

# Informe Laboratorio 1

## Sección 1

Isidora Bravo Ortiz  
e-mail: isidora.bravo2@mail.udp.cl

Agosto de 2025

## 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
3.2. Actividad 2 . . . . .	5
3.3. Actividad 3 . . . . .	9

## 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.

```

$ 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.

```

$ 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.

```

0 larycxpajorj h bnpdarmjm nw anmnb
1 kzqxbwozinqi g amoczqlil mv zmlma
2 jypwavyhmpfh f zlnbypkhk lu ylk lz
3 ixovzumxglog e ykmaxojgj kt xkjky
4 hwnuytlwfknd d xjlzwnifi js wji jx
5 gvmtxskvejme c wikyvmeheh ir vihiw
6 fulswrjudild b vhxulgdg hq uhghv
7 etkrvqitchkc a ugiwtkfcef gp tgfgu
8 dsjquphsbgjb z tfhvsjebe fo sfeft
9 criptografia y seguridad en redes
10 bqhosnfqzehz x rdftqhczc dm qdcdr
11 apgnrmepdygy w qcespgbyb cl pcbcq
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 vkbmhzktybt r lxznkbwtw xg kxw xl
17 ujahlgysxas q kwymjavsv wf jwv wk
18 tizgkfxirwzr p jvxlizuru ve ivuvj
19 shyfjewhqvyq o iuwkhytqt ud hutui
20 rgxeidvgpuxp n htvjgxspstc gtsth
21 qfwdhcufotwo m gsuifwrorsb fsrsg
22 pevcbtensvn l frthevqnq ra erqrf
23 odubfasdmrum k eqsgdupmp qz dqpqe
24 nctaezrclqtl j dprfctolopy cpopd
25 mbszdyqbkski coqebnskn ox bonoc

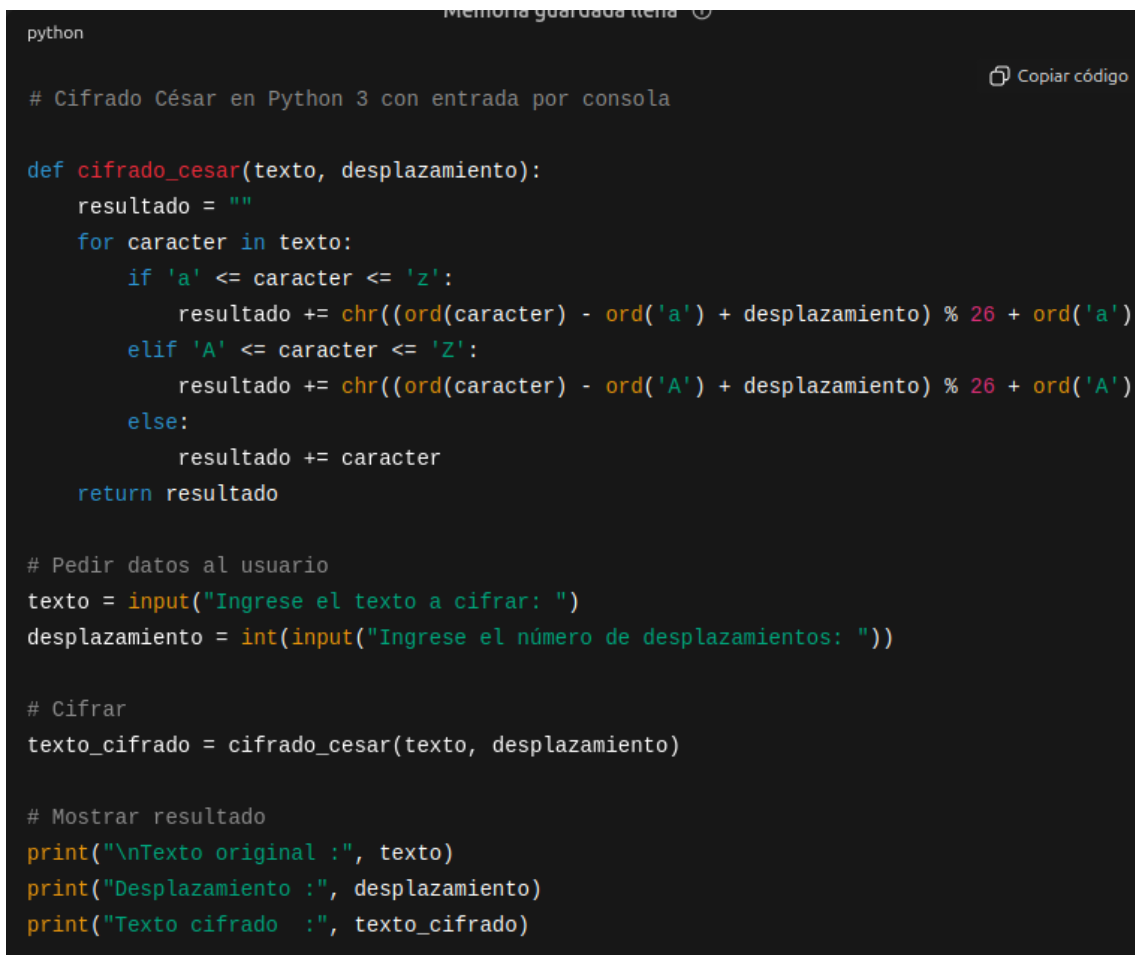
```

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 comenzar con la actividad 1, se le pidió a ChatGPT que generara un programa en python3 que recibiera el texto "criptografía y seguridad en redes" y el número 9 que indica los corrimientos a usar en el cifrado César, obteniendo el siguiente código:

A screenshot of a code editor with a dark background. The code is written in Python and implements a Caesar cipher. It includes a function definition, user input prompts, and print statements to show the original text, the shift value, and the encrypted result. The code is color-coded with syntax highlighting. At the top right, there is a button labeled 'Copiar código'.

```
python

# Cifrado César en Python 3 con entrada por consola

def cifrado_cesar(texto, desplazamiento):
    resultado = ""
    for caracter in texto:
        if 'a' <= caracter <= 'z':
            resultado += chr((ord(caracter) - ord('a') + desplazamiento) % 26 + ord('a'))
        elif 'A' <= caracter <= 'Z':
            resultado += chr((ord(caracter) - ord('A') + desplazamiento) % 26 + ord('A'))
        else:
            resultado += caracter
    return resultado

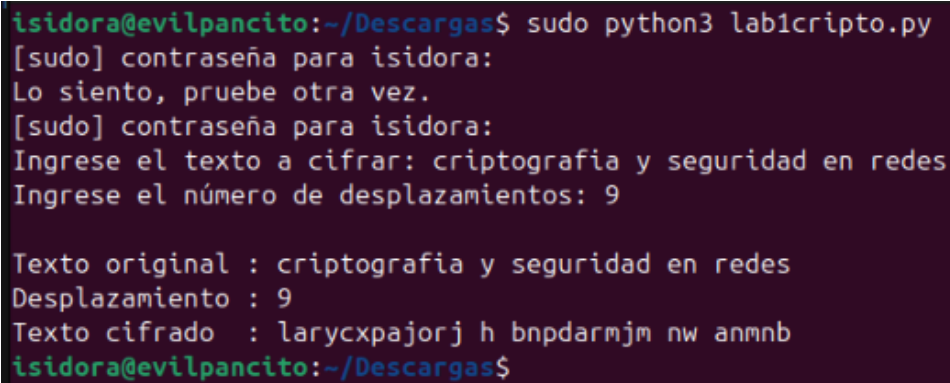
# Pedir datos al usuario
texto = input("Ingrese el texto a cifrar: ")
desplazamiento = int(input("Ingrese el número de desplazamientos: "))

# Cifrar
texto_cifrado = cifrado_cesar(texto, desplazamiento)

# Mostrar resultado
print("\nTexto original :", texto)
print("Desplazamiento :", desplazamiento)
print("Texto cifrado :", texto_cifrado)
```

Figura 1: Código obtenido para realizar cifrado César

Al usar el código generado y con la ayuda de la terminal, podemos ver que el mensaje se cifra de forma correcta.



```
isidora@evilpancito:~/Descargas$ sudo python3 lab1cripto.py
[sudo] contraseña para isidora:
Lo siento, pruebe otra vez.
[sudo] contraseña para isidora:
Ingrese el texto a cifrar: criptografia y seguridad en redes
Ingrese el número de desplazamientos: 9

Texto original : criptografia y seguridad en redes
Desplazamiento : 9
Texto cifrado  : larycxpajorj h bnpdarmjm nw anmnb
isidora@evilpancito:~/Descargas$
```

Figura 2: Mensaje cifrado exitosamente

### 3.2. Actividad 2

Para comenzar la actividad 2, se le pidió nuevamente a ChatGTP que generara un código en python3, esta vez el código generado debe enviar los caracteres generados en la actividad anterior como paquetes icmp request, como respuesta se obtuvo el siguiente código:

```
python
Memoria guardada llena ⓘ

from scapy.all import IP, ICMP, send

# --- Función Cifrado César ---
def cifrado_cesar(texto, desplazamiento):
    resultado = ""
    for caracter in texto:
        if 'a' <= caracter <= 'z':
            resultado += chr((ord(caracter) - ord('a') + desplazamiento) % 26 + ord('a'))
        elif 'A' <= caracter <= 'Z':
            resultado += chr((ord(caracter) - ord('A') + desplazamiento) % 26 + ord('A'))
        else:
            resultado += caracter
    return resultado

# --- Entrada del usuario ---
texto = input("Ingrese el texto a cifrar: ")
desplazamiento = int(input("Ingrese el número de desplazamientos: "))
destino = input("Ingrese la IP de destino: ")

# --- Cifrar el texto ---
texto_cifrado = cifrado_cesar(texto, desplazamiento)
print(f"\nTexto cifrado: {texto_cifrado}")

# --- Enviar un carácter por paquete ICMP ---
for caracter in texto_cifrado:
    paquete = IP(dst=destino) / ICMP(type=8) / caracter
    send(paquete, verbose=False)
    print(f"Enviado carácter '{caracter}' a {destino}")

print("\nTodos los caracteres fueron enviados.")
```

Figura 3: Código generado para enviar paquetes ICMP request

Para realizar esta actividad de forma correcta, previamente se hizo un ping a google.cl como se puede ver a continuación:

```

isidora@evilpancito:~/Descargas$ ping google.cl
PING google.cl (142.250.0.94) 56(84) bytes of data.
64 bytes from cg-in-f94.1e100.net (142.250.0.94): icmp_seq=1 ttl=255 time=29.3 m
s
64 bytes from cg-in-f94.1e100.net (142.250.0.94): icmp_seq=2 ttl=255 time=15.0 m
s
64 bytes from cg-in-f94.1e100.net (142.250.0.94): icmp_seq=3 ttl=255 time=12.3 m
s
64 bytes from cg-in-f94.1e100.net (142.250.0.94): icmp_seq=4 ttl=255 time=12.9 m
s

```

Figura 4: Ping previo a envío de paquetes ICMP en terminal

No.	Time	Source	Destination	Protocol	Length	Info
5	2.064696245	10.0.2.15	142.250.0.94	ICMP	100	Echo (ping) request id=0x20d9, seq=118/30208, ttl=64 (reply in 6)
6	2.022729067	142.250.0.94	10.0.2.15	ICMP	100	Echo (ping) reply id=0x20d9, seq=118/30208, ttl=255 (request in 5)
7	3.006870114	10.0.2.15	142.250.0.94	ICMP	100	Echo (ping) request id=0x20d9, seq=119/30464, ttl=64 (reply in 8)
8	3.022245687	142.250.0.94	10.0.2.15	ICMP	100	Echo (ping) reply id=0x20d9, seq=119/30464, ttl=255 (request in 7)
9	4.008471773	10.0.2.15	142.250.0.94	ICMP	100	Echo (ping) request id=0x20d9, seq=120/30720, ttl=64 (reply in 10)
10	4.024142583	142.250.0.94	10.0.2.15	ICMP	100	Echo (ping) reply id=0x20d9, seq=120/30720, ttl=255 (request in 9)
11	5.010753296	10.0.2.15	142.250.0.94	ICMP	100	Echo (ping) request id=0x20d9, seq=121/30976, ttl=64 (reply in 12)
12	5.026524090	142.250.0.94	10.0.2.15	ICMP	100	Echo (ping) reply id=0x20d9, seq=121/30976, ttl=255 (request in 11)
13	6.011709028	10.0.2.15	142.250.0.94	ICMP	100	Echo (ping) request id=0x20d9, seq=122/31232, ttl=64 (reply in 14)
14	6.028785362	142.250.0.94	10.0.2.15	ICMP	100	Echo (ping) reply id=0x20d9, seq=122/31232, ttl=255 (request in 13)
15	7.013544566	10.0.2.15	142.250.0.94	ICMP	100	Echo (ping) request id=0x20d9, seq=123/31488, ttl=64 (reply in 16)
16	7.028919188	142.250.0.94	10.0.2.15	ICMP	100	Echo (ping) reply id=0x20d9, seq=123/31488, ttl=255 (request in 15)
17	8.022158946	10.0.2.15	142.250.0.94	ICMP	100	Echo (ping) request id=0x20d9, seq=124/31744, ttl=64 (reply in 18)
18	8.035857580	142.250.0.94	10.0.2.15	ICMP	100	Echo (ping) reply id=0x20d9, seq=124/31744, ttl=255 (request in 17)
19	9.023598355	10.0.2.15	142.250.0.94	ICMP	100	Echo (ping) request id=0x20d9, seq=125/32000, ttl=64 (reply in 20)
20	9.040081792	142.250.0.94	10.0.2.15	ICMP	100	Echo (ping) reply id=0x20d9, seq=125/32000, ttl=255 (request in 19)
21	10.025905846	10.0.2.15	142.250.0.94	ICMP	100	Echo (ping) request id=0x20d9, seq=126/32256, ttl=64 (reply in 22)
22	10.037078802	142.250.0.94	10.0.2.15	ICMP	100	Echo (ping) reply id=0x20d9, seq=126/32256, ttl=255 (request in 21)
23	11.028847182	10.0.2.15	142.250.0.94	ICMP	100	Echo (ping) request id=0x20d9, seq=127/32512, ttl=64 (reply in 24)
24	11.049404697	142.250.0.94	10.0.2.15	ICMP	100	Echo (ping) reply id=0x20d9, seq=127/32512, ttl=255 (request in 23)

<ul style="list-style-type: none"> <li>Frame 1: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface any, id 0</li> <li>Linux cooked capture v1</li> <li>Internet Protocol Version 4, Src: 10.0.2.15, Dst: 142.250.0.94</li> <li>Internet Control Message Protocol</li> </ul>	<pre> 0000  00 04 00 01 00 06 08 00 27 d5 a6 25 00 00 08 00 .....!%.... 0010  45 00 00 54 1f d2 40 00 40 01 7f 70 0a 00 02 0f E..T..@..p.... 0020  8e fa 00 5e 08 00 a4 55 20 d9 00 74 5b a7 b3 68 ...A...U...t[...h 0030  00 00 00 00 5c 7a 08 00 00 00 00 00 10 11 12 13 ....\Z:..... 0040  14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 .....!"# 0050  24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 \$%&amp;'()*+,-./0123 0060  34 35 36 37 4567 </pre>
---	--

Figura 5: Ping previo a envío de paquetes ICMP en Wireshark

Luego se ejecuta el código anterior y se envían los caracteres:

```

Texto cifrado: larycxpajorj h bnpdarmjm nw anmnb
Enviado carácter 'l' a 10.0.2.15
Enviado carácter 'a' a 10.0.2.15
Enviado carácter 'r' a 10.0.2.15
Enviado carácter 'y' a 10.0.2.15
Enviado carácter 'c' a 10.0.2.15
Enviado carácter 'x' a 10.0.2.15
Enviado carácter 'p' a 10.0.2.15
Enviado carácter 'a' a 10.0.2.15
Enviado carácter 'j' a 10.0.2.15
Enviado carácter 'o' a 10.0.2.15
Enviado carácter 'r' a 10.0.2.15
Enviado carácter 'j' a 10.0.2.15
Enviado carácter ' ' a 10.0.2.15
Enviado carácter 'h' a 10.0.2.15
Enviado carácter ' ' a 10.0.2.15
Enviado carácter 'b' a 10.0.2.15
Enviado carácter 'n' a 10.0.2.15
Enviado carácter 'p' a 10.0.2.15
Enviado carácter 'd' a 10.0.2.15
Enviado carácter 'a' a 10.0.2.15
Enviado carácter 'r' a 10.0.2.15
Enviado carácter 'm' a 10.0.2.15
Enviado carácter 'j' a 10.0.2.15
Enviado carácter 'm' a 10.0.2.15
Enviado carácter ' ' a 10.0.2.15
Enviado carácter 'n' a 10.0.2.15
Enviado carácter 'w' a 10.0.2.15
Enviado carácter ' ' a 10.0.2.15
Enviado carácter 'a' a 10.0.2.15
Enviado carácter 'n' a 10.0.2.15
Enviado carácter 'm' a 10.0.2.15
Enviado carácter 'n' a 10.0.2.15
Enviado carácter 'b' a 10.0.2.15

Todos los caracteres fueron enviados.

```

Figura 6: Envío de paquetes ICMP en terminal

No.	Time	Source	Destination	Protocol	Length	Info
13250	882.991133589	10.0.2.15	10.0.2.15	ICMP	45	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
13251	883.039925743	10.0.2.15	10.0.2.15	ICMP	45	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
13252	883.103297244	10.0.2.15	10.0.2.15	ICMP	45	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
13253	883.156983860	10.0.2.15	10.0.2.15	ICMP	45	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
13254	883.17750497	10.0.2.15	10.0.2.15	ICMP	45	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
13255	883.197360259	10.0.2.15	10.0.2.15	ICMP	45	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
13256	883.239214844	10.0.2.15	10.0.2.15	ICMP	45	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
13257	883.268185011	10.0.2.15	10.0.2.15	ICMP	45	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
13258	883.297994891	10.0.2.15	10.0.2.15	ICMP	45	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
13259	883.319691446	10.0.2.15	10.0.2.15	ICMP	45	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
13260	883.371623302	10.0.2.15	10.0.2.15	ICMP	45	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
13261	883.436211759	10.0.2.15	10.0.2.15	ICMP	45	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
13262	883.469600406	10.0.2.15	10.0.2.15	ICMP	45	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
13263	883.498325011	10.0.2.15	10.0.2.15	ICMP	45	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
13264	883.559076949	10.0.2.15	10.0.2.15	ICMP	45	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
13265	883.666813575	10.0.2.15	10.0.2.15	ICMP	45	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
13266	883.709537904	10.0.2.15	10.0.2.15	ICMP	45	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
13267	883.748885707	10.0.2.15	10.0.2.15	ICMP	45	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
13268	883.782702430	10.0.2.15	10.0.2.15	ICMP	45	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
13269	883.822723367	10.0.2.15	10.0.2.15	ICMP	45	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)

<ul style="list-style-type: none"> <li>Frame 13269: 45 bytes on wire (360 bits), 45 bytes captured (360 bits) on interface any, id 0</li> <li>Linux cooked capture v1</li> <li>Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.15</li> <li>Internet Control Message Protocol <ul style="list-style-type: none"> <li>Type: 8 (Echo (ping) request)</li> <li>Code: 0</li> <li>Checksum: 0x95ff [correct]</li> <li>[Checksum Status: Good]</li> <li>Identifier (BE): 0 (0x0000)</li> <li>Identifier (LE): 0 (0x0000)</li> <li>Sequence Number (BE): 0 (0x0000)</li> <li>Sequence Number (LE): 0 (0x0000)</li> <li>[No response seen]</li> <li>Data (1 byte)</li> </ul> </li> </ul>	<pre> 0000  00 02 03 04 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0010  45 00 00 1d 00 01 00 00 40 01 02 c2 0a 00 02 0f  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0020  0a 00 02 0f 00 00 95 ff 00 00 00 00 02 00 00  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 </pre>
--	--

Figura 7: Envío de paquetes ICMP en Wireshark

Se puede evidenciar que se logró con lo pedido, ya que el último carácter enviado se transmite como una b. Además en este tipo de paquetes se puede identificar campos relevantes en el análisis de patrones, como por ejemplo el campo timestamp.



92	58.817244490	10.0.2.15	10.0.2.15	ICMP	45 Echo (ping) request	id=0x0000,
93	58.846571776	10.0.2.15	10.0.2.15	ICMP	45 Echo (ping) request	id=0x0000,
94	58.887848765	10.0.2.15	10.0.2.15	ICMP	45 Echo (ping) request	id=0x0000,
95	58.953392802	10.0.2.15	10.0.2.15	ICMP	45 Echo (ping) request	id=0x0000,
96	58.988538963	10.0.2.15	10.0.2.15	ICMP	45 Echo (ping) request	id=0x0000,
97	59.032755412	10.0.2.15	10.0.2.15	ICMP	45 Echo (ping) request	id=0x0000,
98	59.075146665	10.0.2.15	10.0.2.15	ICMP	45 Echo (ping) request	id=0x0000,
99	59.108903968	10.0.2.15	10.0.2.15	ICMP	45 Echo (ping) request	id=0x0000,
100	59.170912410	10.0.2.15	10.0.2.15	ICMP	45 Echo (ping) request	id=0x0000,

```

Frame 84: 45 bytes on wire (360 bits), 45 bytes captured (360 bits) on interface any, id 0
  Section number: 1
    Interface id: 0 (any)
      Encapsulation type: Linux cooked-mode capture v1 (25)
        Arrival Time: Aug 30, 2025 22:12:25.395658472 -04
        UTC Arrival Time: Aug 31, 2025 02:12:25.395658472 UTC
        Epoch Arrival Time: 1756606345.395658472
        [Time shift for this packet: 0.000000000 seconds]
        [Time delta from previous captured frame: 0.052164353 seconds]
        [Time delta from previous displayed frame: 0.052164353 seconds]
        [Time since reference or first frame: 58.493411529 seconds]
        Frame Number: 84
        Frame Length: 45 bytes (360 bits)
        Capture Length: 45 bytes (360 bits)
        [Frame is marked: False]
        [Frame is ignored: False]
        [Protocols in frame: sll:ethertype:ip:icmp:data]
        [Coloring Rule Name: ICMP]
        [Coloring Rule String: icmp || icmpv6]

```

Figura 8: Identificación timestamp paquete ICMP

### 3.3. Actividad 3

En la actividad 3, se solicitó a ChatGPT un código capaz de descifrar el mensaje generado en la actividad 2, sin conocer el corrimiento utilizado. Para ello, el programa debía generar todas las combinaciones posibles e imprimirlas y resaltar en verde la opción más probable del mensaje original. EL código generado es el siguiente:



```
python                                     Memoria guardada llena ⓘ

#!/usr/bin/env python3
# recibir_cesar_icmp.py
# Lee caracteres enviados en el campo data de ICMP Echo Request (uno por paquete),
# reconstruye la cadena y prueba todas las rotaciones César (0..25).
# Resalta en verde la opción con mayor puntaje según heurística en español.

import argparse
import re
import sys

try:
    from scapy.all import rdpcap, sniff, IP, ICMP
except Exception as e:
    print("Error importando scapy. Instala con: pip install scapy")
    raise

# --- Configuración de colores ANSI ---
GREEN = "\033[92m"
YELLOW = "\033[93m"
RESET = "\033[0m"

# --- Lista corta de palabras comunes en español para puntuar candidatas ---
SPANISH_COMMON_WORDS = [
    r"\bde\b", r"\bla\b", r"\bel\b", r"\by\b", r"\ben\b",
    r"\bque\b", r"\bpara\b", r"\bcon\b", r"\bpor\b", r"\buna\b",
    r"\bres\b", r"\bredes\b", r"\bseguridad\b", r"\bcriptograf", r"\bcriptografia\b"
]

def extraer_payloads_de_pcap(path_pcap):
    """Lee un pcap y devuelve la lista de bytes (payloads) de ICMP Echo Request en orden."""
    pkts = rdpcap(path_pcap)
    payload_bytes = []
    for p in pkts:
```

Figura 9: Código generado actividad 3 parte 1

```
if p.haslayer(ICMP) and p.haslayer(IP):
    # ICMP type 8 = Echo Request
    if int(p[ICMP].type) == 8:
        # p[ICMP].payload may be Raw or something; extraemos los bytes crudos
        raw = bytes(p[ICMP].payload)
        if len(raw) > 0:
            # supondremos que cada paquete contiene exactamente 1 carácter en data
            # si hay más de 1 byte, tomamos tal cual (ej. soporta multi-byte)
            payload_bytes.append(raw)
    return payload_bytes

def extraer_payloads_en_vivo(timeout=None, count=None, iface=None):
    """
    Captura paquetes ICMP Echo Request en vivo y devuelve la lista de bytes payload.
    WARNING: requiere ejecutar como root y permiso para sniffing.
    """
    payload_bytes = []

    def procesar(p):
        if p.haslayer(ICMP) and p.haslayer(IP):
            if int(p[ICMP].type) == 8:
                raw = bytes(p[ICMP].payload)
                if len(raw) > 0:
                    payload_bytes.append(raw)

    # filtro BPF para ICMP echo requests
    bpf = "icmp and icmp[icmptype] == 8"
    sniff(filter=bpf, prn=procesar, timeout=timeout, count=count, iface=iface)
    return payload_bytes

def reconstruct_from_payloads(payload_bytes):
    """Concatena bytes en orden: intentaremos decodificar como latin-1 para mapear 1:1 byte"""
    parts = []
    for b in payload_bytes:
        # b puede ser secuencia de bytes; convertimos conservando valores (latin-1)
```

Figura 10: Código generado actividad 3 parte 2

```

    try:
        s = b.decode("latin-1")
    except:
        # si falla, iterar por bytes
        s = "".join(chr(x) for x in b)
    parts.append(s)
    return "".join(parts)

def caesar_shift(text, shift):
    """Aplica shift positivo (desplazar hacia la derecha). Para descifrar, usar shift negativo"""
    res = []
    for ch in text:
        if 'a' <= ch <= 'z':
            res.append(chr((ord(ch) - ord('a') + shift) % 26 + ord('a')))
        elif 'A' <= ch <= 'Z':
            res.append(chr((ord(ch) - ord('A') + shift) % 26 + ord('A')))
        else:
            res.append(ch)
    return "".join(res)

def score_spanish_candidate(candidate):
    """Cuenta coincidencias de palabras comunes (mayor = mejor). Añade pequeño bonus por \
    score = 0
    lc = candidate.lower()
    for pat in SPANISH_COMMON_WORDS:
        matches = re.findall(pat, lc)
        score += len(matches) * 10 # cada palabra común suma 10
    # bonus por porcentaje de vocales típico en español
    vowels = len(re.findall(r"[aeiouâêíóúü]", lc))
    total_letters = len(re.findall(r"[a-zAÉÍÓÚÜÑ]", lc))
    if total_letters > 0:
        vowel_ratio = vowels / total_letters
        # ideal ratio (heurístico) ~0.4 -> acercarse suma más
        score += max(0, (0.4 - abs(0.4 - vowel_ratio)) * 5)
    return score

```

Figura 11: Código generado actividad 3 parte 3

```
def generar_y_evaluar(cadena):
    """Genera todas las 26 rotaciones posibles (descifrado) y las puntúa."""
    candidates = []
    for shift in range(26):
        # Para probar una posible clave k (corrimiento usado en cifrado),
        # si texto fue cifrado con desplazamiento k (to right), entonces
        # para recuperar el original aplicamos desplazamiento -k.
        # Aquí definimos shift_decrypt = -k, pero al recorrer k 0..25
        # podemos calcular candidate = caesar_shift(cadena, -k)
        candidate = caesar_shift(cadena, -shift)
        s = score_spanish_candidate(candidate)
        candidates.append((shift, candidate, s))

    # ordenar por puntaje descendente
    candidates_sorted = sorted(candidates, key=lambda x: x[2], reverse=True)
    return candidates_sorted

def print_results(candidates_sorted):
    best_score = candidates_sorted[0][2]
    # puede haber empate; resaltamos todos los que empatan con la mejor puntuación
    bests = [c for c in candidates_sorted if abs(c[2] - best_score) < 1e-6]
    best_shifts = [c[0] for c in bests]

    print("\nTodas las opciones (shift usado en cifrado -> texto resultante):\n")
    for shift, text, score in candidates_sorted:
        prefix = f"[k={shift:2d}] "
        if shift in best_shifts:
            print(f"{GREEN}{prefix}{text}    (score={score:.2f}){RESET}")
        else:
            print(f"{prefix}{text}    (score={score:.2f})")

    print("\n" + "-"*60)
    print(f"{YELLOW}Opción(es) más probable(s) (resaltada(s) en verde). k considerado como")
    print("-"*60)
    for shift, text, score in bests:
        print(f"{GREEN}k={shift} -> {text} (score={score:.2f}){RESET}")

def main():
```

Figura 12: Código generado actividad 3 parte 4

```
def main():
    parser = argparse.ArgumentParser(description="Recuperar mensaje enviado carácter-por-í
    group = parser.add_mutually_exclusive_group(required=True)
    group.add_argument("--pcap", "-p", help="Archivo pcap que contiene los paquetes ICMP í
    group.add_argument("--live", "-l", action="store_true", help="Capturar paquetes ICMP í
    parser.add_argument("--timeout", "-t", type=int, default=10, help="Timeout para captu
    parser.add_argument("--count", "-c", type=int, default=0, help="Número de paquetes a í
    parser.add_argument("--iface", help="Interfaz para captura en vivo (opcional).")
    args = parser.parse_args()

    if args.pcap:
        print(f"Leyendo pcap: {args.pcap} ...")
        payloads = extraer_payloads_de_pcap(args.pcap)
        if not payloads:
            print("No se encontraron payloads ICMP Echo Request con datos en el pcap.")
            sys.exit(1)
    else:
        print("Capturando en vivo (Ctrl-C para detener si count=0). Requiere sudo/root.")
        payloads = extraer_payloads_en_vivo(timeout=args.timeout, count=(args.count or No
        if not payloads:
            print("No se capturaron paquetes ICMP Echo Request con payload antes del time
            sys.exit(1)

    mensaje = reconstruct_from_payloads(payloads)
    print("\nMensaje reconstruido (raw):")
    print(repr(mensaje))
    print("\nProbando todas las rotaciones César (0..25)...")

    candidates_sorted = generar_y_evaluar(mensaje)
    print_results(candidates_sorted)

if __name__ == "__main__":
    main()
```

Figura 13: Código generado actividad 3 parte 5

Posteriormente, se compiló y ejecutó el código obteniendo las diferentes combinaciones y la opción más probable destacada en verde como se puede ver a continuación:

```

Leyendo pcap: capturalab1.pcapng ...

Mensaje reconstruido (raw):
'larycxpajorj h bnpdarmjm nw anmnb'

Probando todas las rotaciones César (0..25)...

Todas las opciones (shift usado en cifrado -> texto resultante):

[k= 9] criptografia y seguridad en redes (score=51.93)
[k=13] ynelpkcnwbew u oacqnezvz aj nazao (score=1.72)
[k=18] tizgkfxirwzr p jvxlizuru ve ivuvj (score=1.38)
[k=19] shyfjewhqvyq o iuwkhytqt ud hutui (score=1.38)
[k= 1] kzqxbwozinqi g amoczqlil mv zmlma (score=1.21)
[k= 3] ixovzunxglog e ykmaxojgj kt xkjky (score=1.21)
[k= 5] gvmtxskvejme c wikyvmheh ir vihiw (score=1.21)
[k=12] zofmqldoxcfx v pbdrofaxa bk obabp (score=1.21)
[k=15] wlcjniauzcu s myaolcxux yh lyxym (score=1.21)
[k=23] odubfasdmrum k eqsgdupmp qz dqpqe (score=1.21)
[k= 7] etkrvqitchkc a ugiwtkfcf gp tgfgu (score=1.03)
[k=21] qfwdhucufotwo m gsuifwror sb fsrsg (score=1.03)
[k=25] mbszdyqbksk i coqebnskn ox bonoc (score=1.03)
[k= 0] larycxpajorj h bnpdarmjm nw anmnb (score=0.86)
[k= 6] fulswrjudild b vhjxulgdg hq uhghv (score=0.86)
[k= 8] dsjquphsbgjb z tfhvsjebe fo sfefr (score=0.86)
[k=22] pevcbgtensvn l frthevqng ra erqrf (score=0.86)
[k=24] nctaezrclatl j dprfctolo py cpopd (score=0.86)
[k= 4] hwnuytlwfkf d xjlzwnifi js wjijx (score=0.69)
[k=17] ujahlgysxas q kwymjavsv wf jwvwx (score=0.69)
[k=11] apgnrmepdygy w qcespgbyb cl pcbcq (score=0.52)
[k=14] xmdkojbmadv t nzbpmdyvy zi mzyzn (score=0.52)
[k=20] rgxeidvgpuxp n htvjgxspz tc gtsth (score=0.52)
[k= 2] jypwavyhmpf f zlnbyphkh lu ylkz (score=0.34)
[k=10] bqhosnfqzehz x rdfthqhczc dm qdcdr (score=0.34)
[k=16] vkbinhzktybt r lxznkbtw xg kxwxl (score=0.17)

```

Figura 14: Combinaciones generadas y opción más probable destacada en verde

Para finalizar, se indicarán 4 issues que se tuvieron al lidiar con ChatGPT en la realización de este laboratorio:

- Al inicio, ChatGPT asumió que la captura de red era un archivo `.pcap` estándar, pero el archivo disponible estaba en formato `.pcapng`, lo que generó errores de lectura y obligó a buscar soluciones como convertir el archivo o actualizar Scapy.
- El código sugerido presentó errores debido a que el archivo de captura no se encontraba en la carpeta de ejecución, y ChatGPT no indicó de forma explícita la necesidad de usar la ruta completa del archivo.
- ChatGPT recomendó el uso de la librería Scapy, pero no advirtió desde el principio que algunas versiones preinstaladas en Linux no ofrecen soporte completo para el formato `.pcapng`, lo que obligó a considerar actualizar la librería o convertir el archivo.
- El primer script entregado era más extenso y con más funciones de las necesarias, lo que dificultó la prueba inicial. Fue necesario solicitar ajustes para que se adaptara mejor al caso de uso, como el ingreso de datos por consola y el procesamiento del formato correcto.

## Conclusiones y comentarios

Históricamente, la humanidad ha buscado formas de transmitir mensajes sin ser interceptados y, en caso de serlo, que el contenido no pudiera ser descifrado. En ese contexto, durante las campañas militares del Imperio Romano se utilizó el cifrado César, una técnica simple de sustitución que permitía desplazar letras del alfabeto. De esta manera, aunque los enemigos lograran interceptar los mensajes, no podían comprenderlos con facilidad.

Hoy en día, este método es ampliamente conocido y resulta trivial: es sencillo detectar cuándo se está frente a él y aún más fácil revertirlo aplicando el proceso inverso. Por ello, en la actualidad se emplean algoritmos de cifrado mucho más robustos, cuya inversión no solo requiere un conocimiento especializado, sino también un costo computacional altísimo. Con la capacidad de cómputo actual, descifrar algunos de estos algoritmos podría tomar miles o incluso millones de años.

Del mismo modo, comprender estas técnicas clásicas permite extrapolarlas a contextos modernos. Tal como se experimentó en este laboratorio, un protocolo de red como ICMP puede convertirse en un canal para el envío encubierto de información. En este caso, el mensaje, la red y los usuarios asumen los roles de remitente, canal y destinatario, respectivamente, demostrando cómo incluso mecanismos simples pueden servir para evadir detección si no se aplican medidas de seguridad avanzadas.

Finalmente, los desafíos actuales en ciberseguridad ya no se centran en técnicas de sustitución elementales, sino en la capacidad de detectar flujos anómalos de datos en medio de un tráfico masivo, en el diseño de algoritmos cuántico-resistentes y en el balance entre privacidad y seguridad. Este laboratorio, al rescatar un enfoque histórico y aplicarlo en un escenario práctico, no solo fortaleció la comprensión de los fundamentos de la criptografía, sino que también permitió reflexionar sobre la vigencia de estas problemáticas y la necesidad de mantenerse siempre un paso adelante en el ámbito de la seguridad informática.