**Project Title:** Smart RFID Door Lock with ESP32 and MQTT

**Overview:**  
This project aims to create a secure, connected smart door lock system using an ESP32, RFID authentication, and MQTT communication. The system will allow unlocking via RFID tags, MQTT commands, or a manual switch, with real-time status feedback on an OLED display and audible signals from a buzzer. It will also support remote RFID tag management through a web interface, while still functioning offline using locally stored tag data.

**Objectives:**

1. **Access Control**
   * Unlock the door using an authorized RFID tag.
   * Unlocking the door via MQTT client commands.
   * Unlock the door using a physical switch inside the home.
   * Continue RFID unlocking functionality even without Wi-Fi, using tag data stored in EEPROM.
2. **Tag Management**
   * Register new RFID tags via an MQTT command.
   * Add or remove RFID tags stored in ESP32’s EEPROM through MQTT.
   * Fetch stored RFID tag information from EEPROM via MQTT.
3. **User Feedback & Status Display**
   * Show lock status and messages on an OLED display.
   * Use a buzzer with different beep patterns to indicate:
     + RFID authentication success or failure.
     + Successful unlock via MQTT or switch.
   * Use a green LED to indicate Wi-Fi connectivity.
4. **Network Configuration & Reliability**
   * If Wi-Fi is disconnected, allow reconfiguration via Wi-Fi Manager.
   * Ensure MQTT connection reliability for remote control and tag management.

**Key Features:**

* Secure local and remote unlocking options.
* Works fully offline for RFID access using EEPROM storage.
* Visual and audible user feedback.
* Web-based RFID tag management.
* Easy Wi-Fi credential updates without reprogramming.

Component Pin Connection:

1. ESP32 WROOM32 dev board

2. Relay connected to pin D26 of ESP32

3. Buzzer connected to pin D25 of ESP32

4. Green led(wifi led) connected to pin D14 of ESP32

5. Oled connected to I2C of ESP32 (SCK->D22 and SDA->D21)

6. RFID reader connected to SPI of ESP32 (RST->D15, MISO->D19, MOSI->D23, SCK->D18, SDA/SS->D5)

7. Switch connected to pin D27 of ESP32

Arduino Code:

/\* Smart RFID Lock — add/delete/fetch via MQTT

   Topics:

     - Commands (web -> ESP32):   smartlock/esp32/cmd

         payloads: "addTag" | "deleteTag:<UID>" | "fetchTags" | "OPEN" ...

     - Events (ESP32 -> web):     smartlock/esp32/events

         payloads: "tagAdded:<UID>" | "tagAddFailed" | "tagDeleted:<UID>" | "tagDeleteFailed"

     - Tags (ESP32 -> web):       smartlock/esp32/tags

         payload: JSON: {"count":N,"tags":["UID1","UID2",...]}

\*/

#include <vector>

#include <WiFi.h>

#include <PubSubClient.h>

#include <SPI.h>

#include <MFRC522.h>

#include <Preferences.h>

#include <Wire.h>

#include <Adafruit\_SSD1306.h>

// ----------------- CONFIG -----------------

const char \*WIFI\_SSID = "YOUR\_SSID";

const char \*WIFI\_PASS = "YOUR\_PASS";

// MQTT

String mqttServer = "broker.hivemq.com";  // override if needed

uint16\_t mqttPort = 1883;

const char \*TOPIC\_CMD = "smartlock/esp32/cmd";

const char \*TOPIC\_EVENTS = "smartlock/esp32/events";

const char \*TOPIC\_TAGS = "smartlock/esp32/tags";

WiFiClient wifiClient;

PubSubClient mqttClient(wifiClient);

// Pins (adjust if needed)

#define RELAY\_PIN 26  // safe pin for relay

#define BUZZER\_PIN 25

#define WIFI\_LED\_PIN 14

#define SWITCH\_PIN 27

// RFID (MFRC522) VSPI pins

#define RST\_PIN 15

#define SS\_PIN 5

#define MOSI\_PIN 23

#define MISO\_PIN 19

#define SCK\_PIN 18

// OLED (optional)

#define OLED\_SDA 21

#define OLED\_SCL 22

#define SCREEN\_WIDTH 128

#define SCREEN\_HEIGHT 64

Adafruit\_SSD1306 display(SCREEN\_WIDTH, SCREEN\_HEIGHT, &Wire);

// MFRC522

MFRC522 mfrc522(SS\_PIN, RST\_PIN);

// Preferences (NVS) storage

Preferences prefs;

const char \*PREF\_NS = "rfid\_v1";

const char \*PREF\_KEY\_COUNT = "count";

// Limits

const uint8\_t MAX\_TAGS = 50;

// Add-mode behavior

bool addMode = false;

unsigned long addModeStart = 0;

const unsigned long ADD\_MODE\_TIMEOUT\_MS = 20000UL;  // wait 20s for a scanned card

// Helpers

unsigned long lastReconnectAttempt = 0;

// ----------------- Constants used earlier -----------------

const unsigned long UNLOCK\_DURATION\_MS = 5000UL;

// ----------------- Forward declarations -----------------

void setupWiFi();

void mqttCallback(char \*topic, byte \*payload, unsigned int length);

bool mqttConnect();

String uidToString(MFRC522::Uid uid);

bool tagExists(const String &uid);

bool addTagToStorage(const String &uid);

bool removeTagFromStorage(const String &uid);

void publishTagsJSON();

void publishEvent(const char \*msg);

void beepSuccess();

void beepFail();

void enterAddMode();

void checkAddMode();

String readTagFromIndex(int idx);

void saveTagAtIndex(int idx, const String &uid);

int getTagCount();

void setTagCount(int n);

// ----------------- Setup -----------------

void setup() {

  Serial.begin(115200);

  delay(200);

  pinMode(RELAY\_PIN, OUTPUT);

  digitalWrite(RELAY\_PIN, LOW);  // locked

  pinMode(BUZZER\_PIN, OUTPUT);

  digitalWrite(BUZZER\_PIN, LOW);

  pinMode(WIFI\_LED\_PIN, OUTPUT);

  digitalWrite(WIFI\_LED\_PIN, LOW);

  pinMode(SWITCH\_PIN, INPUT\_PULLUP);

  // OLED init (optional)

  Wire.begin(OLED\_SDA, OLED\_SCL);

  if (display.begin(SSD1306\_SWITCHCAPVCC, 0x3C)) {

    display.clearDisplay();

    display.setTextSize(1);

    display.setTextColor(SSD1306\_WHITE);

    display.setCursor(0, 0);

    display.println("Smart Lock");

    display.display();

  }

  // RFID

  SPI.begin(SCK\_PIN, MISO\_PIN, MOSI\_PIN, SS\_PIN);

  mfrc522.PCD\_Init();

  // Preferences NVS

  prefs.begin(PREF\_NS, false);

  if (prefs.getUInt(PREF\_KEY\_COUNT, UINT32\_MAX) == UINT32\_MAX) {

    prefs.putUInt(PREF\_KEY\_COUNT, 0);

  }

  setupWiFi();

  mqttClient.setServer(mqttServer.c\_str(), mqttPort);

  mqttClient.setCallback(mqttCallback);

  // initial connect attempt

  mqttConnect();

  publishTagsJSON();  // publish existing tags at boot

}

// ----------------- Loop -----------------

void loop() {

  if (!mqttClient.connected()) {

    unsigned long now = millis();

    if (now - lastReconnectAttempt > 5000) {

      lastReconnectAttempt = now;

      mqttConnect();

    }

  } else {

    mqttClient.loop();

  }

  // check add mode scanning

  if (addMode) checkAddMode();

  // physical switch (optional): open door if pressed

  if (digitalRead(SWITCH\_PIN) == LOW) {

    // simple unlock pulse

    digitalWrite(RELAY\_PIN, HIGH);

    delay(2500);

    digitalWrite(RELAY\_PIN, LOW);

    beepSuccess();

    mqttClient.publish(TOPIC\_EVENTS, "OpenedBySwitch");

    delay(300);  // debounce-ish

  }

}

// ----------------- WiFi & MQTT -----------------

void setupWiFi() {

  WiFi.begin(WIFI\_SSID, WIFI\_PASS);

  unsigned long start = millis();

  while (WiFi.status() != WL\_CONNECTED && millis() - start < 20000) {

    digitalWrite(WIFI\_LED\_PIN, !digitalRead(WIFI\_LED\_PIN));

    delay(300);

  }

  if (WiFi.status() == WL\_CONNECTED) {

    digitalWrite(WIFI\_LED\_PIN, HIGH);

    Serial.printf("WiFi connected: %s\n", WiFi.localIP().toString().c\_str());

  } else {

    digitalWrite(WIFI\_LED\_PIN, LOW);

    Serial.println("WiFi not connected");

  }

}

bool mqttConnect() {

  if (WiFi.status() != WL\_CONNECTED) return false;

  if (mqttClient.connected()) return true;

  String clientId = "ESP32Lock-" + String((uint32\_t)ESP.getEfuseMac(), HEX);

  Serial.print("Connecting to MQTT... ");

  if (mqttClient.connect(clientId.c\_str())) {

    Serial.println("connected");

    mqttClient.subscribe(TOPIC\_CMD);

    // publish online

    mqttClient.publish(TOPIC\_EVENTS, "Online");

    publishTagsJSON();

    return true;

  } else {

    Serial.printf("failed rc=%d\n", mqttClient.state());

    return false;

  }

}

// ----------------- MQTT message handling -----------------

void mqttCallback(char \*topic, byte \*payload, unsigned int length) {

  String msg;

  for (unsigned int i = 0; i < length; i++) msg += (char)payload[i];

  msg.trim();

  Serial.printf("MQTT recv on %s: %s\n", topic, msg.c\_str());

  if (String(topic) == TOPIC\_CMD) {

    if (msg.equalsIgnoreCase("addTag")) {

      enterAddMode();

    } else if (msg.startsWith("deleteTag:")) {

      String uid = msg.substring(strlen("deleteTag:"));

      uid.trim();

      if (removeTagFromStorage(uid)) {

        publishEvent(("tagDeleted:" + uid).c\_str());

        publishTagsJSON();

      } else {

        publishEvent("tagDeleteFailed");

      }

    } else if (msg.equalsIgnoreCase("fetchTags")) {

      publishTagsJSON();

    } else if (msg.equalsIgnoreCase("OPEN")) {

      // optional: open door via mqtt

      digitalWrite(RELAY\_PIN, HIGH);

      delay(UNLOCK\_DURATION\_MS);

      digitalWrite(RELAY\_PIN, LOW);

      mqttClient.publish(TOPIC\_EVENTS, "OpenedByMQTT");

    }

  }

}

// ----------------- Add mode -----------------

void enterAddMode() {

  addMode = true;

  addModeStart = millis();

  Serial.println("Entered ADD MODE: waiting for card");

  display.clearDisplay();

  display.setCursor(0, 0);

  display.println("Add mode: Scan tag");

  display.display();

  mqttClient.publish(TOPIC\_EVENTS, "AddModeStarted");

}

void checkAddMode() {

  // If card present, read and add

  if (mfrc522.PICC\_IsNewCardPresent() && mfrc522.PICC\_ReadCardSerial()) {

    String uid = uidToString(mfrc522.uid);

    Serial.printf("Card read during add mode: %s\n", uid.c\_str());

    if (addTagToStorage(uid)) {

      publishEvent(("tagAdded:" + uid).c\_str());

      publishTagsJSON();

      beepSuccess();

    } else {

      publishEvent("tagAddFailed");

      beepFail();

    }

    // exit add mode

    addMode = false;

    mfrc522.PICC\_HaltA();

    delay(300);

    return;

  }

  // timeout check

  if (millis() - addModeStart > ADD\_MODE\_TIMEOUT\_MS) {

    addMode = false;

    Serial.println("Add mode timed out");

    publishEvent("tagAddFailed");

    display.clearDisplay();

    display.setCursor(0, 0);

    display.println("Add timed out");

    display.display();

  }

}

// ----------------- Tag storage using Preferences -----------------

String uidKey(int idx) {

  return String("tag") + String(idx);

}

int getTagCount() {

  return prefs.getUInt(PREF\_KEY\_COUNT, 0);

}

void setTagCount(int n) {

  prefs.putUInt(PREF\_KEY\_COUNT, n);

}

String readTagFromIndex(int idx) {

  String key = uidKey(idx);

  return prefs.getString(key.c\_str(), "");

}

void saveTagAtIndex(int idx, const String &uid) {

  String key = uidKey(idx);

  prefs.putString(key.c\_str(), uid);

}

bool tagExists(const String &uid) {

  int cnt = getTagCount();

  for (int i = 0; i < cnt; ++i) {

    if (readTagFromIndex(i).equalsIgnoreCase(uid)) return true;

  }

  return false;

}

bool addTagToStorage(const String &uid) {

  if (uid.length() == 0) return false;

  if (tagExists(uid)) {

    Serial.println("Tag already exists");

    return false;

  }

  int cnt = getTagCount();

  if (cnt >= MAX\_TAGS) {

    Serial.println("Tag storage full");

    return false;

  }

  saveTagAtIndex(cnt, uid);

  setTagCount(cnt + 1);

  Serial.printf("Stored tag %s at index %d\n", uid.c\_str(), cnt);

  return true;

}

bool removeTagFromStorage(const String &uid) {

  int cnt = getTagCount();

  if (cnt == 0) return false;

  // build new list without uid

  std::vector<String> list;

  for (int i = 0; i < cnt; ++i) {

    String v = readTagFromIndex(i);

    if (!v.equalsIgnoreCase(uid)) list.push\_back(v);

  }

  // rewrite

  for (int i = 0; i < cnt; ++i) {

    String key = uidKey(i);

    prefs.remove(key.c\_str());

  }

  for (size\_t i = 0; i < list.size(); ++i) {

    saveTagAtIndex(i, list[i]);

  }

  setTagCount(list.size());

  Serial.printf("Removed tag %s\n", uid.c\_str());

  return true;

}

// ----------------- Publishing -----------------

void publishEvent(const char \*msg) {

  if (mqttClient.connected()) {

    mqttClient.publish(TOPIC\_EVENTS, msg);

    Serial.printf("Published event: %s\n", msg);

  }

}

void publishTagsJSON() {

  if (!mqttClient.connected()) return;

  int cnt = getTagCount();

  String payload = "{ \"count\": " + String(cnt) + ", \"tags\": [";

  for (int i = 0; i < cnt; ++i) {

    String t = readTagFromIndex(i);

    payload += "\"" + t + "\"";

    if (i < cnt - 1) payload += ",";

  }

  payload += "] }";

  mqttClient.publish(TOPIC\_TAGS, payload.c\_str());

  Serial.printf("Published tags: %s\n", payload.c\_str());

}

// ----------------- Utilities -----------------

String uidToString(MFRC522::Uid uid) {

  String s = "";

  for (byte i = 0; i < uid.size; i++) {

    if (uid.uidByte[i] < 0x10) s += "0";

    s += String(uid.uidByte[i], HEX);

  }

  s.toUpperCase();

  return s;

}

void beepSuccess() {

  digitalWrite(BUZZER\_PIN, HIGH);

  delay(80);

  digitalWrite(BUZZER\_PIN, LOW);

  delay(60);

  digitalWrite(BUZZER\_PIN, HIGH);

  delay(80);

  digitalWrite(BUZZER\_PIN, LOW);

}

void beepFail() {

  digitalWrite(BUZZER\_PIN, HIGH);

  delay(250);

  digitalWrite(BUZZER\_PIN, LOW);

}

Connection Diagram:

