Muhammet Taha ÇAKMAK
29183

Q1) Since we know the length of the message (129-bits) we can implement a brute force attack to find the cipher text starting from the shortest message. Q1.py do this for us.

```python
n = 2236578597661440233239751207512463082703496247483496750837699259187895034976146624899681362879032402174196764503614296944218715118439577009702182357594984436148333 1809
c = 10996907317744048201180239191184801878870026359954251163808786991823988392040721694956774540613511870672915186967674497345695382645430111824083800355570536408550748237
e = 17

# We will do an  exhaustive search to find the message strating by the smallest 129 bit plaintext candidate

s = "1" + ("0"*128)
x = int(s,2)

while True:
    result = (x ** e) % n
    if result == c:
        print("Plaintext is:",x)
        break
    x += 1
```

Luckily, the first message we try is the actual cipher text. **So the answer is: 2^128**

Q2)

a) If we know cp and cq, since n is public (known by everybody) we can easily get p and q only by checking gcd (cp,n) and gcd(cq,n).

$$GCD(c_p, n) = GCD(k * p, p * q) = p * GCD(k, q) = p$$

$(Since\ q\ is\ prime, GCD(k,q)\ must\ be\ 1\ or\ q.\ It\ cannot\ be\ q\ because\ that\ makes\ the\ mod\ n\ 0)$

$$GCD(c_q, n) = GCD(k * q, p * q) = q * GCD(k, p) = q$$

$(Since\ p\ is\ prime, GCD(k,p)\ must\ be\ 1\ or\ p.\ It\ cannot\ be\ p\ because\ that\ makes\ the\ mod\ n\ 0)$

By calculating these two we can easily find the primes and the remining part is just classical RSA decryption. Calculate $\phi(n)$, get the inverse modulus and calculate the power to obtain the message.

b) Following code exactly does the what I mentioned above:

```python
cp = 8519815578170286631805017896096474075672015420898414556273273563985458286184873008881921592838421126195389275500629440392317613244976879803
cq = 4818402986643636044952843129829705922725988299097705030918401041334297083874026699605895648827453267994332533399138283856617689029155909510
n = 19183611594110704047944268098409679747777693533574914342013275888624116342577979312303403621441094928490848608709350039227543100666636336845
e = 65537
m = 17284828821141668560328235493236055797130405967076368219938343596386850014870638653469721063839006553022826713358194804149890450517899381470

p = egcd(cp,n)[0]
q = egcd(cq,n)[0]
# print(p * q == n) # Check wheter the result is correct or not

phiN = (p - 1) * (q -1)
d = modinv(e, phiN)
mDecrypt = binaryLeftRight(m,d,n)
print("Decryption key = ", d)
print("Message = ", mDecrypt)
# print(binaryLeftRight(mDecrypt,e,n) == m) -> check wheter the answer is correct or not
```

Answers:

Decryption key =
1896700307404353708889092787238738450976930906045521956481734345110665527335742138790795496373832534417842619128949684135415277314945808880809118610153086033495801903906206958465984135301994041265547967576613347296448980200073170347937760079853038768280039101345869640359016135380246462730384162102080456512517923518283448933843988644106650056625228789124934730144661195192292875776136666400406579377306399811166886722893890946149420113877228120239495227349630491347663620593836674899663370295623786744116606475877671278073475942785273513881946647291846972144397881721497964494880403774165518746408816724612273920805759236254422413920821874362972296520983076035185827080849755507808817011954369304819915225584496686457225956836675

Muhammet Taha ÇAKMAK
29183

07738193367068934042328701706827195662351437964287898635813603191594544235895551438253374111500952411487206603126317432263716673156411261693918514640240582032504394723912833598449022529402571998290395211161528438208673101703575836109766989637333052867937596368882305743574375515622277603616910691317234778567544389019008242607156754619872588374545964274041806115182642079932182426464872228757004385817912300828004460715688047336600067491457686243441920511420146314656399999697441607171009884005245176278158648357175732385620886657814391534419879308103417416622632430501662897600834224532295110131504072741432593535570378686730497662893860336914686154525506771240463622933140219120646013493265342985950065936212341698072021667485145981154994856139526174374932903227960941230693069318645454926118052586147536401790394463081089433994717725329640827954202409677126319923161622471306317477208482856369736513398792557901220193906727530546558017282812383962494170211695236757711273746893283793553883884753597457665018204553157952013728637584608550893661354955655625903864775302424406109757035312939687617336013939129388811713 93

Message =
638430993074629687195837936481159006693552285988028013368030841243284972629648604448109734668091466074510200197045977080275481509858813617320605969234825499359653656530495654596514919349064617114149002930871172405935648860559671540513381615320335361620724965447066373540766149046617848968335576132622105768790894594492599420083997078646795696149541761157224739200181378368968266501730694986378199803088188901855697322097918673080384564212532124778222986091542091264548448179852908954422806679427419346552561737

Q3)

Let’s simulate the F(x1,x2,x3,x4) by using following python program. (Q3.py)

```
# This code will produce the truth table of the given function

X1 = X2 = X3 = X4 = [0,1]

def F(x1,x2,x3,x4):
    return x4 ^ (x1 & x4) ^ (x1 & x4 & x2) ^ (x1 & x4 & x2 & x3)

counter1 = 0
counter0 = 0
print("X1 | X2 | X3 | X4 | F")
for x1 in X1:
    for x2 in X2:
        for x3 in X3:
            for x4 in X4:
                result = F(x1,x2,x3,x4)
                print(x1,"  ",x2
                      ,"  ",x3,"  ",x4,"  ",result)
                if result == 1: counter1 += 1
                if result == 0: counter0 += 1


print("\n \nNumber of zeroes : ", counter0,"Number of ones : " ,counter1)

```

Muhammet Taha ÇAKMAK
29183

The resulting table looks like that:

X1	X2	X3	X4	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Number of zeroes : 11 Number of ones : 5

By using this table, we can analyze the function.

Nonlinearity degree: The highest degree term is 4. So it is 4.

Balance: The result is not balanced, function is more likely to produce 0 than 1. That's why it is not well balanced.

Correlation: There is a strong correlation between x4 and F. Outputs are almost identical to input value of x4.

That is not a good combining function due to the 2 reasons. **Firstly,** it does not produce balanced outputs. P(F = 0) = 11/16 and that is far away from 1 / 2. **Second reason is** output is highly correlated with x4, that makes correlation attacks possible.

Q4)

- To find the decryption key, we need to find phi(N). Since N is multiplication of 2 prime numbers, it is (p-1) * (q -1).

Muhammet Taha ÇAKMAK
29183

- My strategy to find the primes is to check numbers in form of 6k + 1 or 6k – 1. That's because every prime number larger than 3 give us reminder 1 or 5. (That is very simple to prove. In mod 6 there are 6 different possible values which are 0,1,2,3,4,5. If the reminder were 0,2 or 4 that means the number is even. Also if the reminder is 3 that means the number is divisible by 3. That's why only possible reminders are 1 and 5)
- Thanks to that number theory trick, we can search 6x times faster than looking for all the numbers. Also, it is more efficient than checking the odd number 33% percent since we always pass the 3-divisible numbers.
- The second trick is to look only for the numbers smaller than the square root of the numbers.
- Since we expect large primes, we will start to search the numbers from top to down.
- The implementation below (Q4.py) does that for us.

```python
def findPrimeFactor(N):
    n2 = math.ceil(math.sqrt(N))          # One of the multpiles must be less or equal to square root of the number.
    # print(n2)
    # print(n2 % 6)

    # We will start to check from the largest multiple of 6 which is also smaller than the square root of the number.
    # THEOREM: Every prime larger than 3 is in form of either 6k + 1 or 6k - 1

    for i in range(n2- (n2 % 6) , 5, -6):
        if N % (i - 1) == 0:
            print("########################")
            print("Number 1 is: ", i -1)
            print("Number 2 is: ", N  // (i -1))
            print("########################")
            p1 = i - 1
            p2 = N  // (i -1)
            return p1,p2
        if N % (i + 1) == 0:
            print("########################")
            print("Number 1 is: ", i + 1)
            print("Number 2 is: ", N  // (i +1))
            print("########################")
            p1 = i + 1
            p2 = N  // (i +1)
            return p1,p2

    return None


p1, p2 = findPrimeFactor(N)
phiN = (p1 - 1) * (p2 - 1)
inv =  modinv(e,phiN)
print("Decryption key is: ", inv)
```

Answer:
########################
Number 1 is: 2485770689
Number 2 is: 3718940131
########################
Decryption key is: 4032669742276769153

Q5)

Muhammet Taha ÇAKMAK
29183

```python
a, b = get_poly()

GF = galois.GF(2**8, irreducible_poly=int('111000011',2))
aPol = GF(int(a, 2))
bPol = GF(int(b, 2))
print(bin(aPol * bPol)[2:].zfill(8))
print(bin((aPol ** -1))[2:].zfill(8))

check_mult(bin(aPol * bPol)[2:].zfill(8))
check_inv(bin((aPol ** -1))[2:].zfill(8))
```

I used galois library of python (Q5.py), basically turns the binary strings into field elements and turns the operations into field operations. I obtained the two binary polynomials which are correct:

Answer:

a * b = 00110010

a ^-1 = 11001101

Q6)

By using Gauss algorithm we can built a useful R. The principle behind that is as follows:

$\sum a_i N_i M_i = R \; mod \; Q$ where a_i is the individual modulus and $M_i = N_i^{-1} mod \; n_i$. Every term will hold the following equations:

$$a_i N_i M_i = a_i \; mod \; n_k \; if \; k = i$$
$$a_i N_i M_i = 0 \; mod \; n_k \; if \; k \neq i$$

So, whenever we want to extract a specific a_i from R, all we need to calculate the $mod \; n_i$ of the R.
Following python program written based on this principle.

Muhammet Taha ÇAKMAK
29183

```python
a1  = 2700926558
b1  = 967358719
q1  = 3736942861

a2  = 1759062776
b2  = 1106845162
q2  = 3105999989

a3  = 2333074535
b3  = 2468838480
q3  = 2681377229

Q = q1 * q2 * q3

N1 = q2 * q3
N2 = q1 * q3
N3 = q1 * q2

m1 = modinv(N1, q1)
m2 = modinv(N2, q2)
m3 = modinv(N3, q3)

R = (a1*b1*N1*m1 + a2*b2*N2*m2 + a3*b3*N3*m3) % Q

r1 = R % q1
r2 = R % q2
r3 = R % q3


print(r1,r2,r3)
# To check we can run the following piece
print(r1 == ((a1 * b1) % q1))
print(r2 == ((a2 * b2) % q2))
print(r3 == ((a3 * b3) % q3))
```

Answer:

R = 17531516279242048504396112056

R1 = 1643182479 **R2** = 363289399 **R3** = 2376063578