“ESP32 + ENC28J60 MQTT Client with TLS”

Design Notes and Experimental Observations

📘 Project Overview

This document presents a deep technical breakdown of building a fully secure, TLS-encrypted MQTT client on an ESP32 using the ENC28J60 Ethernet module. The implementation is done without relying on high-level client libraries such as **PubSubClient** or **WiFiClientSecure**, showcasing raw control over every layer: **Ethernet**, **TLS**, **MQTT**, and task management via **FreeRTOS**.

The project uses:

- **ESP32 microcontroller**
- **ENC28J60 Ethernet module (EthernetENC)**
- **mbedTLS** for **TLS 1.2** encryption
- **Manual MQTT packet implementation**
- **FreeRTOS** for multitasking architecture

🎯 Motivation & Purpose

The core motivation was to achieve **TLS-secured MQTT over ENC28J60**, something considered highly uncommon in the embedded world. Most developers rely on **Wi-Fi** or **W5500** Ethernet modules due to simpler TLS support. **ENC28J60** lacks native TLS capability and is RAM-hungry (because all the **TLS** handling is handled by **ESP-32** itself)—but this challenge was taken up intentionally to explore its full potential and document what it takes to make it work.

The project successfully implements:

- **TLS handshak**e manually using **mbedTLS**
- **MQTT CONNECT/SUBSCRIBE/PUBLISH/PINGREQ/PINGRESP** packets from scratch

- Switching between **RAW** and **JSON** payloads
- Robust reconnection strategies
- Memory-safe cleanup and recovery
- Task-based concurrency using **FreeRTOS**

🔧 What Is ENC28J60?

ENC28J60 is a low-cost **SPI**-based Ethernet controller by Microchip. It's minimal and lacks onboard **TCP/IP** stack, **TLS offload**, or **advanced buffering**, making it less preferred for secure applications.

Contrast with W5500:

W5500 offers better buffer management and lower overhead. It’s commonly used in secure projects due to mature library support. However, with proper timing, validation, and custom TLS bridging, **ENC28J60 proves itself capable—just underestimated**.

🔐 What Is TLS & Why mbedTLS?

TLS (Transport Layer Security) is the backbone of encrypted internet communication, securing **MQTT** messages on port 8883.

mbedTLS is an embedded-friendly implementation of the **TLS protocol**. It supports:

- **TLS 1.2**
- **X.509 certificate validation**
- **Encrypted streams**

It’s modular but RAM-heavy, requiring careful handling on ESP-32. This project used:

- **mbedtls/ssl.h, mbedtls/net_sockets.h**
- **mbedtls/x509_crt.h for root CA validation**
- **mbedtls/platform.h for platform abstraction**

Entropy Seeding:

A secure handshake needs a good source of randomness. We initialized entropy pools using:

mbedtls_entropy_init(&entropy);

mbedtls_ctr_drbg_seed(&ctr_drbg, mbedtls_entropy_func, &entropy, ...);

This randomness is critical to **TLS security**.

BIO Callbacks:

mbedTLS must **read/write** via custom functions:

mbedtls_ssl_set_bio(&ssl, &ethernetClient, mbedtls_custom_send, mbedtls_custom_recv, NULL);

These callbacks adapt mbedTLS to EthernetClient.

🧱 Core Features

✅ Secure TLS Layer

- **X.509 root CA** validation (embedded directly)
- **TLS 1.2 handshake** with broker (otplai.com:8883)
- Username/password authentication in **CONNECT** packet

✅ Custom MQTT Protocol Implementation

- Manual creation of **CONNECT**, **CONNACK**, **SUBSCRIBE**, **SUBACK**, **PUBLISH**, **PINGREQ**, **PINGRESP**
- Packet format aligns with **MQTT 3.1.1 spec**
- **MQTT packets** built using proper byte structures:
    - Remaining length encoding
    - Flags
    - Topic length
    - Payload size

✅ FreeRTOS-Based Architecture

- Separate task for **MQTT** loop **(MqttTask)**
- Allows non-blocking operation

- Keeps connection alive, handles **PINGREQ**
- Reconnects on timeout or broker disconnect

🔄 Challenge Highlights & Engineering Breakthroughs

🔸 MQTT Packet Building (Manual)

Each packet adheres to the **MQTT 3.1.1 spec**:

- **Byte 1:** Control Packet Type (e.g., 0x10 for CONNECT)
- **Byte 2+:** Remaining Length (variable encoding)
- **Payload:** Topic, client ID, message, QoS flags, etc.

This manual approach allowed precise control and debug visibility.

🔸 Partial Packet Reception

TLS streams split data across reads. This caused broken packets.

Solution: Implemented a 1024-byte circular buffer that accumulates and parses each complete MQTT packet. Used memmove to manage shifting unprocessed data.

🔸 Root CA & Certificate Trust

Root CA was stored in firmware and used to verify the **MQTT** broker's certificate. Only if trust is established then only does the **TLS handshake** complete.

🔸 Debugging & Attempts

- Initially tried **WiFiClientSecure** + **PubSubClient**.
    - Could not adapt **WiFiClientSecure** to Ethernet, since it's **Wi-Fi** specific.
    - Attempted to steal its TLS logic—**failed**.
- Switched to **EthernetClient**, which offered **raw TCP** over **ENC28J60**
    - Combined it with **mbedTLS** via **BIO bridging**

📡 PubSubClient vs Raw MQTT

PubSubClient is great for simple setups. But it requires **WiFiClient(**especially when using **Wi-Fi)** or **W5500**-compatible Ethernet.

This project: Uses no **PubSubClient**. It constructs and parses MQTT manually for maximum flexibility.

📥 Receiving MQTT Packets

CONNACK

- Broker's response to **CONNECT**
- Contains session flags and connection result code

SUBACK

- Response to **SUBSCRIBE**
- Includes packet ID and return codes

PINGREQ / PINGRESP

- Keep-alive mechanism
- Client sends **PINGREQ**, broker must reply **PINGRESP**

QoS 1 Publishing

- QoS 1 requires **PUBACK**
- We detect packet ID but haven't sent **PUBACK** yet

🔄 Reconnection & Memory Cleanup

- On disconnect, **mbedTLS** contexts are fully freed
- Ethernet connection reset
- Reconnect logic waits for valid IP + **Ethernet.linkStatus**()

📊 JSON vs Raw Payloads

We alternate between:

{"id":5,"status":"ok","type":"json"}

...and raw:

data5 ok

This ensures compatibility with both logging tools **(RAW)** and structured parser **(JSON)** like dashboards.

📉 Memory Efficiency Surprises

Despite **TLS + JSON + Ethernet + FreeRTOS**, the heap usage was shockingly low:

Free RAM: 285 KB

Allocated: 72 KB

Largest Free Block: 108 KB

This suggests **FreeRTOS + TLS** runs inside **ROM**-segmented areas, minimizing **RAM footprint**.

🤯 Unheard Of: TLS over ENC28J60

We scoured GitHub and Reddit. Most suggested "Use W5500" or "**WiFiClientSecure** only." Almost no full example of TLS over ENC28J60 was found.

This may be among the first open, clean examples of mbedTLS + ENC28J60 + MQTT.

💡 How AI Helped — and How You Led It

This wasn't about copying code or repackaging tutorials. Most of what went into this project came from **trial-and-error, observation, and iteration**, often sparked by small cues AI pointed out.

The steps weren't always clear. The logic wasn't always clean from the start. Some paths—like trying **WiFiClientSecure** logic on an Ethernet setup—were long shots that didn't work but helped **clarify what wouldn't**. Ideas like bridging **mbedTLS** to **EthernetClient** didn't come pre-packaged either—they emerged gradually from attempting different layers and watching what failed.

In truth:

- The move to **FreeRTOS** wasn't planned—it became necessary.
- The use of root CA certificates came after repeated failed MQTT connections.
- The working publish only came after **calculating the right byte lengths**.
- There wasn't much restructuring—just **consistent poking, testing, and noticing**.

AI was there as a support—suggesting functions, concepts, documentation. But the actual flow—how pieces were **connected**, **verified**, **retried**—that was manual. That was learning. And it was slow at times, but that's what made it valuable.

What emerged wasn't just a working system. It was an understanding of how **embedded networking, TLS, and MQTT really behave under constraints**.

I didn't avoid **FreeRTOS.**

RESULTS

Memory Usage Info:

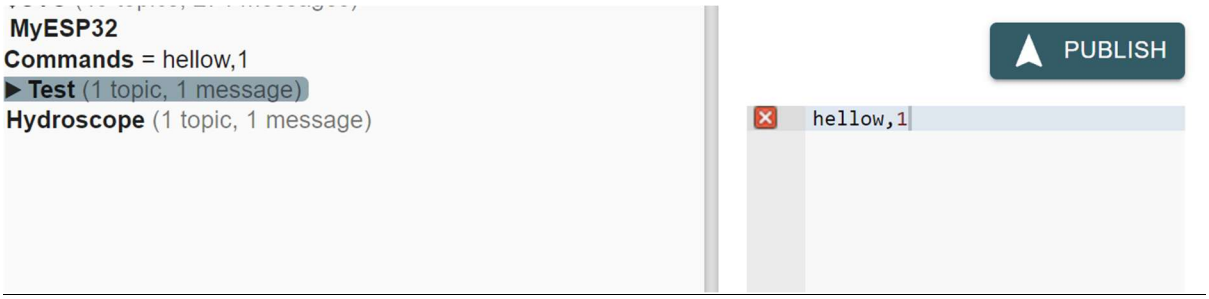
```
-----------------------------------------
INTERNAL Memory Info:
-----------------------------------------
  Total Size        :   374892 B ( 366.1 KB)
  Free Bytes        :   333120 B ( 325.3 KB)
  Allocated Bytes   :    34424 B (  33.6 KB)
  Minimum Free Bytes:   326944 B ( 319.3 KB)
  Largest Free Block:   110580 B ( 108.0 KB)
-----------------------------------------
Flash Info:
-----------------------------------------
  Chip Size         :  4194304 B (4 MB)
  Block Size        :    65536 B (  64.0 KB)
  Sector Size       :     4096 B (   4.0 KB)
  Page Size         :      256 B (   0.2 KB)
  Bus Speed         : 80 MHz
  Bus Mode          : DIO
-----------------------------------------
```

TLS Handshake Success + MQTT Connect + MQTT Publish:

```
16:09:03.485 -> ✅ EthernetClient connected to MQTT broker.
16:09:03.485 -> [🤝] Performing TLS handshake...
16:09:03.529 -> TLS Handshake: Bytes available from broker: 512
16:09:03.560 -> TLS Handshake: Bytes available from broker: 512
16:09:03.594 -> TLS Handshake: Bytes available from broker: 512
16:09:03.594 -> TLS Handshake: Bytes available from broker: 512
16:09:03.638 -> TLS Handshake: Bytes available from broker: 512
16:09:03.638 -> TLS Handshake: Bytes available from broker: 512
16:09:03.670 -> TLS Handshake: Bytes available from broker: 512
16:09:03.978 -> ✅ TLS connected.
16:09:03.978 -> Sending MQTT CONNECT Packet (Bytes: 54): 10 34 00 04 4D 51 54 54 04 C2 00 3C 00 18 4D 79 55 6E 69 71 75 65 4
16:09:04.024 -> ✅ MQTT CONNECT sent.
16:09:04.024 -> ✅ MQTT CONNACK received. Connection Accepted!
16:09:04.056 -> ✅ MQTT PUBLISH sent.
16:09:04.056 -> 📩 Secure message published!
16:09:04.056 -> MQTT Task finished. Deleting task.
```

MQTT Subscribed Topic Data Received:

```
12:58:47.204 -> ✅ PINGREQ sent.
12:58:47.283 -> Received MQTT Packet Type: 0x3, Total Length: 28
12:58:47.283 ->   -> PUBLISH Packet detected!
12:58:47.283 ->     Topic: MyESP32/Commands
12:58:47.329 ->     Payload: hellow,1
12:58:47.329 -> Heartbeat: Sending PINGREQ...
```



ARDUINO IDE Code

```
#include <SPI.h>
#include <EthernetENC.h>
#include "mbedtls/platform.h"
#include "mbedtls/net_sockets.h"
#include "mbedtls/ssl.h"
#include "mbedtls/ctr_drbg.h"
#include "mbedtls/entropy.h"
#include "mbedtls/error.h"
#include "mbedtls/debug.h"
#include "mbedtls/x509_crt.h"
#include "freertos/task.h"

// Define for string formatting (like sprintf)
#include <stdio.h> // For sprintf

#define ENC_CS 5

#define MQTT_RECV_BUFFER_SIZE 1024 // Adjust as
needed for your maximum expected message size
static unsigned char
mqtt_recv_buffer[MQTT_RECV_BUFFER_SIZE];
static int mqtt_recv_buffer_len = 0; // Current data
length in buffer
//#define MBEDTLS_DEBUG_C // Keep this commented out
for now
```

```
void print_mbedtls_error(int err_code); // Add this
line!
bool subscribe_mqtt_topic(const char* topic, uint8_t
qos); // You might already have this
bool publish_mqtt_message(const char* topic, const
char* payload); // You might already have this
void handle_incoming_mqtt_packet();

// Ethernet setuplike if i send anything
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192, 168, 1, 177);
IPAddress dnsIP(192, 168, 1, 1);
IPAddress gateway(192, 168, 1, 1);
IPAddress subnet(255, 255, 255, 0);

// MQTT broker details
const char* mqtt_host = "otplai.com";
const int mqtt_port = 8883;
const char* mqtt_user = "oyt";
const char* mqtt_pass = "123456789";
const char* mqtt_publish_topic =
"MyESP32/Test/Status"; // Topic for publishing
const char* mqtt_subscribe_topic =
"MyESP32/Commands";   // Topic for subscribing

// Global buffers for MQTT packets (increased size
for JSON)
static uint8_t mqtt_connect_packet[256];
static uint8_t mqtt_publish_packet[256]; // Increased
size for JSON payloads
static uint8_t mqtt_subscribe_packet[128]; // For
SUBSCRIBE packet

// LOCATION: After 'static uint8_t
mqtt_publish_packet[128];' (or your existing
definition)
//           and before the new global variables.

const char* root_ca = \
"-----BEGIN CERTIFICATE-----\n" \
```

```
"MIIGiTCCBXGgAwIBAgIIbmCv9bjYeYUwDQYJKoZIhvcNAQELBQAw
gbQxCzAJBgNV\n" \
"BAYTAlVTMRAwDgYDVQQIEwdBcml6b25hMRMwEQYDVQQHEwpTY290
dHNkYWxlMRow\n" \
"GAYDVQQKExFHb0RhZGR5LmNvbSwgSW5jLjEtMCsGA1UECxMkaHR0
cDovL2NlcnRz\n" \
"LmdvZGFkZHkuY29tL3JlcG9zaXRvcnkvMTMwMQYDVQQDEypHbyBE
YWRkeSBTZWN1\n" \
"cmUgQ2VydGlmaWNhdGUgQXV0aG9yaXR5IC0gRzIwHhcNMjUwNjI3
MTI1NTI1WhcN\n" \
"MjYwNjI2MTMyMzU4WjAVMRMwEQYDVQQDEwpvdHBsYWkuY29tMIIB
IjANBgkqhkiG\n" \
"9w0BAQEFAAOCAQ8AMIIBCgKCAQEAul/nKvnAJNaK/h2MFhmLuini
3X9qriM+QDEt\n" \
"X3zGH2WOETBaGft70mAUT8Pe31n+qBAWDdwTo1Hma4EuHuUxYmtW
xWkSUKw0E5Sf\n" \
"pQD0SWXa81WRVz5w6rZtzp/nuE/RBvlv5NM3aYYPMfbNSis/Ju9V
rD6Hi/DwA0M5\n" \
"VdLc30jlrLwV9Ga7mnukiV+Xf9m4Ma2rP4ZV5HN1ZYGG/7H4YEul
vqt1f1Iu64Qr\n" \
"VPbdEEFbCdaoLkyoPgZuSISWJtk1SqgwnU1xuJSqAn1I3cQeg2Tn
I1fTIZ6hRLZe\n" \
"EYPriMpffZoyHkHmp81pxZSbTcD2jD31SZiK91nzvQa7Yr8y9wID
AQABo4IDOzCC\n" \
"AzcwDAYDVR0TAQH/BAIwADAdBgNVHSUEFjAUBggrBgEFBQcDAQYI
KwYBBQUHAwIw\n" \
"DgYDVR0PAQH/BAQDAgWgMDkGA1UdHwQyMDAwLqAsoCqGKGh0dHA6
Ly9jcmwuZ29k\n" \
"YWRkeS5jb20vZ2RpZzJzMS01MTc5Mi5jcmwwXQYDVR0gBFYwVDBI
BgtghkgBhv1t\n" \
"AQcXATA5MDcGCCsGAQUFBwIBFitodHRwOi8vY2VydGlmaWNhdGVz
LmdvZGFkZHku\n" \
"Y29tL3JlcG9zaXRvcnkvMAgGBmeBDAECATB2BggrBgEFBQcBAQRq
MGgwJAYIKwYB\n" \
"BQUHMAGGGGh0dHA6Ly9vY3NwLmdvZGFkZHkuY29tLzBABggrBgEF
BQcwAoY0aHR0\n" \
"cDovL2NlcnRpZmljYXRlcy5nb2RhZGR5LmNvbS9yZXBvc2l0b3J5
L2dkaWcyLmNy\n" \
"dDAfBgNVHSMEGDAWgBRAwr0njsw0gzCiM9f7bLPwtCyAzjAlBgNV
HREEHjAcggpv\n" \
```

```
"dHBsYWkuY29tgg53d3cub3Rwbai5jb20AdBgNVHQ4EFgQUa8lHql
pLMFdOmXHu\n" \
"ZvnPg2cfm8gwggF9BgorBgEEAdZ5AgQCBIIBbQSCAWkBZwB2AA5X
lLzzrqk+Mxss\n" \
"mQez95Dfm8I9cTIl3SGpJaxhxU4hAAABl7F1B9sAAAQDAEcwRQIg
QPiAoR7F+InT\n" \
"zNjWu9qGvZXaLTaYbUBqSOorjvOkcUQCIQDzkIpv0g7y9/T5VPbo
HGyYk329LOIO\n" \
"5DAQkRNpmlkvIgB2AGQRxGykEuyniRyiAi4AvKtPKAfUHjUnq+r+
1QPJfc3wAAAB\n" \
"l7F1COUAAAQDAEcwRQIgT4AcFDnUsoFqjiM3Kw5FHAvJki6AyvMD
8mfsr9Fj2O4C\n" \
"IQCT6KhRCGPTJHFmDUxqR7akZSaINtzAn5aUf0r5KufbpQB1AMs4
9xWJfIShRF9b\n" \
"wd37yW7ymlnNRwppBYWwyxTDFFjnAAABl7F1CXcAAAQDAEYwRAIg
SYpJkXr2llP+\n" \
"SVNHrJTiQ6LHy4+zO6u5WxiX5tzmMcgCIDR/2jZr5OBf7vpHuZrs
I688sQr6u20l\n" \
"7xlgp8Lgqw+YMA0GCSqGSIb3DQEBCwUAA4IBAQBXYyXIH78tw6b+
WRdaAKvhP+h8\n" \
"z6IIO3EsM41XF9bOweAMpWqzqlcy2e5zIDpM+WYoPAEdWJgOwSLq
ouIpaN2mVlS1\n" \
"jvz69Zr+dE/jvUCwdMsN87fE4noWV/hHU6+kFnHn1wPdOxpmSZtE
cHUCT4jT+wC4\n" \
"l/k8xAfIG8KQM5H7n+dJcHCREAYF1kJvZvW/XVZGapId1jhG2S6N
ckSofcRRsWsu\n" \
"GyhgZOZSMgN7yIEYVIJr30hXDxT0HTsXUO8SNdDJh0uUEX4eKWvJ
mboc+JAcoOjt\n" \
"fXPHMXan+k5RpOpiv5bAdmgJ7A4WmXaqs9r0y0vzkwG5h5j/SrYa
bL2D+x8i\n" \
"-----END CERTIFICATE-----\n" \
"-----BEGIN CERTIFICATE-----\n" \
"MIIE0DCCA7igAwIBAgIBBzANBgkqhkiG9w0BAQsFADCBgzELMAkG
A1UEBhMCVVMx\n" \
"EDAOBgNVBAgTB0FyaXpvbmExEzARBgNVBAcTClNjb3R0c2RhbGUx
GjAYBgNVBAoT\n" \
"EUdvRGFkZHkuY29tLCBJbmMuMTEwLwYDVQQDEyhHbyBEYWRkdyBS
b290IENlcnRp\n" \
"ZmljYXRlIEF1dGhvcml0eSAtIEcyMB4XDTExMDUwMzA3MDAwMFoX
DTMxMDUwMzA3\n" \
```

```
"MDAwMFowgbQxCzAJBgNVBAYTAlVTMRAwDgYDVQQIEwdBcml6b25h
MRMwEQYDVQQH\n" \
"EwpTY290dHNkYWxlMRowGAYDVQQKExFHb0RhZGR5LmNvbSwgSW5j
LjEtMCsGA1UE\n" \
"CxMkaHR0cDovL2NlcnRzLmdvZGFkZHkuY29tL3JlcG9zaXRvcnkv
MTMwMQYDVQQD\n" \
"EypHbyBEYWRkeSBTZWN1cmUgQ2VydGlmaWNhdGUgQXV0aG9yaXR5
IC0gRzIwggEi\n" \
"MA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQC54MsQ1K92vdST
YuswZLiBCGzD\n" \
"BNliF44v/z5lz4/OYuY8UhzaFkVLVat4a2ODYpDOD2lsmcgaFItM
zEUz6ojcnqOv\n" \
"K/6AYZ15V8TPLvQ/MDxdR/yaFrzDN5ZBUY4RS1T4KL7QjL7wMDge
87Am+GZHY23e\n" \
"cSZHjzhHU9FGHbTj3ADqRay9vHHZqm8A29vNMDp5T19MR/gd71vC
xJ1gO7GyQ5HY\n" \
"pDNO6rPWJ0+tJYqlxvTV0KaudAVkV4i1RFXULSo6Pvi4vekyCgKU
ZMQWOlDxSq7n\n" \
"eTOvDCAHf+jfBDnCaQJsY1L6d8EbyHSHyLmTGFBUNUtpTrw700ku
H9zB0lL7AgMB\n" \
"AAGjggEaMIIBFjAPBgNVHRMBAf8EBTADAQH/MA4GA1UdDwEB/wQE
AwIBBjAdBgNV\n" \
"HQ4EFgQUQMK9J47MNIMwojPX+2yz8LQsgM4wHwYDVR0jBBgwFoAU
OpqFBxBnKLbv\n" \
"9r0FQW4gwZTaD94wNAYIKwYBBQUHAQEEKDAmMCQGCCsGAQUFBzAB
hhhodHRwOi8v\n" \
"b2NzcC5nb2RhZGR5LmNvbS8wNQYDVR0fBC4wLDAqoCigJoYkaHR0
cDovL2NybC5n\n" \
"b2RhZGR5LmNvbS9nZHJvb3QtZzIuY3JsMEYGA1UdIAQ/MD0wOwYE
VR0gADAzMDEG\n" \
"CCsGAQUFBwIBFiVodHRwczovL2NlcnRzLmdvZGFkZHkuY29tL3Jl
cG9zaXRvcnkv\n" \
"MA0GCSqGSIb3DQEBCwUAA4IBAQAIfmyTEMg4uJapkEv/oV9PBO9s
PpyIBslQj6Zz\n" \
"91cxG7685C/b+LrTW+C05+Z5Yg4MotdqY3MxtfWoSKQ7CC2iXZDX
tHwlTxFWMMS2\n" \
"RJ17LJ3lXubvDGGqv+QqG+6EnriDfcFDzkSnE3ANkR/0yBOtg2DZ
2HKocyQetawi\n" \
"DsoXiWJYRBuriSUBAA/NxBti21G00w9RKpv0vHP8ds42pM3Z2Czq
rpv1KrKQ0U11\n" \
```

```
"GIo/ikGQI31bS/6kA1ibRrLDYGCD+H1QQc7CoZDDu+8CL9IVVO5E
FdkKrqeKM+2x\n" \
"LXY2JtwE65/3YR8V3Idv7kaWKK2hJn0KCacuBKONvPi8BDAB\n"
\
"-----END CERTIFICATE-----\n" \
"-----BEGIN CERTIFICATE-----\n" \
"MIIDxTCCAq2gAwIBAgIBADANBgkqhkiG9w0BAQsFADCBgzELMAkG
A1UEBhMCVVMx\n" \
"EDAOBgNVBAgTB0FyaXpvbmExEzARBgNVBAcTClNjb3R0c2RhbGUx
GjAYDVQQKExFHb0RhZGR5LmNvbSwgSW5jLjExMC8GA1UEAxMoR28g
\n" \
"RGFkZHkgUm9vdCBDZXJ0aWZpY2F0ZSBBdXRob3JpdHkgLSBHMjAe
Fw0wOTA5MDExMDAwMDBaFw0zNzEyMzEyMzU5NTlaMIGDMRMwEQYD\
n" \
"VQQDEwpzY290dHNkYWxlMRowGAYDVQQKExFHb0RhZGR5LmNvbSwg
SW5jLjExMC8GA1UEAxMoR28gRGFkZHkgUm9vdCBDZXJ0aWZpY2F0\
n" \
"ZSBBdXRob3JpdHkgLSBHMjCCASIwDQYJKoZIhvcNAQEBBQADggEP
ADCCAQoCggEBAL9xYgjx+lk09xvJGKP3gElY6SKDE6bFIEMBO4Tx\
n" \
"5oVJnyfq9oQbTqC023CYxzIBsQU+B07u9PpPL1kwIuerGVZr4oAH
/PMWdYA5UXvl+TW2dE6pjYIT5LY/qQOD+qK+ihVqf94Lw7YZFAXK\
n" \
"6sOoBJQ7RnwyDfMAZiLIjWltNowRGLfTshxgtDj6AozO091GB94K
PutdfMh8+7ArU6SSYmlRJQVhGkSBjCypQ5Yj36w6gZoOKcUcqeldH
\n" \
"raenjAKOc7xiID7S13MMuyFYkMlNAJWJwGRtDtwKj9useiciAF9n
9T521NtYJ2/LOdYq7hfRvzOxBsDPAnrSTFcaUaz4EcCAwEAASaNC\
n" \
"MEAwDwYDVR0TAQH/BAUwAwEB/zAOBgYDVR0PAQH/BAQDAgEGMBAw
DgYDVR0lAQH/BAoDATAbBgNVHSUEFDASBglghkgBhv1tAQcXBggr\
n" \
"BgEFBQcDAQYHd3d3LmdvZGFkZHkuY29tMA4GA1UdDwEB/wQEAwIB
AjAdBgNVHQ4EFgQUOpqFBxBnKLbv9r0FQW4gwZTaD94wHwYDVR0j\
n" \
"BBgwFoAUOpqFBxBnKLbv9r0FQW4gwZTaD94wNAYIKwBBQUHAQEEK
DAmMCQGCCsGAQUFBzABhhhodHRwOi8vb2NzcC5nb2RhZGR5LmNv\n
" \
```

```
"bS8wNQYDVR0fBC4wLDAqoCigJoYkaHR0cDovL2NybC5nb2RhZGR5
LmNvbS9nZHJvb3QtZzIuY3JsMEYGA1UdIAQ/MD0wOwYEVR0gADAz\
n" \
"MDEGCCsGAQUFBwIBFiVodHRwczovL2NlcnRzLmdvZGFkZHkuY29t
L3JlcG9zaXRvcnkvMA0GCSqGSIb3DQEBCwUAA4IBAQCZ21151fmX\
n" \
"WWcDYfF+OwYxdS2hII5PZYe096acvNjpL9DbWu7PdIxztDhC2gV7
+AJ1uP2lsdeu\n" \
"9tfeE8tTEH6KRtGX/rcuKxGrkLAngPnon1rpN5+r5N9ss4UXnT3Z
JE95kTXWXwTr\n" \
"gIOrmgIttRD02JDHBHNA7XIloKmf7J6raBKZV8aPEjoJpL1E/QYV
N8Gb5DKj7Tjo\n" \
"2GTzLH4U/ALqn83/B2gX2yKQOC16jdFU8WnjXzPKej17CuPKf185
5eJ1usV2GDPO\n" \
"LPAvTK33sefOT6jEm0pUBsV/fdUID+Ic/n4XuKxe9tQWskMJDE32
p2u0mYRlynqI\n" \
"4uJEvlz36hz1\n" \
"-----END CERTIFICATE-----\n";

static uint8_t mqtt_pingreq_packet[] = {0xC0, 0x00};
// PINGREQ packet (Fixed header 0xC0, Remaining
Length 0x00)

// LOCATION: After your existing root_ca definition,
and before the mbedTLS global structs.

// Global for the publish counter and packet ID
static int publish_counter = 0;
static uint16_t current_packet_id = 1; // Start
packet IDs from 1 for SUBSCRIBE/PUBACK/SUBACK
// Buffer for "dataX ok" or JSON string
static bool payload_type_json = false; // true for
JSON, false for raw

// Global flags and timers for MQTT state
static bool mqtt_connected = false;
static unsigned long last_mqtt_activity = 0; //
Timestamp of last sent/received MQTT packet
```

```
const unsigned long MQTT_KEEP_ALIVE_INTERVAL_MS = 50
* 1000; // Send PINGREQ after 50 seconds of
inactivity (broker's keep-alive is 60s)
const unsigned long PUBLISH_INTERVAL_MS = 5 * 1000;
// Publish every 5 seconds
static unsigned long last_publish_time = 0;

// Global for the publish counter and packet ID

// Start packet IDs from 1 for
SUBSCRIBE/PUBACK/SUBACK
char dynamic_payload_buffer[100]; // Buffer for
"dataX ok" or JSON string

 // Timestamp of last sent/received MQTT packet
 // Send PINGREQ after 50 seconds of inactivity
(broker's keep-alive is 60s)
 // Publish every 5 seconds

// Globals for TLS
mbedtls_ssl_context ssl;
mbedtls_ssl_config conf;
mbedtls_ctr_drbg_context ctr_drbg;
mbedtls_entropy_context entropy;
mbedtls_x509_crt cacert;

// Global EthernetClient to be used by mbedTLS BIO
functions
EthernetClient ethClient;

void handle_incoming_mqtt_packet() {
    int ret;

    // Try to read data from the SSL stream
    // Read directly into the buffer, starting after
any previously read data
    ret = mbedtls_ssl_read(&ssl, mqtt_recv_buffer +
mqtt_recv_buffer_len, MQTT_RECV_BUFFER_SIZE -
mqtt_recv_buffer_len);
```

```
    if (ret > 0) {
        mqtt_recv_buffer_len += ret;
        last_mqtt_activity = millis(); // Update
activity on any incoming data

        // --- Start of very basic MQTT packet
parsing ---
        // This is a minimal parser for PUBLISH (QoS
0) and PINGRESP
        // A full MQTT parser is significantly more
complex and would handle
        // fragmented packets, various control packet
types, etc.

        while (mqtt_recv_buffer_len >= 2) { // Need
at least fixed header (1 byte) + remaining length
(min 1 byte)
            uint8_t fixed_header =
mqtt_recv_buffer[0];
            uint8_t packet_type = (fixed_header >> 4)
& 0x0F;
            // uint8_t qos_flags = (fixed_header >>
1) & 0x03; // For PUBLISH QoS

            // Decode Remaining Length (variable byte
integer)
            int remaining_length = 0;
            int multiplier = 1;
            int i = 1; // Start checking from second
byte
            uint8_t encoded_byte;
            do {
                if (i >= mqtt_recv_buffer_len) {
                    // Not enough bytes for remaining
length, wait for more data
                    return;
                }
                encoded_byte = mqtt_recv_buffer[i++];
                remaining_length += (encoded_byte &
127) * multiplier;
```

```
                multiplier *= 128;
                if (multiplier > 128*128*128) { // Max 4 bytes for remaining length (should not happen for normal messages)
                    Serial.println("❌ Remaining length encoding error or packet too large.");
                    mqtt_connected = false; // Treat as fatal error
                    return;
                }
            } while ((encoded_byte & 128) != 0);

            int total_packet_length = i + remaining_length; // i is the bytes used for fixed header + remaining length

            if (mqtt_recv_buffer_len < total_packet_length) {
                // Not enough data for the full packet, wait for more
                // (This assumes the buffer is large enough for a single max packet)
                return;
            }

            // Process the full packet
            Serial.print("Received MQTT Packet Type: 0x");
            Serial.print(packet_type, HEX);
            Serial.print(", Total Length: ");
            Serial.println(total_packet_length);

            if (packet_type == 0x03) { // PUBLISH (type 3)
                Serial.println("  -> PUBLISH Packet detected!");

                int current_idx = i; // Start after fixed header and remaining length bytes
```

```
                // Parse Topic
                uint16_t topic_len =
(mqtt_recv_buffer[current_idx] << 8) |
mqtt_recv_buffer[current_idx + 1];
                current_idx += 2;

                if (current_idx + topic_len >
total_packet_length) {
                    Serial.println("❌ Malformed
PUBLISH: Topic length extends beyond packet.");
                    mqtt_connected = false;
                    return;
                }

                char topic_name[topic_len + 1]; // +1
for null terminator
                memcpy(topic_name,
&mqtt_recv_buffer[current_idx], topic_len);
                topic_name[topic_len] = '\0';
                current_idx += topic_len;

                Serial.print("    Topic: ");
                Serial.println(topic_name);

                // Handle QoS 1/2 Packet ID if
necessary (skip for QoS 0 for now)
                uint8_t qos = (fixed_header >> 1) &
0x03;
                if (qos > 0) {
                    uint16_t publish_packet_id =
(mqtt_recv_buffer[current_idx] << 8) |
mqtt_recv_buffer[current_idx + 1];
                    current_idx += 2;
                    Serial.print("    Packet ID (QoS
> 0): ");
                    Serial.println(publish_packet_id
);
                    // TODO: For QoS 1, send PUBACK
here if you want to support QoS 1
```

```
                    // You would need a
send_puback(packet_id) function
                }

                // Parse Payload
                int payload_len = total_packet_length
- current_idx;
                if (payload_len < 0) {
                    Serial.println("❌ Malformed
PUBLISH: Negative payload length.");
                    mqtt_connected = false;
                    return;
                }

                // Check if payload_len is too large
for the buffer
                if (payload_len + 1 >
sizeof(mqtt_recv_buffer)) { // Simple check, assuming
payload is <= total buffer
                    Serial.println("❌ Payload too
large to fit in temporary buffer.");
                    // You might choose to truncate
or handle differently
                }

                char payload[payload_len + 1]; // +1
for null terminator
                memcpy(payload,
&mqtt_recv_buffer[current_idx], payload_len);
                payload[payload_len] = '\0';

                Serial.print("    Payload: ");
                Serial.println(payload);

                // --- YOUR APPLICATION LOGIC GOES
HERE ---
                // This is where you decide what to
do with the received message.
                // Example: Control an LED based on
topic/payload
```

```
                if (strcmp(topic_name,
"MyESP32/Test/Status") == 0) {
                    if (strcmp(payload, "ON") == 0) {
                        Serial.println("  -> Received
'ON' command for status topic!");
                        // Add your code here, e.g.:
digitalWrite(LED_BUILTIN, HIGH);
                    } else if (strcmp(payload, "OFF")
== 0) {
                        Serial.println("  -> Received
'OFF' command for status topic!");
                        // Add your code here, e.g.:
digitalWrite(LED_BUILTIN, LOW);
                    } else {
                        Serial.print("  ->
Unrecognized command for status topic: ");
                        Serial.println(payload);
                    }
                }
                // You can add more topic checks
here, e.g.:
                // if (strcmp(topic_name,
"your/other/topic") == 0) { ... }

            } else if (packet_type == 0x0D) { //
PINGRESP (type 13)
                Serial.println("  -> PINGRESP Packet
detected. Good!");
            } else if (packet_type == 0x09) { //
SUBACK (type 9) - Already handled in
subscribe_mqtt_topic, but good to know
                Serial.println("  -> SUBACK Packet
detected (already handled by
subscribe_mqtt_topic).");
            }
            // Add other packet types (CONNACK,
PUBACK, etc.) as needed for a full client

            // Remove the processed packet from the
buffer by shifting remaining data
```

```
            memmove(mqtt_recv_buffer,
mqtt_recv_buffer + total_packet_length,
mqtt_recv_buffer_len - total_packet_length);
            mqtt_recv_buffer_len -=
total_packet_length;
        }

    } else if (ret == MBEDTLS_ERR_SSL_WANT_READ ||
ret == MBEDTLS_ERR_SSL_WANT_WRITE) {
        // No data available yet, or underlying
transport wants to write. This is normal.
        // Serial.println("No data to read
(WANT_READ/WANT_WRITE)"); // Uncomment for verbose
debugging
    } else if (ret == 0) {
        // Connection closed by peer
        Serial.println("MQTT connection closed by
broker (mbedtls_ssl_read returned 0).");
        mqtt_connected = false; // Mark as
disconnected so MqttTask can try to reconnect
    } else {
        // Real error
        print_mbedtls_error(ret);
        Serial.print("Error reading from SSL stream:
");
        Serial.println(ret);
        mqtt_connected = false; // Mark as
disconnected
    }
}
// --- Function Prototypes ---
void print_mbedtls_error(int ret);
static int mbedtls_custom_send(void *ctx, const
unsigned char *buf, size_t len);
static int mbedtls_custom_recv(void *ctx, unsigned
char *buf, size_t len);
bool mqtt_tls_connect(); // Renamed and refactored
bool publish_mqtt_message(const char* topic, const
char* payload_str);
```

```
bool subscribe_mqtt_topic(const char* topic, uint8_t
qos);
void handle_incoming_mqtt_packet();
void MqttTask(void *pvParameters); // Main FreeRTOS
task

// Debug callback for mbedTLS (optional, uncomment
#define MBEDTLS_DEBUG_C to enable)
static void mbedtls_debug( void *ctx, int level,
                           const char *file, int
line,
                           const char *str )
{
    ((void) ctx);
    ((void) file);
    ((void) line);
    Serial.printf( "MBEDTLS DEBUG %d: %s", level, str
);
}

void print_mbedtls_error(int ret) {
    char errbuf[256];
    mbedtls_strerror(ret, errbuf, sizeof(errbuf));
    Serial.print("mbedTLS error: ");
    Serial.println(errbuf);
}

// Custom mbedTLS BIO Send Callback
static int mbedtls_custom_send(void *ctx, const
unsigned char *buf, size_t len) {
    EthernetClient *client = (EthernetClient *)ctx;
    if (!client->connected()) {
        return MBEDTLS_ERR_NET_SEND_FAILED;
    }
    size_t written = client->write(buf, len);
    if (written == 0 && len > 0) {
        if (client->connected()) {
            return MBEDTLS_ERR_SSL_WANT_WRITE;
        } else {
            return MBEDTLS_ERR_NET_SEND_FAILED;
```

```
        }
    }
    return written;
}

// Custom mbedTLS BIO Receive Callback
static int mbedtls_custom_recv(void *ctx, unsigned
char *buf, size_t len) {
    EthernetClient *client = (EthernetClient *)ctx;
    if (!client->connected() && client->available()
== 0) {
        return MBEDTLS_ERR_NET_RECV_FAILED;
    }
    int read_len = client->read(buf, len);
    if (read_len == 0 && len > 0) {
        if (client->connected()) {
            return MBEDTLS_ERR_SSL_WANT_READ;
        } else {
            return MBEDTLS_ERR_NET_RECV_FAILED;
        }
    }
    return read_len;
}

// --- Refactored: Now only connects TLS and MQTT,
does not publish ---
bool mqtt_tls_connect() {
    int ret;

    // Initialize mbedTLS structures (only if not
already initialized or after a disconnect cleanup)
    mbedtls_ssl_init(&ssl);
    mbedtls_ssl_config_init(&conf);
    mbedtls_ctr_drbg_init(&ctr_drbg);
    mbedtls_entropy_init(&entropy);
    mbedtls_x509_crt_init(&cacert);

    // Seed the random number generator
    ret = mbedtls_ctr_drbg_seed(&ctr_drbg,
mbedtls_entropy_func, &entropy, NULL, 0);
```

```
    if (ret != 0) {
        print_mbedtls_error(ret);
        return false;
    }

    // Parse the CA certificate
    ret = mbedtls_x509_crt_parse(&cacert, (const
unsigned char*)root_ca, strlen(root_ca)+1);
    if (ret < 0) {
        print_mbedtls_error(ret);
        return false;
    }

    // Connect EthernetClient
    Serial.print("Connecting EthernetClient to MQTT
broker: ");
    Serial.print(mqtt_host);
    Serial.print(":");
    Serial.println(mqtt_port);
    if (!ethClient.connect(mqtt_host, mqtt_port)) {
        Serial.println("❌ EthernetClient failed to
connect to MQTT broker!");
        return false;
    }
    Serial.println("☑ EthernetClient connected to
MQTT broker.");

    // Configure TLS
    mbedtls_ssl_config_defaults(&conf,
        MBEDTLS_SSL_IS_CLIENT,
        MBEDTLS_SSL_TRANSPORT_STREAM,
        MBEDTLS_SSL_PRESET_DEFAULT);

    mbedtls_ssl_conf_authmode(&conf,
MBEDTLS_SSL_VERIFY_REQUIRED);
    mbedtls_ssl_conf_ca_chain(&conf, &cacert, NULL);
    mbedtls_ssl_conf_rng(&conf,
mbedtls_ctr_drbg_random, &ctr_drbg);
    mbedtls_ssl_setup(&ssl, &conf);
```

```
    // If MBEDTLS_DEBUG_C is defined, uncomment the
next two lines for mbedTLS debug output
    // mbedtls_ssl_conf_dbg(&conf, mbedtls_debug,
NULL);
    // mbedtls_debug_set_threshold(1);
    mbedtls_ssl_set_hostname(&ssl, mqtt_host);

    // Set custom BIO functions
    mbedtls_ssl_set_bio(&ssl, &ethClient,
mbedtls_custom_send, mbedtls_custom_recv, NULL);

    // TLS handshake
    Serial.println("[🤝] Performing TLS
handshake...");
    while ((ret = mbedtls_ssl_handshake(&ssl)) != 0)
{
        if (ret != MBEDTLS_ERR_SSL_WANT_READ && ret
!= MBEDTLS_ERR_SSL_WANT_WRITE) {
            print_mbedtls_error(ret);
            Serial.print("TLS Handshake failed with
raw error: ");
            Serial.println(ret, HEX);
            return false;
        }
        vTaskDelay(10 / portTICK_PERIOD_MS); // Yield
to other tasks

        if (ethClient.available()) {
            Serial.print("TLS Handshake: Bytes
available from broker: ");
            Serial.println(ethClient.available());
        }
    }
    Serial.println("☑ TLS connected.");
    last_mqtt_activity = millis(); // Update last
activity time after TLS connect

    // --- Build MQTT CONNECT packet with username &
password ---
```

```
    const char* client_id =
"MyUniqueESP32_Device_001";
    const char* username = mqtt_user;
    const char* password = mqtt_pass;

    int client_id_len = strlen(client_id);
    int username_len = strlen(username);
    int password_len = strlen(password);

    // Corrected remaining length calculation for
MQTT CONNECT
    int remaining_length = 10 + (2 + client_id_len) +
(2 + username_len) + (2 + password_len);

    int i = 0;
    mqtt_connect_packet[i++] = 0x10;   // CONNECT
    mqtt_connect_packet[i++] = remaining_length;

    // Protocol Name: "MQTT"
    mqtt_connect_packet[i++] = 0x00;
    mqtt_connect_packet[i++] = 0x04;
    mqtt_connect_packet[i++] = 'M';
    mqtt_connect_packet[i++] = 'Q';
    mqtt_connect_packet[i++] = 'T';
    mqtt_connect_packet[i++] = 'T';

    // Protocol Level: 4 (MQTT 3.1.1)
    mqtt_connect_packet[i++] = 0x04;

    // Connect Flags: Clean session + username +
password
    mqtt_connect_packet[i++] = 0xC2;

    // Keep Alive (60 seconds)
    mqtt_connect_packet[i++] = 0x00;
    mqtt_connect_packet[i++] = 0x3C; // 60 seconds

    // Client ID
    mqtt_connect_packet[i++] = 0x00;
    mqtt_connect_packet[i++] = client_id_len;
```

```
    memcpy(&mqtt_connect_packet[i], client_id,
client_id_len);
    i += client_id_len;

    // Username
    mqtt_connect_packet[i++] = 0x00;
    mqtt_connect_packet[i++] = username_len;
    memcpy(&mqtt_connect_packet[i], username,
username_len);
    i += username_len;

    // Password
    mqtt_connect_packet[i++] = 0x00;
    mqtt_connect_packet[i++] = password_len;
    memcpy(&mqtt_connect_packet[i], password,
password_len);
    i += password_len;

    // Debug: Print the raw MQTT CONNECT packet
before sending
    Serial.print("Sending MQTT CONNECT Packet (Bytes:
");
    Serial.print(i);
    Serial.print("): ");
    for (int k = 0; k < i; k++) {
        if (mqtt_connect_packet[k] < 0x10)
Serial.print("0");
        Serial.print(mqtt_connect_packet[k], HEX);
        Serial.print(" ");
    }
    Serial.println();

    // Send the MQTT CONNECT packet
    ret = mbedtls_ssl_write(&ssl,
mqtt_connect_packet, i);
    if (ret < 0) {
        print_mbedtls_error(ret);
        return false;
    }
    Serial.println("☑ MQTT CONNECT sent.");
```

```
    last_mqtt_activity = millis(); // Update last
activity time after CONNECT sent

    unsigned char connack[4];
    // Read with a timeout (e.g., 5 seconds) to
prevent blocking indefinitely
    unsigned long startTime = millis();
    int bytesRead = 0;
    while (bytesRead < sizeof(connack) && (millis() -
startTime < 5000)) {
        int currentRead = mbedtls_ssl_read(&ssl,
connack + bytesRead, sizeof(connack) - bytesRead);
        if (currentRead > 0) {
            bytesRead += currentRead;
        } else if (currentRead !=
MBEDTLS_ERR_SSL_WANT_READ && currentRead !=
MBEDTLS_ERR_SSL_WANT_WRITE && currentRead != 0) {
            print_mbedtls_error(currentRead);
            return false;
        }
        vTaskDelay(10 / portTICK_PERIOD_MS); // Yield
    }

    if (bytesRead < sizeof(connack)) {
        Serial.print("❌ MQTT CONNACK read timeout or
partial read. Read ");
        Serial.print(bytesRead);
        Serial.println(" bytes.");
        Serial.print("Partial received bytes: ");
        for (int k = 0; k < bytesRead; k++) {
            if (connack[k] < 0x10) Serial.print("0");
            Serial.print(connack[k], HEX);
            Serial.print(" ");
        }
        Serial.println();
        return false;
    }

    // --- Check CONNACK response ---
    if (connack[0] != 0x20 || connack[1] != 0x02) {
```

```
        Serial.print("❌ MQTT CONNACK malformed or
unexpected. Received bytes: ");
        Serial.print(connack[0], HEX); Serial.print("
");
        Serial.print(connack[1], HEX); Serial.print("
");
        Serial.print(connack[2], HEX); Serial.print("
");
        Serial.println(connack[3], HEX);
        return false;
    } else if (connack[3] != 0x00) {
        Serial.print("❌ MQTT CONNACK refused. Return
code: ");
        Serial.println(connack[3], HEX);
        return false;
    }
    else {
        Serial.println("☑ MQTT CONNACK received.
Connection Accepted!");
        last_mqtt_activity = millis(); // Update last
activity time after CONNACK received
        return true; // Connection successful
    }
}

// --- New function to publish an MQTT message ---
bool publish_mqtt_message(const char* topic, const
char* payload_str) {
    int ret;
    uint8_t topic_len = strlen(topic);
    uint8_t payload_len = strlen(payload_str);

    // Check if payload fits in buffer (2 bytes for
topic length, topic, payload)
    if (2 + topic_len + payload_len >
sizeof(mqtt_publish_packet) - 2) { // -2 for Fixed
Header + Remaining Length byte
        Serial.println("❌ Payload too large for MQTT
publish packet buffer!");
```

```
        return false;
    }

    int j = 0;
    mqtt_publish_packet[j++] = 0x30; // PUBLISH
command (QoS 0, no DUP, no RETAIN)
    // Remaining length: 2 bytes for topic length,
topic_len bytes for topic, payload_len bytes for
payload
    mqtt_publish_packet[j++] = 2 + topic_len +
payload_len;
    mqtt_publish_packet[j++] = 0x00;
mqtt_publish_packet[j++] = topic_len; // Topic Length
(MSB, LSB)
    memcpy(&mqtt_publish_packet[j], topic,
topic_len); j += topic_len;     // Topic
    memcpy(&mqtt_publish_packet[j], payload_str,
payload_len); j += payload_len; // Payload

    Serial.print("Sending MQTT PUBLISH to '");
    Serial.print(topic);
    Serial.print("' payload: '");
    Serial.print(payload_str);
    Serial.println("'");

    ret = mbedtls_ssl_write(&ssl,
mqtt_publish_packet, j);
    if (ret < 0) {
        print_mbedtls_error(ret);
        Serial.println("❌ Secure publish failed.");
        return false;
    }
    Serial.println("☑ MQTT PUBLISH sent.");
    last_mqtt_activity = millis(); // Update last
activity time after PUBLISH sent
    return true;
}

// --- New function to subscribe to an MQTT topic ---
```

```
bool subscribe_mqtt_topic(const char* topic, uint8_t
qos) {
    int ret;
    uint8_t topic_len = strlen(topic);

    // Remaining length: 2 bytes for Packet ID + 2
bytes for Topic Length + topic_len bytes for Topic +
1 byte for QoS
    int remaining_length = 2 + (2 + topic_len) + 1;

    if (remaining_length >
sizeof(mqtt_subscribe_packet) - 2) { // -2 for Fixed
Header + Remaining Length byte
        Serial.println("❌ Topic too long for MQTT
subscribe packet buffer!");
        return false;
    }

    // Increment packet ID for SUBSCRIBE
    current_packet_id++;
    if (current_packet_id == 0) current_packet_id =
1; // Ensure it never becomes 0

    int i = 0;
    mqtt_subscribe_packet[i++] = 0x82; // SUBSCRIBE
command (Fixed header: 0b10000010 = SUBSCRIBE, QoS 1)
    mqtt_subscribe_packet[i++] = remaining_length; //
Remaining Length

    // Packet Identifier
    mqtt_subscribe_packet[i++] = (current_packet_id
>> 8) & 0xFF; // MSB
    mqtt_subscribe_packet[i++] = current_packet_id &
0xFF;       // LSB

    // Topic Filter
    mqtt_subscribe_packet[i++] = 0x00; // Topic
Length MSB
    mqtt_subscribe_packet[i++] = topic_len; // Topic
Length LSB
```

```
    memcpy(&mqtt_subscribe_packet[i], topic,
topic_len); i += topic_len;

    // Requested QoS (for this topic filter)
    mqtt_subscribe_packet[i++] = qos;

    Serial.print("Sending MQTT SUBSCRIBE to '");
    Serial.print(topic);
    Serial.print("' with QoS ");
    Serial.print(qos);
    Serial.print(" (Packet ID: ");
    Serial.print(current_packet_id);
    Serial.println(")");

    ret = mbedtls_ssl_write(&ssl,
mqtt_subscribe_packet, i);
    if (ret < 0) {
        print_mbedtls_error(ret);
        Serial.println("❌ Secure subscribe
failed.");
        return false;
    }
    Serial.println("☑ MQTT SUBSCRIBE sent.");
    last_mqtt_activity = millis(); // Update last
activity time

    // --- Wait for SUBACK ---
    unsigned char suback[5]; // SUBACK for one topic:
Fixed Header (1) + Remaining Length (1) + Packet ID
(2) + Return Code (1)
    unsigned long startTime = millis();
    int bytesRead = 0;
    while (bytesRead < sizeof(suback) && (millis() -
startTime < 5000)) { // 5-second timeout
        int currentRead = mbedtls_ssl_read(&ssl,
suback + bytesRead, sizeof(suback) - bytesRead);
        if (currentRead > 0) {
            bytesRead += currentRead;
```

```
        } else if (currentRead !=
MBEDTLS_ERR_SSL_WANT_READ && currentRead !=
MBEDTLS_ERR_SSL_WANT_WRITE && currentRead != 0) {
            print_mbedtls_error(currentRead);
            Serial.println("❌ Error reading
SUBACK.");
            return false;
        }
        vTaskDelay(10 / portTICK_PERIOD_MS); // Yield
    }

    if (bytesRead < sizeof(suback)) {
        Serial.print("❌ SUBACK read timeout or
partial read. Read ");
        Serial.print(bytesRead);
        Serial.println(" bytes.");
        return false;
    }

    // Check SUBACK Fixed Header (0x90) and Remaining
Length (0x03)
    if (suback[0] != 0x90 || suback[1] != 0x03) {
        Serial.print("❌ SUBACK malformed or
unexpected header. Received: ");
        Serial.print(suback[0], HEX); Serial.print("
"); Serial.println(suback[1], HEX);
        return false;
    }

    // Check Packet ID
    uint16_t received_packet_id = (suback[2] << 8) |
suback[3];
    if (received_packet_id != current_packet_id) {
        Serial.print("❌ SUBACK Packet ID mismatch.
Expected: ");
        Serial.print(current_packet_id);
        Serial.print(", Received: ");
        Serial.println(received_packet_id);
        return false;
```

```
    }

    // Check Return Code
    if (suback[4] >= 0x80) { // 0x80 indicates
failure
        Serial.print("❌ SUBACK failed for topic '");
        Serial.print(topic);
        Serial.print("'. Return code: ");
        Serial.println(suback[4], HEX);
        return false;
    } else {
        Serial.print("☑ Subscribed to '");
        Serial.print(topic);
        Serial.print("' with granted QoS ");
        Serial.println(suback[4]);
        return true;
    }
}

// --- Handles incoming MQTT packets (PUBLISH,
PINGRESP, etc.) ---


void MqttTask(void *pvParameters) {
    Serial.println("MQTT Task started.");

    // Ensure Ethernet is ready *before* trying to
connect
    while (Ethernet.localIP() == IPAddress(0,0,0,0)
|| Ethernet.linkStatus() == LinkOFF) {
        Serial.println("Waiting for Ethernet link and
IP...");
        vTaskDelay(1000 / portTICK_PERIOD_MS);
    }

    while (true) { // This task will now run
indefinitely
        if (!mqtt_connected) {
            Serial.println("Attempting to connect to
MQTT broker...");
```

```
            // Clean up resources before attempting
to reconnect
            mbedtls_ssl_free(&ssl);
            mbedtls_ssl_config_free(&conf);
            mbedtls_ctr_drbg_free(&ctr_drbg);
            mbedtls_entropy_free(&entropy);
            mbedtls_x509_crt_free(&cacert);
            ethClient.stop(); // Ensure underlying
TCP connection is closed

            if (mqtt_tls_connect()) {
                mqtt_connected = true;
                Serial.println("☑ MQTT Connection
established. Subscribing to topics...");
                // Subscribe to topics immediately
after connecting
                if
(!subscribe_mqtt_topic(mqtt_subscribe_topic, 1)) { //
Subscribe with QoS 1
                    Serial.println("❌ Failed to
subscribe to command topic. Continuing anyway.");
                }
                last_publish_time = millis(); //
Reset publish timer after connection
            } else {
                Serial.println("❌ MQTT Connection
failed. Retrying in 5 seconds...");
                vTaskDelay(5000 /
portTICK_PERIOD_MS); // Wait before retrying
                continue; // Skip to next iteration
of while(true)
            }
        }

        // --- If MQTT is connected, perform
operations ---
        if (mqtt_connected) {
            unsigned long current_time = millis();

            // 1. Handle incoming MQTT packets
```

```
            handle_incoming_mqtt_packet();

            // Check if connection is still valid
after reading
            if (!ethClient.connected()) {
                Serial.println("❌ EthernetClient
disconnected during read. Marking MQTT as
disconnected.");
                mqtt_connected = false;
                continue; // Skip to next loop
iteration to reconnect
            }

            // 2. Publish periodic messages
            if (current_time - last_publish_time >=
PUBLISH_INTERVAL_MS) {
                publish_counter++;

                // Toggle between raw string and JSON
payload every 5 publishes
                if (publish_counter % 5 == 0) {
                    payload_type_json =
!payload_type_json;
                    Serial.print("Switching payload
type to: ");
                    Serial.println(payload_type_json
? "JSON" : "RAW");
                }

                if (payload_type_json) {
                    // Example JSON payload
                    sprintf(dynamic_payload_buffer,
"{\"id\":%d, \"status\":\"ok\", \"type\":\"json\"}",
publish_counter);
                } else {
                    // Raw string payload
                    sprintf(dynamic_payload_buffer,
"data%d ok", publish_counter);
                }
```

```
                if
(!publish_mqtt_message(mqtt_publish_topic,
dynamic_payload_buffer)) {
                    Serial.println("❌ Failed to
publish message. Connection might be lost.");
                    mqtt_connected = false; // Mark
as disconnected if publish fails
                }
                last_publish_time = current_time; //
Reset publish timer
            }

            // 3. Send MQTT PINGREQ for Keep-Alive
            if (current_time - last_mqtt_activity >=
MQTT_KEEP_ALIVE_INTERVAL_MS) {
                Serial.println("Heartbeat: Sending
PINGREQ...");
                int ret = mbedtls_ssl_write(&ssl,
mqtt_pingreq_packet, sizeof(mqtt_pingreq_packet));
                if (ret < 0) {
                    print_mbedtls_error(ret);
                    Serial.println("❌ Failed to send
PINGREQ. Disconnecting.");
                    mqtt_connected = false; // Mark
as disconnected
                } else {
                    Serial.println("☑ PINGREQ
sent.");
                    last_mqtt_activity =
current_time; // Reset activity timer
                }
            }
        }
        vTaskDelay(100 / portTICK_PERIOD_MS); //
Small delay to yield CPU and prevent busy-waiting
    }
    // This part will theoretically never be reached
because of while(true)
    // vTaskDelete(NULL);
}
```

```
void setup() {
    Serial.begin(115200);

    Ethernet.init(ENC_CS);
    Serial.println("Initializing Ethernet...");
    Ethernet.begin(mac, ip, dnsIP, gateway, subnet);
    delay(2000); // Give plenty of time for Ethernet
to initialize

    Serial.println("📡 ENC28J60 started. IP: " +
Ethernet.localIP().toString());

    if (Ethernet.linkStatus() == LinkOFF) {
        Serial.println("❌ Ethernet cable not
connected! Please connect and restart.");
        // In a real application, you might loop here
until connected or reboot.
        while(true) { vTaskDelay(1000 /
portTICK_PERIOD_MS); } // Halt if no cable
    }

    // Increased stack size as mbedTLS can be stack-
heavy
    const uint32_t STACK_SIZE_IN_BYTES = 32768; //
32KB
    const uint32_t STACK_SIZE_IN_WORDS =
STACK_SIZE_IN_BYTES / sizeof(StackType_t);

    xTaskCreate(
        MqttTask,            // Task function
        "MQTT_TLS_Task",     // Name of task
        STACK_SIZE_IN_WORDS, // Stack size in words
        NULL,                // Parameters to pass to
function
        5,                   // Task priority
        NULL                 // Task handle
    );
```

```
    Serial.println("MQTT_TLS_Task created. Main
setup() continues.");
}

void loop() {
    // The main logic is now handled by the MqttTask
FreeRTOS task.
    // This loop can be left empty or used for other
non-critical, low-frequency tasks.
    vTaskDelay(100 / portTICK_PERIOD_MS); // Yield to
other tasks
}
```

Author: Jayant Kumar
Platform: ESP32 + ENC28J60 + mbedTLS + FreeRTOS
Broker: otplai.com:8883