694452700088164817717
0141755476887128502044240300164925440505830343990622920190959934866956
5697534331652019516409514800265887388539283381053937433496994442146
41968202764907970498260085751709 3;  # public modulus value

Cipher=588511908193557145472758995584417156637461398472460756192707453
38657007055698378740637742775361768899700888858087050662614318305443
0644488980265035567576103429384907413616436962850518672602785678969919
27351964557374977619644763633229896668511752432222528159214013173319
85564535161939387143345555058174164329 9; #known cipher text value

Known_password ="This door has RSA encryption with exponent 5 and the
password is";      # Half known password


Known_pass_bin= ''.join(format(ord(x), 'b').zfill(8) for x in
Known_password)  # Known text to Known binary password
# Known_pass_bin =Known_pass_bin+''.join(['0']*512);
Known_pass_int =int(Known_pass_bin, 2)  #known password integer value

print("Known password :",Known_password)
print("Known password binary :",Known_pass_bin)
print("Known password integer value :",Known_pass_int)
print("\n")
# Known_password_int =
```

```
4420798287960858089568459324014695894272440673821128081821764782831683782545477909845049053105265285041200031866649437923254125310572518093995086614391155;


ZmodN = Zmod(N);

P.<x> = PolynomialRing(ZmodN) #implementation='NTL'

for l in range(1,513):
    polynomial= (Known_pass_int*2^l + x) ^5-Cipher           # polynomial = (known_message + x (unknown_message))^e -Cipher
    polynomial_degree = polynomial.degree()

    #parameters for coppersmitha algorithm use
    beta = 1.0                                  # b = N
    epsilon = beta / 7                         # <= beta / 7
    mm = ceil(beta**2 / (polynomial_degree * epsilon))     # optimized value
    tt = floor(polynomial_degree * mm * ((1/beta) - 1))    # optimized value
    XX =  ceil(N**((beta**2/polynomial_degree) - epsilon))  # optimized value

    # Coppersmith
    roots = coppersmith_howgrave_univariate(polynomial, N, beta, mm, tt, XX)
    string_roots = '0'+"".join([str(i) for i in roots])          # unknown password integer string

    sol_int =Integer(string_roots,10)                            # unknown password integer value
    sol_bin ="{0:b}".format(sol_int) ;
    # print("sol_bin: ",sol_bin)                                 # unknown password binary string

    sol_text =int(sol_bin, 2).to_bytes((int(sol_bin, 2).bit_length() + 7) // 8, 'big').decode()  # unknown password text string
```

```
    final_password = Known_password + sol_text;                 # the final message of the given cipher and known text
    print(l," :",final_password)
```

And finally using above program we find the unown password as “`tkigrdrei`”.
The final_password is the full message of the given cipher in the code . Now we have the full message of the given cipher.

Final message =”`This door has RSA encryption with exponent 5 and the password is tkigrdrei`”