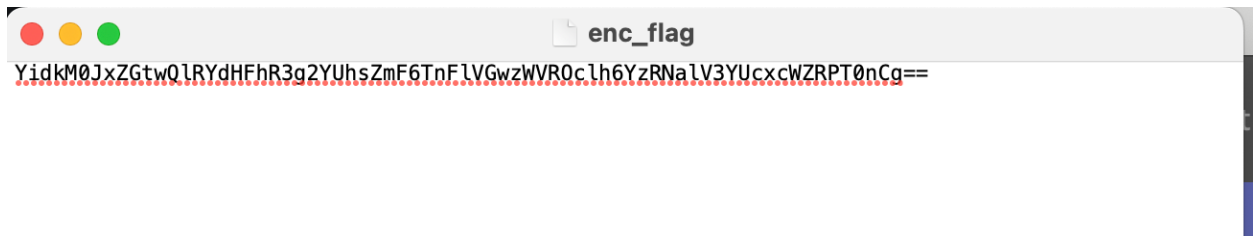


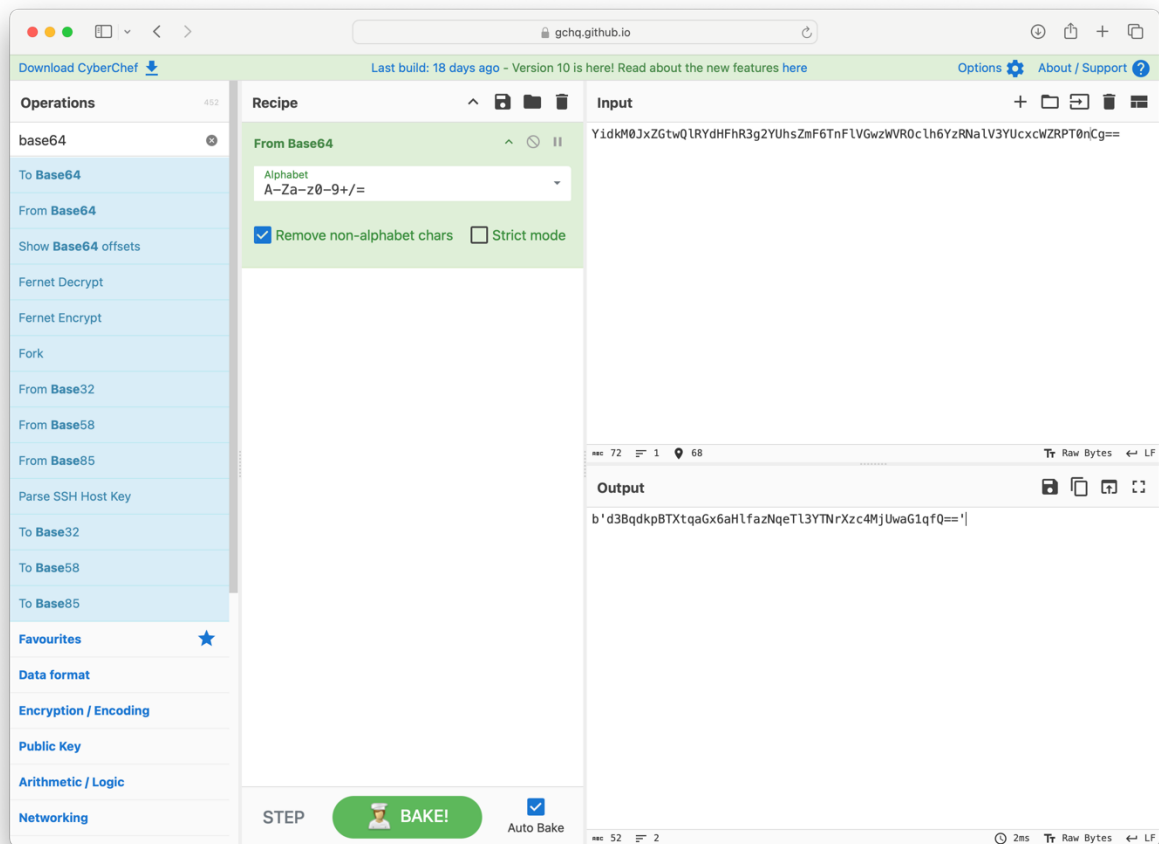
Ejercicio 1

The First step is to download the enc_flag archive which gives us the ciphertext

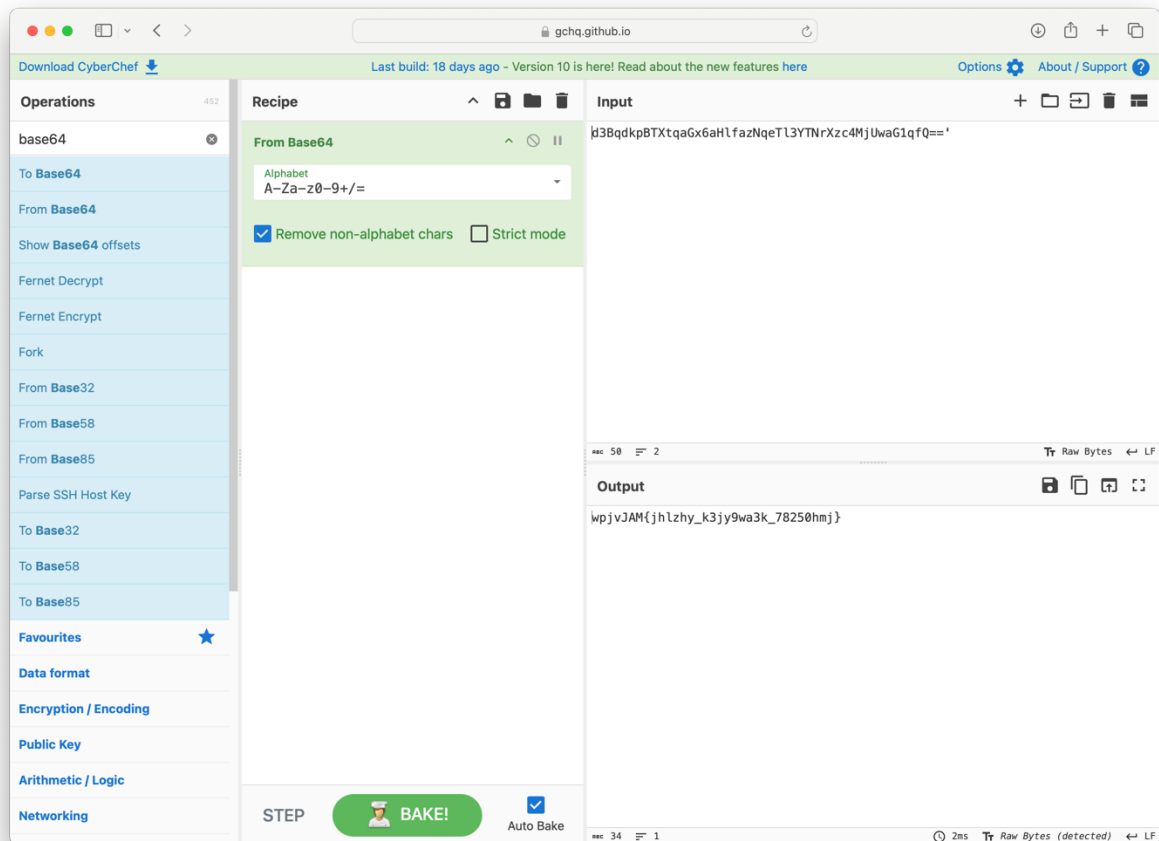


YidkM0JxZGtwQlRYdHFhR3g2YUhsZmF6TnFlVGwzWVR0clh6YzRNa1V3YUcxcWZRPT0nCg==

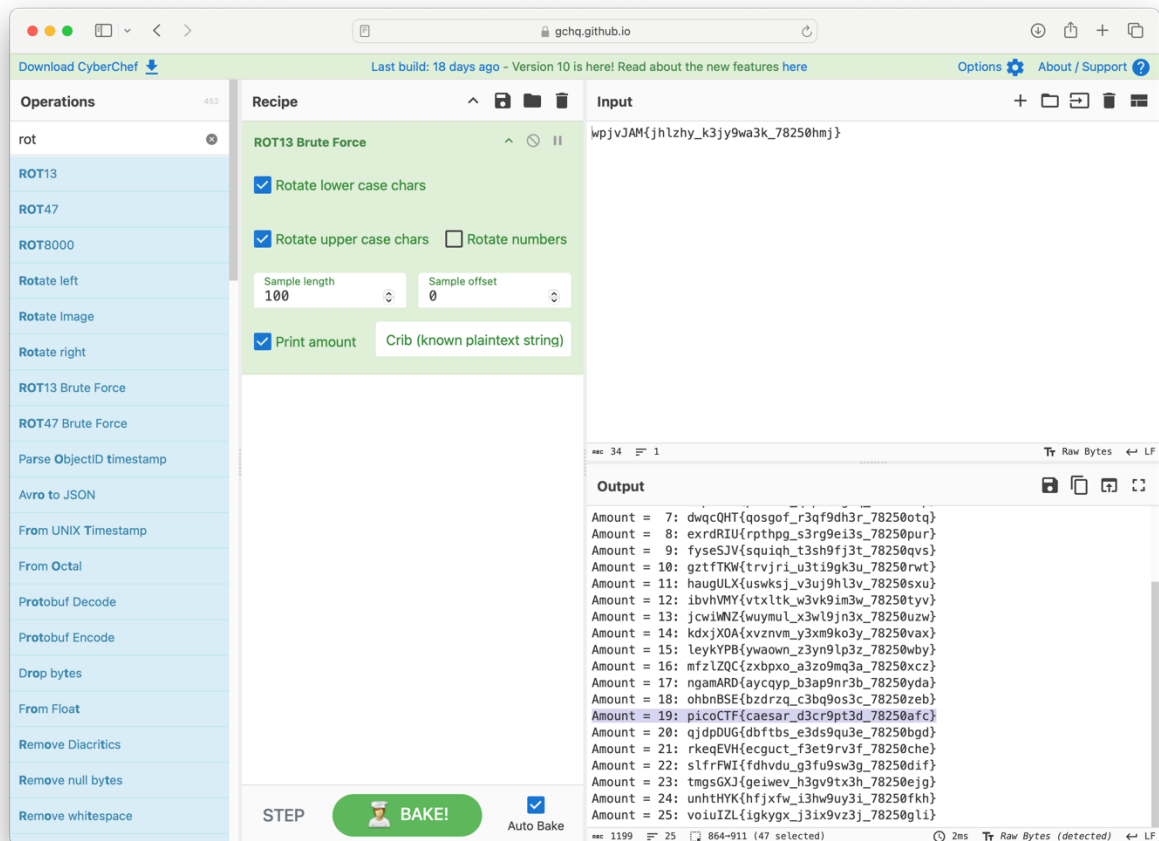
After reviewing the message we can see that the message is encoded in base64, we can assume that because it's a continuous string, it contains letters from (A-Z) with both lower and uppercases, having = at the end of the message is also a good giveaway since in base 64 its used as padding for data that does not firt into 3-byte chunks.



With that information we load the message into the CyberChef website and decode it using the Base 64 function.



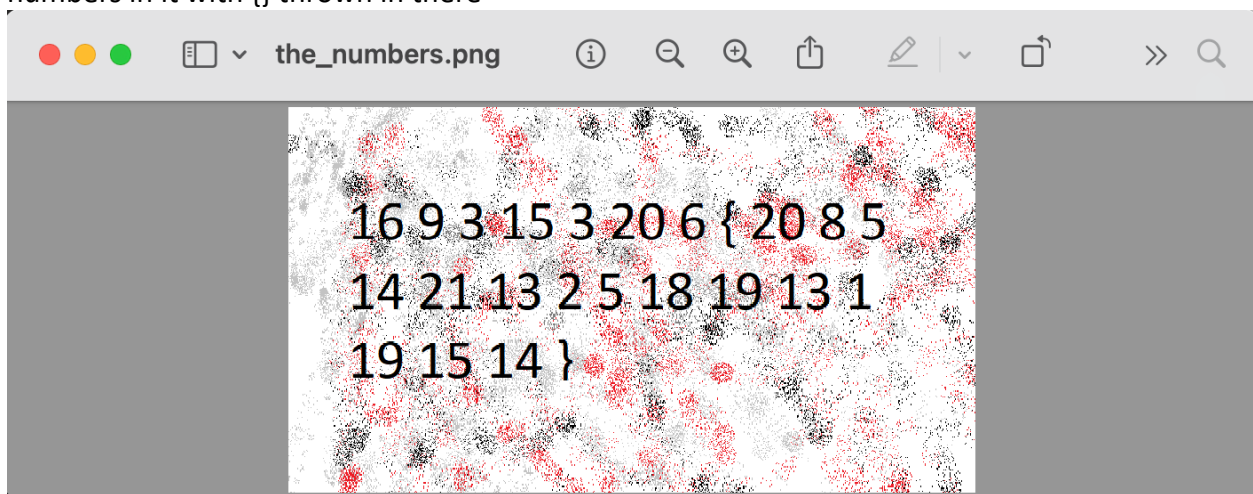
Once that is done we get another encrypted message, looking closely at the message it only contains letters from the A-Z so it might be a Rotation Cypher or ROT, since we don't know which rotation has been used we try to brute force the problem and chose to use the ROT 13 brute Force function.



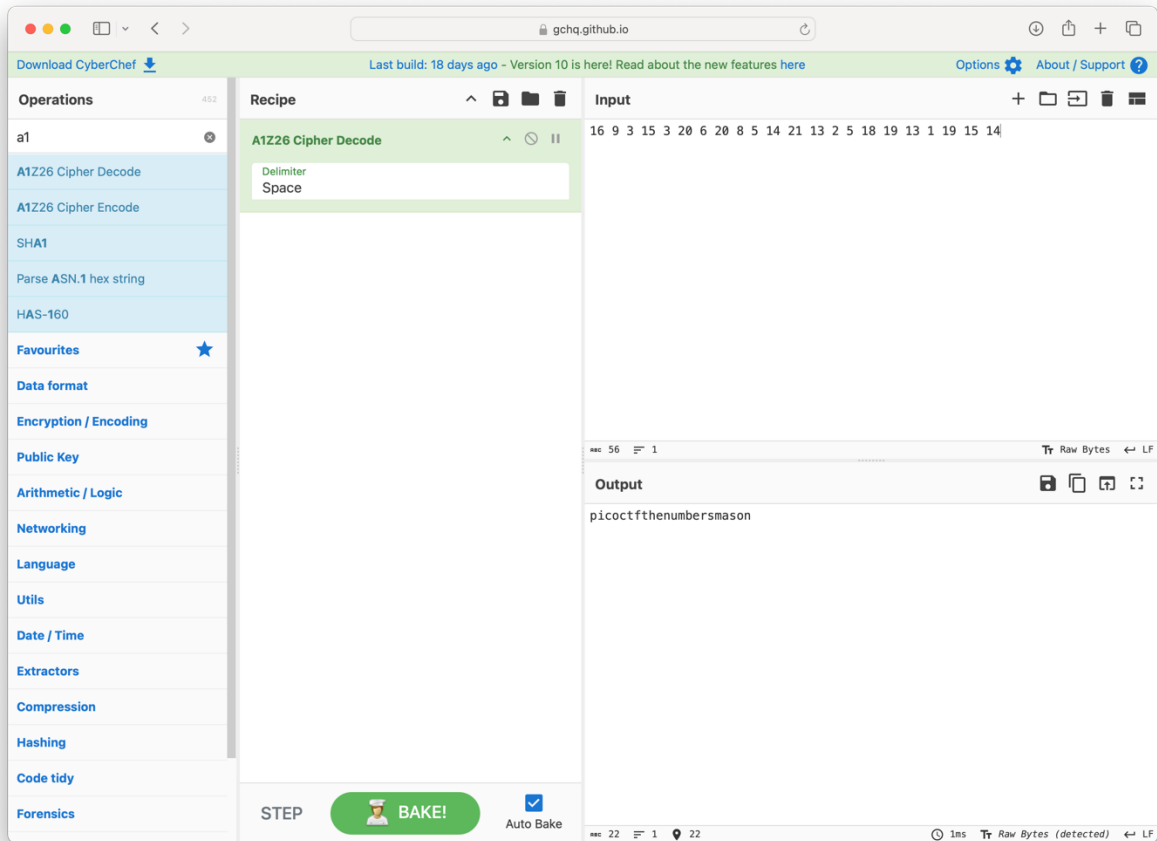
We check the different results and see that the amount 19 gives us a complete message that can be our solution therefore we copy that message and as we thought that is the message decoded.

Ejercicio 2

We download the file that comes with the problem and see its an image with a bunch of numbers in it with {} thrown in there

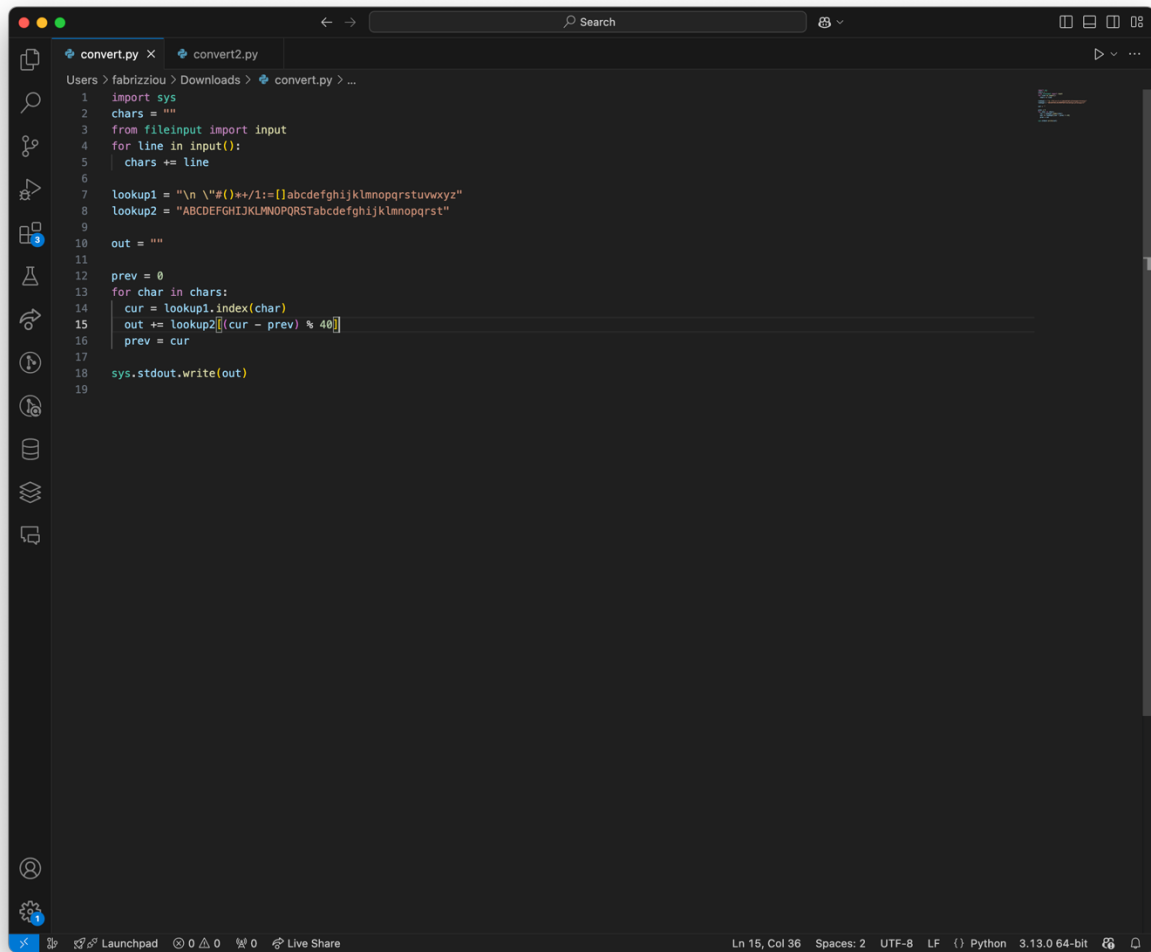


The first thing that comes to mind is that it is a numerical cypher with so we proceed to write the numbers down in the CyberChef page, once that is done we look for the cypher function that can help us which turns out to be A1Z26 Cipher Decode, this gives us the decoded message `picocftthenumbersmason` as an answer which proves to be correct once we tried it.



Ejercicio 3

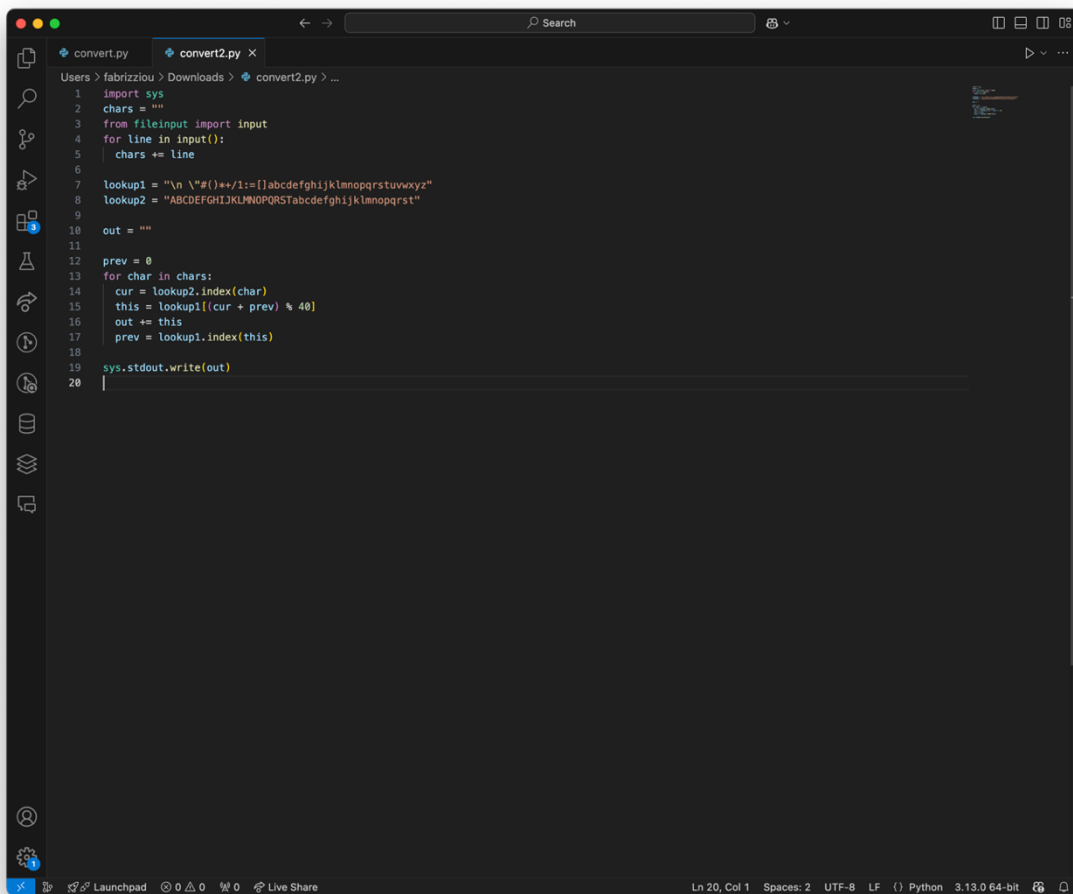
First we download the files in the problem which are a ciphertext and the encoder convert.py



The image shows a code editor window with a dark theme. The title bar at the top indicates the file is 'convert.py' and the current directory is 'Users > fabrizziou > Downloads'. The code is written in Python and implements a Caesar cipher encoder. It reads input from 'input()' and writes the encrypted output to 'sys.stdout.write(out)'. The encryption is performed by shifting each character in the input string by a fixed amount (40) using a lookup table. The lookup table is defined as 'lookup1 = "\n \\"#()*+/,:=[]abcdefghijklmnopqrstuvwxyz"' and 'lookup2 = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"'. The code is as follows:

```
1 import sys
2 chars = ""
3 from fileinput import input
4 for line in input():
5     chars += line
6
7 lookup1 = "\n \\"#()*+/,:=[]abcdefghijklmnopqrstuvwxyz"
8 lookup2 = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
9
10 out = ""
11
12 prev = 0
13 for char in chars:
14     cur = lookup1.index(char)
15     out += lookup2[(cur - prev) % 40]
16     prev = cur
17
18 sys.stdout.write(out)
19
```

The status bar at the bottom shows the current position is 'Ln 15, Col 36', the file encoding is 'UTF-8', and the Python version is '3.13.0 64-bit'.



The image shows a code editor window with two tabs: 'convert.py' and 'convert2.py'. The 'convert2.py' tab is active, displaying a Python script that implements a Vigenere cipher decoder. The script reads input from 'fileinput' and processes it using two lookup tables to shift characters back to their original positions. The status bar at the bottom indicates the current position is Line 20, Column 1, with 2 spaces, in UTF-8 encoding, using the Python 3.13.0 64-bit interpreter.

```
1 import sys
2 chars = ""
3 from fileinput import input
4 for line in input():
5     chars += line
6
7 lookup1 = "\n \\#()+=/!:=[]abcdefghijklmnopqrstuvwxyz"
8 lookup2 = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
9
10 out = ""
11
12 prev = 0
13 for char in chars:
14     cur = lookup2.index(char)
15     this = lookup1[(cur + prev) % 40]
16     out += this
17     prev = lookup1.index(this)
18
19 sys.stdout.write(out)
20
```

We proceed to change the code of convert.py so that instead of encoding it decodes the message, once that is done we can proceed to use the new code convert2.y to decode the ciphertext.

```
Downloads — -zsh — 90x30
(base) fabrizziou@FabrizzioUs-MacBook-Pro Downloads % echo abcd | python3 convert.py
OBBBd%
(base) fabrizziou@FabrizzioUs-MacBook-Pro Downloads % echo -n "OBBBd" | python3 convert2.py
y
abcd
(base) fabrizziou@FabrizzioUs-MacBook-Pro Downloads % cat ciphertext |python3 convert2.py
#asciiororder
#fortychars
#selfinput
#pythontwo

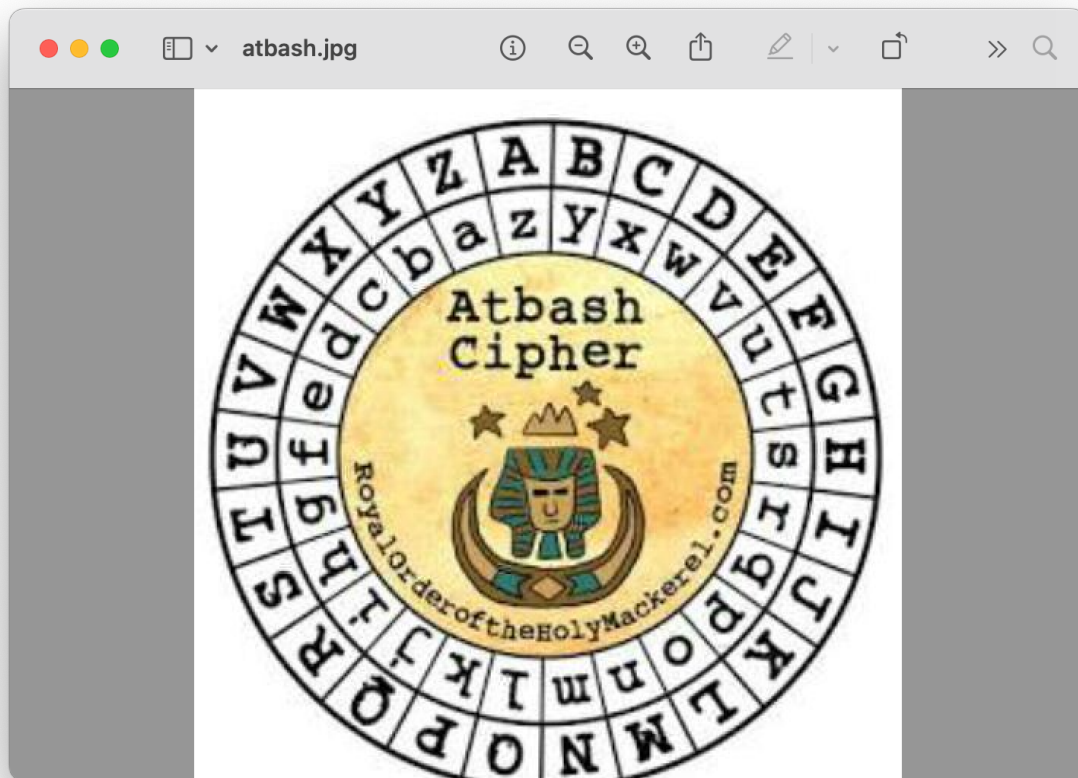
chars = ""
from fileinput import input
for line in input():
    chars += line
b = 1 / 1

for i in range(len(chars)):
    if i == b * b * b:
        print chars[i] #prints
        b += 1 / 1
(base) fabrizziou@FabrizzioUs-MacBook-Pro Downloads % cat ciphertext |python3 convert2.py
>file.py
```

As seen in the image we proceed to test if the new code can indeed decode the message by encoding abcd with the convert.py and using the encoded result in convert2.py once we made sure it decodes we proceed to pass the cyphertext which gave us a new encoded message we save said encoded message in a file.py and using that file as an argument for convert2.py we get the answer which is: picoCTF{adlibs}

Ejercicio 4

First we download the file from the problem which is a JPG with an Atbash Cipher as its shown in the image



The first step would be to get the ciphertext from that image and for that we used an Steganografic decoder

futureboy-us.translate.goog

Google Traductor inglés → español Traducción

Decodificador esteganográfico

Este formulario decodifica la carga útil que estaba oculta en una imagen JPEG o un archivo de audio WAV o AU mediante el [formulario de codificación](#) . Cuando envíe el archivo, se le solicitará que guarde el archivo de carga útil resultante en el disco. Este formulario también puede ayudarlo a adivinar cuál es la carga útil y su tipo de archivo...

Seleccione un archivo JPEG, WAV o AU para decodificar:

Choose File atbash.jpg

Contraseña (puede estar en blanco):

☐ Ver la salida sin procesar como tipo MIME text/plain

☐ Adivina la carga útil

☐ Pregunta para guardar (debe adivinar el tipo de archivo usted mismo).

Submit

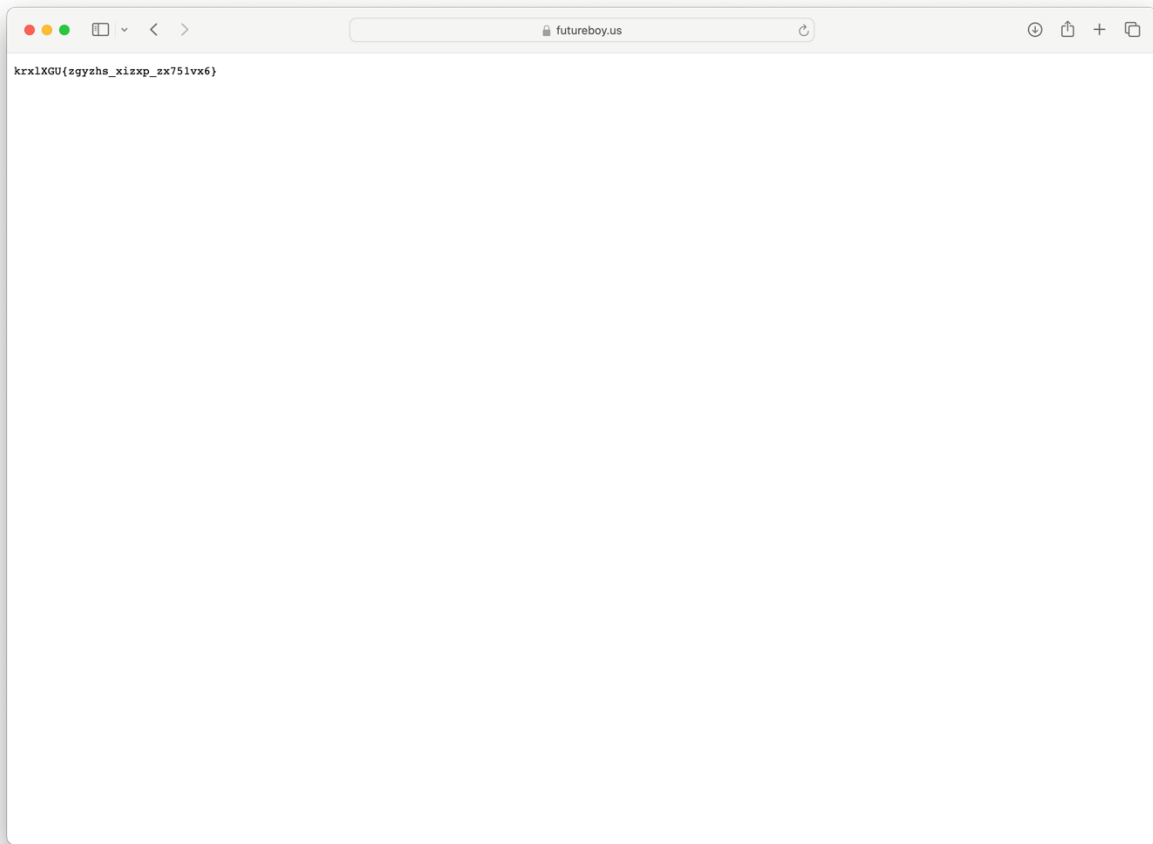
Para utilizar este formulario, primero debe [codificar un archivo](#) .

Estas páginas utilizan el programa [steghide](#) para realizar esteganografía, y los archivos generados son totalmente compatibles con steghide.

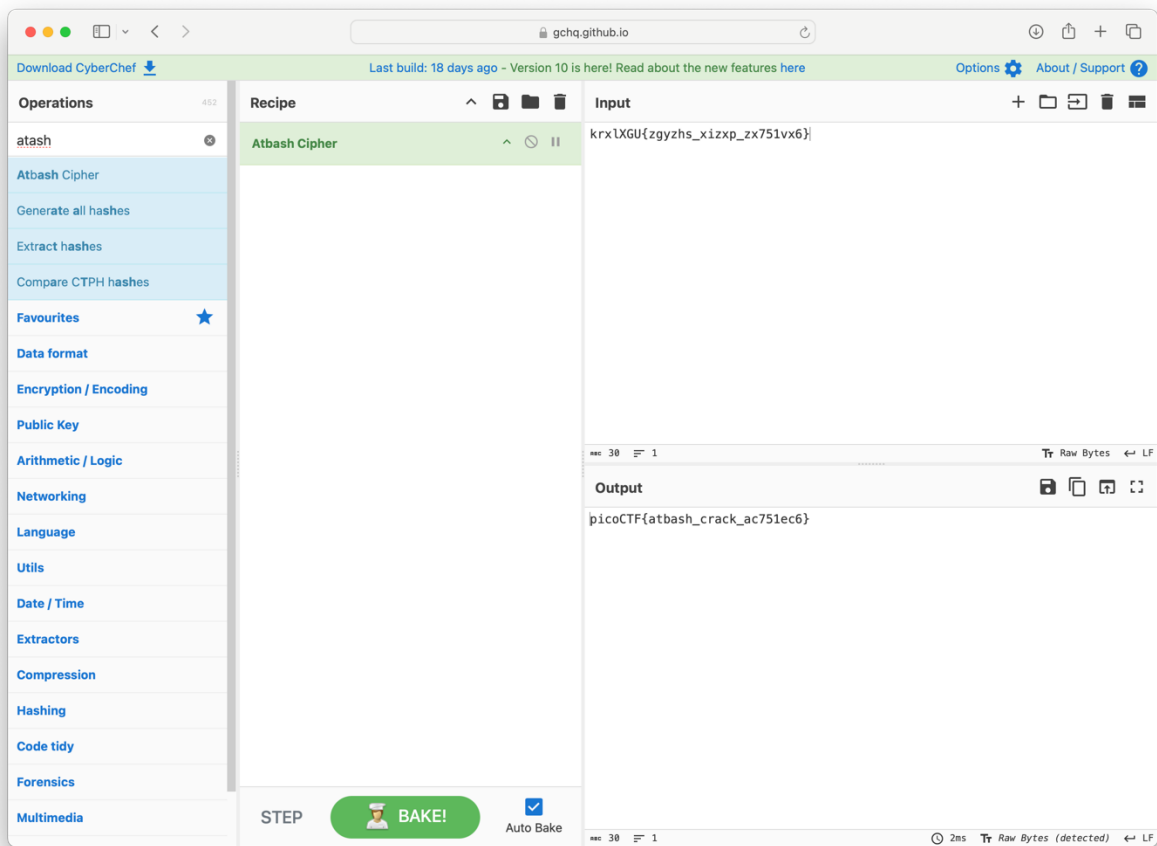
Por favor envíe comentarios o preguntas a [Alan Eliassen](#) .

[Regreso al servidor de inicio de Alan](#)

This decoder gave us the ciphertext which is shown in the next image.



Now that we have the ciphertext we can go to the CyberChef page to help us decode it using the function decoder for Atbash Cipher



This gives us the message picoCTF{atbash_crack_ac751ec6}

Ejercicio 5

First of all we downlado both files te message and the password so that we can begin

rsa_oracle



Medium

Cryptography

picoCTF 2024

browser_webshell_solvable

AUTHOR: GEOFFREY NJOGU

Description

Can you abuse the oracle?

An attacker was able to intercept communications between a bank and a fintech company. They managed to get the [message](#) (ciphertext) and the [password](#) that was used to encrypt the message. Additional details will be available after launching your challenge instance.

This challenge launches an instance on demand.

Its current status is:

NOT_RUNNING

Launch
Instance

Hints ?

1 2 3 4

OpenSSL can be used to decrypt the message. e.g

```
openssl enc -aes-256-cbc  
-d ...
```

1,970 users solved



95%

Liked



picoCTF{FLAG}

Submit
Flag

After that we need to launch instance in the terminal we use the nc titan.picoctf.net in my case 55959 and it shows us that we can encrypt or decrypt if we try the message it tells us that good try but be more creative.

```
Downloads — -zsh — 111x24
(env) (base) fabrizziou@FabrizzioUs-MacBook-Pro Downloads % cat secret.enc
Salted__g???0' ^?      1??h??i???p2E???Lc}5j??H?|UV;??h@?=??
(env) (base) fabrizziou@FabrizzioUs-MacBook-Pro Downloads % cat password.enc
257513595098311731523456852285799527766211312807607183776349206976398976001860473381326592977224529222304628809
8298720343542517375538185662305577375746934
(env) (base) fabrizziou@FabrizzioUs-MacBook-Pro Downloads % nc titan.picoctf.net 55959
*****
*****THE ORACLE*****
*****
what should we do for you?
E --> encrypt D --> decrypt.
E
enter text to encrypt (encoded length must be less than keysize):

encoded cleartext as Hex m: 0

ciphertext (m ^ e mod n) 0

what should we do for you?
E --> encrypt D --> decrypt.
```

```
Users > fabrizziou > Downloads > vi.py > ...
7 msg = conn.recvuntil('decrypt:')
8 print(msg.decode())
9
10 # Send the encryption option
11 conn.sendline(b'E')
12
13 # Receive the server's response
14 msg = conn.recvuntil('keysize:')
15 print(msg.decode())
16
17 # Send the number 2 for encryption
18 conn.sendline(b'\x02')
19 msg = conn.recvuntil('ciphertext (m ^ e mod n)')
20 msg = conn.recvline()
21
22 # Get the value of 2^e and multiply it by m^e from the password.enc file
23 cipher_value = int(msg.decode()) * 42282734711525709938577552090406111432273362451908758476491428075018489608478515
24
25 # Select the decryption option
26 msg = conn.recvuntil('decrypt:')
27 print(msg.decode())
28 conn.sendline(b'D')
29
30 # Send the value of 2^e * m^e for decryption
31 msg = conn.recvuntil('decrypt:')
32 print(msg.decode())
33 conn.sendline(str(cipher_value))
34
35 # Receive the decrypted response
36 msg = conn.recvuntil('hex (c ^ d mod n):')
37 print(msg.decode())
38 msg = conn.recvline()
39 print(msg.decode())
40
41 # Convert the response from hexadecimal to an integer and divide by 2
42 plaintext = int(msg, 16) // 2
43 print(hex(plaintext))
44
45 # Convert the result to ASCII
46 ascii_text = bytes.fromhex(hex(plaintext)[2:]).decode('ascii')
```

To bypass this restriction we used a padding attack base in RSA multiplicative properties, to be more precise we took the encrypted message and modified it multiplying it by a value of 2. Because RSA follows the mathematical property $D(E(m) * 2^e \text{ mod } n) = 2 * m \text{ mod } n$

Now we ask the server to decrypt this modified ciphertext, and we get a scaled version of the decrypted text dividing by 2 the results we can recover the actual text.

```
Downloads — -zsh — 111x24
: //docs.pwntools.com/#bytes
response = connection.recvuntil('decrypt:')

Enter text to decrypt:
/Users/fabrizziou/Downloads/vi.py:31: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https
: //docs.pwntools.com/#bytes
connection.send(str(num)+'\n')
/Users/fabrizziou/Downloads/vi.py:33: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https
: //docs.pwntools.com/#bytes
response = connection.recvuntil('hex (c ^ d mod n):')
decrypted ciphertext as hex (c ^ d mod n):
c8c2607272

0x6461303939
da099
[*] Closed connection to titan.picoctf.net port 55959
(venv) (base) fabrizziou@FabrizzioUs-MacBook-Pro Downloads % openssl enc -aes-256-cbc -d -in secret.enc -k da09
9
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
bad decrypt
8518635328:error:06065064:digital envelope routines:EVP_DecryptFinal_ex:bad decrypt:crypto/evp/evp_enc.c:612:
X?p8?g??/? ?uj??'t{[?}6?il*%
(venv) (base) fabrizziou@FabrizzioUs-MacBook-Pro Downloads %
```

As seen in this image I could not recover the plaintext in my computer for some reason it said the key was deprecated.

Upon reviewing the key that decrypted the Python code, we realized that both my classmates and I received the same answer, so I proceeded to use the decoding they came up with, obtaining the correct answer:

Answer: picoCTF({su((3ss_(r@ck1ng_r3@_da099d93}

The terminal image is proof that even using the hint provided by the exercise, openssl enc -aes-256-cbc -d. does not decrypt the message mentioned.