

# Informe Laboratorio 1

## Sección 1

Ignacio Santiago Medina Díaz  
e-mail: ignacio.medina1@mail.udp.cl

Agosto de 2024

## Índice

<b>1. Descripción</b>	<b>2</b>
<b>2. Actividades</b>	<b>2</b>
2.1. Algoritmo de cifrado . . . . .	2
2.2. Modo stealth . . . . .	2
2.3. MitM . . . . .	3
<b>3. Desarrollo de Actividades</b>	<b>4</b>
3.1. Actividad 1 . . . . .	4
3.1.1. Funcionamiento del código . . . . .	5
3.1.2. Ejecución del código . . . . .	6
3.2. Actividad 2 . . . . .	6
3.2.1. Funcionamiento del código . . . . .	8
3.2.2. Ejecución del código . . . . .	9
3.3. Actividad 3 . . . . .	11
3.3.1. Funcionamiento del código . . . . .	12
3.3.2. Ejecución del código . . . . .	13

## 1. Descripción

1. Usted empieza a trabajar en una empresa tecnológica que se jacta de poseer sistemas que permiten identificar filtraciones de información a través de Deep Packet Inspection (DPI). A usted le han encomendado auditar si efectivamente estos sistemas son capaces de detectar las filtraciones a través de tráfico de red. Debido a que el programa ping es ampliamente utilizado desde dentro y hacia fuera de la empresa, su tarea será crear un software que permita replicar tráfico generado por el programa ping con su configuración por defecto, pero con fragmentos de información confidencial. Recuerde que al comparar tráfico real con el generado no debe gatillar alarmas. De todas formas, deberá hacer una prueba de concepto, en la cual se demuestre que al conocer el algoritmo, será fácil determinar el mensaje en claro. Para los pasos 1,2,3 indicar el texto entregado a ChatGPT y validar si el código resultante cumple con lo requerido.

## 2. Actividades

### 2.1. Algoritmo de cifrado

1. Generar un programa, en python3 utilizando chatGPT, que permita cifrar texto utilizando el algoritmo Cesar. Como parámetros de su programa deberá ingresar el string a cifrar y luego el corrimiento.

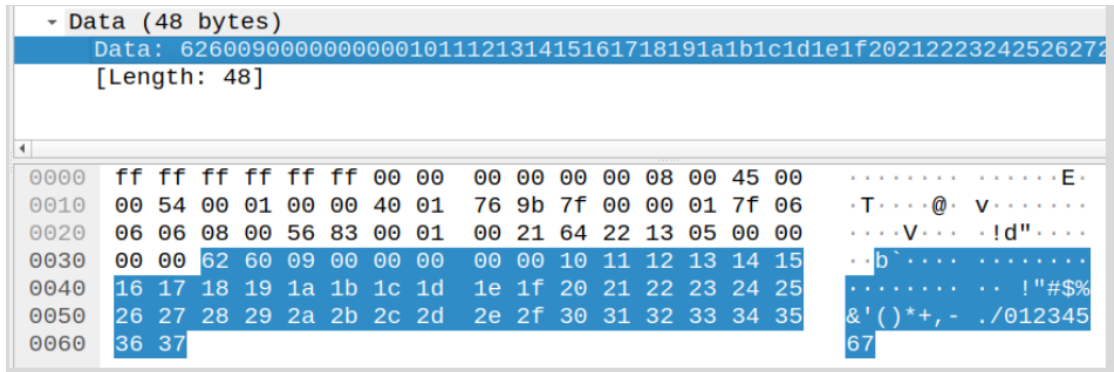
```
└─$ ~/Desktop $ sudo python3 cesar.py "criptografia y seguridad en redes" 9
larycxpajorj h bnpdarmjm nw anmnb
```

### 2.2. Modo stealth

1. Generar un programa, en python3 utilizando ChatGPT, que permita enviar los caracteres del string (el del paso 1) en varios paquetes ICMP request (un caracter por paquete en el campo data de ICMP) para de esta forma no gatillar sospechas sobre la filtración de datos. Deberá mostrar los campos de un ping real previo y posterior al suyo y demostrar que su tráfico consideró todos los aspectos para pasar desapercibido.

```
└─$ ~/Desktop $ sudo python3 pingv4.py "larycxpajorj h bnpdarmjm nw anmnb"
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

El último carácter del mensaje se transmite como una b.



## 2.3. MitM

1. Generar un programa, en python3 utilizando ChatGPT, que permita obtener el mensaje transmitido en el paso2. Como no se sabe cual es el corrimiento utilizado, genere todas las combinaciones posibles e imprímalas, indicando en verde la opción más probable de ser el mensaje en claro.

```

$ sudo python3 readv2.py cesar.pcapng
0      larycxpajorj h bnpdarmjm nw anmnb
1      kzqxbwozinqi g amoczqlil mv zmlma
2      jypwavnymph f zlnbypkhk lu yklklz
3      ixovzumxglog e ykmaxojgj kt xkjky
4      hwnuytlwfkf d xjlzwnifi js wjiix
5      gvmtxskvejme c wikyvmeheh ir vihiw
6      fulswrjudild b vhjxulgdg hq uhghv
7      etkrvqitchkc a ugiwtkfcf gp tgfgu
8      dsjquphsbgjb z tfhvsjebe fo sfef
9      criptografia y seguridad en redes
10     bqhosnfqzehz x rdftqhczc dm qdcdr
11     apgnrmepydgy w qcespgbyb cl pcabcq
12     zofmqldoxcfx v pbdrofaxa bk obabp
13     ynelpkcnwbew u oacqnezwz aj nazao
14     xmdkojbmadv t nzbpmdivy zi mzyzn
15     wlcjnia luzcu s myaolcxux yh lyxym
16     vkbimhzktybt r lxznkbwtw xg kxwxl
17     ujahlgysxas q kwymjavsv wf jwvwk
18     tizgkfxirwzr p jvxlizuru ve ivuvj
19     shyfjewhqvyq o iuwkhytqt ud hutui
20     rgxeidvgpuxp n htvjgxspc tc gtsth
21     qfwdhcufotwo m gsuifwrwr sb fsrsg
22     pevcbtensvn l frthevqng ra erqrf
23     odubfasdmrum k eqsgdupmp qz dqpqe
24     nctaezrclqtl j dprfctolo py cpopd
25     mbszdyqbksk i coqebnkn ox bonoc

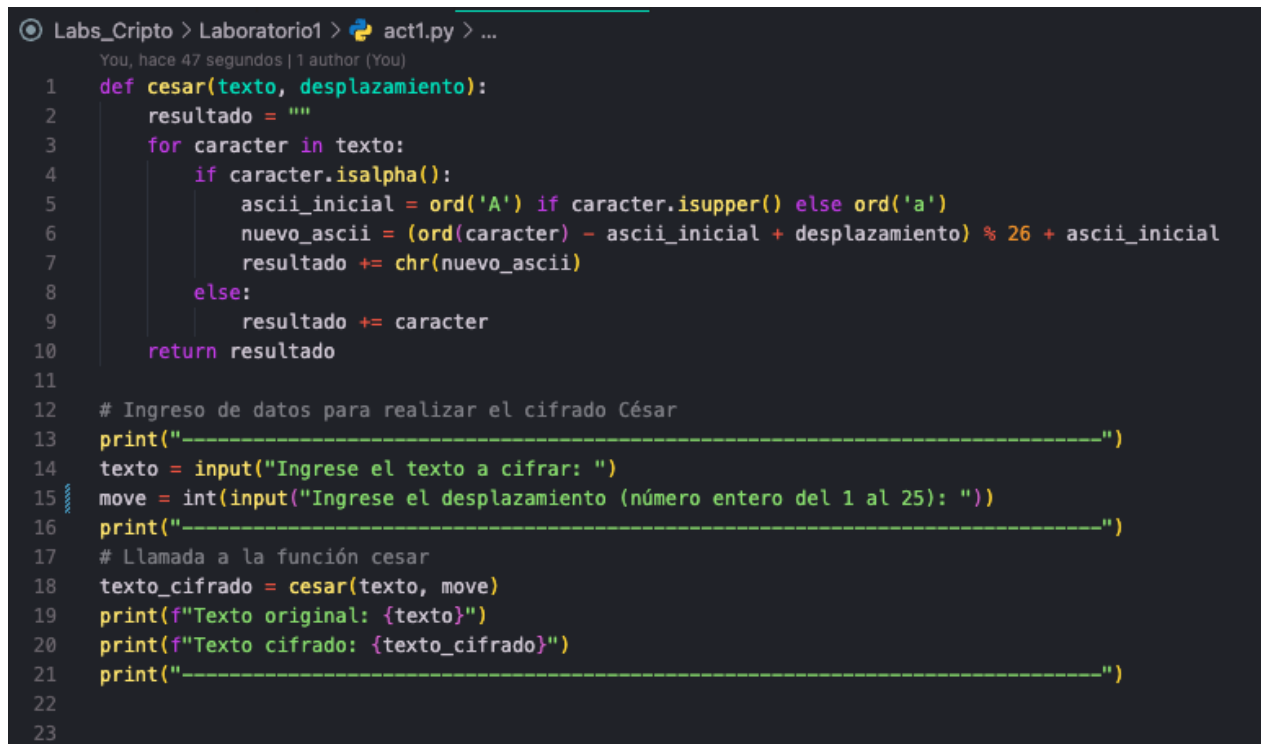
```

Finalmente, deberá indicar 4 issues que haya tenido al lidiar con ChatGPT, netamente para reflejar cuál fue su experiencia al trabajar con esta tecnología.

### 3. Desarrollo de Actividades

#### 3.1. Actividad 1

Para este primer apartado, se generó el siguiente programa en Python 3 junto con Claude, el cual permite el cifrado de texto utilizando el algoritmo César.



```

1  def cesar(texto, desplazamiento):
2      resultado = ""
3      for caracter in texto:
4          if caracter.isalpha():
5              ascii_inicial = ord('A') if caracter.isupper() else ord('a')
6              nuevo_ascii = (ord(caracter) - ascii_inicial + desplazamiento) % 26 + ascii_inicial
7              resultado += chr(nuevo_ascii)
8          else:
9              resultado += caracter
10     return resultado
11
12     # Ingreso de datos para realizar el cifrado César
13     print("-----")
14     texto = input("Ingrese el texto a cifrar: ")
15     move = int(input("Ingrese el desplazamiento (número entero del 1 al 25): "))
16     print("-----")
17     # Llamada a la función cesar
18     texto_cifrado = cesar(texto, move)
19     print(f"Texto original: {texto}")
20     print(f"Texto cifrado: {texto_cifrado}")
21     print("-----")
22
23

```

Figura 1: Código algoritmo César.

El código anterior recibe dos parámetros, uno llamado **texto**, de tipo string, que recibe el texto a cifrar, y el otro, llamado **move** (*considerando desplazamientos del 1 al 25*), de tipo int, que se utilizará para realizar el desplazamiento del texto para llevar a cabo el cifrado. A continuación, se validará el funcionamiento del código durante su ejecución.

Para recibir este código, se entregó el siguiente prompt a Claude para generarlo.

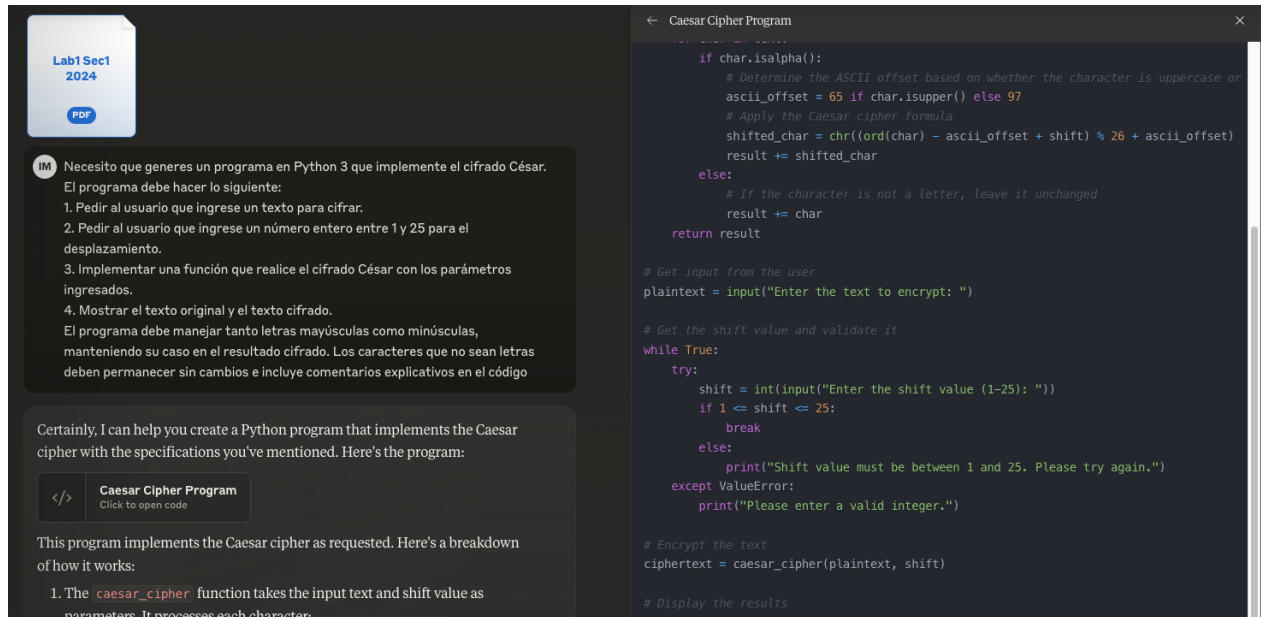


Figura 2: Prompt utilizado para la actividad 1.

A este prompt de la Figura 2 se le realizaron más consultas para corregir el código y solucionar errores posteriormente; finalmente, se modificaron manualmente los inputs del mismo.

### 3.1.1. Funcionamiento del código

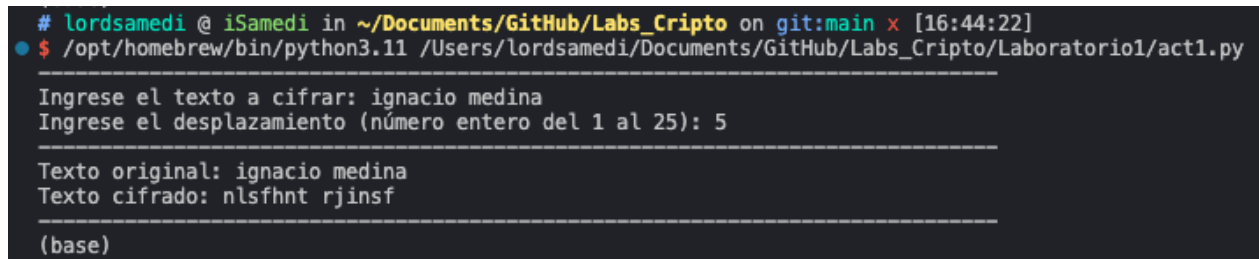
El programa define una función llamada '**cesar**' que toma dos parámetros: el texto a cifrar y el desplazamiento. Esta función realiza el cifrado César carácter por carácter.

Para cada carácter del texto de entrada, la función verifica si es una letra. Si lo es, determina si es mayúscula o minúscula y aplica el desplazamiento correspondiente dentro del rango del alfabeto (*26 letras a desplazar*). Cabe mencionar que los caracteres que no son letras se mantienen sin cambios.

Después de definir la función, el programa solicita al usuario que ingrese el texto a cifrar y el valor de desplazamiento. Utiliza la función `input()` para capturar estas entradas. Luego, llama a la función '**cesar**' con los parámetros proporcionados por el usuario y almacena el resultado en la variable '*texto\_cifrado*'.

Finalmente, el programa muestra por pantalla tanto el *texto original* como el *texto cifrado*, permitiendo al usuario ver el resultado del cifrado César aplicado a su entrada.

### 3.1.2. Ejecución del código



```
# lordsamedi @ iSamedi in ~/Documents/GitHub/Labs_Cripto on git:main x [16:44:22]
$ /opt/homebrew/bin/python3.11 /Users/lordsamedi/Documents/GitHub/Labs_Cripto/Laboratorio1/act1.py

-----
Ingrese el texto a cifrar: ignacio medina
Ingrese el desplazamiento (número entero del 1 al 25): 5
-----
Texto original: ignacio medina
Texto cifrado: nlsfhnt rjinsf
-----
(base)
```

Figura 3: Ejecución del código del algoritmo César.

Dada la ejecución anterior, obtenemos el texto cifrado correspondiente al texto original y al desplazamiento dado, generándolo exitosamente.

## 3.2. Actividad 2

En esta segunda sección del laboratorio, se continuó trabajando con el código previamente desarrollado, realizando algunas modificaciones con el apoyo de Claude durante su implementación. Las alteraciones efectuadas permiten enviar los caracteres de la cadena (*del paso 1*) en varios paquetes de solicitud ICMP, **transmitiendo un carácter por paquete en el campo data** de ICMP. Este enfoque está diseñado para minimizar el riesgo de generar sospechas sobre una posible filtración de datos.

Uno de los prompts utilizados para esta actividad fue el siguiente. Cabe destacar que anteriormente se le entregó el código de la Actividad 1 (*esto se repite para cada actividad; se pasa el archivo de la Actividad 2 para la Actividad 3*).

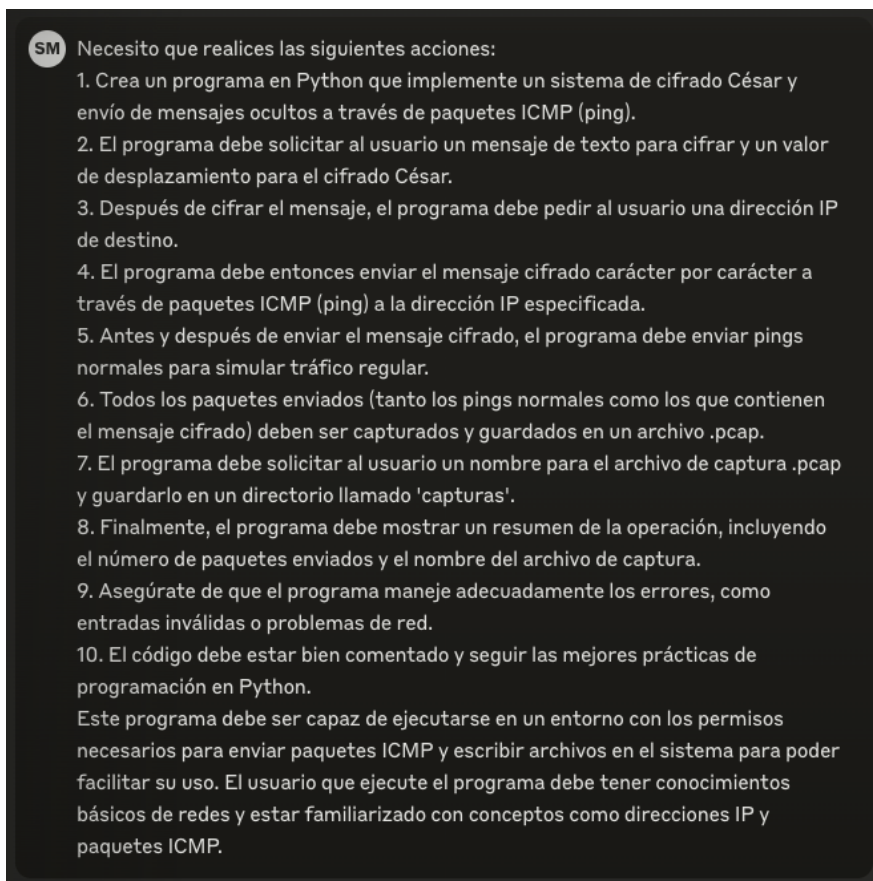


Figura 4: Prompt utilizado para la actividad 2.

En el prompt anterior se explica paso a paso las acciones a realizar conforme a lo solicitado por el laboratorio. Anteriormente, se otorgó el laboratorio a Claude para que muestre las acciones a seguir y dar una idea de los pasos necesarios para lograr la actividad. Una vez generado el código, se le añadieron modificaciones para que el guardado del archivo sea automático, sin necesidad de editar una línea de código. Cabe señalar que a posteriori se le fueron subiendo prompt de errores que surgían a la hora de su ejecución.

***Anotación:*** El código se encuentra en un repositorio de *GitHub*, su link se encuentra al final del informe.

En este apartado se generó una complicación, ya que, no se podía ejecutar Scapy en el código que generó Claude, ya que, pedía modo administrador, por lo tanto se realizó el siguiente prompt.

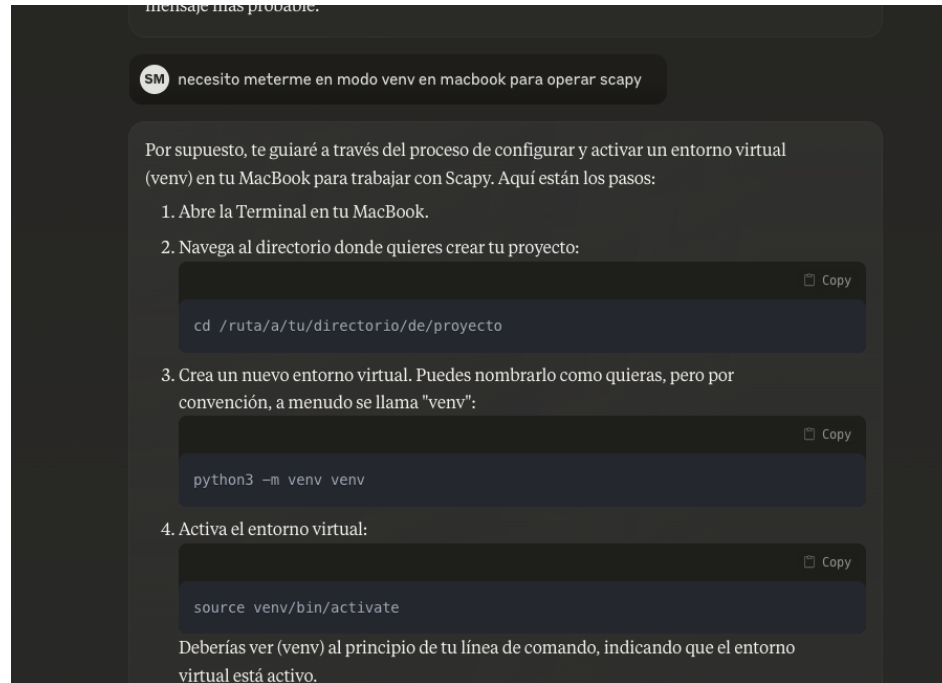


Figura 5: Prompt de proceso de configurar y activar un entorno virtual (venv) en MacBook.

Al utilizar los siguientes comandos, se genero una carpeta en el repositorio llamada 'venv' para que habilite el entorno virtual para poder operar con el comando 'sudo' y lograr utilizar Scapy, y a su vez obtener permisos de administrador. *Cabe destacar que, a la hora de trabajar en este proyecto, necesitarás activar el entorno virtual usando el comando **source venv/bin/activate** desde el directorio de tu proyecto.*

### 3.2.1. Funcionamiento del código

El programa comienza solicitando al usuario un texto para cifrar y un valor de desplazamiento. Utiliza estos datos para aplicar un cifrado César al texto ingresado, cifrado que se realizo en la actividad anterior. Luego, el programa pide al usuario que ingrese una dirección IP de destino. Esta dirección será el objetivo de los paquetes ICMP que se enviarán (*por ejemplo la dns.google 8.8.8.8*).

Antes de enviar el mensaje cifrado, el programa realiza una serie de pings normales a la dirección de destino para establecer un patrón de tráfico '*normal*'.

A continuación, el programa procede a enviar el texto cifrado. Para cada carácter del mensaje cifrado, **crea un paquete ICMP individual** y lo envía a la dirección de destino. Después de enviar todos los caracteres del mensaje cifrado, el programa realiza otra serie de pings normales para '*cerrar*' la comunicación de manera que parezca tráfico regular.

Finalmente, el programa guarda todos los paquetes enviados en un archivo de captura con



extensión .pcap (el nombre del archivo la especifica el usuario guardandolo en una carpeta programada dentro del archivo del Laboratorio 1.

### 3.2.2. Ejecución del código

```
-----
Ingrese el texto a cifrar: ignacio medina
Ingrese el desplazamiento (del 1 al 25): 5
-----
Texto cifrado: nlsfhnt rjinsf
-----
Ingrese la dirección IP de destino (Ejemplo 8.8.8.8): 8.8.8.8
-----
Ping real previo:
Ping real: IP / ICMP 8.8.8.8 > 192.168.100.91 echo-reply 0
Ping real: IP / ICMP 8.8.8.8 > 192.168.100.91 echo-reply 0
Ping real: IP / ICMP 8.8.8.8 > 192.168.100.91 echo-reply 0
Ping real: IP / ICMP 8.8.8.8 > 192.168.100.91 echo-reply 0

Enviando datos como pings:
Enviado 1 paquete con datos: 6e
Enviado 1 paquete con datos: 6c
Enviado 1 paquete con datos: 73
Enviado 1 paquete con datos: 66
Enviado 1 paquete con datos: 68
Enviado 1 paquete con datos: 6e
Enviado 1 paquete con datos: 74
Enviado 1 paquete con datos: 20
Enviado 1 paquete con datos: 72
Enviado 1 paquete con datos: 6a
Enviado 1 paquete con datos: 69
Enviado 1 paquete con datos: 6e
Enviado 1 paquete con datos: 73
Enviado 1 paquete con datos: 66

Ping real posterior:
Ping real: IP / ICMP 8.8.8.8 > 192.168.100.91 echo-reply 0 / Raw
Ping real: IP / ICMP 8.8.8.8 > 192.168.100.91 echo-reply 0 / Raw
Ping real: IP / ICMP 8.8.8.8 > 192.168.100.91 echo-reply 0
Ping real: IP / ICMP 8.8.8.8 > 192.168.100.91 echo-reply 0
-----
Ingrese el nombre del archivo de captura (agrega el '.pcap'): archivo.p
-----
Ingrese el nombre del archivo de captura (agrega el '.pcap'): archivo.pcap
-----
Captura guardada nombreda archivo.pcap contiene: 14 paquetes.
-----
(venv) (base)
# lordsamedi @ iSamedi in ~/Desktop/myFolder/Universidad/8mo/act2 venv [16:57:11]
$
```

Figura 6: Consola al ejecutar act2.py.

La figura 6 se genero al ejecutar en modo administrador (*venv*, esto se especifica en el *README.md* de la carpeta *Laboratorio 1* en el Repositorio) el código de *act2.py*, inicialmente se registra la palabra a cifrar y el desplazamiento (igual forma que la actividad anterior), posteriormente, tal como se señalo anteriormente, se ingresa la IP del destinatario a recibir el paquete de datos ICMP, en el caso del ejemplo se enviaron 14 paquetes con sus respectivas 'Longitudes' (paquete 1 posee una longitud de 6e, paquete 2 de 6c, etc). Una vez finalizada se ingresa el nombre del archivo '.pcap' (en la figura anterior fue nombrado como *archivo.pcap*) y es guardado en la carpeta llamada 'capturas' dentro del repositorio.

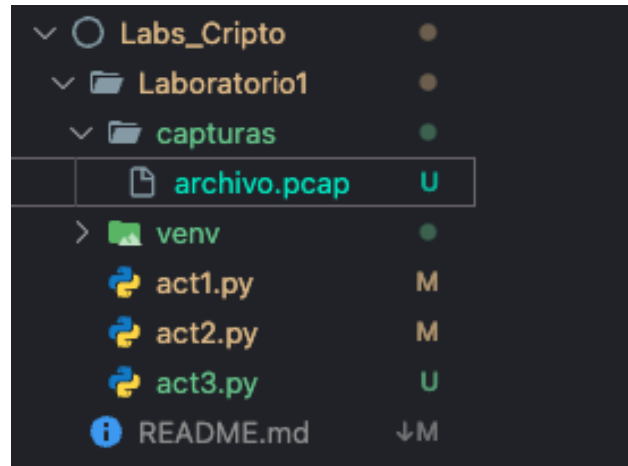


Figura 7: Creación del archivo .pcap de los paquetes enviados.

A continuación, se ejecutó el archivo generado en Wireshark con privilegios de administrador para corroborar que los paquetes ICMP fueron enviados correctamente. El envío fue exitoso hacia el destinatario dns.google, capturando solo los paquetes enviados desde el host.

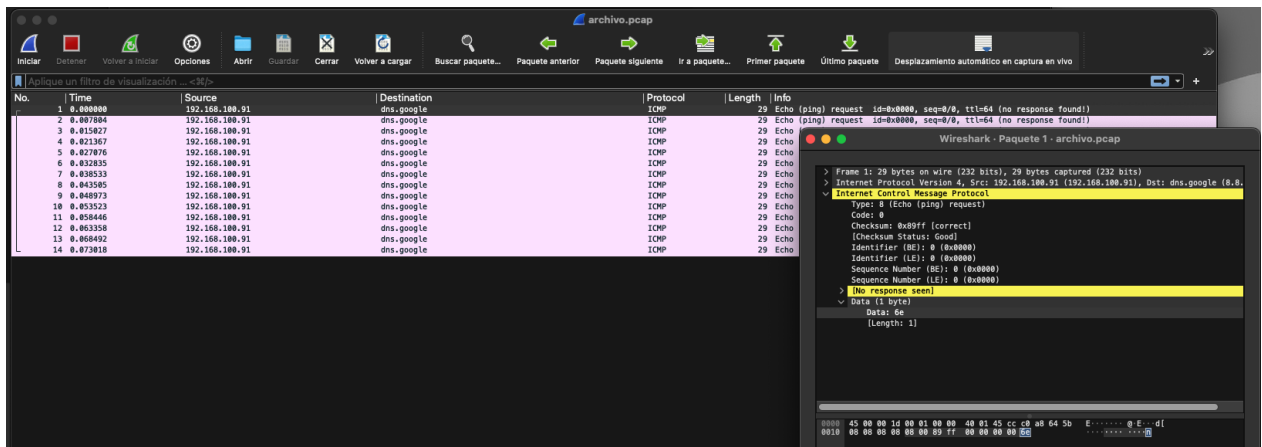


Figura 8: Archivo .pcap abierto en Wireshark.

En la Figura 8 se pueden observar los 14 paquetes del protocolo ICMP enviados, con una longitud idéntica a la señalada en la Figura 5, cada uno de ellos conteniendo una letra de la palabra cifrada por el algoritmo de César, creado en la Actividad 1 y reutilizado en esta actividad.

Finalmente, se puede corroborar que se enviaron correctamente a través de este protocolo al destinatario previamente señalado.

### 3.3. Actividad 3

Para finalizar el laboratorio, se creó un nuevo código que recibirá el archivo **.pcap** generado en la actividad pasada, con el propósito de desencriptar el mensaje enviado a través del protocolo ICMP hacia un destinatario *X*, generando **26 combinaciones** (*por los desplazamientos del 0 al 25*) con el objetivo de poder interceptar y analizar cuál es el corrimiento más probable para descifrar el mensaje en claro.

Al igual que en las actividades anteriores, se utilizó Claude, entregándole el PDF del laboratorio para que lo explique detalladamente. Una vez descrito, se realizó el siguiente prompt explicando punto por punto lo que se necesita para poder realizar el código.

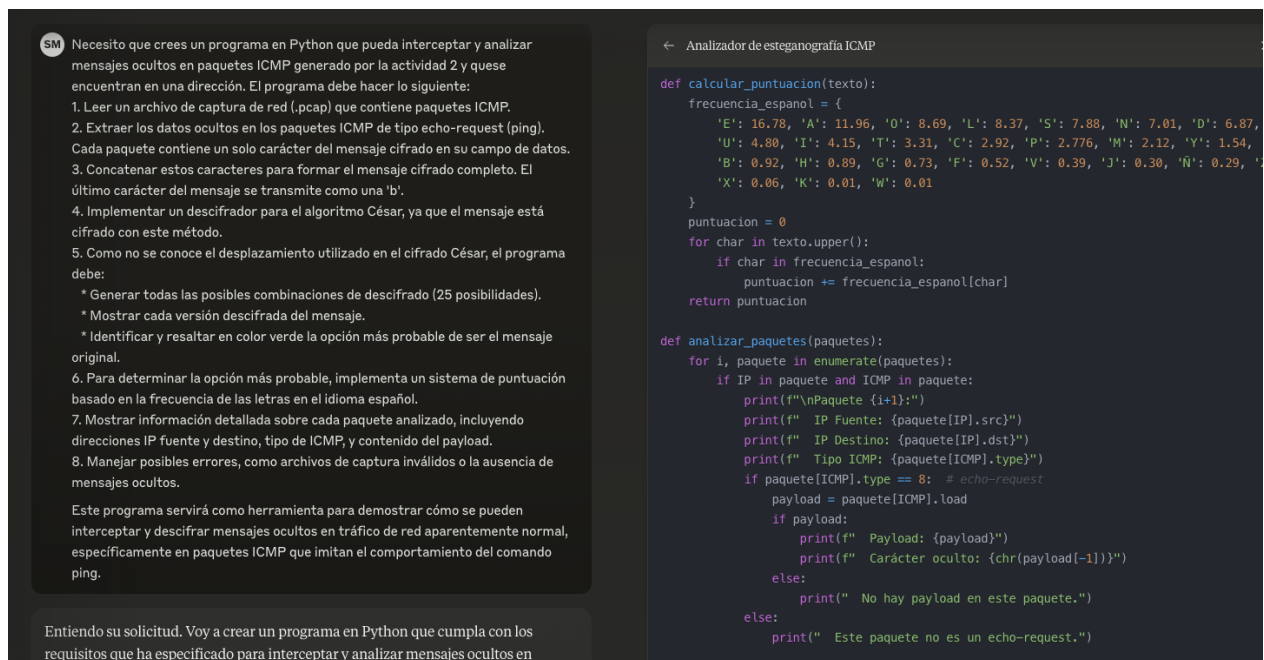


Figura 9: Prompt utilizado para la actividad 3.

Del texto generado por Claude, se le fueron modificando partes del código de forma estética, y los errores que presentaba se le pasaban a la inteligencia para que los corrigiera.

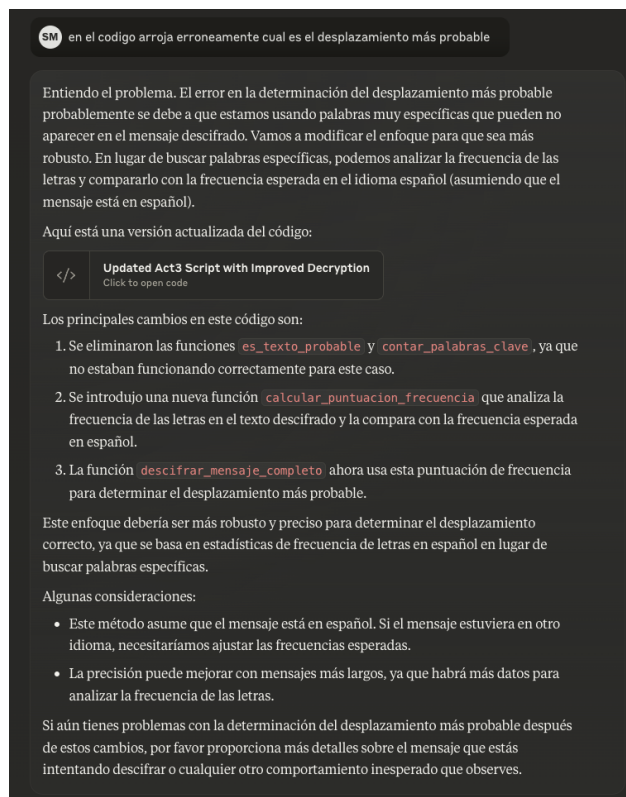


Figura 10: Problemas en los desplazamientos más probables.

En la Figura 10 se logra observar los problemas que se presentaron a la hora de realizar esta actividad, dejando en evidencia uno de los problemas al desarrollar este código a la hora de determinar el desplazamiento más probable.

### 3.3.1. Funcionamiento del código

El programa comienza definiendo varias funciones clave. La primera función '*leer\_pcap*' lee un archivo *.pcap*, extrae los paquetes ICMP de tipo *echo-request* y recopila los datos de la carga útil para formar un mensaje cifrado.

A continuación, la función '*descifrar\_cesar*' implementa el **algoritmo de descifrado César**, que desplaza cada letra del alfabeto un número determinado de posiciones. Posteriormente, la función '*calcular\_puntuacion\_frecuencia*' **evalúa la probabilidad** de que un texto descifrado sea legible comparando la frecuencia de las letras con las frecuencias esperadas en español (*otorgada por la IA de Claude*). La función '*descifrar\_mensaje\_completo*' utiliza las funciones anteriores para **probar todos los posibles desplazamientos** (0-25) del cifrado César, *puntuando cada resultado y presentando el más probable*.

En la ejecución principal del script, se solicita al usuario que ingrese el nombre de un archivo *.pcap*. El programa entonces lee este archivo, **extrae el mensaje cifrado de los paquetes ICMP, y procede a descifrar el mensaje utilizando el método descrito**. Finalmente, muestra todas las posibles versiones descifradas del mensaje, donde finalmente se resalta en color **verde** la opción más probable según el análisis de frecuencia de letras, y luego es imprimido en la consola cual fue el desplazamiento más probable.

### 3.3.2. Ejecución del código

```
# lordsamed1 @ 15amed1 in ~/Desktop/myfolder/Universidad/8mo/act2 view [16:57:27]
$ sudo python3 act3.py
WARNING: No IPv4 address found on anp10 !
WARNING: No IPv4 address found on anp11 !
WARNING: more No IPv4 address found on en3 !

-----
Total de paquetes en el archivo: 14
-----

Analizando paquete 0:
IP src: 192.168.100.91, dst: 8.8.8.8
ICMP type: 8 'IC
Raw payload: b'n'
Caracter extraído: n

Analizando paquete 1:
IP src: 192.168.100.91, dst: 8.8.8.8
ICMP type: 8
Raw payload: b'l'
Caracter extraído: l

Analizando paquete 2:
IP src: 192.168.100.91, dst: 8.8.8.8
ICMP type: 8
Raw payload: b's'
Caracter extraído: s

Analizando paquete 3:
IP src: 192.168.100.91, dst: 8.8.8.8
ICMP type: 8
Raw payload: b'f'
Caracter extraído: f

Analizando paquete 4:
IP src: 192.168.100.91, dst: 8.8.8.8
ICMP type: 8
Raw payload: b'h'
Caracter extraído: h

Analizando paquete 5:
IP src: 192.168.100.91, dst: 8.8.8.8
ICMP type: 8
Raw payload: b'n'
Caracter extraído: n

Analizando paquete 6:
IP src: 192.168.100.91, dst: 8.8.8.8
ICMP type: 8
Raw payload: b't'
Caracter extraído: t

Analizando paquete 7:
IP src: 192.168.100.91, dst: 8.8.8.8
ICMP type: 8
Raw payload: b' '
Caracter extraído:

Analizando paquete 8:
IP src: 192.168.100.91, dst: 8.8.8.8
ICMP type: 8
Raw payload: b'r'
Caracter extraído: r

Analizando paquete 9:
IP src: 192.168.100.91, dst: 8.8.8.8
ICMP type: 8
Raw payload: b'j'
Caracter extraído: j

Analizando paquete 10:
IP src: 192.168.100.91, dst: 8.8.8.8
ICMP type: 8
Raw payload: b'i'
Caracter extraído: i

Analizando paquete 11:
IP src: 192.168.100.91, dst: 8.8.8.8
ICMP type: 8
Raw payload: b'n'
Caracter extraído: n
```

Figura 11: Ejecución parte 1 del código act3.py.

La Figura 11 detalla el examen de los primeros 11 paquetes de un total de 14 encontrados en el archivo *.pcap*. Para cada paquete, se proporciona información crucial: la dirección IP de origen (*consistentemente 192.168.100.91*), la dirección IP de destino (*8.8.8.8 en todos los casos*), el tipo de mensaje ICMP (*tipo 8, correspondiente a echo request*), la carga útil raw, y el carácter extraído de esta carga. Esta metódica presentación permite observar el patrón emergente del mensaje oculto.

```
Analizando paquete 12:
  IP src: 192.168.100.91, dst: 8.8.8.8
  ICMP type: 8
  Raw payload: b's'
  Caracter extraído: s

Analizando paquete 13:
  IP src: 192.168.100.91, dst: 8.8.8.8
  ICMP type: 8
  Raw payload: b'f'
  Caracter extraído: f

-----
Mensaje cifrado extraído: nlsfhnt rjnsf
-----

Opciones de descifrado:
Desplazamiento 0: nlsfhnt rjnsf
Desplazamiento 1: mkregms qihmre
Desplazamiento 2: ljqdflr phglqd
Desplazamiento 3: kipceqk ogfkpc
Desplazamiento 4: jhobdjp nfejob
Desplazamiento 5: ignacio medina
Desplazamiento 6: hfmzbhn ldchmz
Desplazamiento 7: gelyagm kcbgly
Desplazamiento 8: fdkxzfj jbafox
Desplazamiento 9: ecjwyek iazejw
Desplazamiento 10: dbivxdj hzydiv
Desplazamiento 11: cahuwei gyxchu
Desplazamiento 12: bzgtvbh fxwbgt
Desplazamiento 13: ayfsuag ewvafs
Desplazamiento 14: zxertzf dvuzer
Desplazamiento 15: ywdqsyw cutydw
Desplazamiento 16: xvcprxd btsxcp
Desplazamiento 17: wubqwcw asrwbo
Desplazamiento 18: vtanpwb zrqvan
Desplazamiento 19: uszmoua yqpuzm
Desplazamiento 20: trylntz xpotyl
Desplazamiento 21: sqxkmsy wonsxx
Desplazamiento 22: rpwjlxw vnmrwj
Desplazamiento 23: qovikqw umlqvi
Desplazamiento 24: pnuhjpv tlkpuh
Desplazamiento 25: omtgiou skjotg

Desplazamiento más probable: 5
(venv) (base)
# lordsamedi @ iSamedi in ~/Desktop/myFolder/Universidad/8mo/act2 venv [17:02:13]
$
```

Figura 12: Ejecución parte 2 del código *act3.py*.

Por su parte, la Figura 12 completa el análisis con los paquetes restantes y revela el mensaje cifrado completo: **'nlsfhnt rtjnsf'**. Lo más significativo de esta segunda figura es la presentación de las 26 posibles decodificaciones utilizando el cifrado César, una técnica clásica de criptografía. Mediante un análisis de frecuencia de letras, el script identifica el desplazamiento más probable como **5**, lo que sugiere que el mensaje original oculto es **"ignacio medina"**.

En conjunto, estas imágenes ilustran la eficacia del script en la extracción y descifrado de información oculta en tráfico de red, proporcionando una visión detallada y secuencial del proceso, desde la captura inicial de paquetes hasta la revelación final del mensaje secreto.

## Conclusiones y comentarios

Para concluir este laboratorio, sobre las actividades realizadas, podemos destacar que se logró una exitosa implementación del código César en Python, permitiendo al usuario ingresar el mensaje y el desplazamiento deseado. Esto demuestra una comprensión básica de los principios de criptografía. Luego se le añadió un sistema que envía mensajes cifrados a través de paquetes ICMP, imitando el tráfico normal de ping. Esta técnica muestra cómo se puede ocultar información en protocolos comunes de red, lo que podría evadir sistemas de detección de intrusiones superficiales. Por otro lado, se desarrolló un programa capaz de extraer y descifrar mensajes ocultos en archivos de captura de paquetes (`\textit{.pcap}`). Este programa utiliza análisis de frecuencia para identificar el desplazamiento más probable del cifrado César, lo que ilustra técnicas básicas de criptoanálisis.

Cabe destacar la experiencia con IA en este proyecto, específicamente el uso de Claude, que resalta tanto el potencial como las limitaciones actuales de las herramientas de IA en el desarrollo de software. Mientras que la IA proporcionó una base valiosa para el código, la necesidad de ajustes manuales subraya la importancia continua del conocimiento humano en programación y en cualquier ámbito. Paralelamente, el proyecto reveló la facilidad con la que se puede ocultar y transmitir información sensible a través de tráfico de red aparentemente normal.

Como anotación, en la Figura 10 se logra observar los problemas que se presentaron a la hora de realizar esta actividad, dejando en evidencia uno de los problemas al desarrollar este código a la hora de determinar el desplazamiento más probable. Dada la limitación de tokens de Claude, se tuvieron que hacer consultas en tres cuentas diferentes, ya que expiraban mostrando el siguiente mensaje: *"You are out of free messages until 12 AM"*. Por otro lado, cabe destacar que las consultas se hicieron en más de un chat por cuenta, ya que cada chat con la IA está reducido a una cierta cantidad de respuestas.

En resumen, estas actividades proporcionaron una valiosa experiencia práctica en criptografía básica, análisis de red y programación en Python, mientras se exploraban conceptos importantes de seguridad informática, demostrando cómo las técnicas simples de cifrado pueden ser tanto implementadas como vulneradas.

Para terminar, como comentario, adjunto el link del repositorio de **GitHub** del presente laboratorio.

- <https://github.com/i-samedi/Lab-Cripto/tree/main/Laboratorio1>
- <https://github.com/i-samedi/Lab-Cripto> (*Repositorio general del semestre*)