Assignment: TCP Sockets & Caesar Cipher (ESP-IDF & Linux)

Objective

Build end-to-end TCP communication between an ESP32 and a Linux PC using:

- ESP-IDF TCP examples (client/server) in **station mode** connected to a common Wi-Fi network (you may use a phone hotspot).
- Linux TCP client/server implemented in C and compiled with `gcc` and `CMake`.
- A simple **Caesar cipher** scheme (with the shift encoded in the first byte) to encrypt/decrypt a short message (student ID or name).
- Manual packet sniffing with **Wireshark** (screenshots with filters), then automated sniffing/decryption with a **Python** script.

Reference Material

- ESP-IDF TCP server example: https://github.com/espressif/esp-idf/tree/master/examples/protocols/sockets/tcp_server
- ESP-IDF TCP client example: https://github.com/espressif/esp-idf/tree/master/examples/protocols/sockets/tcp_client
- Linux socket programming article (GeekForGeeks): https://www.geeksforgeeks.org/c/socket-programming-cc/
- Caesar cipher validator (web tool): https://cryptii.com/pipes/caesar-cipher

Network Assumptions (Important)

- The ESP32 will operate in **station mode** (STA) and connect to a common Wi-Fi access point (e.g., your phone hotspot or campus AP).
- Configure ESP-IDF example Wi-Fi SSID/PASS via `menuconfig` (*Example Configuration*) before flashing.
- Both ESP32 and the Linux PC must be on the same IP subnet.

Pair Work & Scenarios

Work in pairs and complete the four code runs:

1. **Linux server ↔ ESP32 client**

2. **ESP32 server ↔ Linux client**
3. Repeat each direction (server/client swap) so that both codes (Linux and ESP) are tested as server and client.
4. Use a single well-known TCP port (e.g., `3333`) for convenience.

Message Format & Caesar Cipher Rule

- Message payload over TCP is: `[ 1-byte shift ][ ciphertext bytes... ]`.
- The **first byte** (unsigned) is the Caesar **shift** (0 . . . 25).
- Caesar applies to letters `A-Z` and `a-z` (wrap-around); digits `0-9` rotate modulo 10; other characters remain unchanged.
- **Ciphertext** is what you send; the receiver reads the first byte (shift) and **decrypts** the rest.
- The plaintext content should be the student's **ID or full name**.

Submission as a Git Pull Request (One PR per Pair)

Single PR Policy

Each pair submits **exactly one** Pull Request (PR). Both members must appear as contributors in the commit history.

Required Repository Layout (deliverables)

```
repo-root/
        linux/
             server.c
             client.c
             caesar.c
             caesar.h
             CMakeLists.txt
             README.md                  # how to build/run on Linux
        esp/
             server/                    # ESP-IDF server project (station mode)
                 main/...
                 CMakeLists.txt
                 sdkconfig.defaults     # put your Wi-Fi config notes in README
             client/                    # ESP-IDF client project (station mode)
                main/...
                CMakeLists.txt
                sdkconfig.defaults
        python/
             sniff_caesar.py            # sniffer + decrypt (Scapy)
        docs/
             Report.pdf                 # final report
             wireshark/                 # screenshots (*.png/*.jpg)
                filter_port_3333.png
```

```
                payload_example.png
        .gitignore                      # exclude build artifacts
        README.md                       # top-level instructions summary
```

Branch, Title, and Reviewers

- **Branch name:** `pair/<lastname1-lastname2>`
- **PR title:** `[TCP+Caesar] <Lastname1 & Lastname2>`
- **Assign reviewers:** Tonix22.

What the PR *must* contain

1. **Linux sockets (C):** client & server, buildable with `gcc` and `CMake` (see `linux/`).
2. **ESP-IDF sockets (C):** client & server projects in STA mode (see `esp/`).
3. **Python sniffer:** python/sniffcaesar.py
4. **Screenshots:** Wireshark display filters and captured payloads showing first byte (shift) and ciphertext.
5. **Report (PDF):**

PR Description Template (copy/paste into PR body)

```
## Pair
- Student A: <Full Name, ID, GitHub handle>
- Student B: <Full Name, ID, GitHub handle>

## Whats included
- Linux sockets: `linux/server.c`, `linux/client.c`, `linux/CMakeLists.txt`
- ESP-IDF sockets: `esp/server/`, `esp/client/` (station mode)
- Python sniffer: `python/sniff_caesar.py`
- Report + screenshots: `docs/Report.pdf`, `docs/wireshark/*.png`

## Network & Run Info
- Common Wi-Fi SSID: <ssid> (ESP in STA mode)
- Server TCP port: 3333
- PC interface: <e.g., enp0s31f6 or wlan0>
- ESP IP: <x.x.x.x>, PC IP: <x.x.x.x>
- Wireshark filters used: `tcp.port == 3333`, and/or `ip.addr == <ESP_IP>`

## Caesar Cipher
- Shift byte used (0  25 ): <N>
- Example plaintext: "<Your Name or ID>"
- Example ciphertext: <captured in Wireshark>
- Online validation: https://cryptii.com/pipes/caesar-cipher

## How to build/run (quick)
### Linux
```bash
mkdir -p linux/build && cd linux/build
cmake ..
```
```

```
cmake --build . --config Release
./tcp_server 0.0.0.0 3333
# in another terminal:
./tcp_client <server_ip> 3333 "<Name_123>"
```

Deliverables (PDF Report)

1. **Build instructions** (Linux & ESP-IDF): exact commands, port used, and how you found the network interface (`ifconfig` or `ip a`).
2. **Screenshots** of Wireshark showing:
   - Correct capture filters (IP/port).
   - Packets carrying your payload.
3. **C code** (Linux server & client) with `CMakeLists.txt`.
4. **ESP-IDF config** notes: how you set SSID/PASS and which example you adapted.
5. **Python sniffer** output (console screenshot) that shows the decrypted message.
6. **Validation** using the online Caesar tool (URL above): show that your shift and plaintext/ciphertext match.

Grading Rubric (100 pts)

- Connectivity & runs in both directions (ESP↔Linux) – 30 pts
- Correct Caesar format (first byte = shift) and decryption – 20 pts
- Wireshark filters & screenshots – 20 pts
- Python sniffer that extracts first byte & decrypts payload – 20 pts
- Code quality (comments, minimal errors, clear build steps) – 10 pts

Wireshark Instructions

- Identify your interface (e.g., `enp0s31f6`, `wlan0`, `enps0`). Use `ifconfig` or `ip link`.
- Suggested display filters:
  - `tcp.port == 3333`
  - `ip.addr == <ESP_IP> or ip.addr == <PC_IP>`
  - Combine: `tcp.port == 3333 and ip.addr == <ESP_IP>`
- Click a TCP segment that contains `[Raw]` payload and verify the first byte corresponds to your configured shift.

Build & Run (Linux)

GCC commands (example):

```
# Build
gcc -Wall -O2 -o tcp_server server.c caesar.c
gcc -Wall -O2 -o tcp_client client.c  caesar.c

# Run server first (choose an IP/port you listen on):
./tcp_server 0.0.0.0 3333

# In another terminal, run client pointing to server IP:
./tcp_client 192.168.1.50 3333 "Alice_123"
```

CMakeLists.txt (top-level, builds both):

```
cmake_minimum_required(VERSION 3.10)
project(CaesarTCP C)

set(CMAKE_C_STANDARD 11)
set(CMAKE_C_STANDARD_REQUIRED ON)

add_executable(tcp_server server.c caesar.c)
add_executable(tcp_client client.c  caesar.c)

# On Linux no extra libs needed; on macOS you may need flags for
    sockaddr_in.
# target_link_libraries(tcp_server PRIVATE ...)
# target_link_libraries(tcp_client PRIVATE ...)
```

CMake build:

```
mkdir -p build && cd build
cmake ..
cmake --build . --config Release
```

ESP-IDF Notes

- Set target (e.g., `esp32`), configure Wi-Fi SSID/PASS in `menuconfig` (*Example Configuration*).
- Adapt the official examples:
  - **TCP server**: https://github.com/espressif/esp-idf/tree/master/examples/protocols/sockets/tcp_server
  - **TCP client**: https://github.com/espressif/esp-idf/tree/master/examples/protocols/sockets/tcp_client
- Replace their payload with your Caesar-encrypted message and prepend the **shift byte**.

Code Sketches (You may copy/adapt)

Caesar Cipher (Python)

Listing 1: caesar.py (Python sketch)

```python
def _rot_alpha(ch: str, shift: int) -> str:
    # rotate letters; keep case; non-letters unchanged here
    if 'a' <= ch <= 'z':
        base = ord('a')
        return chr((ord(ch) - base + shift) % 26 + base)
    if 'A' <= ch <= 'Z':
        base = ord('A')
        return chr((ord(ch) - base + shift) % 26 + base)
    return ch

def _rot_digit(ch: str, shift: int) -> str:
    if '0' <= ch <= '9':
        base = ord('0')
        return chr((ord(ch) - base + shift) % 10 + base)
    return ch

def caesar_encrypt(plaintext: str, shift: int) -> bytes:
    shift = shift % 26
    out = []
    for ch in plaintext:
        if ch.isalpha():
            out.append(_rot_alpha(ch, shift))
        elif ch.isdigit():
            out.append(_rot_digit(ch, shift))
        else:
            out.append(ch)
    # Prepend shift as a single byte
    return bytes([shift]) + ''.join(out).encode('utf-8')

def caesar_decrypt(payload: bytes) -> str:
    if not payload:
        return ""
    shift = payload[0] % 26
    ciphertext = payload[1:].decode('utf-8', errors='ignore')
    inv = (26 - shift) % 26
    out = []
    for ch in ciphertext:
        if ch.isalpha():
            out.append(_rot_alpha(ch, inv))
        elif ch.isdigit():
            out.append(_rot_digit(ch, (10 - (shift % 10)) % 10))
        else:
            out.append(ch)
    return ''.join(out)

if __name__ == "__main__":
    msg = "Alice_123"
    s = 5
    pkt = caesar_encrypt(msg, s)
    print("TX bytes:", pkt)
    print("Decrypted:", caesar_decrypt(pkt))
```

Caesar Cipher (C)

Listing 2: caesar.c / caesar.h (C sketch)

```c
// caesar.h
#ifndef CAESAR_H
#define CAESAR_H
#include <stddef.h>
#include <stdint.h>

size_t caesar_encrypt_bytes(const char* plaintext, uint8_t shift,
                            uint8_t* out, size_t out_cap);
/* out[0] = shift; returns total bytes written (including first shift
   byte) */

size_t caesar_decrypt_bytes(const uint8_t* in, size_t in_len,
                            char* out, size_t out_cap);
/* reads in[0] as shift; writes NUL-terminated plaintext if space
   permits */

#endif // CAESAR_H
```

```c
// caesar.c
#include "caesar.h"
#include <ctype.h>

static char rot_alpha(char c, int shift) {
    if ('a' <= c && c <= 'z') {
        int base = 'a';
        return (char)((((c - base) + shift) % 26) + base);
    }
    if ('A' <= c && c <= 'Z') {
        int base = 'A';
        return (char)((((c - base) + shift) % 26) + base);
    }
    return c;
}
static char rot_digit(char c, int shift) {
    if ('0' <= c && c <= '9') {
        int base = '0';
        return (char)((((c - base) + (shift % 10)) % 10) + base);
    }
    return c;
}

size_t caesar_encrypt_bytes(const char* plaintext, uint8_t shift,
                            uint8_t* out, size_t out_cap) {
    if (!out || out_cap == 0) return 0;
    size_t w = 0;
    out[w++] = (uint8_t)(shift % 26);
    for (const char* p = plaintext; *p; ++p) {
        char c = *p;
        if (isalpha((unsigned char)c)) c = rot_alpha(c, shift % 26);
        else if (isdigit((unsigned char)c)) c = rot_digit(c, shift);
        if (w < out_cap) out[w++] = (uint8_t)c; else break;
    }
    return w;
}

size_t caesar_decrypt_bytes(const uint8_t* in, size_t in_len,
                            char* out, size_t out_cap) {
    if (!in || in_len == 0 || !out || out_cap == 0) return 0;
```

```
    uint8_t shift = in[0] % 26;
    int inv = (26 - shift) % 26;
    int inv_d = (10 - (shift % 10)) % 10;
    size_t w = 0;
    for (size_t i = 1; i < in_len; ++i) {
        char c = (char)in[i];
        if (isalpha((unsigned char)c)) c = rot_alpha(c, inv);
        else if (isdigit((unsigned char)c)) c = rot_digit(c, inv_d);
        if (w + 1 < out_cap) out[w++] = c; else break;
    }
    if (w < out_cap) out[w] = '\0';
    return w;
}
```

Linux TCP Server (C)

Listing 3: server.c (Linux)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include "caesar.h"

#define BUFSZ 1024

int main(int argc, char** argv) {
    if (argc < 3) {
        fprintf(stderr, "Usage: %s <bind_ip> <port>\n", argv[0]);
        return 1;
    }
    const char* bind_ip = argv[1];
    int port = atoi(argv[2]);

    int sfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sfd < 0) { perror("socket"); return 1; }

    int opt = 1;
    setsockopt(sfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));

    struct sockaddr_in addr = {0};
    addr.sin_family = AF_INET;
    addr.sin_port = htons((uint16_t)port);
    inet_pton(AF_INET, bind_ip, &addr.sin_addr);

    if (bind(sfd, (struct sockaddr*)&addr, sizeof(addr)) < 0) { perror(
        "bind"); return 1; }
    if (listen(sfd, 1) < 0) { perror("listen"); return 1; }
    printf("Server listening on %s:%d\n", bind_ip, port);

    struct sockaddr_in cli = {0}; socklen_t clen = sizeof(cli);
    int cfd = accept(sfd, (struct sockaddr*)&cli, &clen);
    if (cfd < 0) { perror("accept"); return 1; }

```

```
    uint8_t buf[BUFSZ];
    ssize_t n = recv(cfd, buf, sizeof(buf), 0);
    if (n > 0) {
        char plain[BUFSZ];
        caesar_decrypt_bytes(buf, (size_t)n, plain, sizeof(plain));
        printf("Received %zd bytes. Decrypted: %s\n", n, plain);

        // Echo back the same payload (as received), or re-encrypt your own reply:
        send(cfd, buf, (size_t)n, 0);
    }

    close(cfd);
    close(sfd);
    return 0;
}
```

Linux TCP Client (C)

Listing 4: client.c (Linux)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include "caesar.h"

#define BUFSZ 1024

int main(int argc, char** argv) {
    if (argc < 4) {
        fprintf(stderr, "Usage: %s <server_ip> <port> <message>\n", argv[0]);
        return 1;
    }
    const char* ip = argv[1];
    int port = atoi(argv[2]);
    const char* msg = argv[3];

    int sfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sfd < 0) { perror("socket"); return 1; }

    struct sockaddr_in addr = {0};
    addr.sin_family = AF_INET;
    addr.sin_port = htons((uint16_t)port);
    inet_pton(AF_INET, ip, &addr.sin_addr);

    if (connect(sfd, (struct sockaddr*)&addr, sizeof(addr)) < 0) {
        perror("connect"); return 1;
    }

    uint8_t pkt[BUFSZ];
    uint8_t shift = 5; // choose your shift (0..25), also put in first byte
```

```
    size_t n = caesar_encrypt_bytes(msg, shift, pkt, sizeof(pkt));
    send(sfd, pkt, n, 0);

    uint8_t reply[BUFSZ];
    ssize_t r = recv(sfd, reply, sizeof(reply), 0);
    if (r > 0) {
        char plain[BUFSZ];
        caesar_decrypt_bytes(reply, (size_t)r, plain, sizeof(plain));
        printf("Reply (%zd bytes). Decrypted: %s\n", r, plain);
    }

    close(sfd);
    return 0;
}
```

Python Sniffer & Decrypter (Local Use Only)

Run with `sudo` if required. Replace interface and optional IP filter as needed. Keep messages short so they fit in a single TCP segment (simplifies extraction from `Raw`).

Listing 5: sniff$_c$*aesar.py*(*Scapy*)

```
from scapy.all import sniff, TCP, Raw
import sys

def _rot_alpha(ch, shift):
    if 'a' <= ch <= 'z':
        base = ord('a'); return chr((ord(ch)-base+shift)%26 + base)
    if 'A' <= ch <= 'Z':
        base = ord('A'); return chr((ord(ch)-base+shift)%26 + base)
    return ch

def _rot_digit(ch, shift):
    if '0' <= ch <= '9':
        base = ord('0'); return chr((ord(ch)-base+(shift%10))%10 + base
           )
    return ch

def caesar_decrypt(payload: bytes) -> str:
    if not payload:
        return ""
    s = payload[0] % 26
    inv = (26 - s) % 26
    inv_d = (10 - (s % 10)) % 10
    text = payload[1:].decode('utf-8', errors='ignore')
    out = []
    for ch in text:
        if ch.isalpha():
            out.append(_rot_alpha(ch, inv))
        elif ch.isdigit():
            out.append(_rot_digit(ch, inv_d))
        else:
            out.append(ch)
    return ''.join(out)

def handle(pkt):
    if pkt.haslayer(TCP) and pkt.haslayer(Raw):
```

```
        data = bytes(pkt[Raw].load)
        if len(data) >= 2:
            try:
                plain = caesar_decrypt(data)
                print(f"[+] Decrypted: {plain}")
            except Exception as e:
                print(f"[!] Decode error: {e}")

if __name__ == "__main__":
    iface = sys.argv[1] if len(sys.argv) > 1 else "enp0s31f6"  # change as needed
    # Optional BPF filter: set your port
    bpf = "tcp port 3333"
    print(f"Sniffing on {iface} with filter: {bpf}")
    sniff(iface=iface, filter=bpf, prn=handle, store=False)
```

Safety & Ethics

- Sniff only **your own** lab traffic on the local network you control (ESP ↔ your PC).
- Do not capture third-party traffic. Follow institutional network policies.

Tips

- Validate Caesar shifts with https://cryptii.com/pipes/caesar-cipher.
- If you cannot see payload in Wireshark, ensure messages are short and not fragmented, or use TCP stream reassembly view.
- If your interface name is different (e.g., `wlan0`, `enps0`), check with `ifconfig` or `ip a`.