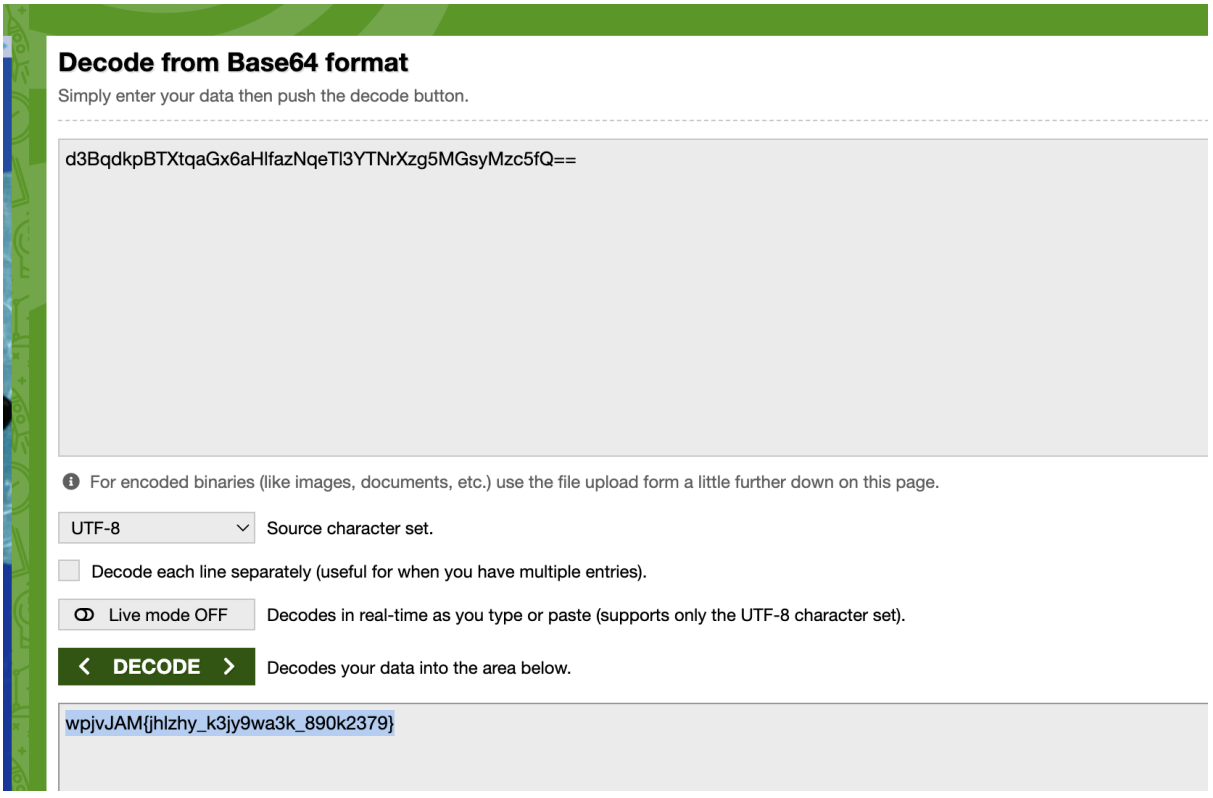
DEBER 1

1. **interencdec**

   Method 1:

   - First, download the file given.
   - Open File in type editor
   - Into the file, it is a text in 64 base encoding.
   - In a browser, search for a web that decodes this type of text (we use https://www.base64decode.org/).
   - In Pico CTF hint, it is mentioned that “Engaging in various decoding processes is of utmost importance”, so we need to decode many times.
   - Especifically, we entered the first chain (YidkM0JxZGtwQlRYdHFhR3g2YUhsZmF6TnFlVGwzWVROclh6ZzVNR3N5TXpjNWZRPT0nCg==) and we got b'd3BqdkpBTXtqaGx6aHlfazNqeTl3YTNrXzg5MGsyMzc5fQ==’. So we need to enter this to decode again.



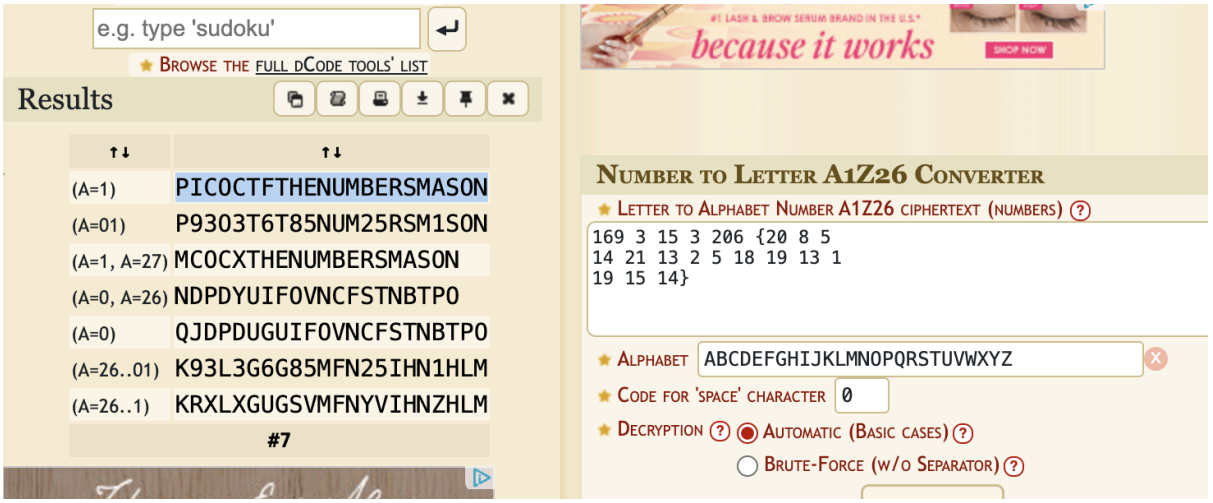
   - We got wpjvJAM{jhlzhy_k3jy9wa3k_890k2379}. This is supposed to be the key, and we entered to a Caesar cipher to decode (https://www.dcode.fr/caesar-cipher).
   - Then, in the left side of the web, we got the final flag.

Method 2:

- Open the two codes (ex1.1.py and ex1.2.py), that decodes for base 64 and decodes for caesars cypher.
- Execute ex1.1.py, and put in terminal the code given in txt downloaded, and repeat like option 1.
- Execute ex1.2.py, and put in terminal the output of ex1.1.py. Search into all the responses the plain text (Shift 19: picoCTF{caesar_d3cr9pt3d_890d2379}).

2. The Numbers

- First, download the image given.
- As the exercise 1, the image have a code that needs to be decoded with Caesar cipher (https://www.dcode.fr/caesar-cipher).
- Entered the code, we got:



- And we have to put into the format given in the hint (The flag is in the format PICOCTF{}), so the final flag was : picoctf{thenumbersmason}.

Method 2:

- Open the code (ex2.py), that decodes for A1Z26.
- Execute ex2.py, and put in terminal the code given in image downloaded.
- And we have to put into the format given in the hint (The flag is in the format PICOCTF{}), so the final flag was : picoctf{thenumbersmason}.

3. C3

- Download the python file and the txt file.
- Open txt in vs code and analyze it.
- the file txt is read line by line and stored in the variable chars. For each character in chars, its position is searched in lookup1. The difference of this index with the index of the previous character (prev) is calculated. This difference is used to select the character in lookup2 cyclically (modulo 40). prev is updated with the current index.
- We execute erick_dsuarez10@MacBook-Pro-de-Erick-2 downloads % python3 convert2.py < ciphertext and get this code and save into final.py

```
zsh: command not found: python
erick_dsuarez10@MacBook-Pro-de-Erick-2 downloads % python3 convert2.py < ciphertext
#asciiorder
#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
```

- Ask ChatGPT to convert this code to python3.
- Execute python3 final.py < ciphertext.
- The final flag is: **adlibs**

4. **HideToSee**

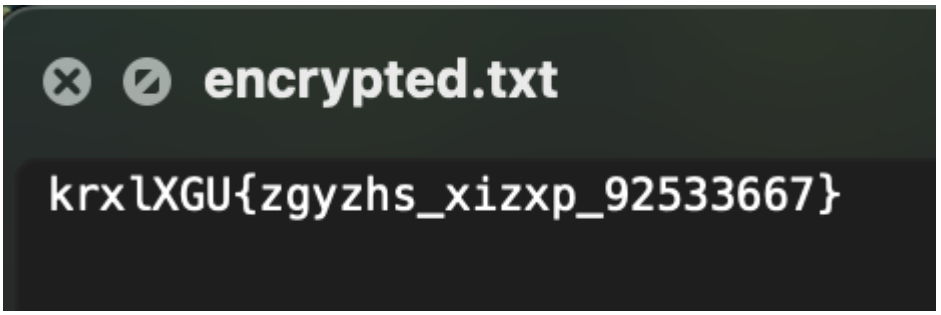
For Mac

- Download the image that is given.
- Open the image and there is no clear information to solve this.
- After investigating, the solution comes along searching for the hidden information in the jpg file. For this, we need steghide.
- Open directory in terminal, install steghide with macports “sudo port install steghide”
- Then execute steghide info atbash.jpg
- give enter and there is this output
- There is a new file named encrypted.txt that contains the key.

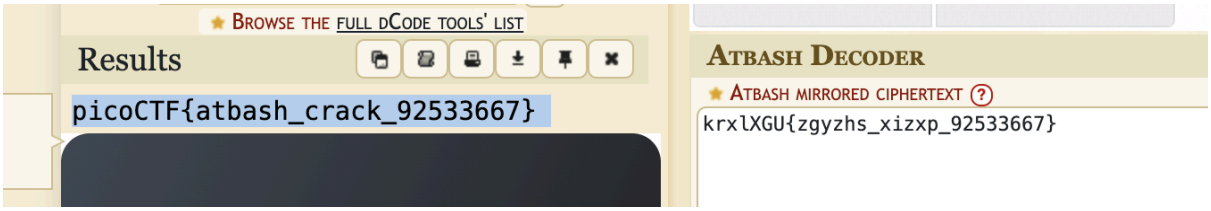
```
erick_dsuarez10@MacBook-Pro-de-Erick-2 downloads % steghide info atbash.jpg

"atbash.jpg":
  formato: jpeg
  capacidad: 2.4 KB
?Intenta informarse sobre los datos adjuntos? (s/n) s
[Anotar salvoconducto:
  archivo adjunto "encrypted.txt":
    tama?o: 31.0 Byte
    encriptado: rijndael-128, cbc
    compactado: si
erick_dsuarez10@MacBook-Pro-de-Erick-2 downloads % steghide extract -sf atbash.
pg

[Anotar salvoconducto:
anot? los datos extra?dos e/"encrypted.txt".
```



- Then we need to go to (https://www.dcode.fr/atbash-cipher) and decode this.
- We got the flag: picoCTF{atbash_crack_92533667}



For Linux

- Download the image that is given.
- Open the image and there is no clair information to solve this.
- After investigating, the solution cames along searching of the hidden information in the jpg file. For this, we need steghide.
- Open directory in terminal, install steghide with macports “sudo apt update && sudo apt install sssehide -y”
- Then execute steghide info atbash.jpg
- give enter and there is this output
- There is a new file named encrypted.txt that contains the key.
- Then we need to go to (https://www.dcode.fr/atbash-cipher) and decode this.
- We got the flag: picoCTF{atbash_crack_92533667}.

5. **rsa_oracle**

In this challenge, we were given two files:

- password.enc: An RSA-encrypted password that the decryption oracle refuses to decrypt directly.
- secret.enc: An AES-256-CBC–encrypted file that uses the password from password.enc as its key.

The goal was to recover the plaintext password from password.enc by exploiting RSA’s multiplicative property and then use that password to decrypt secret.enc (revealing the flag).

Opcion 1: Using a Pwntools Python Script

1. Download the Challenge Files
   Use wget to download the encrypted files by code or by clicking the links:
2. Connect to the RSA Oracle
   We connect to the server (e.g., titan.picoctf.net on port 61923) using pwntools.
3. Read the Forbidden Ciphertext
   The script opens password.enc, reads its content, and converts it to an integer:

```
with open("password.enc") as file:
    c = int(file.read())
```

4. Encrypt a Known Value
   The server is prompted to encrypt the integer 2 (sent as \x02). The result, denoted as c_a, equals:
5. Construct the Blinded Ciphertext
   Multiply c_a (encryption of 2) by c (the forbidden ciphertext):
6. Send Blinded Ciphertext for Decryption
   By sending c_blinded to the server, the oracle decrypts it (since it is not the forbidden ciphertext) and returns:
7. Unblind the Result
   The script converts the hexadecimal response to an integer and divides by 2 to recover ppp:

```
password = int(p.recvline(), 16) // 2
```

Finally, the integer is converted to a byte string and then decoded to yield the plaintext password (in our case, "4955e").

8. Decrypt the Secret File
   With the recovered password, we decrypt secret.enc using OpenSSL:

The resulting file flag.txt contains the final flag.

Script Code:

```
from pwn import *

context.log_level = 'critical'

p = remote("titan.picoctf.net", 61923)

p.recvuntil(b"decrypt.")

# Read forbidden ciphertext from password.enc

with open("password.enc") as file:

        c = int(file.read())

# Request encryption: send "E" then encrypt the number 2

p.sendline(b"E")
```

```
p.recvuntil(b"keysize): ")

p.sendline(b"\x02")

p.recvuntil(b"mod n) ")

c_a = int(p.recvline())

# Request decryption: send "D" then the blinded ciphertext (c_a * c)

p.sendline(b"D")

p.recvuntil(b"decrypt: ")

p.sendline(str(c_a * c).encode())

p.recvuntil(b"mod n): ")

# The server returns (2 * password) in hex, so divide by 2 to get the password

password = int(p.recvline(), 16) // 2

password = password.to_bytes(len(str(password))-7, "big").decode("utf-8")

print("Password:", password)
```

When executed, this script prints the password (e.g., "4955e"), which is then used to decrypt secret.enc.

Opcion 2: Manual RSA Oracle Exploitation Using Interactive Python

1. Convert password.enc to an Integer
   Open a Python shell and run:

```
with open("password.enc", "rb") as f:
    ciphertext_bytes = f.read()
c = int.from_bytes(ciphertext_bytes, "big")
print("Forbidden ciphertext (c):", c)
```

2. Manually Obtain Enc(2) from the Server
   You can use netcat or telnet to connect to the oracle server:

```
nc titan.picoctf.net 61923
```

3. Calculate the Blinded Ciphertext
   In your Python shell, compute:

```
c_blinded = c_a * c  # Ensure you use the exact value received for c_a
print("Blinded ciphertext:", c_blinded)
```

4. Unblind to Recover the Password
   Back in Python, convert the hex string to an integer and divide by 2:

```
response_hex = "the_hex_string_returned_from_server"  # Replace with actual response
decrypted_value = int(response_hex, 16)
p_int = decrypted_value // 2
# Convert integer to bytes, adjusting for proper length
p_bytes = p_int.to_bytes((p_int.bit_length() + 7) // 8, "big")
password = p_bytes.decode("utf-8")
print("Recovered password:", password)
```

You should see the password "4955e".

5. Decrypt secret.enc with OpenSSL
   Finally, use the password to decrypt the AES-encrypted file the same way:

Open flag.txt to reveal the final flag.

ANEXOS:

ex1.1.py:

```
import base64


def decode_base64_recursive(encoded_str):
    while True:
        try:
            decoded_bytes = base64.b64decode(encoded_str, validate=True)
            decoded_str = decoded_bytes.decode('utf-8')
```

```
            print(f"[DEBUG] Decodificado: {decoded_str}")  # Muestra cada paso de decodificación
            encoded_str = decoded_str  # Asignamos la nueva cadena para seguir decodificando
        except (base64.binascii.Error, UnicodeDecodeError):
            break  # Si ya no se puede decodificar más, salimos del bucle
    return encoded_str

if __name__ == "__main__":
    input_str = input("Ingrese la cadena en Base64: ")
    decoded_str = decode_base64_recursive(input_str)
    print(f"Texto decodificado: {decoded_str}")
```

ex1.2.py:

```
def caesar_cipher(text, shift):
    result = ""
    for char in text:
        if char.isalpha():
            shift_amount = shift % 26
            new_char = chr(((ord(char.lower()) - 97 + shift_amount) % 26) + 97)
            result += new_char.upper() if char.isupper() else new_char
        else:
            result += char
    return result

if __name__ == "__main__":
```

```
    input_text = input("Ingrese el texto a cifrar/descifrar con Caesar Cipher: ")
    print("Probando todos los desplazamientos posibles:")
    for shift in range(26):
        flag = caesar_cipher(input_text, shift)
        print(f"Shift {shift}: {flag}")
```

ex2.py:

```
import re

def a1z26_decoder(text):
    numbers = re.findall(r'\b\d+\b', text)  # Extrae todos los números separados
    decoded_text = ''.join(chr(int(num) + 64) if 1 <= int(num) <= 26 else '?' for num in numbers)
    return decoded_text

if __name__ == "__main__":
    input_text = input("Ingrese el texto con números para decodificar (A1Z26): ")
    flag = a1z26_decoder(input_text)
    print(f"Decodificado: {flag if flag else '[ERROR] No se encontraron números válidos en A1Z26'}")
```