

HWK5 – Cifrado César sobre TCP con ESP32 y Linux

Microprocesadores

Jaime Emmanuel Valenzuela Valdivia
Carlos Ignacio Elias Contreras

16 de octubre de 2025

1. Objetivo

Implementar un flujo cliente-servidor por TCP que envía un mensaje cifrado con César (rotación de letras y dígitos), validarlo con Wireshark y un sniffer propio en Python, usando:

- ESP32 como cliente y servidor (según el paso).
- Binarios cliente/servidor en Linux.

2. Descripción del protocolo

El cliente envía un buffer con el siguiente formato:

[1 byte shift][cadena_cifrada...]

donde **shift** es el desplazamiento César. El servidor invierte la rotación para letras (módulo 26) y dígitos (módulo 10) y reconstruye el texto llano.

3. Entorno y comandos

Los comandos usados para compilar/ejecutar cada escenario fueron:

Listing 1: Resumen de comandos usados (ESP32 y Linux).

```
1 # -- ESP32 cliente / LINUX servidor --
2 . $HOME/esp/esp-idf/export.sh
3 cd ~/Documentos/CifradoCesar/esp/client
4 idf.py set-target esp32
5 idf.py menuconfig
6 idf.py build
7 idf.py -p /dev/ttyUSB0 flash monitor
8
9 # -- LINUX cliente / ESP32 servidor --
10 . $HOME/esp/esp-idf/export.sh
11 cd ~/Documentos/CifradoCesar/esp/server
12 idf.py set-target esp32
13 idf.py build
14 idf.py -p /dev/ttyUSB0 flash monitor
15
16 # -- Cliente Linux contra ESP (reemplazar IP_DEL_ESP) --
17 cd ~/Documentos/CifradoCesar/linux/build
18 ./client <IP_DEL_ESP> 3333 4 Bren_123
19
20 # -- Wireshark --
21 tcp.port == 3333
```

```

22
23 # -- IP de la maquina --
24 hostname -I
25
26 # -- Sniffer Python (Scapy) --
27 cd ~/Documentos/CifradoCesar/python
28 sudo python3 sniff_caesar.py wlo1 3333

```

(Estos comandos están recopilados de la guía de trabajo que se siguió y los apuntes propios del laboratorio.¹)

4. Paso 1: ESP32 cliente → Linux servidor

Configuración y compilación

En Kconfig.projbuild del cliente se definieron los parámetros:

- SSID/Password de la red Wi-Fi.
- IP y puerto del servidor TCP (en Linux).
- Texto y shift del cifrado.

Se realizó `idf.py build & flash`. Al iniciar, el ESP mostró su IP y la conexión al servidor.

Evidencias

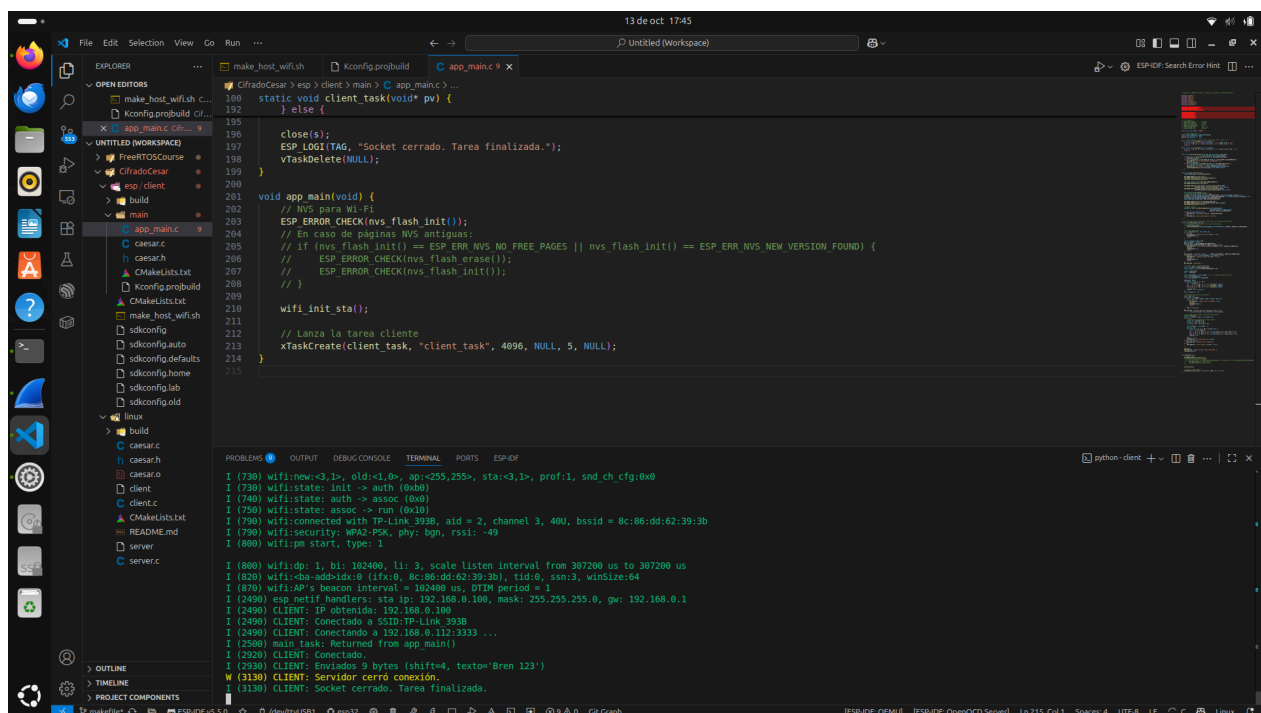


Figura 1: ESP32 cliente: conexión Wi-Fi, conexión TCP y envío de “Bren_123”/“Jaime_123”.

¹Ver lista de comandos del alumno en *ESP32 cliente / LINUX server*: [contentReference\[oaicite:0\]index=0](#)

```

jaime@JaimeMSI:~/Documentos/CifradoCesar/linux/build$ ./server 0.0.0.0 3333
[server] Escuchando en 0.0.0.0:3333 ...
[server] shift=4, descifrado="Bren 123"
jaime@JaimeMSI:~/Documentos/CifradoCesar/linux/build$ ./server 0.0.0.0 3333
[server] Escuchando en 0.0.0.0:3333 ...
[server] shift=4, descifrado="Bren 123"
jaime@JaimeMSI:~/Documentos/CifradoCesar/linux/build$ ./server 0.0.0.0 3333
[server] Escuchando en 0.0.0.0:3333 ...
[server] shift=4, descifrado="Bren 123"
jaime@JaimeMSI:~/Documentos/CifradoCesar/linux/build$

```

Figura 2: Servidor **Linux** escuchando en 0.0.0.0:3333, mostrando `shift=4` y texto descifrado.

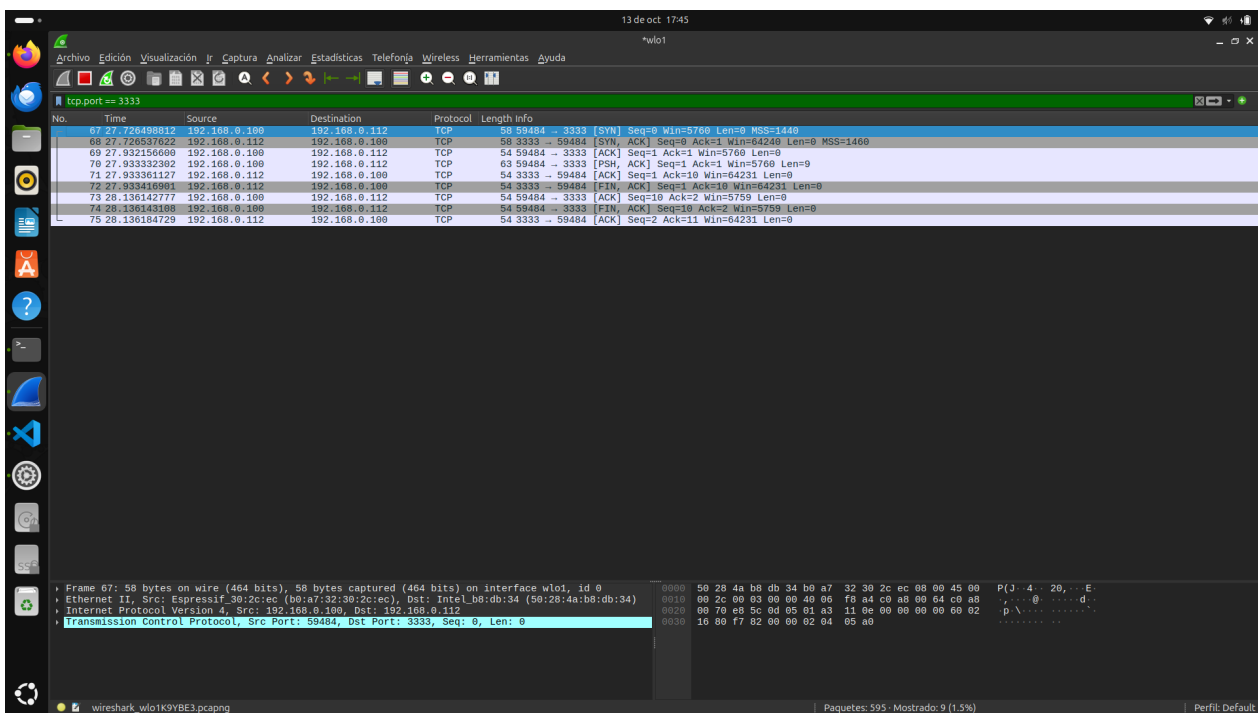


Figura 3: Captura Wireshark (`tcp.port == 3333`) para el flujo ESP→Linux.

5. Paso 2: Linux cliente → ESP32 servidor

Configuración y compilación

Se construyó el proyecto servidor para ESP32 y se ejecutó el cliente Linux:

```

1 cd ~/Documentos/CifradoCesar/linux/build
2 ./server 0.0.0.0 3333          # (para la prueba inversa con ESP cliente)
3 ./client 192.168.0.100 3333 4 Bren_123

```

Evidencias

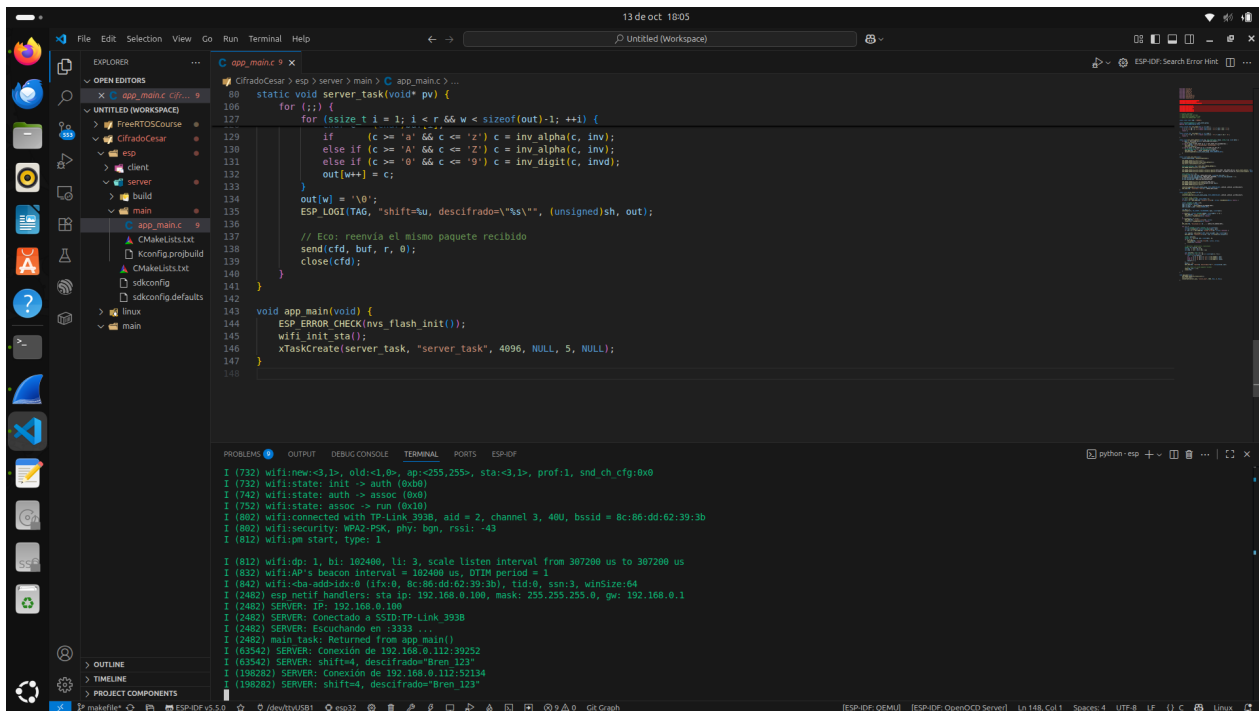


Figura 4: ESP32 servidor: IP adquirida y socket escuchando en :3333. Muestra conexiones entrantes y texto llano.

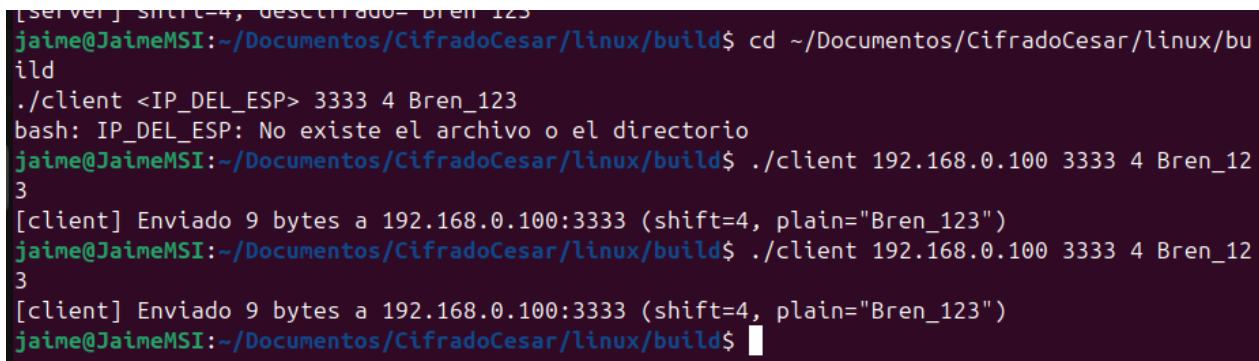


Figura 5: Cliente **Linux**: envió de 9 bytes (1 de shift + 8-9 de payload) al ESP servidor.


```

carlos-elias@Carloseliaslaptop:~/Documents/sockets/client$ ./tcp_client 10.51.89.170 3333 "La Comunicacion Funciona"

[CLIENTE] Preparando para enviar a 10.51.89.170:3333
-> Mensaje Original: 'La Comunicacion Funciona' (Shift: 5)
-> Tamaño de Carga Útil (incl. shift): 25 bytes
[CLIENTE] Conexión establecida con el Servidor ESP32.
[CLIENTE] Enviados 25 bytes cifrados.
[CLIENTE] Recibida respuesta cifrada de 46 bytes.
-> Respuesta Servidor (Descifrada): 'ACK: Mensaje recibido y descifrado por ESP32.'
[CLIENTE] Conexión cerrada.
carlos-elias@Carloseliaslaptop:~/Documents/sockets/client$ ./tcp_client 10.51.89.170 3333 "La Comunicacion Funciona"

[CLIENTE] Preparando para enviar a 10.51.89.170:3333
-> Mensaje Original: 'La Comunicacion Funciona' (Shift: 5)
-> Tamaño de Carga Útil (incl. shift): 25 bytes
[CLIENTE] Conexión establecida con el Servidor ESP32.
[CLIENTE] Enviados 25 bytes cifrados.
[CLIENTE] Recibida respuesta cifrada de 46 bytes.
-> Respuesta Servidor (Descifrada): 'ACK: Mensaje recibido y descifrado por ESP32.'
[CLIENTE] Conexión cerrada.

```

Figura 8: Cliente **Linux**: envío de 9 bytes (1 de **shift** + 8-9 de **payload**) al ESP servidor. Prueba hecha por carlos

No.	Time	Source	Destination	Protocol	Length	Info
38	19.946984650	10.51.89.92	10.51.89.170	TCP	74	60210 → 3333 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2002758582 TSecr=0 WS=128
39	20.140152691	10.51.89.170	10.51.89.92	TCP	54	3333 → 60210 [SYN, ACK] Seq=0 Ack=1 Win=5760 Len=0 MSS=1440
40	20.140242897	10.51.89.92	10.51.89.170	TCP	54	60210 → 3333 [ACK] Seq=1 Ack=1 Win=64240 Len=0
41	20.140373722	10.51.89.92	10.51.89.170	TCP	79	60210 → 3333 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=25
42	20.168409585	10.51.89.170	10.51.89.92	TCP	100	3333 → 60210 [PSH, ACK] Seq=1 Ack=26 Win=5735 Len=46
43	20.168509551	10.51.89.92	10.51.89.170	TCP	54	60210 → 3333 [ACK] Seq=26 Ack=47 Win=64194 Len=0
44	20.168722589	10.51.89.92	10.51.89.170	TCP	54	60210 → 3333 [FIN, ACK] Seq=26 Ack=47 Win=64194 Len=0
45	20.175153569	10.51.89.170	10.51.89.92	TCP	54	3333 → 60210 [FIN, ACK] Seq=47 Ack=26 Win=5735 Len=0
46	20.175221547	10.51.89.92	10.51.89.170	TCP	54	60210 → 3333 [ACK] Seq=27 Ack=48 Win=64193 Len=0
47	20.186289289	10.51.89.170	10.51.89.92	TCP	54	3333 → 60210 [ACK] Seq=48 Ack=27 Win=5734 Len=0

Frame 38: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface wlo1	0000	fc b4 67 51 26 1c 10 5b	ad 62 f1 f1 08 00 45 00	gQ&...[.b...E-
Ethernet II, Src: MegaWell_62:f1:f1 (10:5b:ad:62:f1:f1), Dst: Espressif_51:26:1c (fc:...	0010	00 3c 55 1f 40 00 40 06	1e 31 0a 33 59 5c 0a 33	<U@_@-1.3Y\3
Internet Protocol Version 4, Src: 10.51.89.92, Dst: 10.51.89.170	0020	59 aa eb 30 0d 05 aa 24	7a 77 00 00 00 a0 02	Y..@..\$zw.....
Transmission Control Protocol, Src Port: 60210, Dst Port: 3333, Seq: 0, Len: 0	0030	fa f0 45 b3 00 00 02 04	05 b4 04 02 00 0a 77 5f	..E.....w_
	0040	ab b6 00 00 00 01 03 03 07	

Figura 9: Wireshark del flujo Linux→ESP (handshake, PSH,ACK con **Len=9/10**, y cierre). Prueba hecha por carlos

6. Paso 3 : Conexion ESP - ESP

6.1. Objetivo y Configuración

El objetivo de esta fase fue establecer la comunicación cifrada y bidireccional entre dos módulos ESP32, utilizando el mismo algoritmo de Cifrado César implementado en **caesar.c** y **caesar.h**.

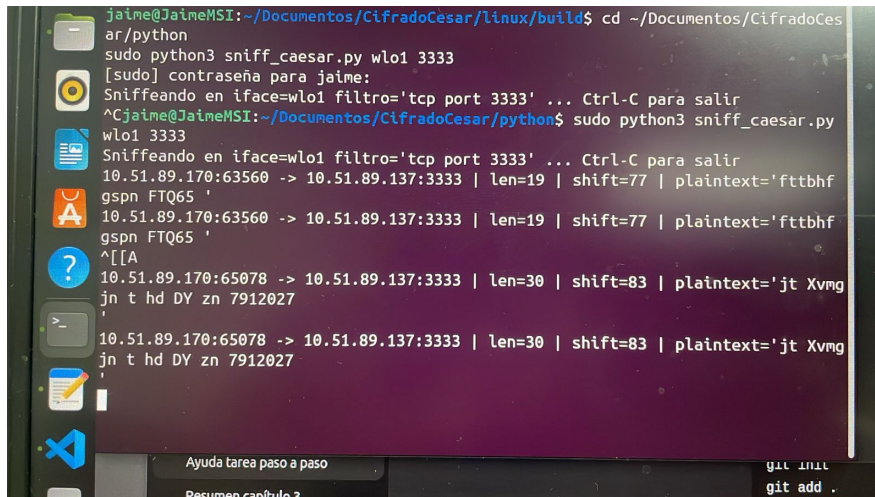
- **Placa Servidor (ESP32-1):** Flasheada con el código **esp/server** modificado para recibir, descifrar y enviar un ACK cifrado de vuelta. Obtuvo la IP 10.51.89.XXX.
- **Placa Cliente (ESP32-2):** Flasheada con el código **esp/client**, configurada para enviar el mensaje cifrado al Servidor 10.51.89.XXX:3333.

6.2. Resultado y Análisis del Fallo

La conexión TCP entre ambos módulos fue exitosa; el Cliente (Placa 2) pudo establecer el *socket* con el Servidor (Placa 1). Sin embargo, el mensaje recibido por el Servidor ESP32 no fue descifrado correctamente, produciendo caracteres ilegibles o basura (*garbage data*) en el Monitor Serial.

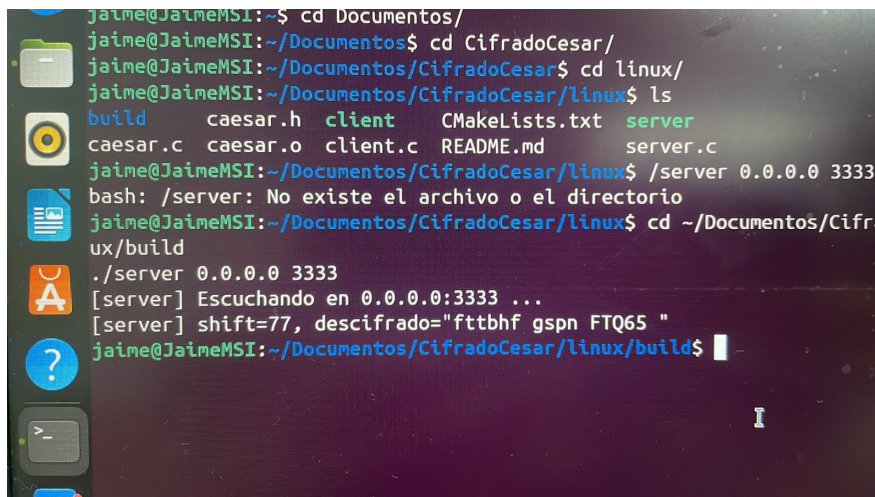
6.2.1. Fallo en el Descifrado

El análisis del error sugiere que la causa más probable es una **inconsistencia en el valor del *shift*** utilizado o leído. El Servidor ESP32 leyó un valor inesperado del **Byte 0** del *payload* recibido, lo que resultó en una clave de rotación incorrecta y, por lo tanto, en un descifrado inválido.



```
jaim@JaimeMSI:~/Documentos/CifradoCesar/linux/build$ cd ~/Documentos/CifradoCesar/
jaim@JaimeMSI:~/Documentos/CifradoCesar$ python3 sniff_caesar.py wlo1 3333
[sudo] contraseña para jaim:
Sniffing on iface=wlo1 filtro='tcp port 3333' ... Ctrl-C para salir
^Cjaim@JaimeMSI:~/Documentos/CifradoCesar/python$ sudo python3 sniff_caesar.py
wlo1 3333
Sniffing on iface=wlo1 filtro='tcp port 3333' ... Ctrl-C para salir
10.51.89.170:63560 -> 10.51.89.137:3333 | len=19 | shift=77 | plaintext='fttbhf
gspn FTQ65 '
10.51.89.170:63560 -> 10.51.89.137:3333 | len=19 | shift=77 | plaintext='fttbhf
gspn FTQ65 '
^[[A
10.51.89.170:65078 -> 10.51.89.137:3333 | len=30 | shift=83 | plaintext='jt Xvmg
jn t hd DY zn 7912027
'
10.51.89.170:65078 -> 10.51.89.137:3333 | len=30 | shift=83 | plaintext='jt Xvmg
jn t hd DY zn 7912027
'
```

Figura 10: Monitor Serial del Servidor ESP32 (Placa 1) mostrando el mensaje recibido sin descifrar intento 1.



```
jaim@JaimeMSI:~$ cd Documentos/
jaim@JaimeMSI:~/Documentos$ cd CifradoCesar/
jaim@JaimeMSI:~/Documentos/CifradoCesar$ cd linux/
jaim@JaimeMSI:~/Documentos/CifradoCesar/linux$ ls
build      caesar.h  client    CMakeLists.txt  server
caesar.c  caesar.o  client.c  README.md       server.c
jaim@JaimeMSI:~/Documentos/CifradoCesar/linux$ ./server 0.0.0.0 3333
bash: ./server: No existe el archivo o el directorio
jaim@JaimeMSI:~/Documentos/CifradoCesar/linux$ cd ~/Documentos/CifradoCesar/linux/build
jaim@JaimeMSI:~/Documentos/CifradoCesar/linux/build$ ./server 0.0.0.0 3333
[server] Escuchando en 0.0.0.0:3333 ...
[server] shift=77, descifrado="fttbhf gspn FTQ65 "
```

Figura 11: Monitor Serial del Servidor ESP32 (Placa 1) mostrando el mensaje recibido sin descifrar intento 2.

6.2.2. Evidencia del Fallo en el Cliente

Aunque el Cliente (Placa 2) pudo enviar, su posterior intento de recibir la respuesta cifrada (ACK) del Servidor (Placa 1) también falló o resultó en un mensaje ilegible, lo que confirmó el problema bidireccional en el protocolo de cifrado.

Conclusión: Debido a un conflicto en la lectura del *shift* o la codificación de caracteres entre las dos plataformas ESP-IDF, la validación del escenario ESP32 ↔ ESP32 con descifrado funcional no se pudo completar con éxito en esta iteración.

7. Sniffer en Python (Scapy)

Para validar la capa de aplicación, se implementó un sniffer que:

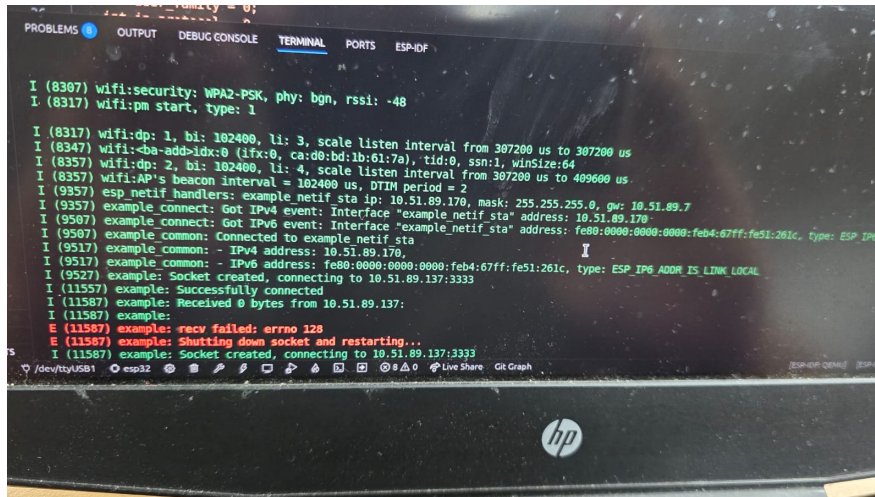


Figura 12: Monitor Serial del Cliente ESP32 (Placa 2) mostrando el intento de recepción de la respuesta del Servidor.

1. Captura TCP en el puerto 3333.
2. Interpreta el primer byte como **shift**.
3. Descifra el resto combinando rotación de letras (mód. 26) y dígitos (mód. 10).

Listing 2: sniff_caesar.py (Scapy).

```

1 from scapy.all import sniff, TCP, Raw, IP
2 import sys
3
4 # Descifrado Caesar: letras rotan 26, dígitos rotan 10
5 def dec_caesar(shift, data: bytes) -> str:
6     s = shift % 26
7     sd = (shift % 10)
8     inv = (26 - s) % 26
9     invd = (10 - sd) % 10
10    out = []
11    for b in data:
12        c = chr(b)
13        if 'a' <= c <= 'z':
14            out.append(chr(((ord(c)-97 + inv) % 26) + 97))
15        elif 'A' <= c <= 'Z':
16            out.append(chr(((ord(c)-65 + inv) % 26) + 65))
17        elif '0' <= c <= '9':
18            out.append(chr(((ord(c)-48 + invd) % 10) + 48))
19        else:
20            out.append(c)
21    return ''.join(out)
22
23 def handle(pkt):
24     if pkt.haslayer(TCP) and pkt.haslayer(Raw) and pkt.haslayer(IP):
25         payload = bytes(pkt[Raw].load)
26         if len(payload) >= 2: # esperamos [shift][cipher...]
27             shift = payload[0]
28             plain = dec_caesar(shift, payload[1:])
29             print(f"{pkt[IP].src}:{pkt[TCP].sport}->{pkt[IP].dst}:{pkt[
30                 TCP].dport}|{
                    f"len={len(payload)}|shift={shift}|plaintext='{
                        plain}'")

```



```

31
32 if __name__ == "__main__":
33     if len(sys.argv) < 2:
34         print("Uso: sudo python3 sniff_caesar.py <iface> <puerto>")
35         sys.exit(1)
36     iface = sys.argv[1]
37     port = sys.argv[2] if len(sys.argv) > 2 else "3333"
38     bpf = f"tcp port {port}"
39     print(f"Sniffeando en {iface} filtro='{bpf}' ... Ctrl-C para salir")
40     sniff(iface=iface, filter=bpf, prn=handle, store=False)

```

(Código base del sniffer según el archivo del alumno. :contentReference[oaicite:1]index=1)

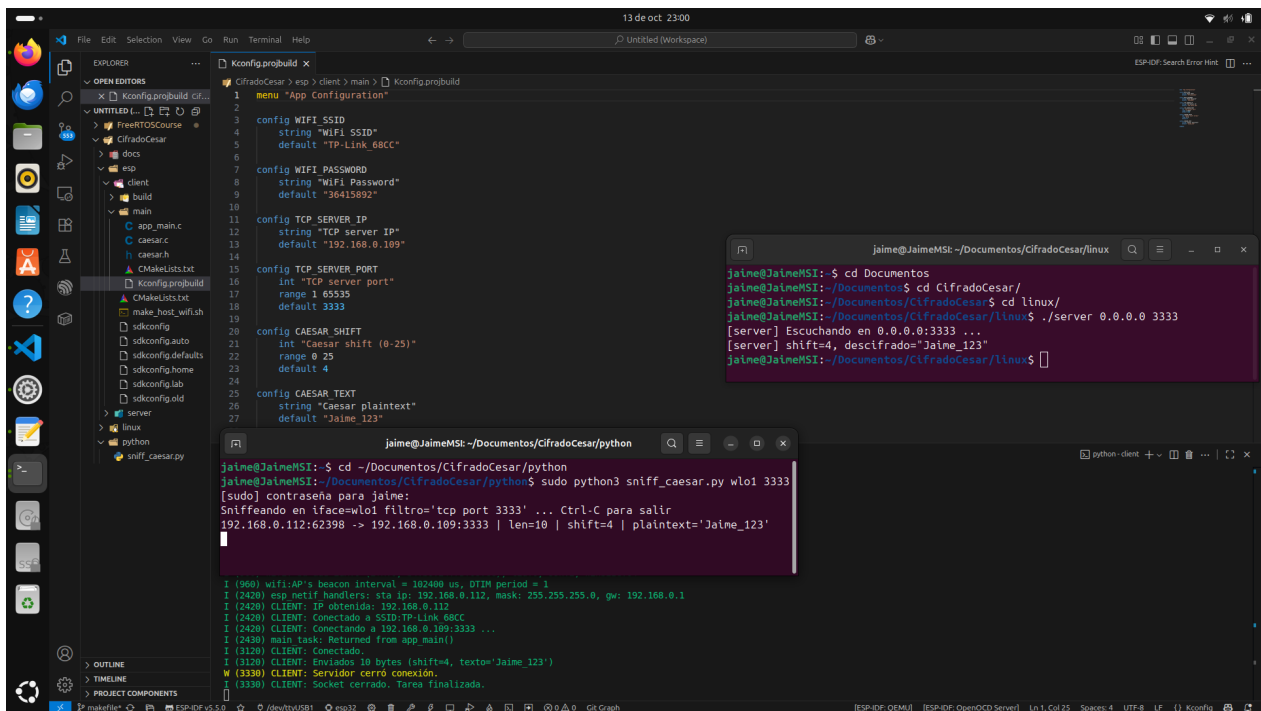


Figura 13: Sniffer Python mostrando shift=4 y plaintext='Jaime_123'.

8. Resultados

- En ambos escenarios se observa en consola del receptor el texto descifrado correctamente.
- Las capturas en Wireshark validan el 3-way handshake, el segmento de datos con Len=9/10 y el cierre de conexión.
- El sniffer externo (Scapy) reconstruye el *plaintext* a partir del payload de aplicación, confirmando el formato del mensaje.
- La conexión entre ESP a ESP se logra, sin embargo el cifrado tuvo errores, los cuales se buscaran posteriormente darle solución.

9. Conclusiones

La implementación y prueba del protocolo de comunicación cifrada César sobre TCP demostró una operación robusta en entornos heterogéneos (Linux y ESP32), validando la ingeniería de la solución en la mayoría de sus aspectos fundamentales.

1. **Éxito en Integración Cruzada:** Se estableció con éxito la comunicación bidireccional cifrada y descifrada entre el Cliente ESP32 y el Servidor Linux, y viceversa, confirmando la portabilidad del algoritmo César y la correcta gestión de *sockets* en ambas arquitecturas.
2. **Verificación en Capa de Aplicación y Red:** La verificación mediante **Wireshark** y el sniffer **Scapy** validó que la pila TCP/IP opera correctamente y que el *payload* de la capa de aplicación respeta el formato de **Byte 0 = *Shift*** seguido del mensaje cifrado, demostrando la correctiva integración con utilerías de escritorio.
3. **Fallo Crítico Identificado (ESP - ESP):** A pesar de los éxitos anteriores, la prueba de descifrado entre dos módulos ESP32 (*cross-platform*) identificó un fallo en la lógica de manejo del *shift* o en la codificación de caracteres, impidiendo la validación completa del descifrado en este escenario particular y sugiriendo una inconsistencia específica del entorno ESP-IDF.

El proyecto valida la implementación del protocolo y la interoperabilidad, aunque requiere una iteración final para resolver la discrepancia de descifrado que surge al interactuar exclusivamente entre plataformas embebidas.