Question 1:

Basic RSA. We have C, N and E.

```
C = 623247839491341191594088165133349125343435173008801376916627808954099927602620 21
N = 1280678415822214057864524798453297819181910621573945477544758171055968245116423923
E = 65537

#from factordb.com; N = P * Q
P = 189910798652748353534451711394853132833 1
Q = 67435786954060093387014589956474649531903 3

#find values of phi and d, as well as deciphered text
phi = (P-1) * (Q-1)
PrK = pow(E,-1,phi) # private key
M = pow(C,PrK,N)

print(phi)
print(PrK)
print(M)

#Ptxt is in numerals, assuming this to be coded in ascii, for each numeral representation:
from Crypto.Util.number import long_to_bytes
plaintext = long_to_bytes(M)
print(plaintext)
```

➜ Assignment4 git:(main) ✗ /bin/python "/home/dashlander/Desktop/Sem 6/CS431/Assignment4/pandq.py"
1280678415822214057864524798453297819181234364596418349127352680639289550089776560
4493327356060849603512044069096102973015747284668209335159428649254592659835561931301638252944910606592729142534253543799622213535290525663955565467740017722764 5
b'picoCTF{sma11_N_n0_g0od_05012767}'

Question 2

From the code we see, this is an XOR implementation. However, the 50000 padding is then reused. So all we have to do is ensure that we just need to cause a wraparound so that the encrypted string is XORed using the same values as it was encrypted with, which should give us the flag. We can do this by bruteforcing and sending large chunks of data to the server to be XORed, and once the wraparound happens, we simply send the encrypted data to the server to be decrypted, and this shall give us the flag.

The flag in question, therefore is: picoCTF{3a16944dad432717ccc3945d3d96421a}

Question 3

From the code given, the b16_encode function breaks translates an ascii character to two letters from a to p; ALPHABET = string.ascii_lowercase[:16]
We know the key len is 1, so the key can be from a to p. Also, the offset is defined. After encoding, the flag is then shifted, which we do not know by how much. But since the offset can be from a to p, we iterate over all 16 combinations.

The output we get is as follows:
➜ Assignment4 git:(main) ✗ /bin/python "/home/dashlander/Desktop/Sem 6/CS431/Assignment4/nc.py"
Key: a, Decrypted message: ËÚµÚÛµÇÇËÌÊËÈÉ
Key: b, Decrypted message: ºÉ¤ÉÊ
¤¶¹¶º¶»»
¹º·¶¸
Key: c, Decrypted message: ©¸¸¹s¥v¨¥u©u|¥ªx}ªzx}|tz}||{|z¨©¦v¥z§
Key: d, Decrypted message: §¨bedgliglkcilkkjkiei
Key: e, Decrypted message: qQqTSSZV[XV[ZRX[ZZYZX
TX

Key: f, Decrypted message: v
`
@`rCurBvBIrwEJwGEJIAGJIIHIGuvsCrGt
Key: g, Decrypted message: et_tu?_a2da1e18af49f649806988786deb2a6c
Key: h, Decrypted message: TcNcd.NP!SP T 'PU#(U%#('/%("&'%STQ!P%R
Key: i, Decrypted message: CR=RS=OBOCODDBC@OA
Key: j, Decrypted message: 2A,AB
12?>0 ,>1>2>33
Key: k, Decrypted message: !01û -ý!ýô-"ðõ"òðõôüòõôôóôò !.þ-ò/
Key: l, Decrypted message: /
/ ê
íìãïäáïäãëáäãâãâáíá
Key: m, Decrypted message: ùÙù
Ü

ÛÛÒ
ÞÓÐÞÓÒÚÐÓÒÒÑÒÐ
Ü
Ð
ÈèúËýúÊþÊÁúÿÍÂÿÏÍÂÁÉÏÂÁÁÀÁÍýþûËúÍü
Key: o, Decrypted message: íü×üý·×é°ìé¹í¹°éî¼±î¾¼±°¸¾±°°¿°¾ìíê°é¾ë
Key: p, Decrypted message: ÜëÆëì¦ÆØ©ÛØ¨Ü¨¯ØÝ« Ý« ¯§ ¯¯®¯ÛÜÙ©ØÚ

The only sensible output we get is for the key g, and as such the flag is :
picoCTF{et_tu?_a2da1e18af49f649806988786deb2a6c}

Question 4:

Since it is said that d is small, searching google says that the weiner attack is possible. Using this, I used the oweiner implementation from orsiano[https://github.com/orisano/owiener].
Using the following script, I found the flag:

```
import owiener

C =
752194934262897745266772245850465993284825740204099297126385122143904272949269992657355683535280644379864200811664630359762195502348281165081008177529779683049123139754134308566434397083955117669370237174018991300053698992719717945063031254048806115331824726885741453446977036600884369921107848744006470252 83
N =
106641942602199082330329788662726792628380590566553936486673293692114080806494454368347375124461744343892184091297964103139205085252612790015095725089549742850810521951483598870935930911618967291047749696511770397130559816010973731655265142894139185848949696660349445416219495482971068960991032960972157649083
E =
595784874735465706992989527141191079819732579127245117885293686339383971313463170370832190897434744028347266517251439917951642009881784819999914360321404724396909406132947337934589336502151666033532781603958847510514464278902557966147291063480185348891686852874543414245156531518673677103739005946567426 5203

d = owiener.attack(E, N)

if d is None:
   print("Failed")
```

```
M = pow(C,d,N)
from Crypto.Util.number import long_to_bytes
plaintext = long_to_bytes(M)
print(plaintext)
```

Therefore, Flag is : picoCTF{proving_wiener_2635457}

Question 5

Could not connect on SSO, connection was blocked. Had to use internet to get the file.

On line 378, user cultiris credentials are found. The password given is cvpbPGS{P7e1S_54I35_71Z3}

We know the flag is of the format picoCTF. Assuming that the cipher is a shift cipher of same displacement, we can find the flag. The shift is of 13. The flag is picoCTF{C7r1F_54V35_71M3}

Question 6

Using online decryption tools, flag is
picoCTF{WH3R3_D035_7H3_F3NC3_8361N_4ND_3ND_4A76B997}

Question 7

Using Substitution Breaker, we find the key from the encrypted study guide, which is
xunmrydfwhglstibjcavopezqk

From which, we find the flag which is picoCTF{perhaps_the_dog_jumped_over_was_just_tired}

Question 8

Simple Vignere Cipher

Question 9

Since the encryption text changes everytime we connect, the attack has to be done in a single connection.

We do not know what the key is, but we can use KPA. Using a and then encrypting and storing all values, we then decrypt the encrypted values, and when the keys match for both, we decrypt the encrypted ciphertext. The flag thus found is: af5fa5d565081bac320f42feaf69b405

Question 10