1. Remote Configuration Message Format Specification

You are expected to implement runtime reconfiguration of parameters such as sampling interval and register list. The format used between the cloud and the device must follow this example:

Cloud → Device Message:

```
{
  "config_update": {
    "sampling_interval": 5,
    "registers": ["voltage", "current", "frequency"]
  }
}
```

Device → Cloud Acknowledgment:

```
{
  "config_ack": {
    "accepted": ["sampling_interval", "registers"],
    "rejected": [],
    "unchanged": []
  }
}
```

- Configuration changes **must take effect after the next upload cycle**, without reboot.
- Device must persist changes using **EEPROM or flash** to survive power cycles.
  All configuration updates must be **validated** and **idempotent**.

2. Command Execution Protocol Specification

You are provided with the writable registers and the Inverter SIM API documentation separately (previously made available in Milestone 2). You must implement the full client-server flow for command execution.

Cloud → Device Command Message:

```
{
  "command": {
    "action": "write_register",
    "target_register": "status_flag",
```

```
    "value": 1
  }
}
```

Device → Cloud Execution Result:

```
{
  "command_result": {
    "status": "success",
    "executed_at": "2025-09-04T14:12:00Z"
  }
}
```

- Commands are queued by the cloud and executed by the device **at the next scheduled communication window**.
- Devices must forward the command to the Inverter SIM and return its result.

3. Security Layer Guidelines

You must implement a **lightweight security layer** appropriate for the NodeMCU platform.

Required Features:

- **Authentication and Integrity:**
  Use **HMAC-SHA256** to sign the full message payload with a **pre-shared key (PSK)**.
- **Confidentiality (Optional / Simplified):**
  Simulated or simplified encryption (e.g., AES-CBC or ChaCha20 without Poly1305) is acceptable. Use `Crypto.h`, `BearSSL`, or similar lightweight Arduino libraries.
- You may simulate encryption by encoding payloads in base64 and including a mock-encryption flag, but full HMAC authentication must still be implemented.
- **Anti-Replay Protection:**
  Use a **sequential nonce** or timestamp in each message. The device should store the last seen nonce and reject reused ones.
- Nonces must be stored persistently using EEPROM or flash if the device reboots during FOTA or resets unexpectedly

Example Secured Payload Format:

```
{
  "nonce": 11003,
  "payload": "<base64-encoded or plaintext JSON>",
  "mac": "<HMAC tag>"
}
```

Note: You are **not expected** to implement hardware-secure key storage. PSKs can be hardcoded or stored in flash.

PSKs may be hardcoded as constants in code or stored in EEPROM using `EEPROM.write()` or `Preferences`. If stored, ensure secure access patterns.

4. FOTA Manifest and Chunk Delivery Format

The FOTA (firmware-over-the-air) process must be chunked and support resumption, verification, and rollback. The firmware size must fit within the available dual-bank OTA region of the ESP8266.

Firmware Manifest Example:

```
{
  "version": "1.0.3",
  "size": 32768,
  "hash": "4c87b57c6f42e20e7a4f1b519b4a2f1f",
  "chunk_size": 1024
}
```

Firmware Chunk Message:

```
{
  "chunk_number": 5,
  "data": "<base64-encoded binary>",
  "mac": "<HMAC tag>"
}
```

Device Acknowledgment:

```
{
  "fota_status": {
    "chunk_received": 5,
    "verified": true
  }
}
```

- You may simulate the actual firmware binary using placeholder data.
- You are expected to verify firmware integrity using a **SHA-256 hash**, and simulate rollback if verification fails.
- Use the **ESP8266 OTA API** (`ESP.updateSketch()` or equivalent) for flash writing and reboot handling.

- Devices must reject chunks with invalid HMAC or duplicate chunk numbers and request retransmission. Devices may skip duplicate chunks that were already confirmed.

5. Cloud Communication Example

All communication between EcoWatt Device and EcoWatt Cloud must follow a defined request/response structure. A single cloud endpoint may deliver configuration updates, commands, and firmware chunks.

Device Upload/Status Request:

```
{
  "device_id": "EcoWatt001",
  "status": "ready"
}
```

Cloud Response (example):

```
{
  "config_update": { ... },
  "command": { ... },
  "fota": {
    "manifest": { ... },
    "next_chunk": 3
  }
}
```

- Cloud-side logic must log all changes, commands, and FOTA events in a structured format.