# **Report: Secure MQTT-Based Device Provisioning and OTA Update System**

**Date:** April 08, 2025

## **Overview**

This report analyzes two codebases: a **Python server script** and an **ESP32 firmware** implementation. Together, they form a secure system for provisioning IoT devices and delivering Over-The-Air (OTA) firmware updates via MQTT. The system leverages modern cryptographic techniques and MQTT communication to ensure secure device management and updates.

## **1. What’s Implemented?**

### **1.1 Python Server Script**

The Python script acts as a central server managing MQTT communication with ESP32 devices. Key features include:

* **MQTT Broker Integration:**
  + Connects to an MQTT broker (e.g., EMQX Cloud) using TLS for encrypted communication.
  + Subscribes to topics for provisioning requests, device status, and commands.
  + Publishes OTA updates and commands.
* **Secure Provisioning:**
  + Handles device provisioning requests by generating and encrypting credentials (username/password) using ECDH-derived keys and AES-CBC.
  + Supports unique device identification via IMEI.
* **OTA Firmware Updates:**
  + Signs firmware with ECDSA (SECP256R1 curve) for integrity and authenticity.
  + Splits firmware into chunks, sends them over MQTT, and ensures delivery with acknowledgment retries.
  + Supports device-specific OTA targeting using IMEI.
* **Command Interface:**
  + Allows sending commands (e.g., soft reset, factory reset, firmware rollback) via MQTT.
  + Interactive CLI for manual OTA initiation and device management.

### **1.2 ESP32 Firmware**

The ESP32 firmware runs on the device, interfacing with a SIM7600 modem for cellular connectivity. Key features include:

* **State Machine:**
  + Manages setup stages: modem initialization, network connection, MQTT setup, and provisioning/OTA handling.
* **Secure Provisioning:**
  + Requests credentials from the server using ECDH key exchange and stores them encrypted in NVS (Non-Volatile Storage) with AES.
  + Uses IMEI as a unique identifier.
* **OTA Updates:**
  + Receives firmware chunks, verifies ECDSA signatures and SHA-256 hashes, and applies updates to OTA partitions.
  + Supports rollback to previous firmware or factory reset.
* **MQTT Communication:**
  + Uses AT commands with the SIM7600 modem for MQTT over TLS.
  + Publishes status updates and processes incoming commands.
* **Security Features:**
  + Encrypts sensitive data in NVS using a device-specific key.
  + Implements a watchdog timer (WDT) for reliability.

## 

## 

## 

## 

## 

## 

## 

## **2. Security Advantages**

The system incorporates robust security mechanisms to protect device communication, provisioning, and updates. Here are the key advantages:

### **2.1 End-to-End Encryption**

* **TLS for MQTT:** Both server and ESP32 use TLS to encrypt communication with the MQTT broker, preventing eavesdropping.
* **ECDH Key Exchange:** During provisioning, ECDH (Elliptic Curve Diffie-Hellman) ensures a secure shared secret between server and device, used to encrypt credentials with AES-CBC.
* **NVS Encryption:** ESP32 stores credentials encrypted with a unique device key, protecting against physical access attacks.

### **2.2 Integrity and Authenticity**

* **ECDSA Signatures:** Firmware updates are signed with ECDSA, ensuring they originate from the trusted server and remain untampered.
* **SHA-256 Hashing:** OTA firmware is verified against a hash, detecting any corruption or malicious alteration.

### **2.3 Secure Provisioning**

* **Unique Device IDs:** Using IMEI prevents credential reuse across devices.
* **Nonce Usage:** A random nonce in provisioning prevents replay attacks.
* **Password Strength:** Server enforces a minimum 12-character password, enhancing credential security.

### **2.4 Resilience**

* **Retry Mechanisms:** Both server and ESP32 implement retries for MQTT and OTA operations, ensuring reliability over unreliable networks.
* **Rollback Capability:** ESP32 can revert to previous or factory firmware, mitigating update failures.
* **Watchdog Timer:** Prevents ESP32 lockups, enhancing operational security.

## 

## 

## **3. Structure and Workflow**

### **3.1 Overall System Architecture**

* **Server (Python):** Central hub for device management, running on a host machine.
* **ESP32 Device:** IoT endpoint with SIM7600 modem, connected via cellular network.
* **MQTT Broker:** Intermediate server (e.g., EMQX Cloud) facilitating secure communication.
* **Topics:**
  + dev\_pass\_req/dev\_pass\_res: Provisioning requests and responses.
  + OTA\_Update: Firmware updates.
  + esp32\_status: Device status reports.
  + server\_cmd: Commands from server to device.

### **3.2 Python Server Workflow**

1. **Initialization:**
   * Loads or generates an ECDSA private key for signing.
   * Connects to MQTT broker with TLS.
2. **Provisioning:**
   * Listens for dev\_pass\_req messages containing device IMEI, nonce, and public key.
   * Performs ECDH to derive a shared secret, encrypts credentials, and sends them via dev\_pass\_res.
3. **OTA Updates:**
   * Computes firmware hash and signature.
   * Sends OTA:BEGIN with metadata, followed by chunked firmware data.
   * Tracks acknowledgments (OTA:PROGRESS) and resends missing chunks.
   * Concludes with OTA:END.
4. **Command Handling:**
   * Processes status updates and sends commands (e.g., reset) via server\_cmd.

### **3.3 ESP32 Firmware Workflow**

1. **Setup:**
   * Initializes modem, connects to GPRS, and retrieves IMEI.
   * Loads credentials from NVS or requests new ones if unprovisioned.
2. **Provisioning:**
   * Generates ECDH key pair and nonce, sends request to dev\_pass\_req.
   * Decrypts response from dev\_pass\_res, stores credentials, and restarts MQTT with new credentials.
3. **OTA Updates:**
   * Receives OTA:BEGIN, initializes OTA partition, and verifies firmware metadata.
   * Processes chunks, updates hash, and sends OTA:PROGRESS acknowledgments.
   * Verifies signature and hash on OTA:END, applies update, and restarts.
4. **Running State:**
   * Monitors network/MQTT, responds to commands (e.g., reset), and reports status.

### **3.4 Interaction Flow**

* **Provisioning:** ESP32 → Server (request) → Server → ESP32 (credentials).
* **OTA:** Server → ESP32 (firmware chunks) → ESP32 → Server (acks) → Server → ESP32 (end).
* **Commands:** Server → ESP32 (command) → ESP32 → Server (confirmation).

## **4. Summary**

The implemented system provides a **secure, scalable, and reliable solution** for managing IoT devices. The Python server offers centralized control, while the ESP32 firmware ensures devices operate securely in the field. Security is enhanced through encryption (TLS, AES), authentication (ECDSA), and integrity checks (SHA-256), making it resistant to common attacks like eavesdropping, tampering, and unauthorized access. The modular workflows ensure maintainability and adaptability for future enhancements.

**Recommendation:** This system is ready for deployment with proper testing of network reliability and OTA edge cases (e.g., interrupted updates). Consider adding logging on the server for auditability and refining error handling for production use.