REPORT

Q1)

N has 7272 bit length, M is 129 bit long int and e is 17. Max value of M^e is smaller than 2192 bit length.(I did not try to find exact bit length). Modulus operation of N does not mean anything because we cannot even reach the value N. Thus, we can try to find M ^e values that is exactly equal to C. This approach looks like a valid way of exhaustive search.

In a loop I started iterating i=2 values that are exponentially increasing. i= pow(i,2). Checking the bit length of i and C I tried to approach the value. Bit length of C is 2177 and I stopped at m^e generates a 2177 bit length int value. Fortunately, when the function returns the m value as i , m^e actually equals to C. Thus I did not need to give effort to iterate more. Please check q1.py script.

M is 340282366920938463463374607431768211456

Message: b'\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'

☺

I could not figure out why this integer turns that byte represantation. However, I am sure that M^e equals to C. I do not know, maybe the hidden message is a smiley face . 😉

Q2

A)

Let's assume we know the value of cp. cp= (kp)^e mod n → cp = (n x T)+ (kp)^e , n= p.q → cp = (p x q x T)+ (kp)^e → kp is divisible by p , (px q x T) is divisible by p which means that cp must be divisible by p. N is a product of p and q which are prime. Thus , gcd value of any number with N has 4 options as 1, p, q, pxq (n). We do know that cp is divisible by p, if we calculate gcd(cp,n) the value is p. We can divide N with p and get the result as q. This is why multiplying public key with a constant integer and taking its power to e does not increase the security.

B)

Using the methodology described in part A we can find gcd of cp and n which gives the value of p. Dividing n with p gives the value of q. Then we can calculate phi of n as (p-1)(q-1). The last step is to find inverse of e in modulus phi of n. cm^e (mod phi_n) gives the integer plaintext value that can be decoded to string. Please check Q2.py to see the implementation and the output of this methodology

OUTPUT:

p is
3483753979604054987132575993009247280677480830776517911674906367140118698804998715510893918811294216144190151725377419129036611339311280826667966376212821895485268741270673718765512724909344201649188032985767661541816613226597379582912263532923988472919188046944004910656786170902863441838615032114533378659329761289506305332794087032070150632593390122770264224811609210671163350290119689657689997952960383894821321613527251385071409627904846687409654162444936114967748783877548682892470049689657139805082419172344757789672099225611173196591361453634559755307304275616218375354141099696170152528762732744618170060822066758645480996345434921211544846465337604603284977417959580060778570045743986609758717410555166086203387827723704157613477609392104343441056824603801205050991269966151149442919855435855576339449021327402385080608147792847466333224840386776566036138419773572714740715000463673060807424917001836481637488168119

q is
5506591942606409786007683694164320552666853760563997278112673956252642062580230341282514609867273797940285064713708705697045430216001808829496025459965762277595065815015312928959182173836874784924700204423500487879545871499053277509540470387842362515320251346826624958017594645145534876296220713938374221682083338460760448977511677765195762206864734020500176548242446545289154236269695091850232180279774486799056250625062409745209690007982157634073337918524689046502794950964394031916498102270905968246681208683669797493641971374341341036682069179230118965760408474627770213910582526956588655421840245359419231811191941978620018710013661856005725155989723730148659469997175814878286107665317557624508522339262171444938321292609872613647002641849742438288664089486939747792445254182820234942822660616550075433187549789920190466334149239889705377631041843807736298683682903374442069445140251407366732295733356931580544159902241

Message: b"I am free. Every single thing that I've done I decided to do. My actions are governed by nothing but my own free will. Do you wanna know what I hate more than everything else in this world? Anyone who isn't free."

By checking the truth table of F(x_1, x_2, x_3, x_4) we can tell that the ratio of 0's and 1's is 5/16 (i.e. occurrence of 0 -> 5 and 1-> 11). This states that function F is not balanced. In terms of nonlinearity degree function looks substantially high with degree 4. However, if we check correlation of Z with respect to x4, there are 13 collisions out of 16 instances which states that F is highly correlated to x4. This is due to the fact that x4 is used in every outputs in the combined function. That is why function F is not a good combining function. The truth table is provided below:

x1	x2	x3	x4	(x1&x2&x3&x4)^(x1&x2&x4)^(x1&x4)^(x4)
1	1	1	1	0
0	1	1	1	1
1	0	1	1	0
0	0	1	1	1
1	1	0	1	1
0	1	0	1	1
1	0	0	1	0
0	0	0	1	1
1	1	1	0	0
0	1	1	0	0
1	0	1	0	0
0	0	1	0	0
1	1	0	0	0
0	1	0	0	0
1	0	0	0	0
0	0	0	0	0

Q4

In this question I tried several methods to factorize N, using sympy.ntheory I did accomplish to factorize N. It is working because N is not really a cryptografic integer. (N is relatively small number.) After finding p, q values the task is easy.

-Compute phi of n

- Find d as modular inverse of e mod phi of n

- Compute M as C^d mod N

Please check q4.py script to check outputs.

p,q values and message is:

p is 2485770689

q is 3718940131

Message: b'Aloha!'

Q5

A) In this task I started with changing a,b string values to a list of integer values. I used a double iterated loop that I found in stackoverflow to multiply polynomials. For reducing, I used np.polydiv function that returns quotient,remainder pairs. I iterated over remainders, I need the bit values as 1 for the remainder indexes that are including odd numbers.(if the remainder includes 2 x (x^4) my str that represents the value of ax x bx will include 0 value for x^4 index). To make my statement more clear the remainder array [-1. 1. 0. 0. 0. -3. -3. 0.] will point to “11000110”.

B) For this task I created a possible candidates array that starts from 00000000 to 11111111. I created a function that evaluates a(x) x b(x) mod p(x) and checks if the value is “00000001” which means b(x) = a^ (-1). When it finds such a value it calls check inv function to test and it correctly works. Please check client.py for both implementations. The output is:

{'a': '11110110', 'b': '10101100'}

a(x)xb(x)is in gf(2^8)is 11000110

Congrats

[0 1 0 0 0 1 0 1]

Multiplicative inverse of a(x)is 01000101

Congrats

Q6)

While computing R we do not know r1,r2,r3 values but we can use a x b instead of r, after implying modulus their value will be the same => a * b = r mod q

Some variables such as M and N must be found to compute R. Q= q1*q2*q3 . Ni values are Q / q1, Q/q2 , Q/q3 in order. Mi values are modular inverse of Ni with respect to qi. (M1 = modinv(Ni1,q1)). After finding these variable's values we can construct R as:

R = ((a1*b1 * M1* N1) +(a2*b2 * M2* N2)+ (a1*b1 * M1* N1)) mod Q

Note that M1 = q2*q3 M2 = q1*q3 M3= q1*q2 . If we want to find r1 we can find R mod q1, M2 and M3 includes the quotient q1 and this yields to 0. Only the first part of the R will be computed and R mod q1 will point to r1 value. For r2 and r3 the same approach is used. Please check q6.py script to see the implementation and the outputs.

Output:

R is 175315162792420485043961120 56

r1 is (R%q1)) 1643182479

r2 is (R%q2) 363289399

r3 is (R%q3) 2376063578