UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

Guatemala 24 de abril de 2025
Mark Albrand 21004

Cifrado de información

Proyecto #1

Este proyecto tiene como finalidad comprender y aplicar conceptos de cifrados de flujo mediante la resolución de retos prácticos basados en algoritmos criptográficos como RC4, XOR y ChaCha20.
A través de estos ejercicios, se explorará la importancia de claves, nonces y seguridad en la implementación de cifrados modernos.

El proyecto consiste en 4 retos, donde en cada uno se debe decodificar un texto oculto dentro de una imagen y además encontrar una flag para obtener las credenciales del siguiente reto. A continuación se muestra el resumen de lo encontrado y la resolución de cada desafío.

Resumen

Texto

When Jack and the Beasts Pirates invaded Zou to look for Raizo of Wano Country,
the Mink Tribe chained him to the Road Poneglyph within the Whale Tree to keep him from showing himself.
Seventeen days later, the Straw Hats, Trafalgar Law, and Raizo's samurai comrades freed him from the Poneglyph.
After Robin deciphered the Road Poneglyph, Inuarashi explained its purpose of revealing the location of Laugh Tale when its information was combined with that of the other three Road Poneglyphs.

Flags

- FLAG_348a56efa179e2911c421c9c6ad83869
- FLAG_7bf472c0a9f8c3bda0057209c9ad1aed
- FLAG_fcbd8689ff72334dbd43f119306e1a9d
- FLAG_914bacd0923f836359edcf71b726b935

Luffy

Para iniciar este desafío, se utiliza las credenciales iniciales, y se buscan archivos que coincidan con el nombre a buscar (poneglyph).

```bash
docker exec -it {challengeX_ctf} bash
```

```bash
su luffy
password: onepiece
```

```bash
find . -name "poneglyph*"
```

Esto da como resultado:

```bash
luffy@2affc5ba053a:~$ find . -name "poneglyph*"
./ONEPIECE/East_Blue/Shells_Town/Casa_de_Helmeppo/poneglyph.zip
```

Usando 7z para extraer el archivo:
```bash
7z x ./East_Blue/Shells_Town/Casa_de_Helmeppo/poneglyph.zip
-oponeglyph_descomprimido
```

Ingresando la contraseña `onepiece` resulta en el siguiente archivo:

```bash
luffy@2affc5ba053a:~/ONEPIECE/poneglyph_descomprimido$ ls -a
.  ..  poneglyph.jpeg
```

```
luffy@2affc5ba053a:~/ONEPIECE$ 7z x ./East_Blue/Shells_Town/Casa_de_Helmeppo/poneglyph.zip -oponeglyph_descomprimido

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=C,Utf16=off,HugeFiles=on,64 bits,16 CPUs AMD Ryzen 9 7940HS w/ Radeon 780M Graphics      (A70F41),ASM,AES-NI)

Scanning the drive for archives:
1 file, 67683 bytes (67 KiB)

Extracting archive: ./East_Blue/Shells_Town/Casa_de_Helmeppo/poneglyph.zip
--
Path = ./East_Blue/Shells_Town/Casa_de_Helmeppo/poneglyph.zip
Type = zip
Physical Size = 67683


Would you like to replace the existing file:
  Path:     poneglyph_descomprimido/poneglyph.jpeg
  Size:     69117 bytes (68 KiB)
  Modified: 2025-04-21 21:30:18
with the file from archive:
  Path:     poneglyph.jpeg
  Size:     69117 bytes (68 KiB)
  Modified: 2025-04-21 21:30:18
? (Y)es / (N)o / (A)lways / (S)kip all / A(u)to rename all / (Q)uit? Y


Enter password (will not be echoed):
Everything is Ok

Size:       69117
Compressed: 67683
```

Ahora se mueve esta imagen al volumen compartido `/luffy_ctf` para trabajar con ella en la máquina local.

```bash
cp poneglyph.jpeg /luffy_ctf
```

Con la imagen en mi máquina local, se puede abrir sin problemas la imagen. Se puede notar que en la esquina superior
izquierda hay un texto que parece ser una llave, empezando con los caracteres 'eYU' y siguiendo con caracteres que parecen
no ser ASCII. Esto es un indicio de que este es lo que hay que descifrar usando XOR.

Al leer el contenido con python, se puede encontrar el segmento de bytes que representan este texto.

```text
eYU^\x14xPS[\x14S_T\x10@ZT\x10rQSBDC\x14bXBQ@WB\x10YZDPTUP\x12k_E\x14F^\x10\\[]Z\x10V[@\x11bQ]H^\x10_R\x12fQ^[\x12r_EZFCl\x1c
```

Usando esto como texto cifrado, y mi carnet `21004` como la llave, se obtiene la flag siguiente:

```text
When Jack and the Beasts Pirates invaded Zou to look for Raizo of Wano Country,
```

> Si se usa la imagen completa, el resultado no contiene ningún texto legible. Por lo tanto se asume
> que el texto cifrado es solo la parte que contiene el texto.

Por último, usando el siguiente comando se pueden encontrar diversos archivos de texto

```bash
luffy@2affc5ba053a:~/ONEPIECE$ find . -type f \( -iname "*.txt" -o -iname "*.enc" -o -iname "*.flag" \)
```

```text
./Dressrosa/Acacia/Casa_de_Rebecca/flag.txt
./Dressrosa/Royal_Palace/Casa_de_Viola/flag.txt
./East_Blue/Syrup_Village/Casa_de_Usopp/flag.txt
./East_Blue/Romance_Dawn/Casa_de_Shanks/flag.txt
./East_Blue/Shells_Town/Casa_de_Morgan/flag.txt
./East_Blue/Loguetown/Casa_de_Dragon/flag.txt
./Zou/Right_Belly_Fortress/Casa_de_Nekomamushi/flag.txt
./Zou/Right_Fore_Leg/Casa_de_Inuarashi/flag.txt
./Wano/Udon/Casa_de_Oden/flag.txt
./Fishman_Island/Coral_Mansion/Casa_de_Otohime/flag.txt
./Fishman_Island/Gyoverly_Hills/Casa_de_Otohime/flag.txt
```

```

```
(base) ~\Coding\UVG\Cifrados\Cifrados-Proyecto-1 git:[main]
docker exec -it 2affc5ba053a bash
luffy@2affc5ba053a:~/ONEPIECE$ find . -type f \(-iname "*.txt" -o -iname "*.enc" -o -iname "*.flag" \)
./Dressrosa/Acacia/Casa_de_Rebecca/flag.txt
./Dressrosa/Royal_Palace/Casa_de_Viola/flag.txt
./East_Blue/Syrup_Village/Casa_de_Usopp/flag.txt
./East_Blue/Romance_Dawn/Casa_de_Shanks/flag.txt
./East_Blue/Shells_Town/Casa_de_Morgan/flag.txt
./East_Blue/Loguetown/Casa_de_Dragon/flag.txt
./Zou/Right_Belly_Fortress/Casa_de_Nekomamushi/flag.txt
./Zou/Right_Fore_Leg/Casa_de_Inuarashi/flag.txt
./Wano/Udon/Casa_de_Oden/flag.txt
./Fishman_Island/Coral_Mansion/Casa_de_Otohime/flag.txt
./Fishman_Island/Gyoverly_Hills/Casa_de_Otohime/flag.txt
luffy@2affc5ba053a:~/ONEPIECE$ cat ./Dressrosa/Royal_Palace/Casa_de_Viola/flag.txt
747d71776b010508510104545651050855020d030053040603520953025355080 30c0408luffy@2affc5ba053a:~/ONEPIECE$
```

De estos, el segundo archivo `./Dressrosa/Royal_Palace/Casa_de_Viola/flag.txt` contiene el único texto que no es legible.

Aplicando el mismo método de XOR, se obtiene la siguiente flag:

```text
FLAG_348a56efa179e2911c421c9c6ad83869
```

```
El cifrado es: b'eYU^\x14xPS[\x14S_T\x10@ZT\x10rQSBDC\x14bXBQ@WB\x10YZDPTUP\x12k_E\x14F^\x10\\[]Z\x10V[@\x11bQ]H^\x10_R\x12fQ^[\x12r_EZFCI\x1c'

La flag es: 'When Jack and the Beasts Pirates invaded Zou to look for Raizo of Wano Country,'

La flag es: 'FLAG_348a56efa179e2911c421c9c6ad83869'

Process finished with exit code 0
```

Zoro

Para este segundo reto, se siguió un procedimiento similar al primero.

```bash
docker exec -it {challengeX_ctf} bash
```

Usando la flag encontrada en el reto anterior, se inicia sesión como `zoro`:

```bash
su zoro
```
```

password: FLAG_348a56efa179e2911c421c9c6ad83869
```

Se vuelve a usar el comando `find` para buscar archivos que contengan la palabra `poneglyph` en su nombre:

```bash
find . -name "poneglyph*"
```

Esto da como resultado:

```bash
zoro@89bf91b1a510:~/ONEPIECE$ find . -name "poneglyph*"
./East_Blue/Shells_Town/Casa_de_Rika/poneglyph.zip
```

Usando 7z para extraer el archivo:
```bash
7z x ./East_Blue/Shells_Town/Casa_de_Rika/poneglyph.zip
-oponeglyph_descomprimido
```

Ingresando la contraseña `FLAG_348a56efa179e2911c421c9c6ad83869` resulta en el siguiente archivo:

```bash
zoro@89bf91b1a510:~/ONEPIECE$ ls
Dressrosa East_Blue Fishman_Island Wano Zou poneglyph_descomprimido
zoro@89bf91b1a510:~/ONEPIECE$ cd poneglyph_descomprimido/
zoro@89bf91b1a510:~/ONEPIECE/poneglyph_descomprimido$ ls
poneglyph.jpeg
```
```

```
zoro@89bf91b1a510:~/ONEPIECE$ find . -name "poneglyph*"
./East_Blue/Shells_Town/Casa_de_Rika/poneglyph.zip
zoro@89bf91b1a510:~/ONEPIECE$ 7z x ./East_Blue/Shells_Town/Casa_de_Rika/poneglyph.zip -oponeglyph_descomprimido

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=C,Utf16=off,HugeFiles=on,64 bits,16 CPUs AMD Ryzen 9 7940HS w/ Radeon 780M Graphics      (A70F41),ASM,AES-NI)

Scanning the drive for archives:
1 file, 54862 bytes (54 KiB)

Extracting archive: ./East_Blue/Shells_Town/Casa_de_Rika/poneglyph.zip
--
Path = ./East_Blue/Shells_Town/Casa_de_Rika/poneglyph.zip
Type = zip
Physical Size = 54862


Enter password (will not be echoed):
Everything is Ok

Size:       54474
Compressed: 54862
zoro@89bf91b1a510:~/ONEPIECE$ ls
Dressrosa  East_Blue  Fishman_Island  Wano  Zou  poneglyph_descomprimido
zoro@89bf91b1a510:~/ONEPIECE$ cd poneglyph_descomprimido/
zoro@89bf91b1a510:~/ONEPIECE/poneglyph_descomprimido$ ls
poneglyph.jpeg
```

Ahora se mueve esta imagen al volumen compartido `/zoro_ctf` para trabajar con ella en la máquina local.

```bash
cp poneglyph.jpeg /zoro_ctf
```

Con la imagen en mi máquina local, se puede abrir sin problemas la imagen. Se puede notar nuevamente que en la esquina superior
izquierda hay un texto que parece ser una llave, empezando con los caracteres 'FYU' y siguiendo con caracteres que parecen
no ser ASCII. Esto es un indicio de que este es lo que hay que descifrar usando XOR.

Al leer el contenido con python, se puede encontrar el segmento de bytes que representan este texto.

```text
FYU\x10y[_[\x10`@XRU\x14QYQYZWU\x10X]_\x11D_\x14FYU\x10f]PT\x10d]_UWXKA
X\x10C[EXYZ\x12EXU\x14eYQ\\Q\x12eBUQ\x12E_\x10_WT@\x10\\[\\\x10VF]\\\x10C\
\]FY^S\x12YY]GW]V\x1e
```

Usando esto como texto cifrado, y mi carnet `21004` como la llave, se obtiene la flag siguiente:

```text
the Mink Tribe chained him to the Road Poneglyph within the Whale Tree to keep him from showing himself.
```

Por último, usando el siguiente comando se pueden encontrar diversos archivos de texto

```bash
zoro@89bf91b1a510:~/ONEPIECE$ find . -type f \( -iname "*.txt" -o -iname "*.enc" -o -iname "*.flag" \)
```

```text
./Dressrosa/Acacia/Casa_de_Riku_Dold_III/flag.txt
./Dressrosa/Acacia/Casa_de_Rebecca/flag.txt
./Dressrosa/Toy_House/Casa_de_Trebol/flag.txt
./East_Blue/Syrup_Village/Casa_de_Usopp/flag.txt
./East_Blue/Arlong_Park/Casa_de_Nami/flag.txt
./East_Blue/Baratie/Casa_de_Patty/flag.txt
./Zou/Left_Hind_Leg/Casa_de_Nekomamushi/flag.txt
./Zou/Left_Fore_Leg/Casa_de_Inuarashi/flag.txt
```

./Wano/Flower_Capital/Casa_de_Oden/flag.txt
./Wano/Udon/Casa_de_Kozuki_Momonosuke/flag.txt
./Fishman_Island/Gyoverly_Hills/Casa_de_Shirahoshi/flag.txt
```

```
zoro@89bf91b1a510:~/ONEPIECE$ find . -type f \( -iname "*.txt" -o -iname "*.enc" -o -iname "*.flag" \)
./Dressrosa/Acacia/Casa_de_Riku_Dold_III/flag.txt
./Dressrosa/Acacia/Casa_de_Rebecca/flag.txt
./Dressrosa/Toy_House/Casa_de_Trebol/flag.txt
./East_Blue/Syrup_Village/Casa_de_Usopp/flag.txt
./East_Blue/Arlong_Park/Casa_de_Nami/flag.txt
./East_Blue/Baratie/Casa_de_Patty/flag.txt
./Zou/Left_Hind_Leg/Casa_de_Nekomamushi/flag.txt
./Zou/Left_Fore_Leg/Casa_de_Inuarashi/flag.txt
./Wano/Flower_Capital/Casa_de_Oden/flag.txt
./Wano/Udon/Casa_de_Kozuki_Momonosuke/flag.txt
./Fishman_Island/Gyoverly_Hills/Casa_de_Shirahoshi/flag.txt
zoro@89bf91b1a510:~/ONEPIECE$ cat ./East_Blue/Baratie/Casa_de_Patty/flag.txt
072d03661519c3535ace983613f97d543b0f82500691e1d6e0eafb10e7bdbb51521c6b2e67zoro@89bf91b1a510:~/ONEPIECE$
zoro@89bf91b1a510:~/ONEPIECE$
```

De estos, el sexto archivo `./East_Blue/Baratie/Casa_de_Patty/flag.txt` contiene el único texto que no es legible.

Aplicando el mismo método de XOR, la llave no se logra descifrar. Como el nombre del reto lo dice, esta esta codificada
usando RC4. Usando el scipt desarrollado de Python para romper RC4, se logra obtener la siguiente flag:

```text
FLAG_7bf472c0a9f8c3bda0057209c9ad1aed
```

```
El cifrado es: b'FYU\x10y[_[\x10`@XRU\x14QYQYZWU\x10X]_\x11D_\x14FYU\x10f]PT\x10d]_UWXKAX\x10C[EXYZ\x12EXU\x14eYQ\\Q\x12eBUQ\x12E_\x10_WT@\x10\\[\\\x10VF]\\\x10C\\]FY^S\x12YY]GW]V\x1e'

La flag es: 'the Mink Tribe chained him to the Road Poneglyph within the Whale Tree to keep him from showing himself.'

La flag es: 'b'FLAG_7bf472c0a9f8c3bda0057209c9ad1aed''

Process finished with exit code 0
```

Usopp

Para este tercer reto, se siguió la misma metodología que en los dos anteriores.

```bash
docker exec -it {challengeX_ctf} bash
```
```

Usando la flag encontrada en el reto anterior, se inicia sesión como `usopp`:

```bash
su usopp
password: FLAG_7bf472c0a9f8c3bda0057209c9ad1aed
```

Se vuelve a usar el comando `find` para buscar archivos que contengan la palabra `poneglyph` en su nombre:

```bash
find . -name "poneglyph*"
```

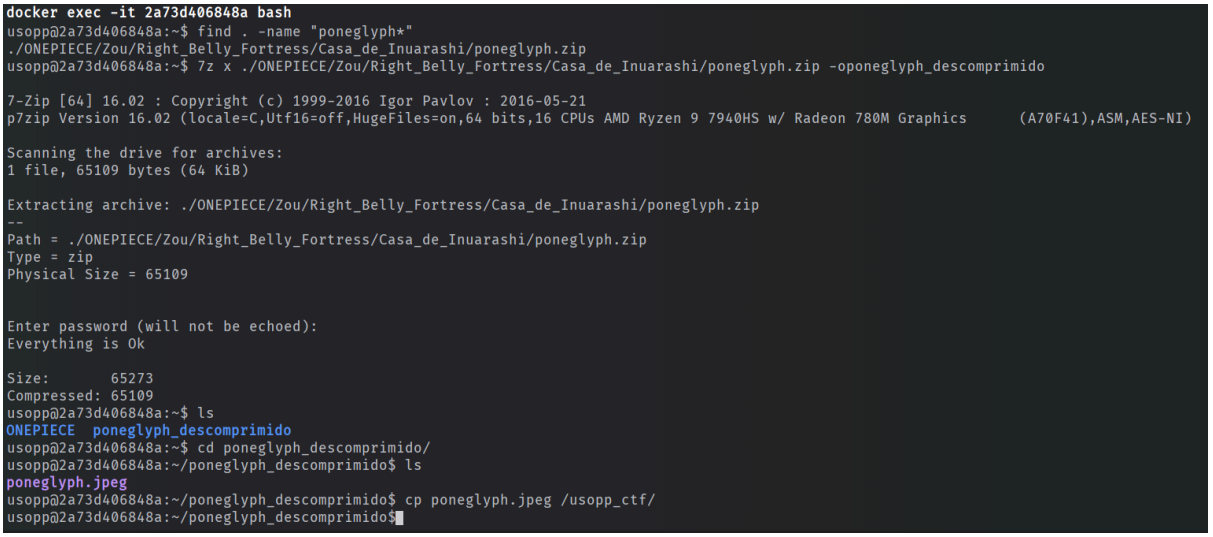
Esto da como resultado:

```bash
usopp@2a73d406848a:~$ find . -name "poneglyph*"
./ONEPIECE/Zou/Right_Belly_Fortress/Casa_de_Inuarashi/poneglyph.zip
```

Usando 7z para extraer el archivo:
```bash
7z x ./ONEPIECE/Zou/Right_Belly_Fortress/Casa_de_Inuarashi/poneglyph.zip
-oponeglyph_descomprimido
```

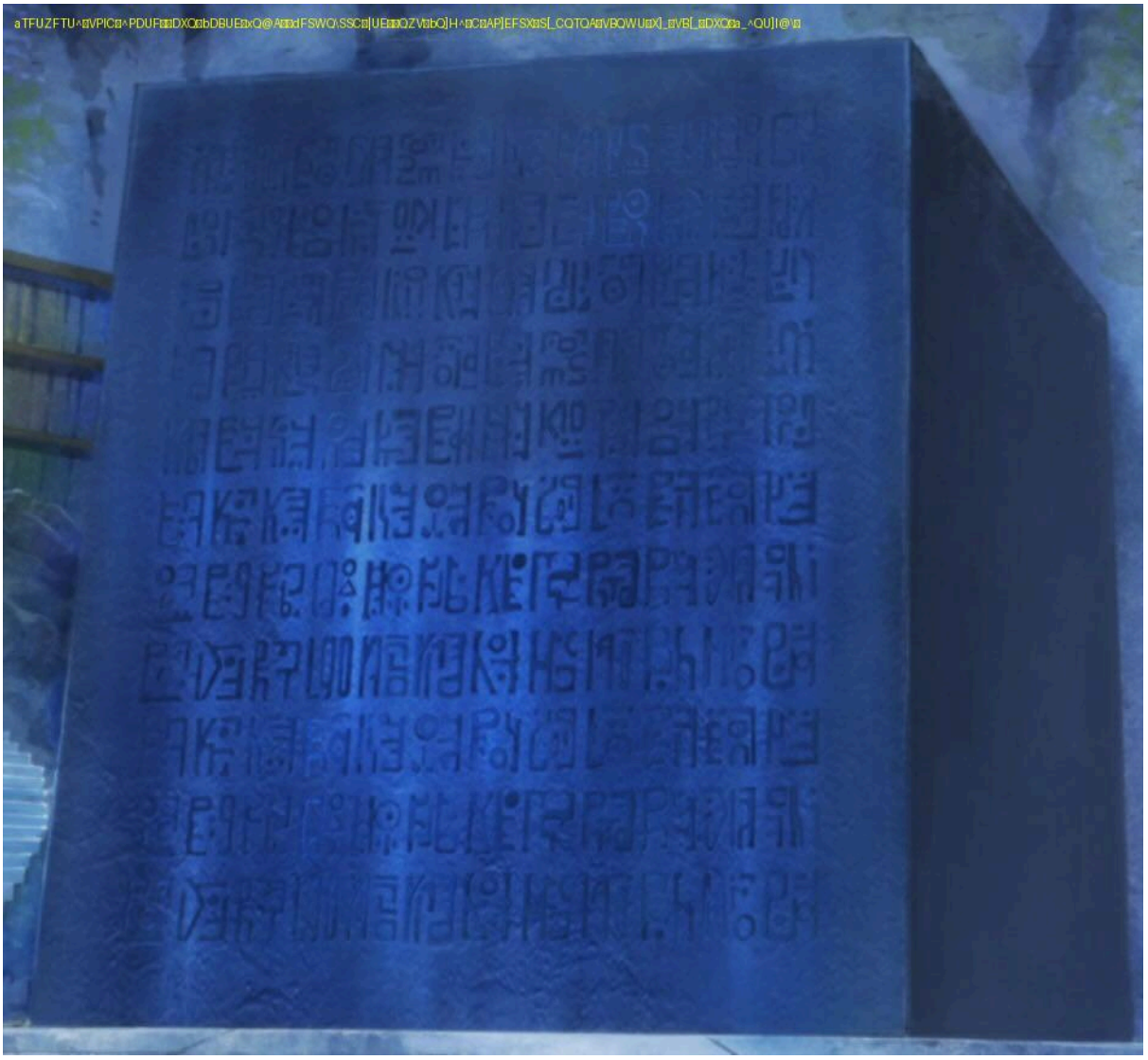
Ingresando la contraseña `FLAG_7bf472c0a9f8c3bda0057209c9ad1aed` resulta en el siguiente archivo:

```bash
usopp@2a73d406848a:~$ ls
ONEPIECE  poneglyph_descomprimido
usopp@2a73d406848a:~$ cd poneglyph_descomprimido/
usopp@2a73d406848a:~/poneglyph_descomprimido$ ls
poneglyph.jpeg
```



Ahora se mueve esta imagen al volumen compartido `/usopp_ctf` para trabajar con ella en la máquina local.

```bash
cp poneglyph.jpeg /usopp_ctf
```



Con la imagen en mi máquina local, se puede abrir sin problemas la imagen. Se puede notar nuevamente que en la esquina superior
izquierda hay un texto que parece ser una llave, empezando con los caracteres 'aTUFZ' y siguiendo con caracteres que parecen
no ser ASCII. Esto es un indicio de que este es lo que hay que descifrar usando XOR.

Al leer el contenido con python, se puede encontrar el segmento de bytes que representan este texto.

```text
aTFUZFTU^\x14VPIC\x14^PDUF\x1e\x11DXQ\x12bDBUE\x11xQ@A\x1d\x10dFSWQ\
\SSC\x10|UE\x1d\x10QZV\x11bQ]H^\x17C\x14AP]EFSX\x10S[_CQTQA\x11VBQWU\
x10X]_\x11VB[_\x11DXQ\x12a_^QU]I@\\\x1c
```

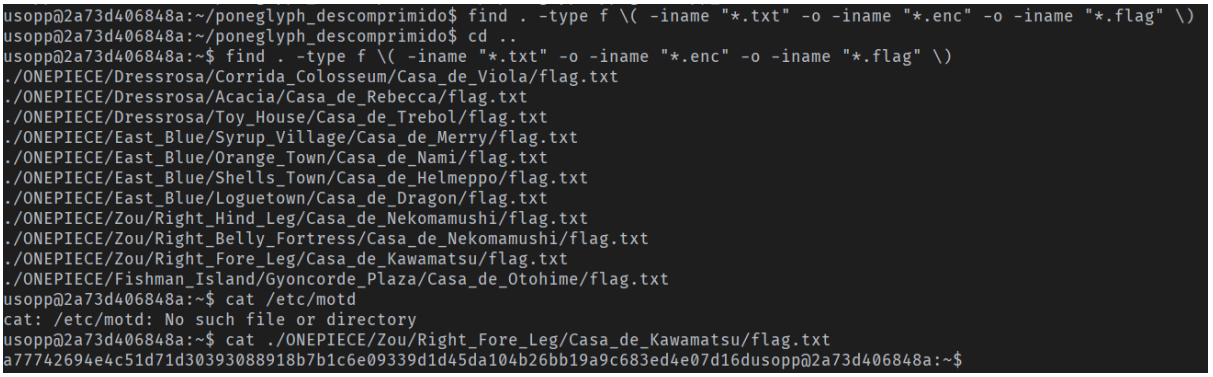
Usando esto como texto cifrado, y mi carnet `21004` como la llave, se obtiene la flag siguiente:

```text
Seventeen days later, the Straw Hats, Trafalgar Law, and Raizo's samurai comrades freed him from the Poneglyph.
```

Por último, usando el siguiente comando se pueden encontrar diversos archivos de texto

```bash
usopp@2a73d406848a:~/poneglyph_descomprimido$ find . -type f \( -iname "*.txt" -o -iname "*.enc" -o -iname "*.flag" \)
```

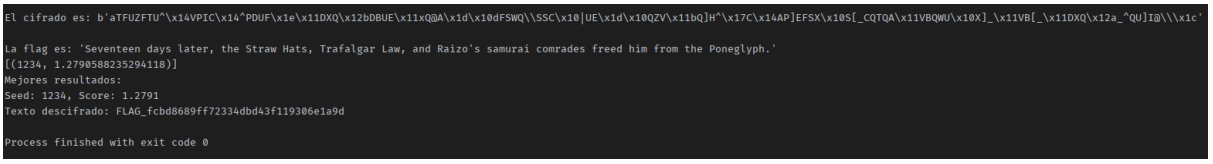
```text
./ONEPIECE/Dressrosa/Corrida_Colosseum/Casa_de_Viola/flag.txt
./ONEPIECE/Dressrosa/Acacia/Casa_de_Rebecca/flag.txt
./ONEPIECE/Dressrosa/Toy_House/Casa_de_Trebol/flag.txt
./ONEPIECE/East_Blue/Syrup_Village/Casa_de_Merry/flag.txt
./ONEPIECE/East_Blue/Orange_Town/Casa_de_Nami/flag.txt
./ONEPIECE/East_Blue/Shells_Town/Casa_de_Helmeppo/flag.txt
./ONEPIECE/East_Blue/Loguetown/Casa_de_Dragon/flag.txt
./ONEPIECE/Zou/Right_Hind_Leg/Casa_de_Nekomamushi/flag.txt
./ONEPIECE/Zou/Right_Belly_Fortress/Casa_de_Nekomamushi/flag.txt
./ONEPIECE/Zou/Right_Fore_Leg/Casa_de_Kawamatsu/flag.txt
./ONEPIECE/Fishman_Island/Gyoncorde_Plaza/Casa_de_Otohime/flag.txt
```



De estos, el archivo `./ONEPIECE/Zou/Right_Fore_Leg/Casa_de_Kawamatsu/flag.txt` contiene el único texto que no es legible.

Como lo dice el enunciado, ahora la flag esta codificada con un algoritmo de Stream Cipher. Para romper el cifrado,
se decidió utilizar un script de python que utiliza fuerza bruta para encontrar la llave. Con este script, se puede ver
que la mayoría de las llaves no funcionan, resultando en una única seed que funciona, la `1234` dando como resultado la siguiente flag:

```text
FLAG_fcbd8689ff72334dbd43f119306e1a9d
```



Nami

Para este el último reto, se siguió la misma metodología que en los dos anteriores.

```bash
docker exec -it {challengeX_ctf} bash
```

Usando la flag encontrada en el reto anterior, se inicia sesión como `usopp`:

```bash
su nami
password: FLAG_fcbd8689ff72334dbd43f119306e1a9d
```

Se vuelve a usar el comando `find` para buscar archivos que contengan la palabra `poneglyph` en su nombre:

```bash
find . -name "poneglyph*"
```

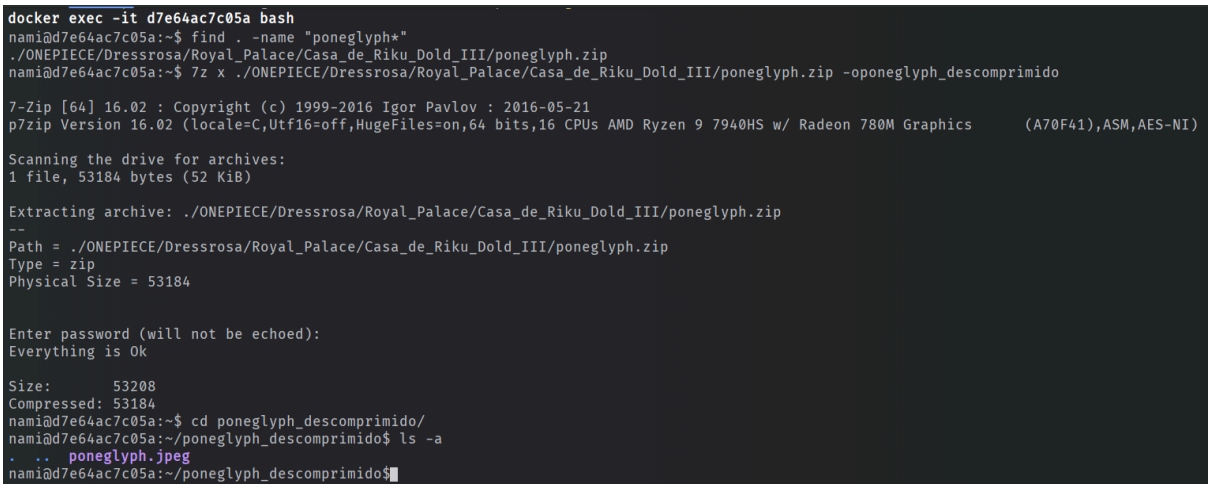
Esto da como resultado:

```bash
nami@d7e64ac7c05a:~$ find . -name "poneglyph*"
./ONEPIECE/Dressrosa/Royal_Palace/Casa_de_Riku_Dold_III/poneglyph.zip
```

Usando 7z para extraer el archivo:
```bash
 7z x ./ONEPIECE/Dressrosa/Royal_Palace/Casa_de_Riku_Dold_III/poneglyph.zip
-oponeglyph_descomprimido
```

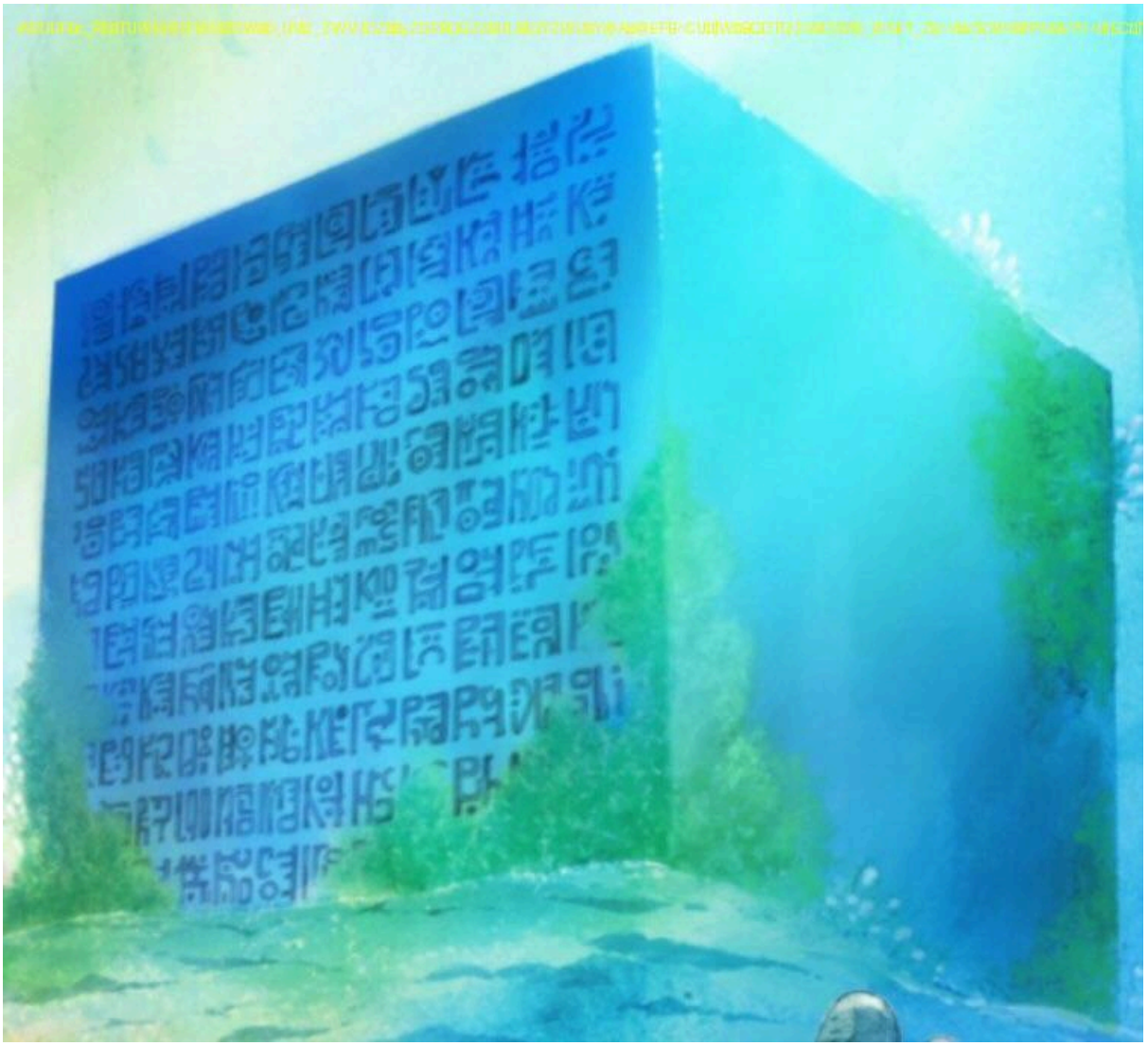
Ingresando la contraseña `FLAG_fcbd8689ff72334dbd43f119306e1a9d` resulta en el siguiente archivo:

```bash
nami@d7e64ac7c05a:~$ cd poneglyph_descomprimido/
nami@d7e64ac7c05a:~/poneglyph_descomprimido$ ls -a
.  ..  poneglyph.jpeg
```



Ahora se mueve esta imagen al volumen compartido `/nami_ctf` para trabajar con ella en la máquina local.

```bash
cp poneglyph.jpeg /nami_ctf
```



Con la imagen en mi máquina local, se puede abrir sin problemas la imagen. Se puede notar nuevamente que en la esquina superior
izquierda hay un texto que parece ser una llave, pero en este caso el texto es prácticamente ilegible a simple vista.
Entonces, imprimiendo los bytes de la imagen, en las primeras líneas se pueden ver caracteres fuera de lugar, con el mismo
patrón que en los ejercicios anteriores. Gracias a esto fue posible encontrar el texto que parece ser la llave.

Al leer el contenido con python, se puede encontrar el segmento de bytes que representan este texto.

```text
sWDUF\x12c_R]\\\x11TUW[AXUFWU\x10D\\W\x11b_UV\x11`_ZWV\\IDZ\x1d\x10yZG
PBQGZX\x10ULB]QYZWU\x10Y@A\x11@EFB^CU\x14]W\x10BQDTQ\\]\\V\x10D\\W\x
11\\_WSEY_Z\x12^V\x10xSDWX\x14fP\\U\x14EYU^\x14[EC\x10]\\W_BYSEY_Z\x12F
QC\x14Q^]R]\\TT\x10C[EX\x10@ZPD\x10[T\x11DXQ\x12^DXQ@\x11DXFWT\x10b[SU
\x10`[\\TW\\MBYC\x1e
```

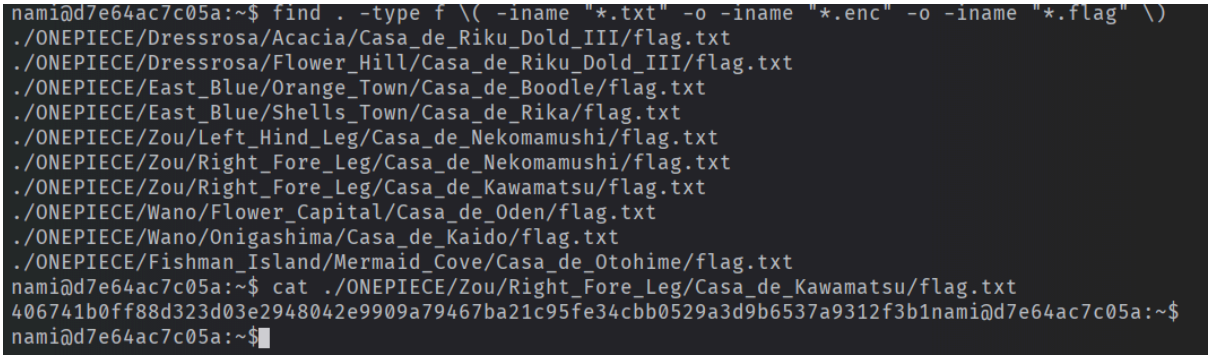
Usando esto como texto cifrado, y mi carnet `21004` como la llave, se obtiene la flag siguiente:

```text
After Robin deciphered the Road Poneglyph, Inuarashi explained its purpose of
revealing the location of Laugh Tale when its information was combined with that of the
other three Road Poneglyphs.
```

Por último, usando el siguiente comando se pueden encontrar diversos archivos de texto

```bash
nami@d7e64ac7c05a:~$ find . -type f \( -iname "*.txt" -o -iname "*.enc" -o -iname
"*.flag" \)
```

```text
./ONEPIECE/Dressrosa/Acacia/Casa_de_Riku_Dold_III/flag.txt
./ONEPIECE/Dressrosa/Flower_Hill/Casa_de_Riku_Dold_III/flag.txt
./ONEPIECE/East_Blue/Orange_Town/Casa_de_Boodle/flag.txt
./ONEPIECE/East_Blue/Shells_Town/Casa_de_Rika/flag.txt
./ONEPIECE/Zou/Left_Hind_Leg/Casa_de_Nekomamushi/flag.txt
./ONEPIECE/Zou/Right_Fore_Leg/Casa_de_Nekomamushi/flag.txt
./ONEPIECE/Zou/Right_Fore_Leg/Casa_de_Kawamatsu/flag.txt
./ONEPIECE/Wano/Flower_Capital/Casa_de_Oden/flag.txt
./ONEPIECE/Wano/Onigashima/Casa_de_Kaido/flag.txt
./ONEPIECE/Fishman_Island/Mermaid_Cove/Casa_de_Otohime/flag.txt
```



De estos, el archivo `./ONEPIECE/Zou/Right_Fore_Leg/Casa_de_Kawamatsu/flag.txt` contiene el único texto que no es legible.

Como lo dice el enunciado, ahora la flag esta codificada con un algoritmo de ChaCha20.

Este algoritmo depende de una llave y un nonce. Es un cifrado difícil de romper con fuerza bruta, así que se probaron primero algunos
diferentes valores, usando mi carnet como punto de partida.

Este algoritmo requiere que la llave sea de 32 bytes, el nounce es el que puede llegar a variar en cuanto a su longitud.

> Los más comunes son de 8 bytes y 12 bytes.

Usando el script desarrollado de Python para romper ChaCha20, se logra obtener la siguiente flag con la configuración de 32 bits / 8 bits
usando mi carnet como base para tanto la llave como el nonce:

```text
[+] Flag descifrada: FLAG_914bacd0923f836359edcf71b726b935
[!] El texto no es UTF-8 legible con la configuración nonce 12
Raw output:
b'#?\xe0\x02\xe3\x98\x17\xa3\x93G\x0e\x13i\xd5\x83\xd2hR\xc6\xa5\xbd\x1c_\x97\xba\x1a\xc1\x98\xa8\x07\xcdU3E9(\xc7'
[!] El texto no es UTF-8 legible con la configuración nonce 24
Raw output: b"x\x88\x1e\\<\xc2\xf4\xeb\t\xb7\xd3F\x84x\xf8Q?\x12l\xa43$k\x95\xc59v\xc4(\xa1~'\xc7\x9cd\xa1"
```

```
El cifrado es: b'sWDUF\x12c_R]\\\x11TUW[AXUFWU\x10D\\W\x11b_UV\x11`_ZWV\\IDZ\x1d\x10yZGPBQGZX\x10ULB]QYZWU\x10Y@A\x11@EFB^CU\x14]W\x10BQDTQ\\]\\V\x10D\\W\x11\\_WSEY_Z\x12^V\x10xSDWX\x14fP\\U\x14EYU^\x14[EC\x10
 ]\\W_BYSEY_Z\x12FQC\x14Q^]R]\\TT\x10C[EX\x10@ZPD\x10[T\x11DXQ\x12^DXQ@\x11DXFWT\x10b[SU\x10`[\\TW\\MBYC\x1e'

La flag es: 'After Robin deciphered the Road Poneglyph, Inuarashi explained its purpose of revealing the location of Laugh Tale when its information was combined with that of the other three Road Poneglyphs.'
[+] Flag descifrada: FLAG_914bacd0923f836359edcf71b726b935
[!] El texto no es UTF-8 legible con la configuración nonce 12
Raw output: b'#?\xe0\x02\xe3\x98\x17\xa3\x93G\x0e\x13i\xd5\x83\xd2hR\xc6\xa5\xbd\x1c_\x97\xba\x1a\xc1\x98\xa8\x07\xcdU3E9(\xc7'
[!] El texto no es UTF-8 legible con la configuración nonce 24
Raw output: b" x\x88\x1e\\<\xc2\xf4\xeb\t\xb7\xd3F\x84x\xf8Q?\x12l\xa43$k\x95\xc59v\xc4(\xa1~'\xc7\x9cd\xa1"

Process finished with exit code 0
```