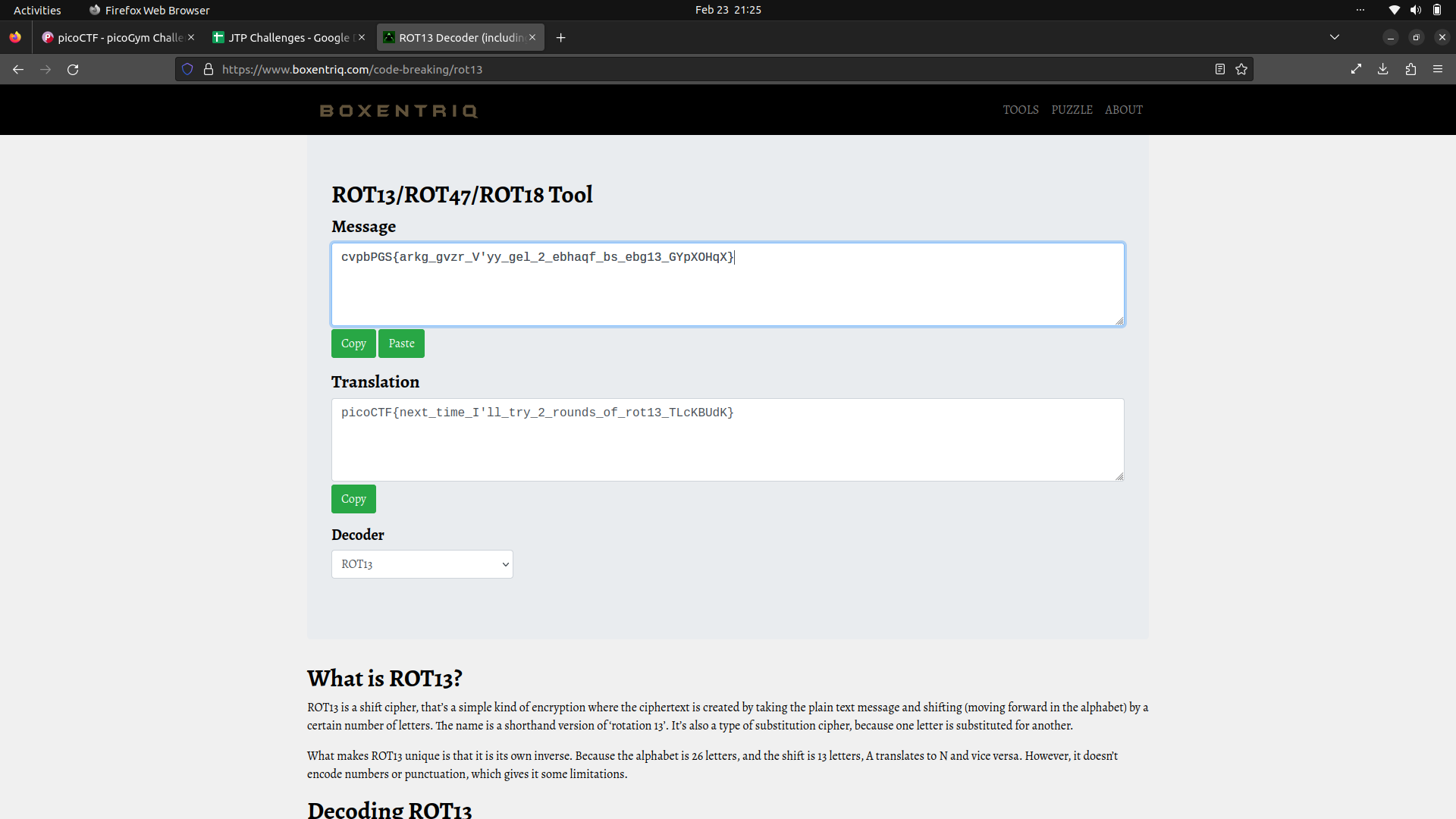
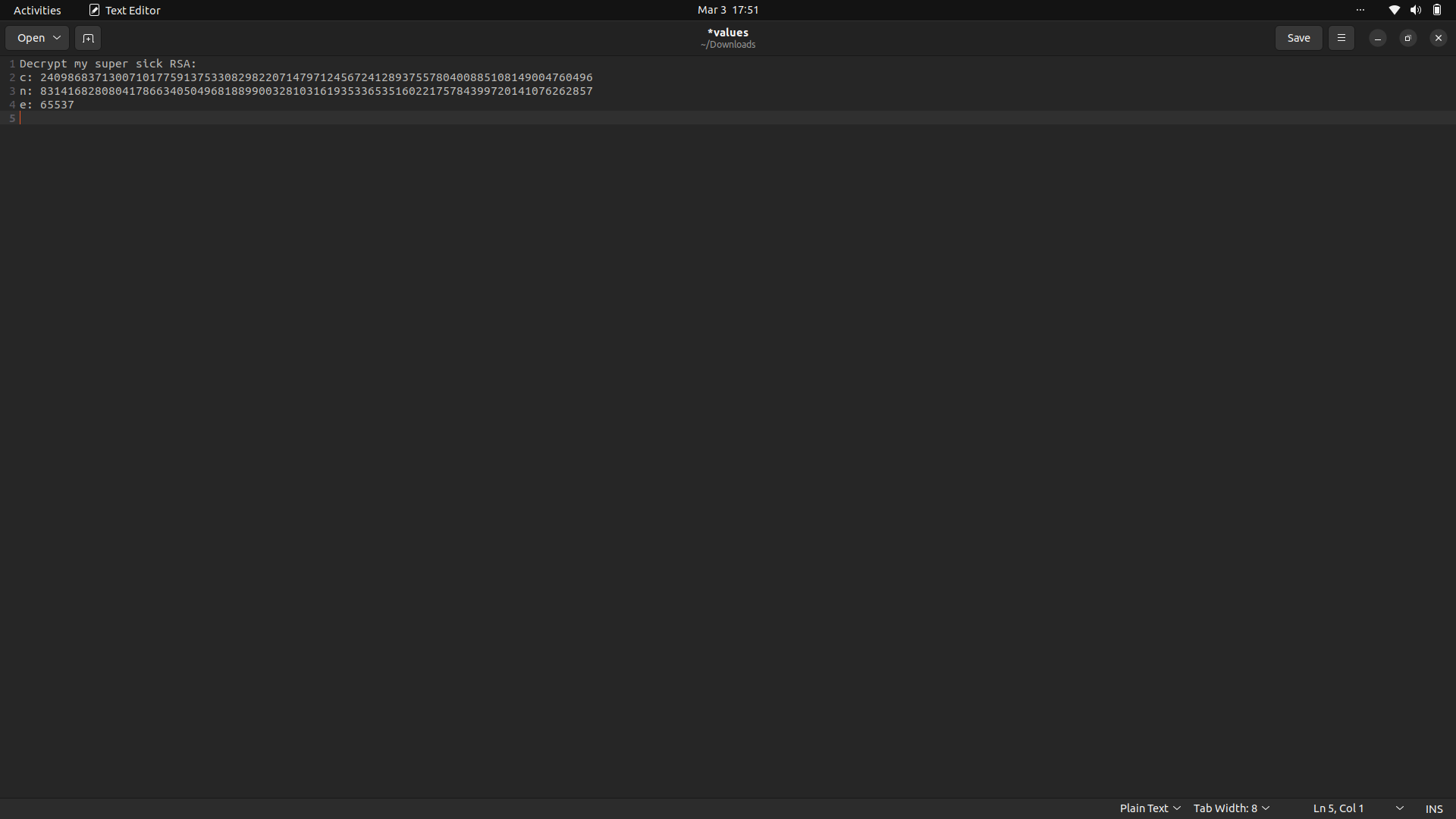
MOD 26

It was simply a matter of applying ROT 13 to the code.



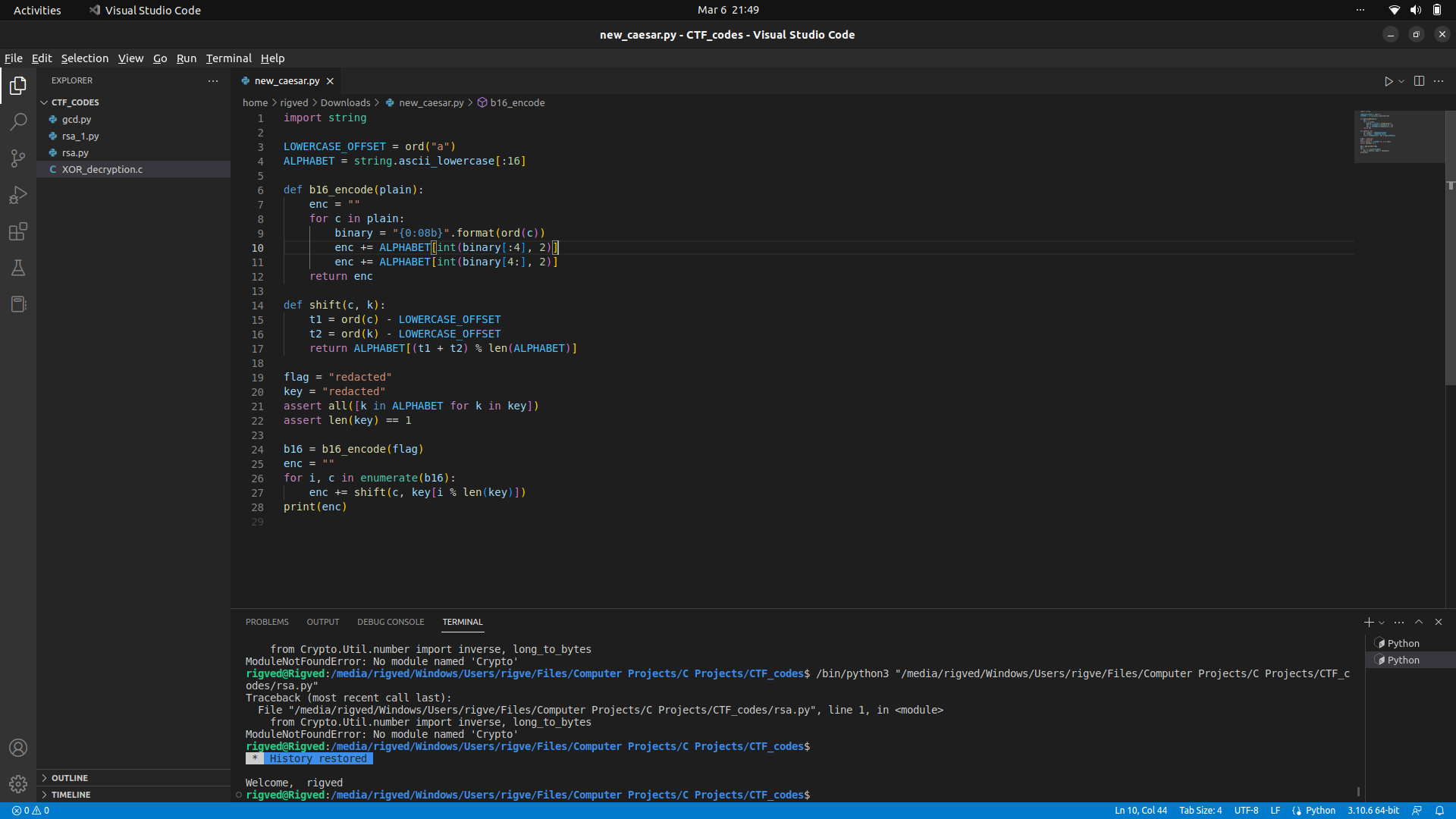
MIND YOUR Ps & Qs



In this, the value of n is the product of ‘p’ and ‘q’. Using an online factor calculator, we can find its two prime factors ‘p’ and ‘q’.

Flag: picoCTF{sma11\_N\_n0\_g0od\_23540368}

NEW CEASAR



Here, we are to decrypt the following string of characters:

kjlijdliljhdjdhfkfkhhjkkhhkihlhnhghekfhmhjhkhfhekfkkkjkghghjhlhghmhhhfkikfkfhm

We are also given the previously inserted python code. Let’s try to understand what it does.

LOWERCASE\_OFFSET = ord("a")

The ‘ord’ function returns the unicode code of any given ASCII character.

String.ascii\_lowercase returns all lowercase alphabets. The [:16] only allows it to return the first 16 alphabets.

The b16 function:

for c in plain:

binary = "{0:08b}".format(ord(c))

enc += ALPHABET[int(binary[:4], 2)]

enc += ALPHABET[int(binary[4:], 2)]

return enc

Uses the binary value of the ASCII value of ‘c’ and breaks it in 2 parts, initially first 4, then last 4.

b16 = b16\_encode(flag)

enc = ""

for i, c in enumerate(b16):

enc += shift(c, key[i % len(key)])

Finally, the above section of the code uses enumerate function which returns two variables instead of one, namely the number of iteration and the value of the iterable. It takes the value of the key, which is 1 in this case, and uses it to shift the character at the place by the variable i%len(key).

ef shift(c, k):

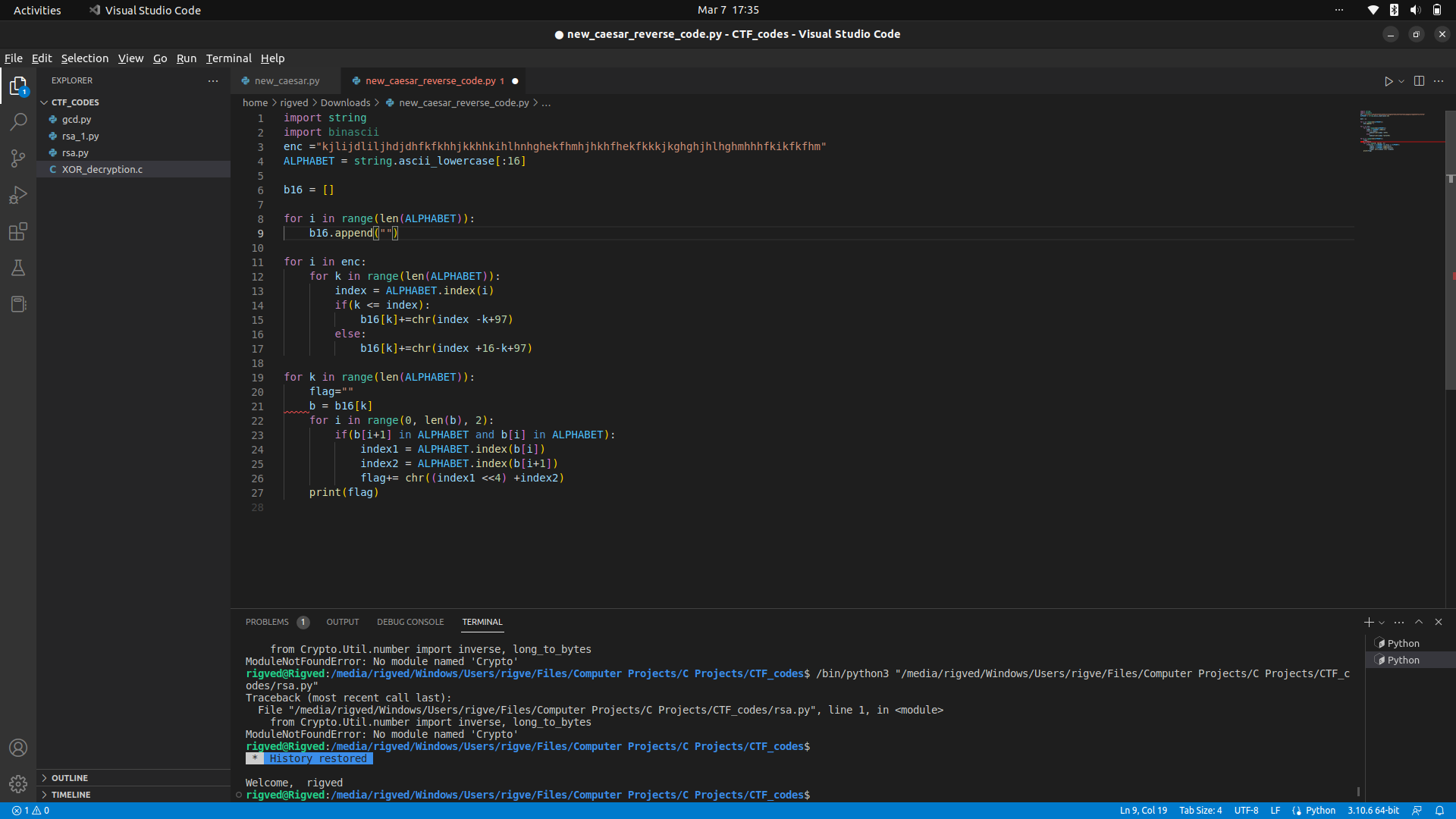
t1 = ord(c) - LOWERCASE\_OFFSET

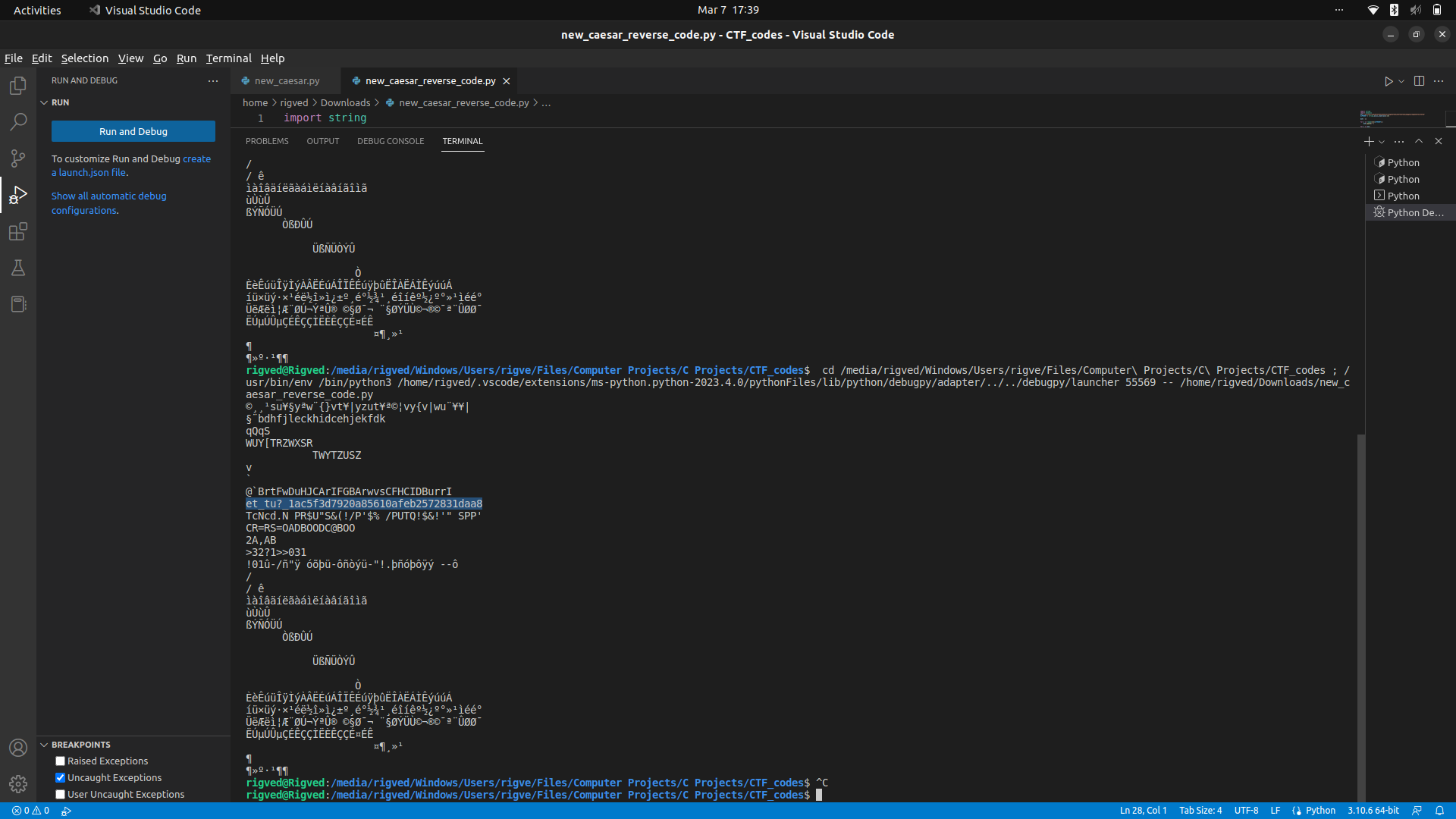
t2 = ord(k) - LOWERCASE\_OFFSET

return ALPHABET[(t1 + t2) % len(ALPHABET)]

The shift function returns a value in ALPHABET at the index ascii value of the c minus 97 plus the ascii value of the key minus 97 mod 16.

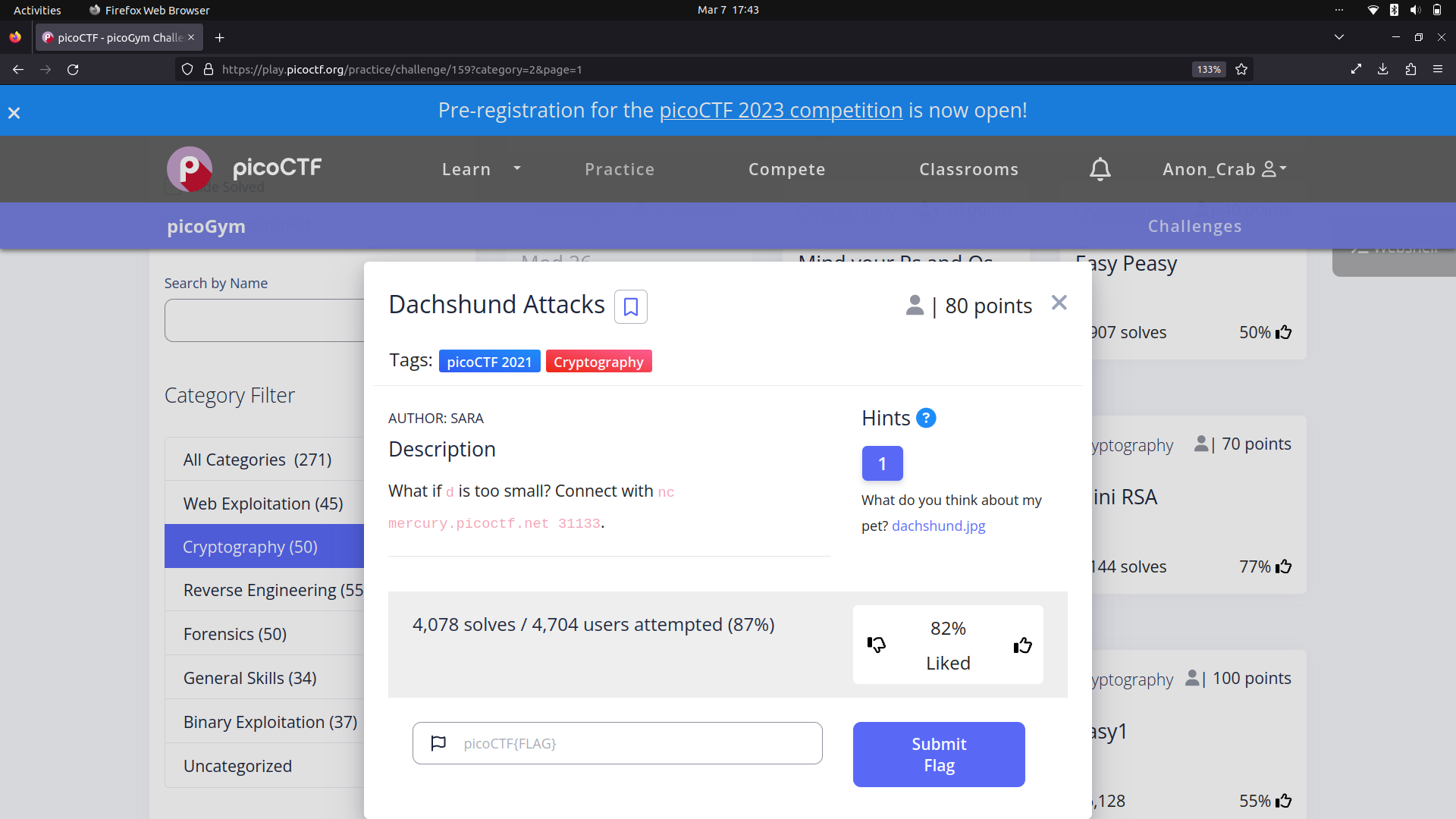
Writing a code to reverse the above encryption:





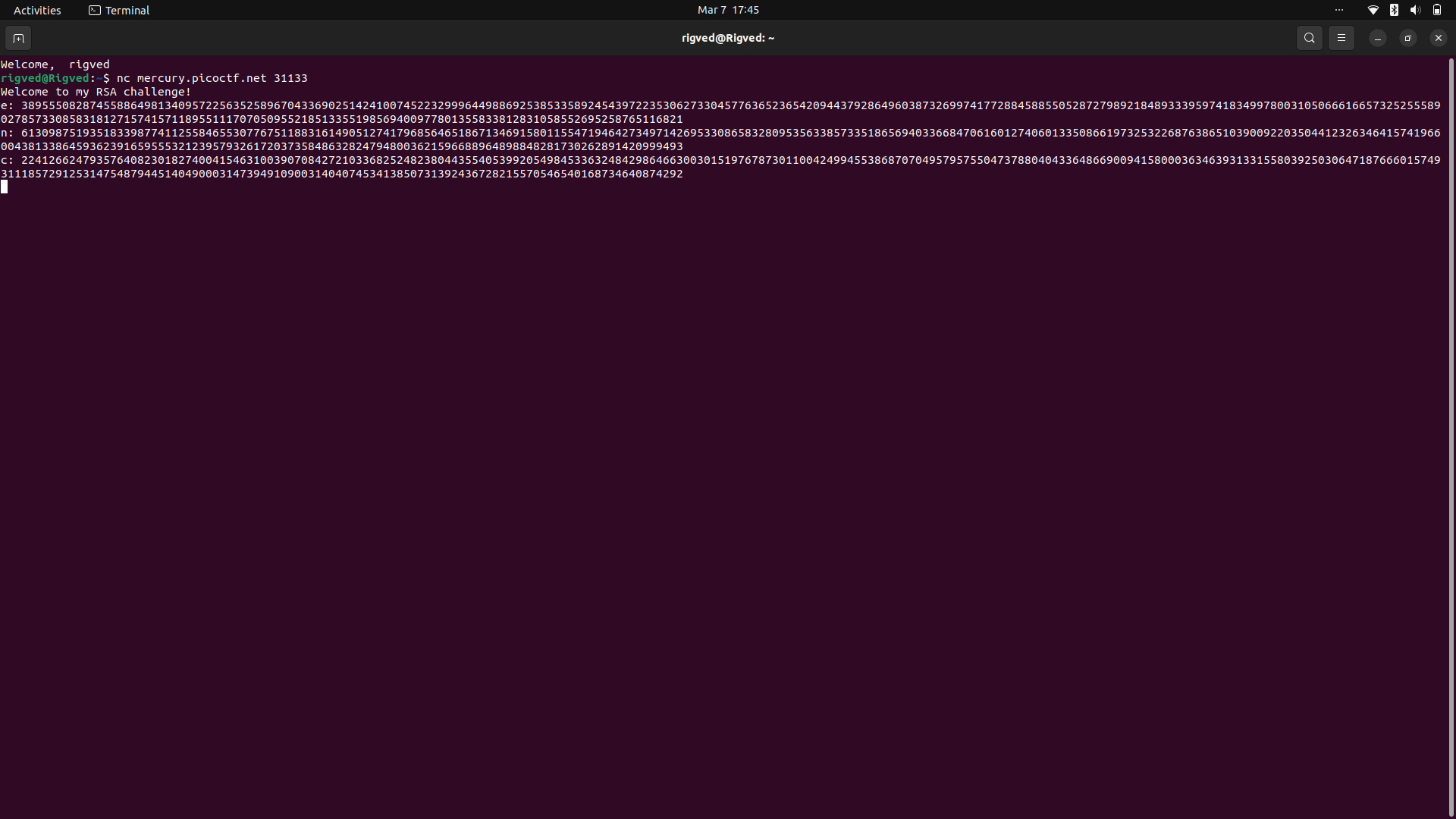
Flag: PicoCTF{et\_tu?\_1ac5f3d7920a85610afeb2572831daa8}

DASCHUND ATTACK





Connecting with the given port, we are returned the values of e, n, c. This obviously hints to RSA encryption.



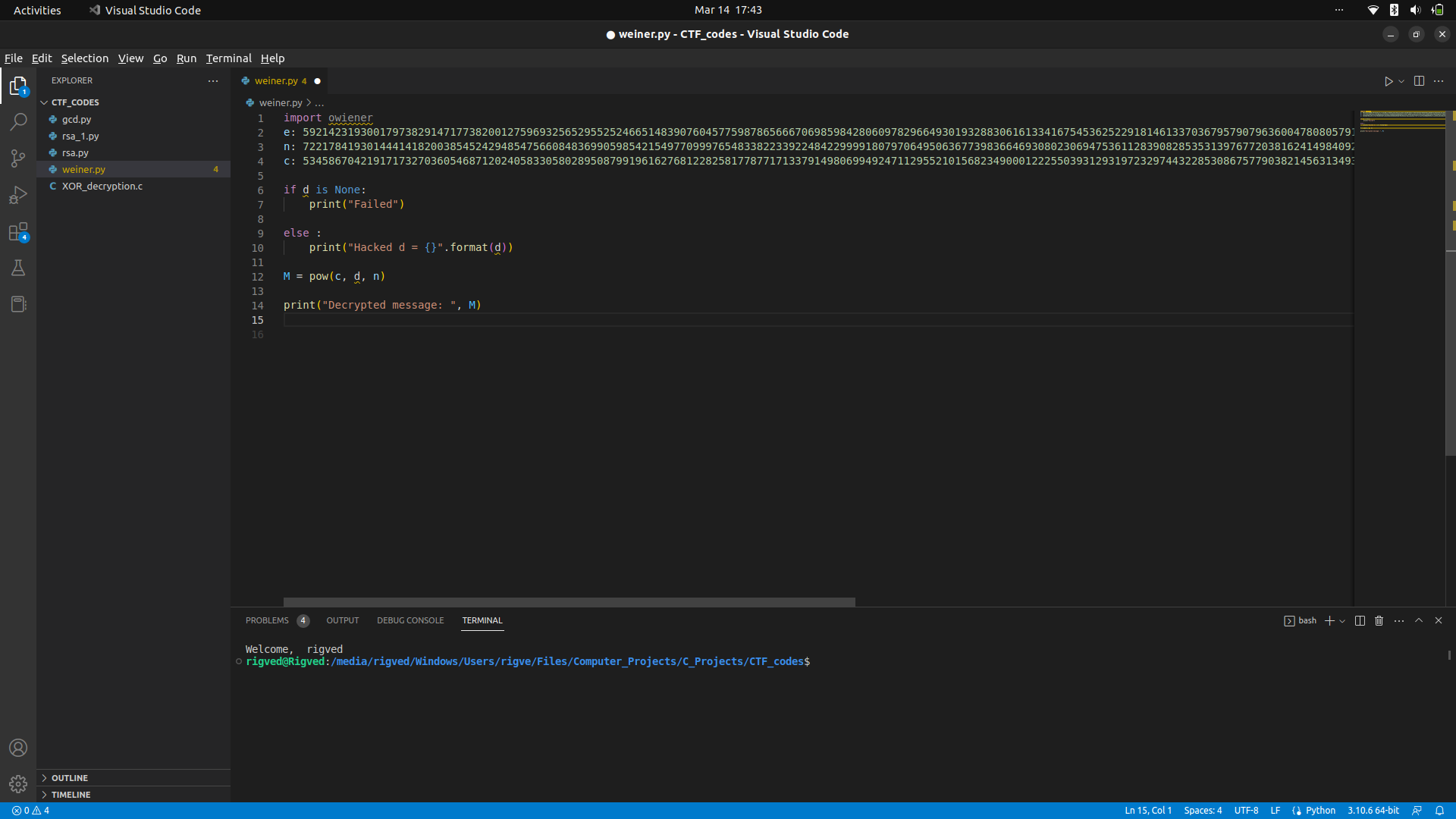
e: 38955508287455886498134095722563525896704336902514241007452232999644988692538533589245439722353062733045776365236542094437928649603873269974177288458855052872798921848933395974183499780031050666166573252555890278573308583181271574157118955111707050955218513355198569400977801355833812831058552695258765116821

n: 61309875193518339877411255846553077675118831614905127417968564651867134691580115547194642734971426953308658328095356338573351865694033668470616012740601335086619732532268763865103900922035044123263464157419660043813386459362391659555321239579326172037358486328247948003621596688964898848281730262891420999493

c: 22412662479357640823018274004154631003907084272103368252482380443554053992054984533632484298646630030151976787301100424994553868707049579575504737880404336486690094158000363463931331558039250306471876660157493111857291253147548794451404900031473949109003140407453413850731392436728215570546540168734640874292

Turns out, when ‘d’ is very small, there is a particular method to break it, which is called a Weiner attack.

Using an online python3 implementation for weiner attack, we can use it to find the value of ‘d’. Simply using the ‘pow’ function for finding exponential values of variables, we can decrypt the value of the cyphertext.

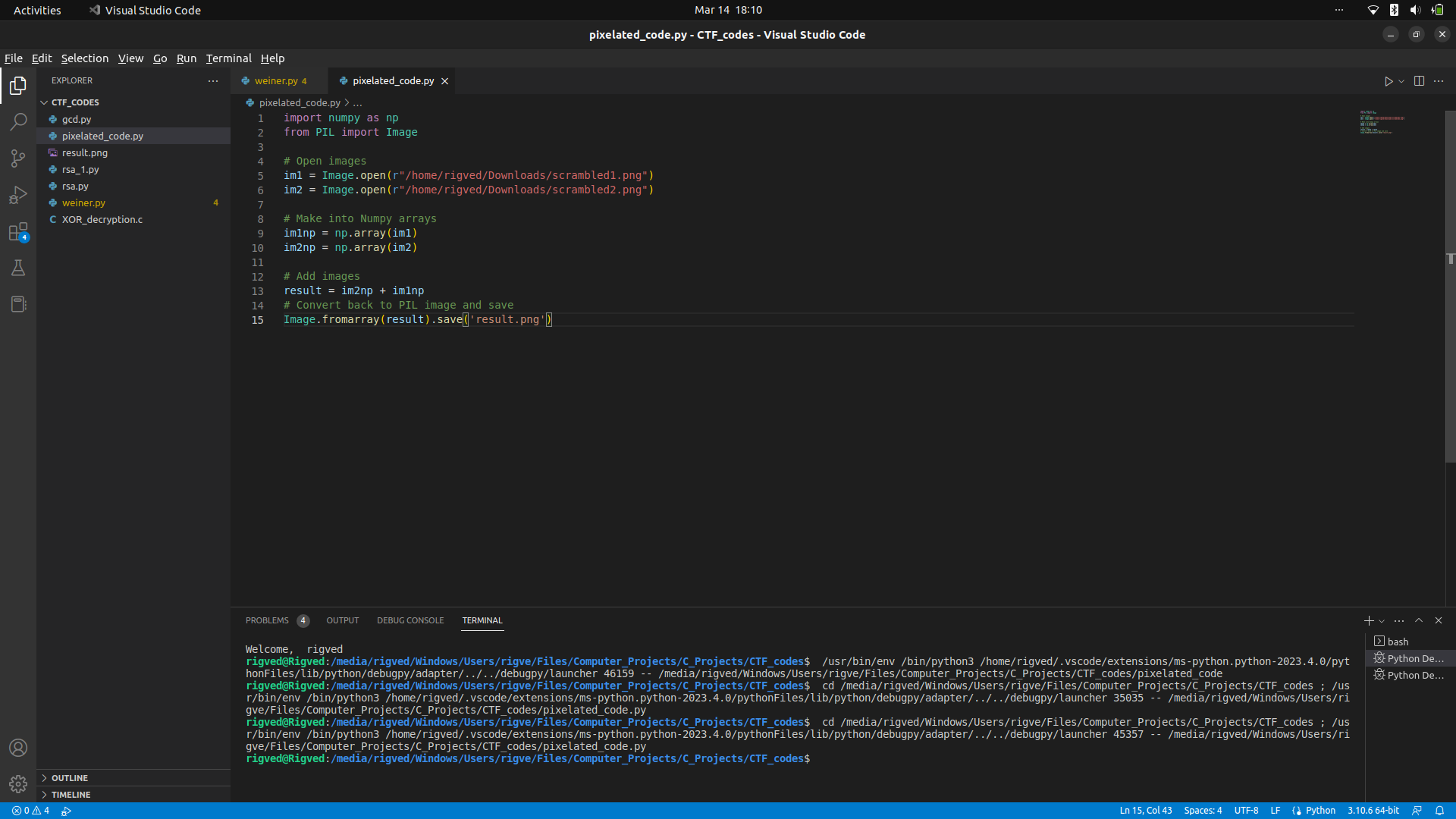


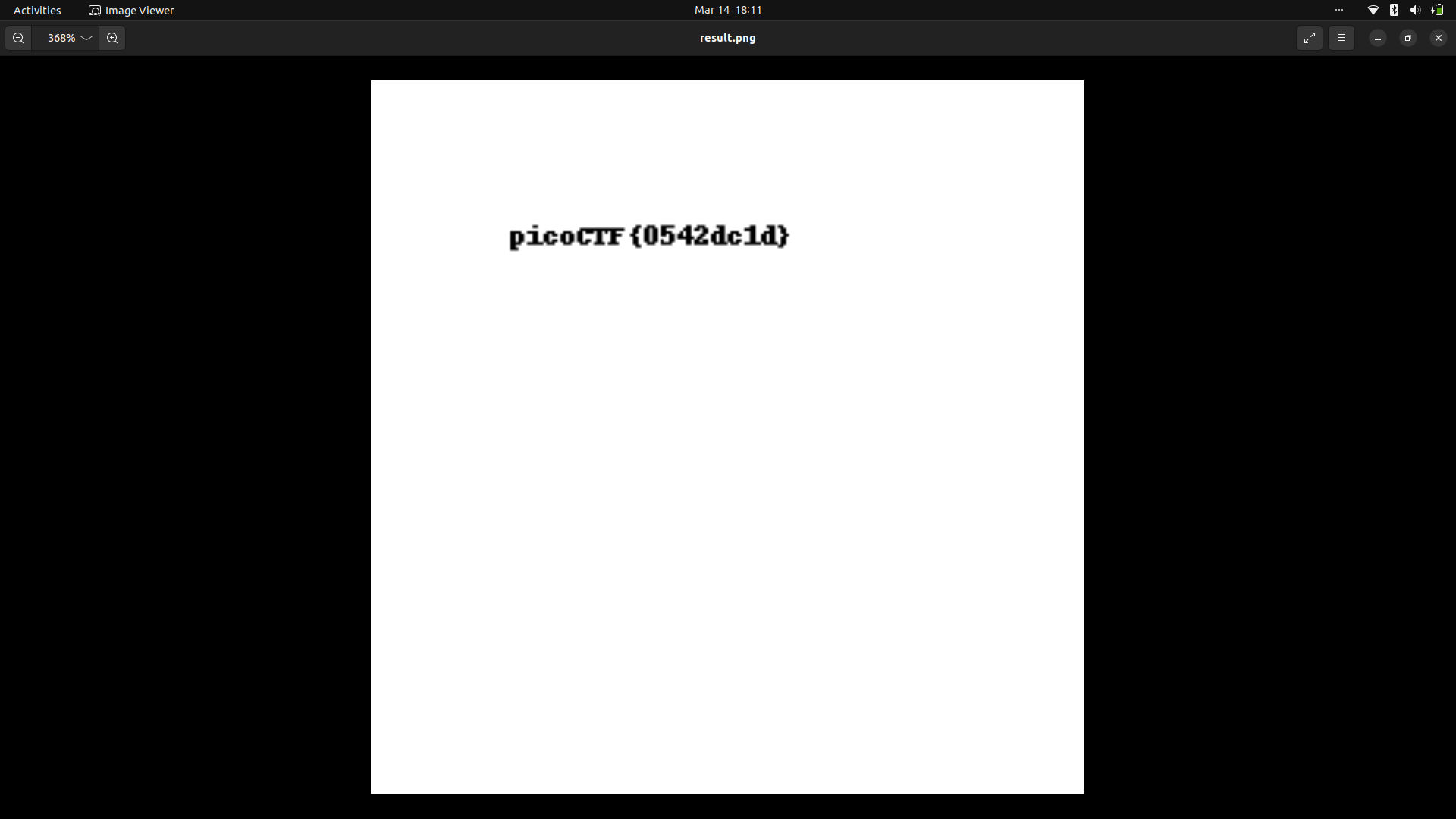
Resources:

<https://pypi.org/project/owiener/>

PIXELATED

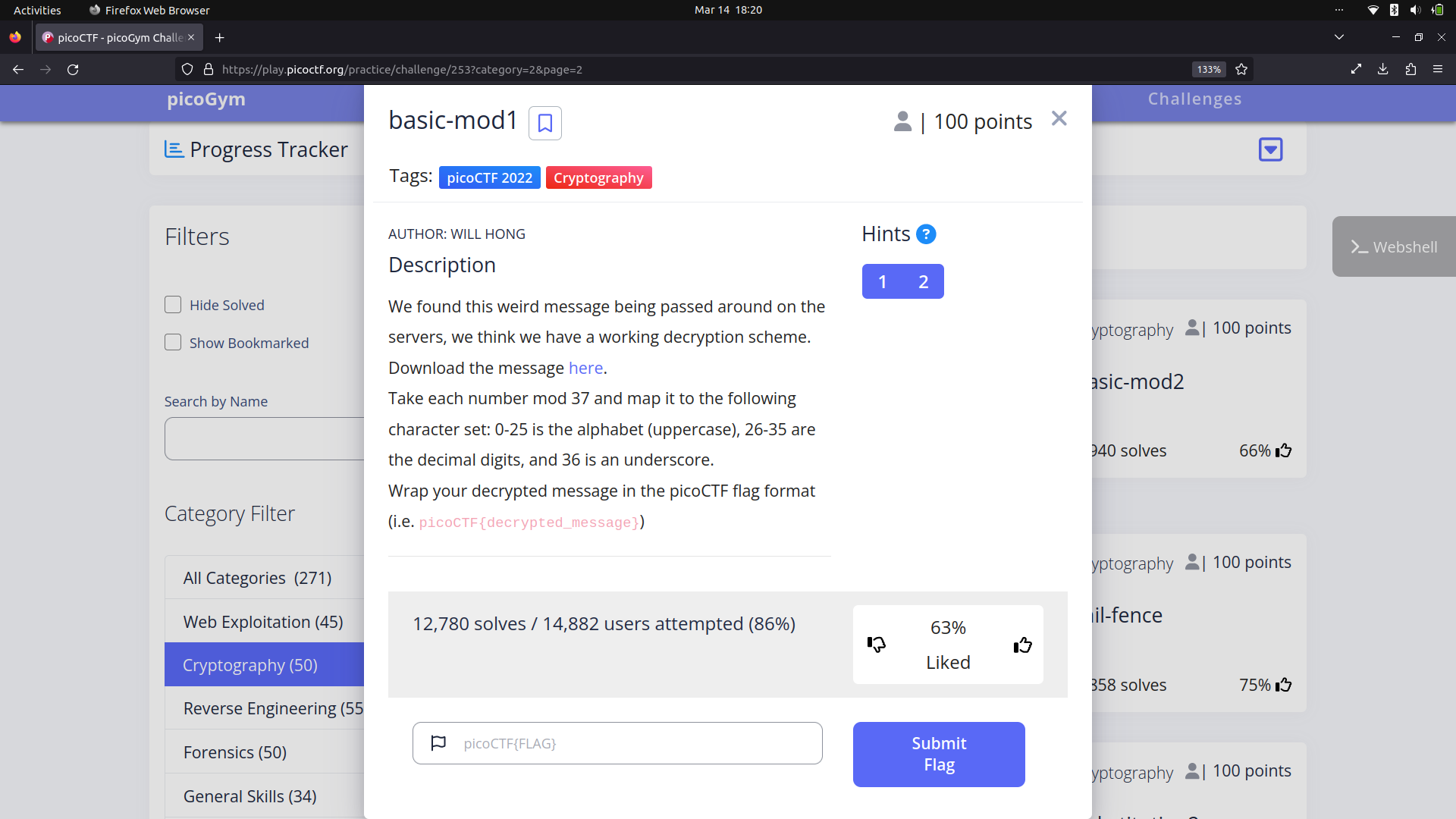
In this, we are given two different image file, which upon being converted to hexadump do not yield any possible clues. However, the naming of the files (scrambled1 and scrambled2) are such that they lead us to conclude that perhaphs we need to add the two image files together.





Upon doing so, we get the flag.

BASIC-MOD1



The solution to the encryption is given in the problem itself. All that remains to be done is to write a program to do the decryption for us.