

UNIVERSIDAD DEL VALLE DE GUATEMALA
Cifrado de Información
Sección 10
Ludwing Cano



Proyecto 1
Cifrados de Flujo y Desafíos de Seguridad

José Daniel Gómez Cabrera 21429

Repository

<https://github.com/JDgomez2002/project-1-cipher.git>

Instrucciones para ejecutar el código

1. Clonar el repositorio con:
 - a. git clone {repository}
2. Instalar las dependencias/requerimientos:
 - a. pip3 install -r resources/requirements.txt

Al ejecutar todos los comandos debería de observarse:

```
~/UVG/cipher/ctf_onepice_symmetric_cipher git:(main) (3.593s)
pip3 install -r resources/requirements.txt
Requirement already satisfied: Pillow in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from -r resources/requirements.txt (line 1)) (18.4.0)
Requirement already satisfied: numpy in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from -r resources/requirements.txt (line 2)) (1.26.4)
Collecting pixief (from -r resources/requirements.txt (line 3))
  Downloading pixief-1.3-py2.py3-none-any.whl.metadata (3.7 kB)
Collecting pyzippier (from -r resources/requirements.txt (line 4))
  Downloading pyzippier-0.3.6-py2.py3-none-any.whl (3.5 kB)
Requirement already satisfied: pycryptodomex in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from -r resources/requirements.txt (line 5)) (3.21.0)
Collecting pycryptodomex (from pyzippier->-r resources/requirements.txt (line 4))
  Downloading pycryptodomex-3.22.0-cp37abi3-macosx_10_9_universal2.whl.metadata (3.4 kB)
  Downloading pycryptodomex-3.22.0-cp37abi3-macosx_10_9_universal2.whl (2.5 MB)
  Downloading pyzippier-0.3.6-py2.py3-none-any.whl (67 kB)
  Downloading pycryptodomex-3.22.0-cp37abi3-macosx_10_9_universal2.whl (2.5 MB/s) 2.5/2.5 MB 2.8 MB/s eta 0:00:00
  Downloading pyzippier-0.3.6-py2.py3-none-any.whl (67 kB)
  Downloading pycryptodomex-3.22.0-cp37abi3-macosx_10_9_universal2.whl (2.5 MB)
Installing collected packages: pixief, pycryptodomex, pyzippier
Successfully installed pixief-1.3 pycryptodomex-3.22.0 pyzippier-0.3.6
```

3. Generar los retos con:
 - a. python3 generate_challenges.py

```
~/UVG/cipher/ctf_onepice_symmetric_cipher git:(main)±2 (13.039s)
python3 generate_challenges.py

Ingrese su carné: 21429
Laberinto generado con éxito.
Ruta1: challenges/luffy/ONEPIECE/East_Blue/Arlong_Park/Casa_de_Genzo
Ruta2: challenges/luffy/ONEPIECE/Alabasta/Alubarna/Casa_de_Igaram
Linux posix
  adding: challenges/luffy/poneglyph.jpeg (deflated 2%)
flag location: challenges/luffy/ONEPIECE/East_Blue/Arlong_Park/Casa_de_Genzo
Laberinto generado con éxito.
Ruta1: challenges/zoro/ONEPIECE/East_Blue/Loguetown/Casa_de_Dragon
Ruta2: challenges/zoro/ONEPIECE/East_Blue/Romance_Dawn/Casa_de_Luffy
Linux posix
  adding: challenges/zoro/poneglyph.jpeg (deflated 0%)
flag location: challenges/zoro/ONEPIECE/East_Blue/Loguetown/Casa_de_Dragon
Laberinto generado con éxito.
Ruta1: challenges/usopp/ONEPIECE/Water_7/GalleyLa_Headquarters/Casa_de_Paulie
Ruta2: challenges/usopp/ONEPIECE/Water_7/Blue_Station/Casa_de_Icebburg
Linux posix
  adding: challenges/usopp/poneglyph.jpeg (deflated 1%)
flag location: challenges/usopp/ONEPIECE/Water_7/GalleyLa_Headquarters/Casa_de_Paulie
Laberinto generado con éxito.
Ruta1: challenges/nami/ONEPIECE/Dressrosa/Flower_Hill/Casa_de_Rebecca
Ruta2: challenges/nami/ONEPIECE/Alabasta/Katorea/Casa_de_Toto
Linux posix
  adding: challenges/nami/poneglyph.jpeg (deflated 1%)
flag location: challenges/nami/ONEPIECE/Dressrosa/Flower_Hill/Casa_de_Rebecca
Retos generados con éxito!
```

4. Ejecutar el contenedor de Docker

a. docker-compose up -d

```

dancer@ip-172-31-1-163:~/Desktop$ docker-compose up -d
[warn] /usr/bin/docker-compose: /usr/bin/docker-compose: symmetric_cipher/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion
Compose can now delegate builds to bake for better performance.
To do so, set COMPOSE_BAKE=true.
[*] Starting 108 services
  => usopp_image: Built
  => usopp_image[internal]: Load build definition from Dockerfile
  => transferring dockerfile: 1.1KB
  => [usopp_image internal]: Load build definition from Dockerfile
  => usopp_image: Built
  => usopp_image[internal]: Load build definition from Dockerfile
  => transferring dockerfile: 1.1KB
  => [usopp_image internal]: Load build definition from Dockerfile
  => usopp_image: Built
  => usopp_image[internal]: Load build definition from Dockerfile
  => transferring dockerfile: 1.08KB
  => [zoro_image internal]: Load build definition from Dockerfile
  => zoro_image: Built
  => zoro_image[internal]: Load build definition from Dockerfile
  => transferring dockerfile: 1.1KB
  => [zoro_image internal]: Load metadata for docker.io/library/ubuntu:22.04
  => [zoro_image internal]: Load dockerignore
  => zoro_image: Built
  => luffy_image: Built
  => luffy_image[internal]: Load dockerignore
  => transferring context: 28
  => [zoro_image internal]: Load dockerignore
  => zoro_image: Built
  => usopp_image: Built
  => [usopp_image internal]: Load dockerignore
  => transferring context: 28
  => [zoro_image internal]: Load dockerignore
  => zoro_image: Built
  => usopp_image: Built
  => usopp_image[internal]: Load context
  => usopp_image: Built
  => usopp_image[internal]: Load context
  => usopp_image: Built
  => [luffy_image internal]: Load build context
  => transferring context: 84.30KB
  => [zoro_image internal]: Load build context
  => transferring context: 69.45KB
  => [usopp_image internal]: Load build context
  => transferring context: 80.82KB
  => usopp_image: Built
  => usopp_image[internal]: Run command: && apt-get install -y sudo python3 python3-pip python3-dev git vim nano curl tesseract-ocr && rm -rf /var/lib/apt/lists/*
  => [zoro_image 3/7]: RUN useradd -ms /bin/bash zoro && echo "zoro:FL4G_51c30a7f4bd01b8e7fd7ed767d508e" | chpasswd && adduser zoro sudo
  => [usopp_image 3/7]: RUN useradd -ms /bin/bash usopp && echo "usopp:FL4G_eac738cd76ea6313a2b2c500bfdfaa" | chpasswd && adduser usopp sudo
  => [usopp_image 3/7]: RUN usermod -aG sudo zoro && usermod -aG sudo usopp && usermod -aG sudo luffy
  => [zoro_image 3/7]: RUN usermod -aG sudo zoro && usermod -aG sudo luffy && usermod -aG sudo usopp
  => [usopp_image 4/7]: RUN passed -l root
  => [zoro_image 4/7]: RUN passed -l root
  => [usopp_image 4/7]: RUN passed -l root
  => [luffy_image 4/7]: RUN passed -l root
  => [luffy_image 5/7]: WORKDIR /home/luffy
  => [zoro_image 5/7]: COPY ./ONEPIECE /home/luffy/ONEPIECE
  => [usopp_image 5/7]: COPY ./ONEPIECE /home/zoro/ONEPIECE
  => [zoro_image 6/7]: COPY ./ONEPIECE /home/zoro/ONEPIECE
  => [luffy_image 6/7]: RUN chmod +R luffy:luffy /home/luffy
  => [zoro_image 6/7]: RUN chmod +R usopp:usopp /home/usopp
  => [usopp_image 6/7]: RUN chmod +R nami:nami /home/nami
  => [luffy_image 6/7]: RUN chmod +R nami:nami /home/nami
  => writing image sha256:b8151b2762326739f9ba1e1ccdb89ff7ecdd4594d60ff1f8c6250f323591624
  => naming to docker.io/library/ctf.onepiece_symmetric_cipher-luffy_image
  => writing image sha256:85815b5708d825d680bf5fadd6ea42421c2492fb1709553a272559c2df19b
  => naming to docker.io/library/ctf.onepiece_symmetric_cipher-usopp_image
  => exporting layers
  => writing image sha256:436b57397680d787eb2fdccc457b87b1cd62082beb9513d2fd49d9b48c546344
  => naming to docker.io/library/ctf.onepiece_symmetric_cipher-zoro_image
  => naming to docker.io/library/ctf.onepiece_symmetric_cipher-nami_image
  => exporting layers
  => writing image sha256:9a216d368c5ee8a0b78339f5d56d4e0bc7d7ae3f3258f779f905d8dcf1ff51a61e35
  => naming to docker.io/library/ctf.onepiece_symmetric_cipher-nami_image
  => naming to docker.io/library/ctf.onepiece_symmetric_cipher-zoro_image
  => naming to docker.io/library/ctf.onepiece_symmetric_cipher-luffy_image
  => [luffy_image] resolving provenance for metadata file
  => [usopp_image] resolving provenance for metadata file
  => [zoro_image] resolving provenance for metadata file
  => [zoro_image] resolving provenance for metadata file
[*] Running 9/9
  => luffy_image: Built
  => usopp_image: Built
  => zoro_image: Built
  => usopp_image[internal]: CACHED
  => Container_usopp_challenge: Started
  => Container_luffy_challenge: Started
  => Container_zoro_challenge: Started
  => Container_nami_challenge: Started

```

Reto 1 - Luffy

Este reto nos ayuda a familiarizarnos con criptografía básica y operaciones con XOR, también a reconocer las vulnerabilidades y amenazas de dichos tipos de criptografía. También nos ayuda a aprender más acerca de la extracción de texto de imágenes y a comprender de manera práctica la desencriptación por medio de una llave específica como lo es en este caso, el número de carné.

Objetivos

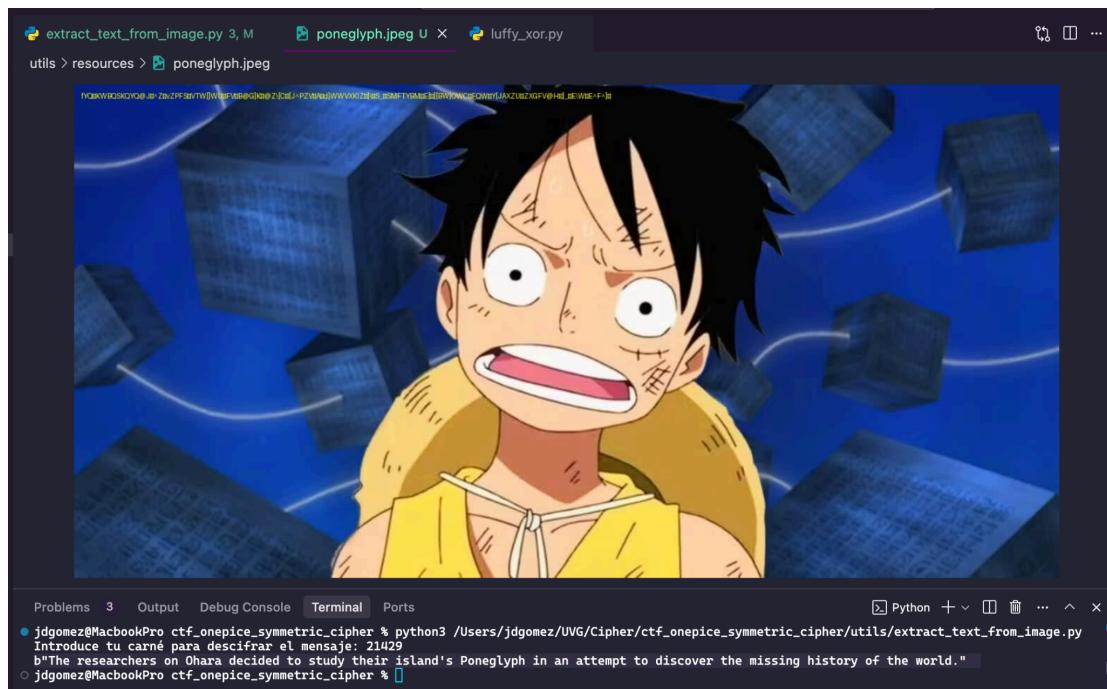
- i. Analizar un mensaje cifrado con XOR.
- ii. Encontrar patrones para descifrar el mensaje original.
- iii. Identificar vulnerabilidades inherentes al uso repetido o débil de claves en cifrados XOR.
- iv. Fortalecer la intuición criptográfica y el pensamiento crítico frente a problemas de seguridad en la información.

Flag encontrada

FLAG_51c530a7fdb01b01e7fd7ed5767d508e

Párrafo del poneglyphs

Imagen encontrada



Texto encontrado

- b"The researchers on Ohara decided to study their island's Poneglyph in an attempt to discover the missing history of the world."

Documentación

Paso 1

Ingresé al challenge 1 y accedí al usuario “luffy” con la contraseña “onepiece”.

```
~/UVG/cipher/ctf_onepice_symmetric_cipher/challenges/luffy git:(main)±3
docker exec -it luffy_challenge bash
nobody@ef270d47b06e:/home/luffy$ su luffy
Password:
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

luffy@ef270d47b06e:~$
```

Paso 2

Encontré la flag por medio de la ayuda de un comando de docker que encuentra archivos con terminaciones específicas.

- find / -type f \(-name "*.txt" -o -name "*.flag" -o -name "*.hidden" -o -name "*.enc"\)
\\) 2>/dev/null

En la siguiente imagen muestro el resultado de la búsqueda...

```
~/UVG/cipher/ctf_onepice_symmetric_cipher/challenges/luffy git:(main)±3
docker exec -it luffy_challenge bash
/usr/share/vim/vim82/doc/windows.txt
/usr/share/vim/vim82/doc/recover.txt
/usr/share/vim/vim82/doc/os_msdos.txt
/usr/share/vim/vim82/doc/version7.txt
/usr/share/vim/vim82/doc/index.txt
/usr/share/vim/vim82/doc/usr_07.txt
/usr/share/vim/vim82/doc/diff.txt
/usr/share/vim/vim82/doc/usr_04.txt
/usr/share/vim/vim82/doc/hebrew.txt
/usr/share/vim/vim82/doc/netbeans.txt
/usr/share/vim/vim82/doc/usr_32.txt
/usr/share/vim/vim82/doc/pi_spec.txt
/usr/share/vim/vim82/doc/options.txt
/usr/share/vim/vim82/doc/usr_45.txt
/usr/share/vim/vim82/pack/dist/opt/matchit/doc/matchit.txt
/usr/share/gnupg/help.nb.txt
/usr/share/gnupg/help.es.txt
/usr/share/gnupg/help.pt_BR.txt
/usr/share/gnupg/help.ro.txt
/usr/share/gnupg/help.sk.txt
/usr/share/gnupg/help.gl.txt
/usr/share/gnupg/help.pl.txt
/usr/share/gnupg/help.eo.txt
/usr/share/gnupg/help.ja.txt
/usr/share/gnupg/help.de.txt
/usr/share/gnupg/help.it.txt
/usr/share/gnupg/help.fi.txt
/usr/share/gnupg/help.zh_TW.txt
/usr/share/gnupg/help.el.txt
/usr/share/gnupg/help.id.txt
/usr/share/gnupg/help.ca.txt
/usr/share/gnupg/help.cs.txt
/usr/share/gnupg/help.hu.txt
/usr/share/gnupg/help.pt.txt
/usr/share/gnupg/help.sv.txt
/usr/share/gnupg/help.et.txt
/usr/share/gnupg/help.be.txt
/usr/share/gnupg/help.txt
/usr/share/gnupg/help.fr.txt
/usr/share/gnupg/help.zh_CN.txt
/usr/share/gnupg/help.da.txt
/usr/share/gnupg/help.tr.txt
/usr/share/gnupg/help.ru.txt
/usr/share/perl/5.34.0/Unicode/Collate/allkeys.txt
/usr/share/perl/5.34.0/Unicode/Collate/keys.txt
/usr/share/perl/5.34.0/unicode/SpecialCasing.txt
/usr/share/perl/5.34.0/unicode/NamedSequences.txt
/usr/share/perl/5.34.0/unicore/Blocks.txt
/home/luffy/ONEPIECE/East_Blue/Shells_Town/Casa_de_Morgan/flag.txt
/home/luffy/ONEPIECE/East_Blue/Shells_Town/Casa_de_Rika/flag.txt
/home/luffy/ONEPIECE/East_Blue/Baratie/Casa_de_Zeff/flag.txt
/home/luffy/ONEPIECE/East_Blue/Arlong_Park/Casa_de_Genzo/flag.txt
/home/luffy/ONEPIECE/Sabaody_Archipelago/Grove_1/Casa_de_Rayleigh/flag.txt
/home/luffy/ONEPIECE/Sabaody_Archipelago/Grove_1/Casa_de_Keimi/flag.txt
/home/luffy/ONEPIECE/Water_7/Carpenters_Cafe/Casa_de_Lucci/flag.txt
/home/luffy/ONEPIECE/Dressrosa/Corrida_Colosseum/Casa_de_Viola/flag.txt
/home/luffy/ONEPIECE/Dressrosa/Flower_Hill/Casa_de_Riku_Dold_III/flag.txt
/home/luffy/ONEPIECE/Dressrosa/Flower_Hill/Casa_de_Rebecca/flag.txt
/home/luffy/ONEPIECE/Dressrosa/Flower_Hill/Casa_de_Viola/flag.txt
luffy@ef270d47b06e:~$
```

Paso 3

Debido a que habían demasiados archivos “flag.txt”, era imposible determinar cuál de todos era nuestra flag objetivo. Sin embargo, con la ayuda de un comando, se puede leer el contenido de todos los archivos “flag.txt”.

- ```
- find /home/luffy/ONEPIECE/ -type f -name "flag.txt" -exec cat {} + 2>/dev/null
```

```
/home/luffy/ONEPIECE/Dressrosa/Corrida_Colosseum/Casa_de_Viola/flag.txt
/home/luffy/ONEPIECE/Dressrosa/Flower_Hill/Casa_de_Riku_Dold_III/flag.txt
/home/luffy/ONEPIECE/Dressrosa/Flower_Hill/Casa_de_Rebecca/flag.txt
/home/luffy/ONEPIECE/Dressrosa/Flower_Hill/Casa_de_Viola/flag.txt
luffy@ef270d47b06e:~$ find /home/luffy/ONEPIECE/ -type f -name "flag.txt" -exec cat {} + 2>/dev/null
Has encontrado un mapa que apunta a Arlong_ParkHas encontrado un enemigo y ha tenido que lucharHas encontrado un lu
gar peligroso747d757566070057070a025003545b5601055009035403545d055450070e04065007090a54Has encontrado un obstáculo
luffy@ef270d47b06e:~$
```

Dentro de este resultado, resalta mucho la secuencia de números:

- 747d757566070057070a025003545b5601055009035403545d055450070e040650070  
90a54

Utilizando un script de python que implemente XOR se puede desencriptar la bandera.

```
XOR.py U ×
utils > XOR.py > ...
1 def xor_decrypt(hex_string, key):
2 encrypted_bytes = bytes.fromhex(hex_string)
3 key_bytes = str(key).encode()
4 key_length = len(key_bytes)
5 decrypted_bytes = bytes([key_bytes[i % key_length] ^ b for i, b in enumerate(encrypted_bytes)])
6
7 return decrypted_bytes.decode(errors='ignore')
8
9 flag = '747d757566070057070a025003545b5601055009035403545d055450070e04065007090a54'
10
11 carnet = 21429
12 decrypted_text = xor_decrypt(flag, carnet)
13 print("Texto desencriptado:", decrypted_text)
14
15

Problems Output Debug Console Terminal Ports
jdgoméz@MacBookPro ctf_onepice_symmetric_cipher % /opt/homebrew/bin/python3 /Users/jdgomez/UVG/Cipher/ctf_onepice_symmetric_cipher/utils/XOR.py
Texto desencriptado: FLAG_51c530a7bd01b01e7fd7ed5767d508e
jdgoméz@MacBookPro ctf_onepice_symmetric_cipher %
```

La bandera que desencripte fue:

- FLAG 51c530a7fdbd01b01e7fd7ed5767d508e

## Paso 5

Para encontrar la imagen utilicé el mismo comando de encontrar archivos.

- ```
- find / -type f \(-name "*.jpg" -o -name "*.png" -o -name "*.jpeg" -o -name "*.gif" -o -name "*.bmp" -o -name "*.zip"\) 2>/dev/null
```

```
find: expected an expression after '-o'
luffy@ef270d47b06e:~$ find / -type f \! -name "*.jpg" -o -name "*.png" -o -name "*.jpeg" -o -name "*.gif" -o -name "*.bmp" -o -name "*.zip" \) 2>/dev/null
/usr/share/info/gnupg-module-overview.png
/usr/share/info/gnupg-card-architecture.png
/usr/share/pixmaps/debian-logo.png
/usr/share/icons/hicolor/48x48/apps/gvim.png
/usr/share/icons/hicolor/16x16/apps/gvim.png
/usr/share/icons/hicolor/32x32/apps/gvim.png
/usr/share/gitweb/static/git-logo.png
/home/luffy/ONEPIECE/Alabasta/Alubarna/Casa_de_Igaram/poneglyph.zip
luffy@ef270d47b06e:~$
```

Se puede observar que existe un archivo zip que coincide con el requerido.

Paso 6

Descomprimi el archivo

```
~/UVG/cipher/ctf_onepice_symmetric_cipher/challenges/luffy git:(main)±4
docker exec -it luffy_challenge bash
luffy@ef270d47b06e:~/ONEPIECE/Alabasta/Alubarna/Casa_de_Igaram$ 7z x poneglyph.zip

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,12 CPUs LE)

Scanning the drive for archives:
1 file, 70329 bytes (69 KiB)

Extracting archive: poneglyph.zip
--
Path = poneglyph.zip
Type = zip
Physical Size = 70329

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

Size:      71664
Compressed: 70329
luffy@ef270d47b06e:~/ONEPIECE/Alabasta/Alubarna/Casa_de_Igaram$
```

Paso 7

Posteriormente, tuve que extraer el texto de la imagen. Guardé la imagen en mi laptop desde docker, para extraer el texto con el programa que nos proporcionó Ludwing

```
extract_text_from_image.py 3 M x poneglyph.jpeg U luffy_xor.py
utils > extract_text_from_image.py > ...
1
2 from PIL import Image
3 import piexif
4 from luffy_xor import xor_cipher
5
6 def extraer_texto_metadata(imagen_path):
7     # Abrir la imagen
8     img = Image.open(imagen_path)
9
10    # Obtener los metadatos EXIF
11    exif_dict = piexif.load(img.info.get('exif', b''))
12
13    # Obtener el texto almacenado en 'Artist' (o en el campo que elegimos)
14    texto = exif_dict['0th'].get(piexif.ImageIFD.Artist)
15    if texto:
16        return texto.decode('utf-8')
17    return None
18
19
20 # Uso del código para descifrar la imagen
21 # Ejemplo de uso
22 image_path = "./utils/resources/poneglyph.jpeg"
23 student_id = input("Introduce tu carné para descifrar el mensaje: ")
24 texto_cifrado = extraer_texto_metadata(image_path)
25 decrypted_text = xor_cipher(texto_cifrado, student_id)
26 print(decrypted_text)
27
28
```

Problems 3 Output Debug Console Terminal Ports

jdgomez@MacBookPro ctf_onepiece_symmetric_cipher % python3 ./Users/jdgomez/UVG/Cipher/ctf_onepiece_symmetric_cipher/utils/extract_text_from_image.py
Introduce tu carné para descifrar el mensaje: 210429
b'The researchers on Ohara decided to study their island's Ponglyph in an attempt to discover the missing history of the world.'
jdgomez@MacBookPro ctf_onepiece_symmetric_cipher %

b"The researchers on Ohara decided to study their island's Poneglyph in an attempt to discover the missing history of the world."

Con la flag encontrada y también el texto de la imagen en base a mi número de carné, se completó el Reto 1.

- FLAG_51c530a7fdb01b01e7fd7ed5767d508e
 - b"The researchers on Ohara decided to study their island's Poneglyph in an attempt to discover the missing history of the world."

Reto 2 - Zoro

El principal objetivo de este reto es profundizar

Objetivos

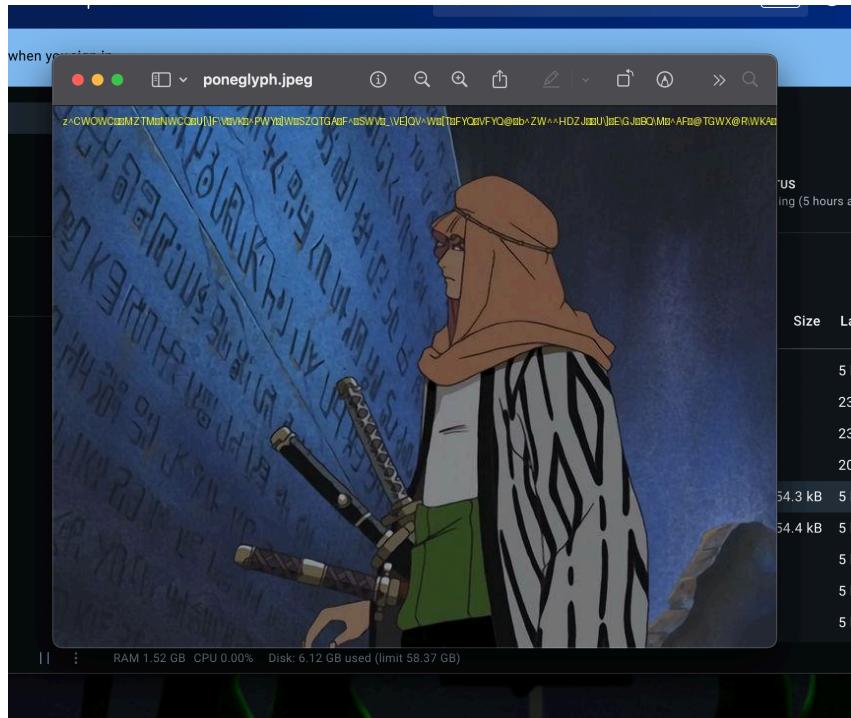
- i. Comprender el funcionamiento del cifrado RC4 y cómo se utiliza una clave para generar un flujo pseudoaleatorio.
 - ii. Analizar un flujo de datos cifrado con el algoritmo RC4.
 - iii. Analizar las debilidades de RC4, como los sesgos en la generación de claves y la vulnerabilidad a ataques por correlación.

Flag encontrada

- FLAG eac8738dc786a6313a2b26c500bfdcaa

Párrafo del poneglyphs

Imagen encontrada



Párrafo extraído

- b'However, they were limited by lack of access to and knowledge of the other Poneglyphs, and thus sent out researchers to find more information on them, knowing the study of the Poneglyphs was illegal.'

Documentación

Paso 1

Ingresamos al siguiente challenge, recordando que la contraseña de este challenge es la flag encontrada del challenge anterior (FLAG_51c530a7fdb01b01e7fd7ed5767d508e).

Ejecutamos el segundo challenge con:

- docker exec -it zoro_challenge bash

Ingresamos al usuario zoro por medio de ingresar la contraseña, la flag del challenge anterior:

- su zoro

```
~/UVG/cipher/ctf_onepiece_symmetric_cipher/challenges/zoro git:(main)±8
docker exec -it zoro_challenge bash
nobody@55af71d2a8d0:/home/zoro$ su zoro
Password:
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

zoro@55af71d2a8d0:~$
```

Paso 2

De la misma manera que el challenge anterior, utilice el comando find para poder encontrar todo tipo de archivos relacionados a las flags.

- find / -type f \(-name "*.txt" -o -name "*.flag" -o -name "*.hidden" -o -name "*.enc"\)
|) 2>/dev/null

```
~/UVU/Cipher/ctf_onepiece_symmetric_cipher/challenges/zoro git:(main)*8
docker exec zoro_challenger bash
/usr/share/doc/gnupg/doc/difile
/usr/share/vim/vim82/doc/windows.txt
/usr/share/vim/vim82/doc/recover.txt
/usr/share/vim/vim82/doc/os_msdos.txt
/usr/share/vim/vim82/doc/os_dos.txt
/usr/share/vim/vim82/doc/index.txt
/usr/share/vim/vim82/doc/usr_07.txt
/usr/share/vim/vim82/doc/usr_08.txt
/usr/share/vim/vim82/doc/usr_94.txt
/usr/share/vim/vim82/doc/hebrew.txt
/usr/share/vim/vim82/doc/utf8.txt
/usr/share/vim/vim82/doc/usr_32.txt
/usr/share/vim/vim82/doc/p1_spec.txt
/usr/share/vim/vim82/doc/utf8.txt
/usr/share/vim/vim82/doc/usr_45.txt
/usr/share/vim/vim82/pack/dist/opt/matchit/doc/matchit.txt
/usr/share/gnugp/help_ar.txt
/usr/share/gnugp/help_es.txt
/usr/share/gnugp/help_pt_BR.txt
/usr/share/gnugp/help_sk.txt
/usr/share/gnugp/help_gl.txt
/usr/share/gnugp/help_de.txt
/usr/share/gnugp/help_en.txt
/usr/share/gnugp/help_ja.txt
/usr/share/gnugp/help_it.txt
/usr/share/gnugp/help_fi.txt
/usr/share/gnugp/help_sl.txt
/usr/share/gnugp/help_id.txt
/usr/share/gnugp/help_tr.txt
/usr/share/gnugp/help_cz.txt
/usr/share/gnugp/help_hu.txt
/usr/share/gnugp/help_pt.txt
/usr/share/gnugp/help_es_ES.txt
/usr/share/gnugp/help_et.txt
/usr/share/gnugp/help_de_DE.txt
/usr/share/gnugp/help_fr.txt
/usr/share/gnugp/help_zh_CN.txt
/usr/share/gnugp/help_da.txt
/usr/share/gnugp/help_tr.txt
/usr/share/gnugp/help_ru.txt
/usr/share/perl/5.34.0/Unicode/Collate/allkeys.txt
/usr/share/perl/5.34.0/Unicode/Collate/keys.txt
/usr/share/perl/5.34.0/Unicode/SpecialCasing.txt
/usr/share/perl/5.34.0/Unicode/NamedSequences.txt
/usr/share/perl/5.34.0/Unicode/Blocks.txt
/home/zoro/ONEPICECE/Alabasta/Alubarna/Casa_de_Vivi/Flag.txt
/home/zoro/ONEPICECE/East_Blue/Loguetown/Casa_de_Dragon/Flag.txt
/home/zoro/ONEPICECE/Sabadoy_Archipelago/Grove_80/Casa_de_Rayleigh/Flag.txt
/home/zoro/ONEPICECE/Sabadoy_Archipelago/Grove_10/Casa_de_Koala/Flag.txt
/home/zoro/ONEPICECE/Sabadoy_Archipelago/Grove_41/Casa_de_Rayleigh/Flag.txt
/home/zoro/ONEPICECE/Mater_7/Gallely_Headquarters/Casa_de_Lucci/Flag.txt
/home/zoro/ONEPICECE/Water_7/Franky_House/Casa_de_Luffy/Flag.txt
/home/zoro/ONEPICECE/Water_7/Franky_House/Casa_de_Luffy/Flag.txt
/home/zoro/ONEPICECE/Dressrosa/Corrida_Colosseum/Casa_de_Riku_Dold_III/Flag.txt
zoro@zoro:~$
```

Paso 3

Mostré el contenido de los archivos “flag.txt” con el comando:

- find /home/zoro/ONEPICECE/ -type f -name "flag.txt" -exec cat {} + 2>/dev/null

```
/usr/share/gnugp/help_zh_CN.txt
/usr/share/gnugp/help_da.txt
/usr/share/gnugp/help_tr.txt
/usr/share/gnugp/help_ru.txt
/usr/share/perl/5.34.0/Unicode/Collate/allkeys.txt
/usr/share/perl/5.34.0/Unicode/Collate/keys.txt
/usr/share/perl/5.34.0/Unicode/SpecialCasing.txt
/usr/share/perl/5.34.0/Unicode/NamedSequences.txt
/usr/share/perl/5.34.0/Unicode/Blocks.txt
/home/zoro/ONEPICECE/Alabasta/Alubarna/Casa_de_Vivi/Flag.txt
/home/zoro/ONEPICECE/East_Blue/Baratie/Casa_de_Sanji/Flag.txt
/home/zoro/ONEPICECE/East_Blue/Loguetown/Casa_de_Dragon/Flag.txt
/home/zoro/ONEPICECE/Sabadoy_Archipelago/Grove_80/Casa_de_Rayleigh/Flag.txt
/home/zoro/ONEPICECE/Sabadoy_Archipelago/Grove_10/Casa_de_Koala/Flag.txt
/home/zoro/ONEPICECE/Sabadoy_Archipelago/Grove_41/Casa_de_Rayleigh/Flag.txt
/home/zoro/ONEPICECE/Mater_7/Gallely_Headquarters/Casa_de_Lucci/Flag.txt
/home/zoro/ONEPICECE/Water_7/Gallely_Headquarters/Casa_de_Paulie/Flag.txt
/home/zoro/ONEPICECE/Water_7/Franky_House/Casa_de_Luffy/Flag.txt
/home/zoro/ONEPICECE/Dressrosa/Corrida_Colosseum/Casa_de_Riku_Dold_III/Flag.txt
zoro@zoro:~$ find /home/zoro/ONEPICECE/ -type f -name "flag.txt" -exec cat {} + 2>/dev/null
Has encontrado un lugar peligrosoHas encontrado un amigo y han decidido viajar juntoseed80b25e409d12ba354b97fb46619226a682d49651f3e21281cabe820b5184629c575c2
aHas encontrado un enemigo y ha tenido que lucharHas encontrado una pista que apunta a LoguetownHas encontrado un lugar peligrosoHas encontrado un lugar aba
zoro@zoro:~$
```

- eed80b25e409d12ba354b97fb46619226a682d49651f3e21281cabe820b5184629c5752
af

Paso 4

De la misma manera que el challenge anterior, desencripte el archivo con mi número de carné y con la ayuda de un script de python que utiliza el algoritmo RC4 para desencriptar dicha flag.

The screenshot shows a terminal window with the following content:

```
extract_text_from_image.py 3, M      poneglyph.jpeg U      RC4.py 1, U      luffy_xor.py
utils > RC4.py > ...
1  from Crypto.Cipher import ARC4
2  import binascii
3
4  # Datos
5  flag = "eed80b25e409d12ba354b97fb46619226a682d49651f3e21281cab...e20b5184629c575c2af"
6  carne = "21429" # Numero de carne
7
8  # Convertir la clave a bytes
9  clave_bytes = carne.encode()
10
11 # Convertir el hexadecimal a bytes
12 cifrado_bytes = binascii.unhexlify(flag)
13
14 # Crear el desifrador RC4
15 rc4 = ARC4.new(clave_bytes)
16
17 # Descifrar
18 mensaje_descifrado = rc4.decrypt(cifrado_bytes)
19
20 print("Raw decrypted bytes:", mensaje_descifrado)
21 print("Hex representation:", mensaje_descifrado.hex())
22 # Try different encodings
23
24 print("decoded:", mensaje_descifrado.decode('utf-8'))
```

Terminal tab bar: extract_text_from_image.py 3, M, poneglyph.jpeg U, RC4.py 1, U, luffy_xor.py

Terminal output:

```
jdgomez@MacBookPro ctf_onepiece_symmetric_cipher % python3 /Users/jdgomez/UVG/Cipher/ctf_onepiece_symmetric_cipher/utils/RC4.py
Raw decrypted bytes: b'FLAG_eac8738dc786a6313a2b26c500bfdfaa'
Hex representation: 464c41475f6561633837333864633738366136333133613262323663353030626664666161
decoded: FLAG_eac8738dc786a6313a2b26c500bfdfaa
jdgomez@MacBookPro ctf_onepiece_symmetric_cipher %
```

Flag:

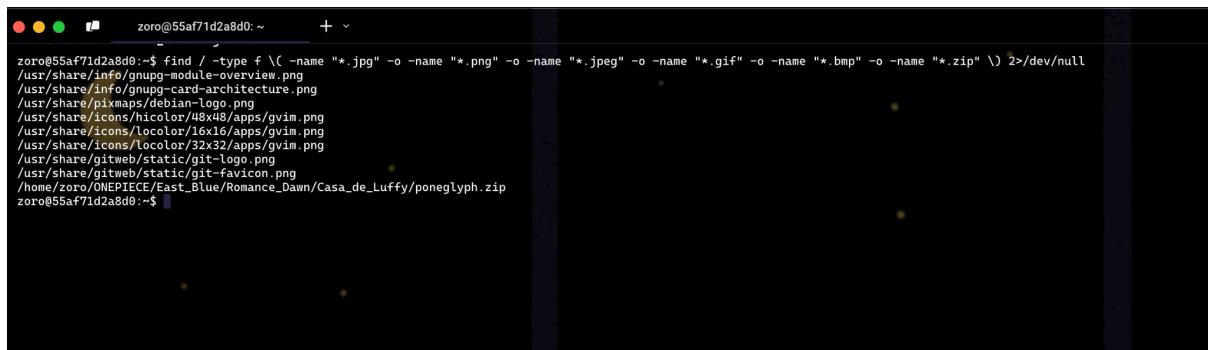
- FLAG_eac8738dc786a6313a2b26c500bfdfa

El script descifra un mensaje cifrado con RC4 por medio de la biblioteca Crypto. Primero, convierte la clave (carné) y el mensaje cifrado (hexadecimal) a bytes. Luego, crea un objeto RC4 con la clave y lo usa para descifrar el contenido. Finalmente, se interpreta el resultado en UTF-8 para poder mostrarlo como texto.

Paso 5

Busqué la imagen con el mismo comando que el challenge anterior.

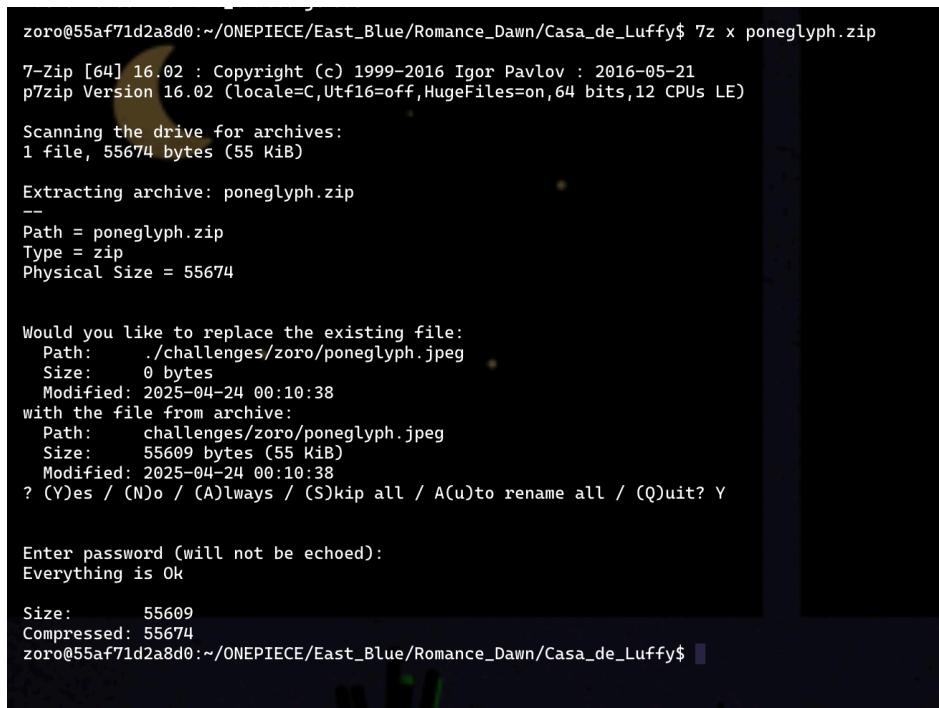
- find / -type f \(| -name "*.jpg" -o -name "*.png" -o -name "*.jpeg" -o -name "*.gif" -o -name "*.bmp" -o -name "*.zip" \) 2>/dev/null



```
zoro@55af71d2a8d0:~ + 
zoro@55af71d2a8d0:~$ find / -type f \(| -name "*.jpg" -o -name "*.png" -o -name "*.jpeg" -o -name "*.gif" -o -name "*.bmp" -o -name "*.zip" \) 2>/dev/null
/usr/share/info/gnupg-module-overview.png
/usr/share/info/gnupg-card-architecture.png
/usr/share/pixmaps/debian-logo.png
/usr/share/icons/hicolor/48x48/apps/gvim.png
/usr/share/icons/locolor/16x16/apps/gvim.png
/usr/share/icons/locolor/32x32/apps/gvim.png
/usr/share/gitweb/static/git-logo.png
/usr/share/gitweb/static/git-favicon.png
/home/zoro/ONEPIECE/East_Blue/Romance_Dawn/Casa_de_Luffy/poneglyph.zip
zoro@55af71d2a8d0:~$
```

Paso 6

Desencripté la imagen. Utilizando la como contraseña la flag del challenge anterior.



```
zoro@55af71d2a8d0:~/ONEPIECE/East_Blue/Romance_Dawn/Casa_de_Luffy$ 7z x poneglyph.zip
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,12 CPUs LE)

Scanning the drive for archives:
1 file, 55674 bytes (55 KiB)

Extracting archive: poneglyph.zip
--
Path = poneglyph.zip
Type = zip
Physical Size = 55674

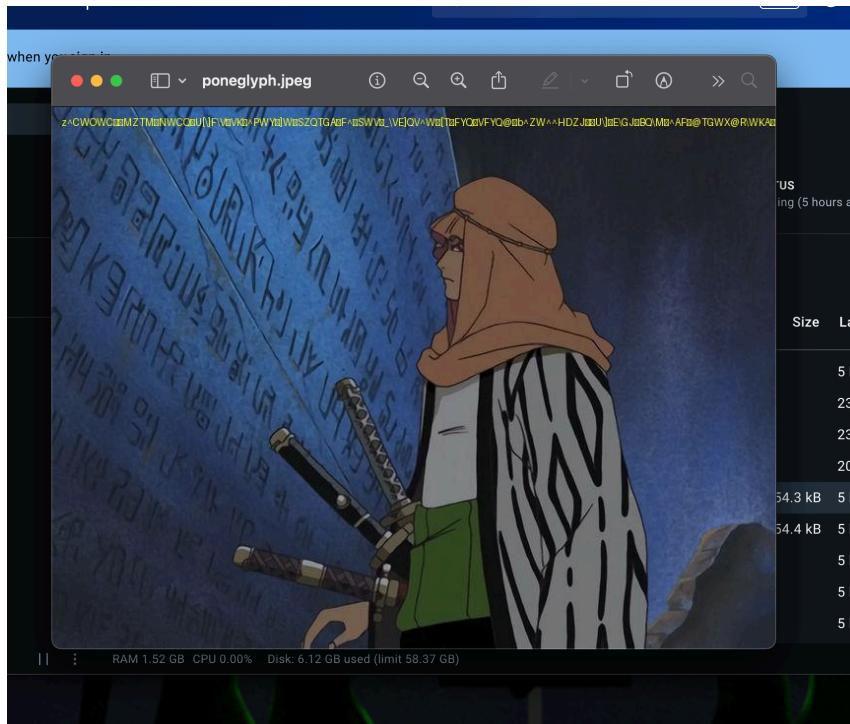
Would you like to replace the existing file:
  Path: ./challenges/zoro/poneglyph.jpeg
  Size:    0 bytes
  Modified: 2025-04-24 00:10:38
with the file from archive:
  Path: challenges/zoro/poneglyph.jpeg
  Size: 55609 bytes (55 KiB)
  Modified: 2025-04-24 00:10:38
? (Y)es / (N)o / (A)lways / (S)kip all / (U)to rename all / (Q)uit? Y

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

Size:      55609
Compressed: 55674
zoro@55af71d2a8d0:~/ONEPIECE/East_Blue/Romance_Dawn/Casa_de_Luffy$
```

Paso 7

Guardar la imagen dentro de mi dispositivo.



Paso 8

Extraer el texto de la imagen

A screenshot of a terminal window. The user is running a Python script named 'extract_text_from_image.py' on a file called 'poneglyph_2.jpeg'. The script uses the PIL library to open the image, the piexif library to extract EXIF metadata, and the xor_cipher library to decrypt the text. The user is prompted to enter a student ID to decrypt the message. The terminal output shows the decrypted text, which is a quote about Poneglyphs being illegal.

b'However, they were limited by lack of access to and knowledge of the other Poneglyphs, and thus sent out researchers to find more information on them, knowing the study of the Poneglyphs was illegal.'

Reto 3 - Usopp

Objetivos

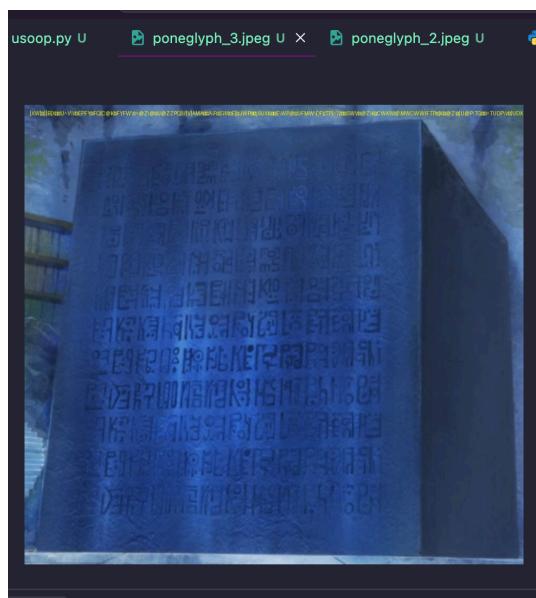
- i. Comprender el funcionamiento de los cifrados de flujo personalizados y su dependencia de claves pseudoaleatorias.
- ii. Analizar la implementación de un cifrado de flujo personalizado y encontrar fallos en la generación de claves para descifrar el mensaje.
- iii. Identificar vulnerabilidades comunes, como la reutilización de claves o el uso de generadores predecibles.
- iv. Fortalecer habilidades de análisis criptográfico en entornos reales con implementaciones no estándar.

Flag encontrada

- FLAG_f6e2eabb946d68a1b76f46512623b700

Párrafo del poneglyphs

Imagen encontrada



Párrafo encontrado

- b'Nico Olvia, along with thirty-three other archaeologists, set out to sea. Sadly, their attempt failed, and they were intercepted by the Marines, leaving Olvia as the sole survivor.'

Documentación

Paso 1

Ingresamos al siguiente challenge, recordando que la contraseña de este challenge es la flag encontrada del challenge anterior (FLAG_eac8738dc786a6313a2b26c500bfdfa).

Ejecutamos el segundo challenge con:

- docker exec -it usopp_challenge bash

Ingresamos al usuario zoro por medio de ingresar la contraseña, la flag del challenge anterior:

- su usopp

```
~/UVG/cipher/ctf_onepice_symmetric_cipher/challenges/zoro git:(main)±9
docker exec -it usopp_challenge bash
nobody@5952576b869d:/home/usopp$ su usopp
Password:
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

usopp@5952576b869d:~$
```

Paso 2

De la misma manera que el challenge anterior, utilice el comando find para poder encontrar todo tipo de archivos relacionados a las flags.

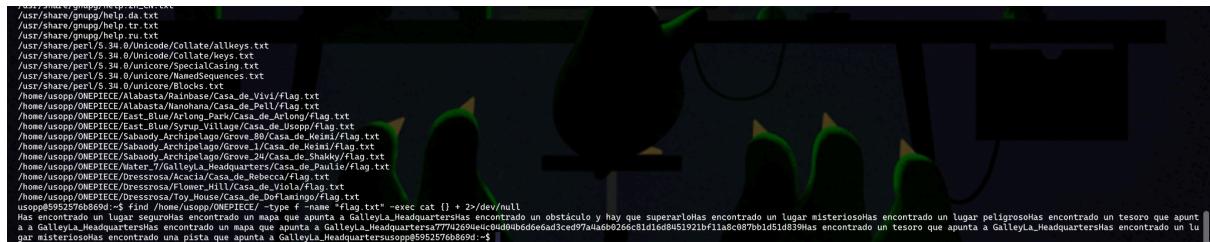
- find / -type f \(| -name "*.txt" -o -name "*.flag" -o -name "*.hidden" -o -name "*.enc" \| 2>/dev/null

```
~/UVG/cipher/ctf_onepice_symmetric_cipher/challenges/zoro git:(main)±9
docker exec -it usopp_challenge bash
/usr/share/vim/vim82/doc/index.txt
/usr/share/vim/vim82/doc/usr_07.txt
/usr/share/vim/vim82/doc/diff.txt
/usr/share/vim/vim82/doc/usr_04.txt
/usr/share/vim/vim82/doc/hebrew.txt
/usr/share/vim/vim82/doc/netbeans.txt
/usr/share/vim/vim82/doc/usr_32.txt
/usr/share/vim/vim82/doc/pi_spec.txt
/usr/share/vim/vim82/doc/options.txt
/usr/share/vim/vim82/doc/usr_45.txt
/usr/share/vim/vim82/pack/dist/opt/matchit/doc/matchit.txt
/usr/share/gnupg/help.nb.txt
/usr/share/gnupg/help.es.txt
/usr/share/gnupg/help.pt_BR.txt
/usr/share/gnupg/help.ro.txt
/usr/share/gnupg/help.sk.txt
/usr/share/gnupg/help.gl.txt
/usr/share/gnupg/help_pl.txt
/usr/share/gnupg/help_eo.txt
/usr/share/gnupg/help_ja.txt
/usr/share/gnupg/help_de.txt
/usr/share/gnupg/help_it.txt
/usr/share/gnupg/help_fi.txt
/usr/share/gnupg/help_zh_TW.txt
/usr/share/gnupg/help_el.txt
/usr/share/gnupg/help_id.txt
/usr/share/gnupg/help_ca.txt
/usr/share/gnupg/help_cs.txt
/usr/share/gnupg/help_hu.txt
/usr/share/gnupg/help_pt.txt
/usr/share/gnupg/help_sv.txt
/usr/share/gnupg/help_et.txt
/usr/share/gnupg/help_be.txt
/usr/share/gnupg/help.txt
/usr/share/gnupg/help_fr.txt
/usr/share/gnupg/help_zh_CN.txt
/usr/share/gnupg/help_da.txt
/usr/share/gnupg/help_tr.txt
/usr/share/gnupg/help_ru.txt
/usr/share/perl/5.34.0/Unicode/Collate/allkeys.txt
/usr/share/perl/5.34.0/Unicode/Collate/keys.txt
/usr/share/perl/5.34.0/unicode/SpecialCasing.txt
/usr/share/perl/5.34.0/unicode/NamedSequences.txt
/usr/share/perl/5.34.0/unicode/Blocks.txt
/home/usopp/ONEPIECE/Alabasta/Rainbase/Casa_de_Vivi/flag.txt
/home/usopp/ONEPIECE/Alabasta/Nanohana/Casa_de_Pell/flag.txt
/home/usopp/ONEPIECE/East_Blue/Arlong_Park/Casa_de_Arlong/flag.txt
/home/usopp/ONEPIECE/East_Blue/Syrup_Village/Casa_de_Usopp/flag.txt
/home/usopp/ONEPIECE/Sabadoy_Archipelago/Grove_80/Casa_de_Keimi/flag.txt
/home/usopp/ONEPIECE/Sabadoy_Archipelago/Grove_1/Casa_de_Keimi/flag.txt
/home/usopp/ONEPIECE/Water_7/Galleyla_Headquarters/Casa_de_Paulie/flag.txt
/home/usopp/ONEPIECE/dressrosa/Acacia/Casa_de_Rebecca/flag.txt
/home/usopp/ONEPIECE/dressrosa/Flower_Hill/Casa_de_Viola/flag.txt
/home/usopp/ONEPIECE/dressrosa/Toy_House/Casa_de_Doflamingo/flag.txt
usopp@5952576b869d:~$
```

Paso 3

Mostré el contenido de los archivos “flag.txt” con el comando:

- find /home/usopp/ONEPIECE/ -type f -name "flag.txt" -exec cat {} + 2>/dev/null



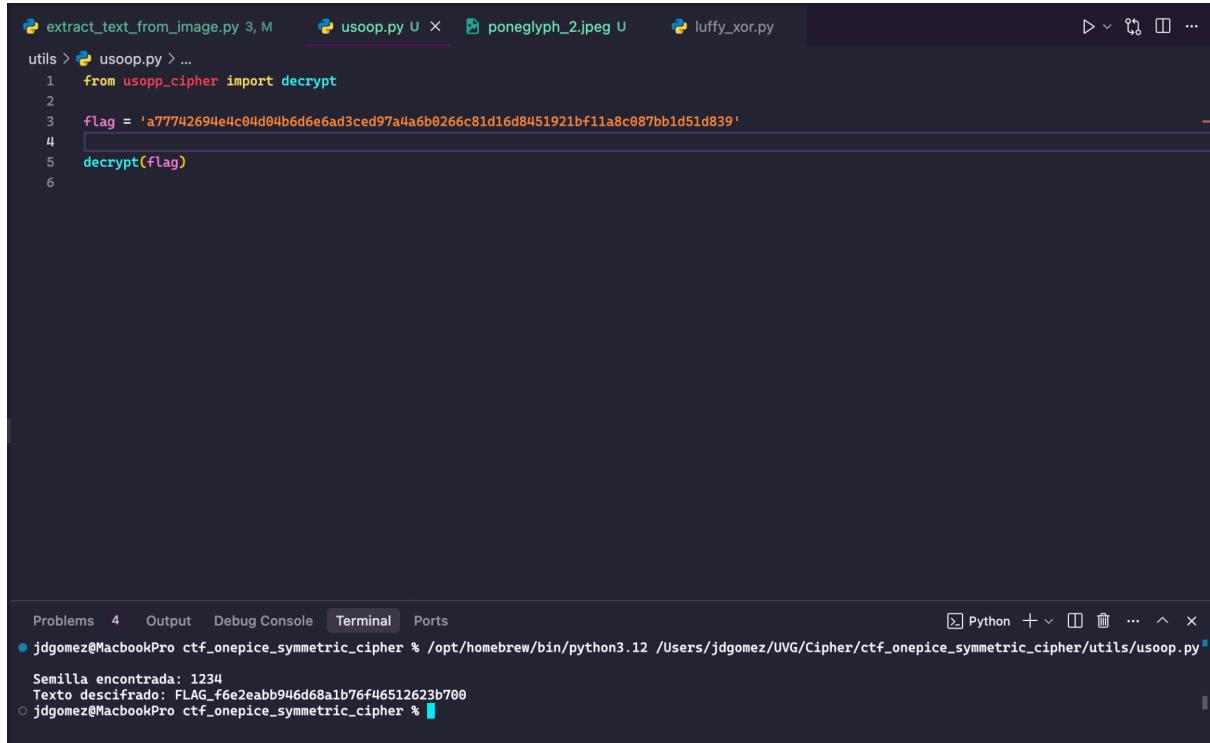
```
/usr/share/grnupg/keyservers.txt
/usr/share/grnupg/help_da.txt
/usr/share/grnupg/help_tr.txt
/usr/share/grnupg/help_ru.txt
/usr/share/grnupg/curve25519/Collate/alleys.txt
/usr/share/perl5/ 34 0/Unicore/Collate/keys.txt
/usr/share/perl5/ 34 0/Unicore/SpecialCasins.txt
/usr/share/perl5/ 34 0/Unicore/Sequences.txt
/usr/share/perl5/ 34 0/Unicore/Block.txt
/home/usopp/ONEPIECE/Alabasta/Rainbase/Casa_de_Vivi/Flag.txt
/home/usopp/ONEPIECE/Alabasta/Ranhana/Casa_de_Veli/Flag.txt
/home/usopp/ONEPIECE/Alabasta/Rebel/Casa_de_King/Flag.txt
/home/usopp/ONEPIECE/East.Blue/Syrup.Village/Casa_de_usopp/Flag.txt
/home/usopp/ONEPIECE/Sabadoy_Archipelago/Grove_38/Casa_de_Keimi/Flag.txt
/home/usopp/ONEPIECE/Sabadoy_Archipelago/Grove_38/Casa_de_Sorini/Flag.txt
/home/usopp/ONEPIECE/Water7_Galleya_Headquarters/Casa_de_Paulie/Flag.txt
/home/usopp/ONEPIECE/Dressrosa/Acacia/Casa_de_Rebecca/Flag.txt
/home/usopp/ONEPIECE/TonyTonyHawk/Casa_de_Tony/Flag.txt
/home/usopp/ONEPIECE/Dressrosa/Toy_House/Casa_de_Doflamingo/Flag.txt
usopp@5952576b8694:~$ find /home/usopp/ONEPIECE/ -type f -name "flag.txt" -exec cat {} + 2>/dev/null
Has encontrado un lugar segurrolas encontrado un mapa que apunta a Galleya_HeadquartersHas encontrado un obstáculo y hay que superarloHas encontrado un lugar misteriosohas encontrado un lugar peligrosoHas encontrado un tesoro que apunt a Galleya_HeadquartersHas encontrado un mapa que apunta a Galleya_HeadquartersHas encontrado una pista que apunta a Galleya_Headquartersusopp@5952576b8694:~$
```

Nuevamente volví a encontrar el contenido de la flag encriptado, que nos servirá para poder encontrar la flag.

- a77742694e4c04d04b6d6e6ad3ced97a4a6b0266c81d16d8451921bf11a8c087bb1d51d839

Paso 4

De la misma manera que el challenge anterior, desencripte el archivo con la ayuda de un script de python que utiliza el algoritmo usopp para desencriptar dicha flag.



```
extract_text_from_image.py 3, M  usoop.py U  poneglyph_2.jpeg U  luffy_xor.py
utils > 🐍 usoop.py ...
1  from usopp_cipher import decrypt
2
3  flag = 'a77742694e4c04d04b6d6e6ad3ced97a4a6b0266c81d16d8451921bf11a8c087bb1d51d839'
4
5  decrypt(flag)
6
```

```
Problems 4 Output Debug Console Terminal Ports Python + ⌂ ⌂ ... ^ x
● jdgoméz@MacbookPro ctf_onepiece_symmetric_cipher % /opt/homebrew/bin/python3.12 /Users/jdgomez/UVG/Cipher/ctf_onepiece_symmetric_cipher/utils/usoop.py
Semilla encontrada: 1234
Texto descifrado: FLAG_f6e2eabb946d68a1b76f46512623b700
○ jdgoméz@MacbookPro ctf_onepiece_symmetric_cipher %
```

Flag:

- FLAG_f6e2eabb946d68a1b76f46512623b700

El código implementa un cifrado que se basa en un proceso conocido como "generador de secuencias de claves" más conocido como keystream, que utiliza un generador de números aleatorios, conocido como PRNG con una semilla dada. Este cifrado utiliza una operación XOR entre cada byte del texto plano y el keystream. Al momento de descifrar, el algoritmo intenta encontrar la semilla correcta para descifrar el texto. La principal diferencia entre un XOR ordinario, es que aquí se utiliza una secuencia de claves generada de forma pseudo-aleatoria para hacer la operación XOR, lo que lo hace dependiente de la semilla para generar el keystream y producir el texto cifrado.

Paso 5

Busqué la imagen con el mismo comando que el challenge anterior.

- find / -type f \(-name "*.jpg" -o -name "*.png" -o -name "*.jpeg" -o -name "*.gif" -o -name "*.bmp" -o -name "*.zip" \) 2>/dev/null

```
usopp@5952576b869d:~$ find / -type f \(-name "*.jpg" -o -name "*.png" -o -name "*.jpeg" -o -name "*.gif" -o -name "*.bmp" -o -name "*.zip" \) 2>/dev/null
/usr/share/info/gnupg-module-overview.png
/usr/share/info/gnupg-card-architecture.png
/usr/share/pixmaps/debian-logo.png
/usr/share/icons/hicolor/48x48/apps/gvim.png
/usr/share/icons/locolor/16x16/apps/gvim.png
/usr/share/icons/locolor/32x32/apps/gvim.png
/usr/share/gitweb/static/git-logo.png
/usr/share/gitweb/static/git-favicon.png
/home/usopp/ONEPIECE/Water_7/Blue_Station/Casa_de_Icebburg/poneglyph.zip
```

Paso 6

Descripté la imagen. Utilizando la como contraseña la flag del challenge anterior.

```
usopp@5952576b869d:~/ONEPIECE/Water_7/Blue_Station/Casa_de_Icebburg$ 7z x poneglyph.zip
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,12 CPUs LE)

Scanning the drive for archives:
1 file, 66837 bytes (66 KiB)

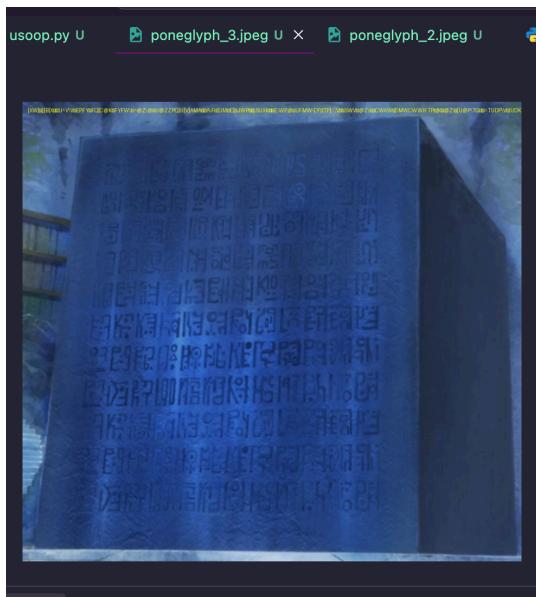
Extracting archive: poneglyph.zip
--
Path = poneglyph.zip
Type = zip
Physical Size = 66837

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

Size:      67218
Compressed: 66837
usopp@5952576b869d:~/ONEPIECE/Water_7/Blue_Station/Casa_de_Icebburg$
```

Paso 7

Guardar la imagen dentro de mi dispositivo.



Paso 8

Extraer el texto de la imagen

```

utils > extract_text_from_image.py 3, M < poneglyph_3.jpeg U ...
utils > extract_text_from_image.py > ...
  6  def extraer_texto_metadata(imagen_path):
  7      # Abrir la imagen
  8      img = Image.open(imagen_path)
  9
 10     # Obtener los metadatos EXIF
 11     exif_dict = piexif.load(img.info.get('exif', b''))
 12
 13     # Obtenemos el texto almacenado en 'Artist' (o en el campo que elegimos)
 14     texto = exif_dict['0th'].get(piexif.ImageIFD.Artist)
 15
 16     if texto:
 17         return texto.decode('utf-8')
 18     return None
 19
 20
 21     # Uso del código para descifrar la imagen
 22     # Ejemplo de uso
 23     image_path = "./utils/resources/poneglyph_3.jpeg"
 24     student_id = input("Introduce tu carné para descifrar el mensaje: ")
 25     texto_cifrado = extraer_texto_metadata(image_path)
 26     decrypted_text = xor_cipher(texto_cifrado, student_id)
 27     print(decrypted_text)
 28

```

Problems 4 Output Debug Console Terminal Ports

jdgoméz@MacBookPro ctf_onepice_symmetric_cipher % python3 ./Users/jdgoméz/UVG/Cipher/ctf_onepice_symmetric_cipher/utils/extract_text_from_image.py
 Introduce tu carné para descifrar el mensaje: 21429
 b'Nico Olvia, along with thirty-three other archaeologists, set out to sea. Sadly, their attempt failed, and they were intercepted by the Marines, leaving Olvia as the sole survivor.'

b'Nico Olvia, along with thirty-three other archaeologists, set out to sea. Sadly, their attempt failed, and they were intercepted by the Marines, leaving Olvia as the sole survivor.'

Reto 4 - Nami

Objetivos

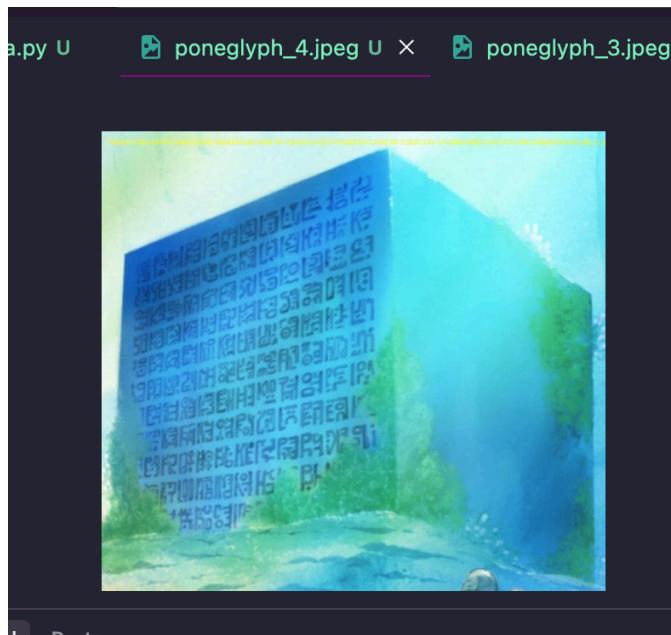
- i. Comprender el funcionamiento del cifrado de flujo ChaCha20 y su importancia en la criptografía moderna.
- ii. Analizar el rol del nonce y su impacto en la seguridad del mensaje cifrado.
- iii. Experimentar con la encriptación y desencriptación de datos utilizando claves derivadas del usuario.
- iv. Fortalecer habilidades prácticas en el uso de cífrados seguros y buenas prácticas criptográficas.

Flag encontrada

- FLAG_e0a5831dc1bf73258131a44ffeb85131

Párrafo del poneglyphs

Imagen encontrada



Párrafo encontrado

- b'The World Government had always been wary of the archaeologists because of their knowledge, but researching the Poneglyphs meant they had an excuse to act against them.'

Documentación

Paso 1

Ingresamos al siguiente challenge, recordando que la contraseña de este challenge es la flag encontrada del challenge anterior (FLAG_f6e2eabb946d68a1b76f46512623b700).

Ejecutamos el segundo challenge con:

- docker exec -it usopp_challenge bash

Ingresamos al usuario zoro por medio de ingresar la contraseña, la flag del challenge anterior:

- su usopp

```
~/UVG/cipher/ctf_onepice_symmetric_cipher/challenges/zoro git:(main)*10
docker exec -it nami_challenge bash
nobody@ef5804d46d73:/home/nami$ su nami
Password:
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

nami@ef5804d46d73:~$
```

Paso 2

De la misma manera que el challenge anterior, utilice el comando find para poder encontrar todo tipo de archivos relacionados a las flags.

- find / -type f \(-name "*.txt" -o -name "*.flag" -o -name "*.hidden" -o -name "*.enc" \) 2>/dev/null

```

~/UVG/cipher/ctf_onepice_symmetric_cipher/challenges/zoro git:(main)*10
docker exec -it nami_challenge bash
/usr/share/vim/vim82/doc/version7.txt
/usr/share/vim/vim82/doc/index.txt
/usr/share/vim/vim82/doc/usr_07.txt
/usr/share/vim/vim82/doc/diff.txt
/usr/share/vim/vim82/doc/usr_04.txt
/usr/share/vim/vim82/doc/hebrew.txt
/usr/share/vim/vim82/doc/netbeans.txt
/usr/share/vim/vim82/doc/usr_32.txt
/usr/share/vim/vim82/doc/pi_spec.txt
/usr/share/vim/vim82/doc/options.txt
/usr/share/vim/vim82/doc/usr_45.txt
/usr/share/vim/vim82/pack/dist/opt/matchit/doc/matchit.txt
/usr/share/gnupg/help_nb.txt
/usr/share/gnupg/help_es.txt
/usr/share/gnupg/help_pt_BR.txt
/usr/share/gnupg/help_ro.txt
/usr/share/gnupg/help_sk.txt
/usr/share/gnupg/help_gl.txt
/usr/share/gnupg/help_pl.txt
/usr/share/gnupg/help_eo.txt
/usr/share/gnupg/help_ja.txt
/usr/share/gnupg/help_de.txt
/usr/share/gnupg/help_it.txt
/usr/share/gnupg/help_fi.txt
/usr/share/gnupg/help_zh_TW.txt
/usr/share/gnupg/help_el.txt
/usr/share/gnupg/help_id.txt
/usr/share/gnupg/help_ca.txt
/usr/share/gnupg/help_cs.txt
/usr/share/gnupg/help_hu.txt
/usr/share/gnupg/help_pt.txt
/usr/share/gnupg/help_sv.txt
/usr/share/gnupg/help_et.txt
/usr/share/gnupg/help_be.txt
/usr/share/gnupg/help_tr.txt
/usr/share/gnupg/help_ru.txt
/usr/share/perl/5.34.0/Unicode/Collate/allkeys.txt
/usr/share/perl/5.34.0/Unicode/Collate/keys.txt
/usr/share/perl/5.34.0/unicore/SpecialCasing.txt
/usr/share/perl/5.34.0/unicore/NamedSequences.txt
/usr/share/perl/5.34.0/unicore/Blocks.txt
/home/nami/ONEPIECE/Alabasta/Rainbase/Casa_de_Vivi/flag.txt
/home/nami/ONEPIECE/Alabasta/Nanohana/Casa_de_Toto/flag.txt
/home/nami/ONEPIECE/East_Blue/Shells_Town/Casa_de_Rika/flag.txt
/home/nami/ONEPIECE/East_Blue/Romance_Dawn/Casa_de_Makino/flag.txt
/home/nami/ONEPIECE/Sabaody_Archipelago/Grove_24/Casa_de_Rayleigh/flag.txt
/home/nami/ONEPIECE/Sabaody_Archipelago/Grove_66/Casa_de_Rayleigh/flag.txt
/home/nami/ONEPIECE/Water_7/Shift_Station/Casa_de_Paulie/flag.txt
/home/nami/ONEPIECE/Water_7/Franky_House/Casa_de_Franky/flag.txt
/home/nami/ONEPIECE/Dressrosa/Corrida_Colosseum/Casa_de_Riku_Dold_III/flag.txt
/home/nami/ONEPIECE/Dressrosa/Flower_Hill/Casa_de_Rebecca/flag.txt
nami@ef5804d46d73:~$
```

Paso 3

Mostré el contenido de los archivos “flag.txt” con el comando:

- find /home/nami/ONEPIECE/ -type f -name "flag.txt" -exec cat {} + 2>/dev/null

```

/home/nami/ONEPIECE/Water_7/Franky_House/Casa_de_Franky/flag.txt
/home/nami/ONEPIECE/Dressrosa/Corrida_Colosseum/Casa_de_Riku_Dold_III/flag.txt
/home/nami/ONEPIECE/Dressrosa/Flower_Hill/Casa_de_Rebecca/flag.txt
nami@ef5804d46d73:~$ find /home/nami/ONEPIECE/ -type f -name "flag.txt" -exec cat {} + 2>/dev/null
Has encontrado un obstáculo y hay que superarloHas encontrado un enemigo y ha tenido que huirHas encontrado un tesoro que apunta a Flower_HillHas encontrado una pista que apunta a Flower_HillHas encontrado un mapa que apunta a Flower_HillHas encontrado un mapa que apunta a Flower_HillHas encontrado un lugar misteriosoHas encontrado un lugar lleno de secretos y misterios que apuntan a Flower_HillHas encontrado un objeto misterioso y no sabes qué es8010e59882f300f3932
1675c7ca2074ac371a1475de25f1540639dd0413c89093c15c8e3d0nami@ef5804d46d73:~$
```

Nuevamente volví a encontrar el contenido de la flag encriptado, que nos servirá para poder encontrar la flag.

- 8010e59882f300f39321675c7ca2074ac371a1475de25f1540639dd0413c89093c15c8e
3d6

Paso 4

De la misma manera que el challenge anterior, desencripte el archivo con la ayuda de un script de python que utiliza el algoritmo usopp para desencriptar dicha flag.

```
extract_text_from_image.py 3, M chacha.py U poneglyph_3.jpeg U
utils > chacha.py > ...
1  from nami_chacha import chacha20_decrypt
2  from binascii import unhexlify
3
4  # El codigo hexadecimal cifrado
5  flag = "8010e59882f300f39321675c7ca2074ac371a1475de25f1540639dd0413c89093c15c8e3d6"
6
7  try:
8      ciphertext_bytes = unhexlify(flag)
9      carne = "21429"
10
11     # Desencriptar
12     plaintext = chacha20_decrypt(ciphertext_bytes, carne)
13     print("Mensaje desencriptado:", plaintext)
14 except Exception as e:
15     print(f"Error al desencriptar: {str(e)}")
16

Problems 4 Output Debug Console Terminal Ports
jdgomez@MacbookPro ctf_onepiece_symmetric_cipher % python3 /Users/jdgomez/UVG/Cipher/ctf_onepiece_symmetric_cipher/utils/chacha.py
Mensaje desencriptado: FLAG_e0a5831dc1bf73258131a44ffeb85131
jdgomez@MacbookPro ctf_onepiece_symmetric_cipher %
```

Flag:

- FLAG_e0a5831dc1bf73258131a44ffeb85131

Este script utiliza la libreria nami_chacha para descifrar un mensaje cifrado en formato hexadecimal con el algoritmo ChaCha20. El primer paso es convertir la cadena de caracteres cifrada a bytes, luego utiliza el carne como base para derivar la clave. Despu s descifra el mensaje.

Paso 5

Busqu  la imagen con el mismo comando que el challenge anterior.

- find / -type f \(-name "*.jpg" -o -name "*.png" -o -name "*.jpeg" -o -name "*.gif" -o -name "*.bmp" -o -name "*.zip"\) 2>/dev/null

```
usopp@5952576b869d:~$ find / -type f \(-name "*.jpg" -o -name "*.png" -o -name "*.jpeg" -o -name "*.gif" -o -name "*.bmp" -o -name "*.zip"\) 2>/dev/null
/usr/share/info/gnupg-module-overview.png
/usr/share/info/gnupg-card-architecture.png
/usr/share/pixmaps/debian-logo.png
/usr/share/icons/hicolor/48x48/apps/gvim.png
/usr/share/icons/locolor/16x16/apps/gvim.png
/usr/share/icons/locolor/32x32/apps/gvim.png
/usr/share/gitweb/static/git-logo.png
/usr/share/gitweb/static/git-favicon.png
/home/usopp/ONEPIECE/Water_7/Blue_Station/Casa_de_Icebburg/poneglyph.zip
```

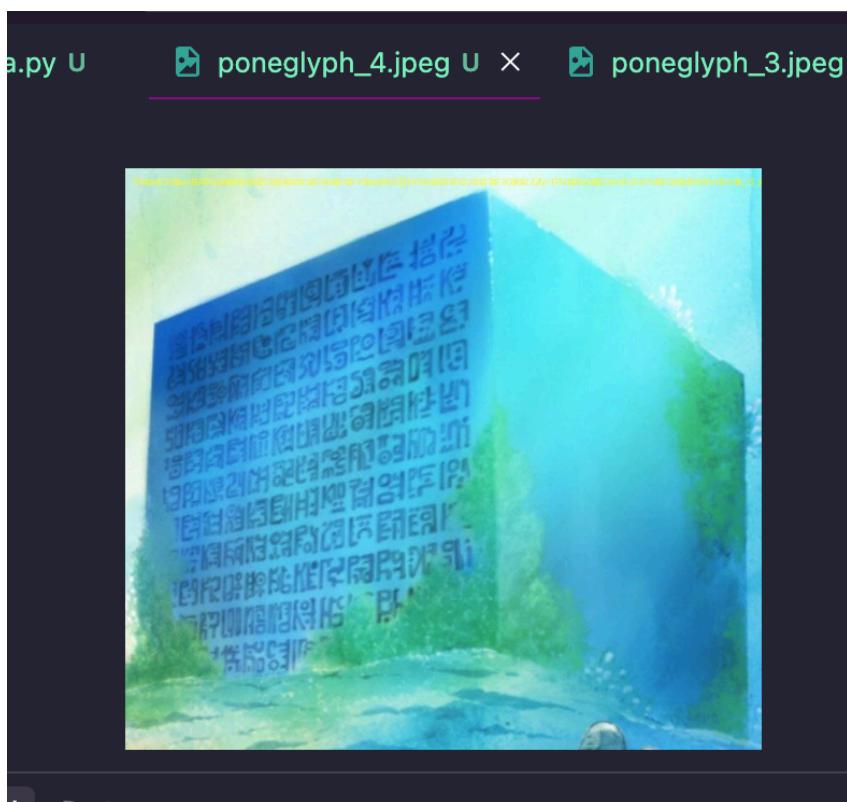
Paso 6

Desencript  la imagen. Utilizando la como contrase a la flag del challenge anterior.

```
nami@ef5804d46d73:~/ONEPIECE/Alabasta/Katorea/Casa_de_Toto$ 7z x poneglyph.zip  
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,12 CPUs LE)  
  
Scanning the drive for archives:  
1 file, 52870 bytes (52 KiB)  
  
Extracting archive: poneglyph.zip  
--  
Path = poneglyph.zip  
Type = zip  
Physical Size = 52870  
  
Enter password (will not be echoed):  
Everything is Ok  
  
Size: 53144  
Compressed: 52870  
nami@ef5804d46d73:~/ONEPIECE/Alabasta/Katorea/Casa_de_Toto$
```

Paso 7

Guardar la imagen dentro de mi dispositivo.



Paso 8

Extraer el texto de la imagen

The screenshot shows a terminal window with several tabs at the top: 'extract_text_from_image.py 3, M', 'chacha.py U', 'poneglyph_4.jpeg U', and 'poneglyph_3.jpeg U'. Below the tabs, there is a code editor with Python code for extracting text from images. The code includes a function 'extraer_texto_metadata' that reads metadata from an image file using PIL and decodes it to UTF-8 if it exists. It also includes an example usage block. The terminal output shows the command 'python3 extract_text_from_image.py' being run, followed by an input prompt 'Introduce tu carén para descifrar el mensaje: 21429'. The response is a decoded message: 'b'The World Government had always been wary of the archaeologists because of their knowledge, but researching the Poneglyphs meant they had an excuse to act against them.'

```
utils > extract_text_from_image.py > ...
  6     def extraer_texto_metadata(imagen_path):
  7         texto = EXTRAE_DICTEL_OCL(j.getpaxelx,ImagenID=ALERTA)
  8         if texto:
  9             return texto.decode('utf-8')
 10         return None
 11
 12
 13     # Uso del código para descifrar la imagen
 14     # Ejemplo de uso
 15     image_path = "./utils/resources/poneglyph_4.jpeg"
 16     student_id = input("Introduce tu carén para descifrar el mensaje: ")
 17     texto_cifrado = extraer_texto_metadata(image_path)
 18     decrypted_text = xor_cipher(texto_cifrado, student_id)
 19     print(decrypted_text)
 20
 21
 22
 23
 24
 25
 26
 27
 28

Problems 4 Output Debug Console Terminal Ports
jdgomomez@MacbookPro ctf_onepice_symmetric_cipher % python3 /Users/jdgomez/UVG/Cipher/ctf_onepice_symmetric_cipher/utils/extract_text_from_image.py
Introduce tu carén para descifrar el mensaje: 21429
b'The World Government had always been wary of the archaeologists because of their knowledge, but researching the Poneglyphs meant they had an excuse to act against them.'
jdgomomez@MacbookPro ctf_onepice_symmetric_cipher %
```

b'The World Government had always been wary of the archaeologists because of their knowledge, but researching the Poneglyphs meant they had an excuse to act against them.'

Reflexión final

Este proyecto número 1 me pareció una inmersión práctica y desafiante en el mundo de los cifrados de flujo y la seguridad de la información. A través de la resolución de los cuatro retos temáticos (Luffy, Zoro, Usopp y Nami), pudimos explorar y aplicar directamente conceptos teóricos de criptografía, desde el básico XOR hasta algoritmos más robustos como RC4 y ChaCha20, e incluso un cifrado de flujo personalizado. La experiencia no sólo reforzó la comprensión de cómo funcionan estos cifrados, sino que también subrayó la importancia crítica de factores como la gestión de claves, el uso de nonces y las vulnerabilidades inherentes a cada método, como la reutilización de claves o los sesgos en la generación de flujos pseudoaleatorios. Resolver cada desafío requirió no solo conocimiento técnico, sino también pensamiento analítico y habilidades de resolución de problemas para identificar patrones, encontrar archivos ocultos en entornos simulados (contenedores Docker) y aplicar correctamente las herramientas y scripts de desencriptación y extracción de metadatos. Para mí, el proyecto fue una valiosa oportunidad para fortalecer la intuición criptográfica y desarrollar habilidades prácticas en el análisis y la superación de desafíos de seguridad informática en un contexto simulado pero realista.

Retos encontrados

Durante el desarrollo del proyecto, surgieron diversos retos que pusieron a prueba tanto los conocimientos técnicos como la capacidad de adaptación. Uno de los principales desafíos fue la familiarización con los distintos entornos de Docker para cada reto y el uso eficiente de

comandos de consola (como find y cat) para navegar por los sistemas de archivos simulados y localizar los archivos clave (flags e imágenes) entre múltiples directorios y archivos distractores. Comprender la lógica específica de cada algoritmo de cifrado (XOR con clave de carné, RC4, el cifrado personalizado basado en PRNG y ChaCha20 con nonce) y aplicar correctamente los scripts de Python para la desencriptación fue crucial y requirió análisis detallado, especialmente al identificar y utilizar las claves correctas, que a menudo dependían del carné del estudiante o de la flag obtenida en el reto anterior. La extracción de texto oculto en los metadatos de las imágenes presentó otro nivel de complejidad, implicando la transferencia de archivos desde Docker y el uso de scripts específicos. Finalmente, la gestión de las dependencias iniciales y la depuración de los scripts propios o proporcionados para asegurar su correcto funcionamiento con las claves y datos cifrados específicos de cada etapa, también formaron parte integral de los obstáculos superados.