22MIC0181

ADVANCED PYTHON PROGRAMMING – CSI3007

NAME : P SURIYA KUMARI

REG NO: 22MIC0181

IOT – DEVICE APPLICATION

QUANTUM-RESILIENT SMART KEYCHAIN PART – 2

ENERGY SOURCE

The Quantum-Resilient Smart Keychain uses a compact Li-Po rechargeable battery managed by a Power Management Unit (PMU) that provides stable power for the MCU and secure element. The PMU handles voltage regulation, peak current during post-quantum operations, and thermal protection. Most of the time, the device stays in deep-sleep mode, waking only on motion or proximity, giving it long battery life. It charges safely through magnetic pogo pins, avoiding exposed ports and ensuring durability.

1)DEVICE SETUP (Hardware & Firmware Stack)

(Quantum-Resilient Smart Keychain — full architecture: Device Setup → Connectivity → Data Storage (SQL/NoSQL) → Application Layer)

Goal: Compact, low-power hardware that can perform PQC ops safely and deterministically.

Hardware components

- **MCU:** ESP32-S3 or ARM Cortex-M33 (dual-core recommended).
- **Secure Element (SE):** TPM-lite or ATECC-like chip with hardware isolation, TRNG, and tamper resist.
- **Connectivity:** BLE 5.2 + NFC controller.
- **Sensors / UI:** 3-axis accelerometer (wake), capacitive touch / RGB LED, optional small buzzer.
- **Power:** 120–300 mAh Li-Po + PMU (magnetic pogo contacts), over-current & thermal protection.
- **Charging:** Magnetic pogo pins, charging IC supporting safe fast/slow modes.
- **Enclosure:** IP54 rated for everyday use.

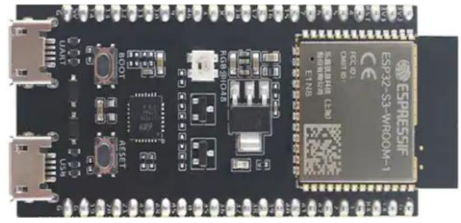

Fig.1 ESP32-S3 Microcontroller Module

Fig.2 Secure Element (Cryptographic Security Chip)

Fig.3 PIR Motion Sensor Module

Fig.4 NFC / RFID Antenna Coil

Firmware architecture (layered)

- **Secure Bootloader** — verifies PQ-signed firmware; anti-rollback.
- **RTOS Kernel** — preemptive scheduler (FreeRTOS recommended).
- **PQ Crypto Engine** — optimized libs for Kyber (KEM) + Dilithium (signatures); include hybrid ECC+PQC mode. Implement constant-time critical parts.
- **SE Interface Module** — isolated calls to SE for keygen, sign, decapsulate (I²C / secure channel).
- **Comm Manager** — BLE GATT profiles, NFC handler, session manager, KDF integration.
- **Power Manager** — sensor wake policies, dynamic gating during PQ ops.
- **App Logic** — auth workflows, telemetry buffering, OTA handler, tamper handling.

Device State Machine (simplified)

- Deep sleep → motion/NFC wake → proximity check → PQ handshake → authenticated session → perform action (unlock/log) → persist event → sleep.

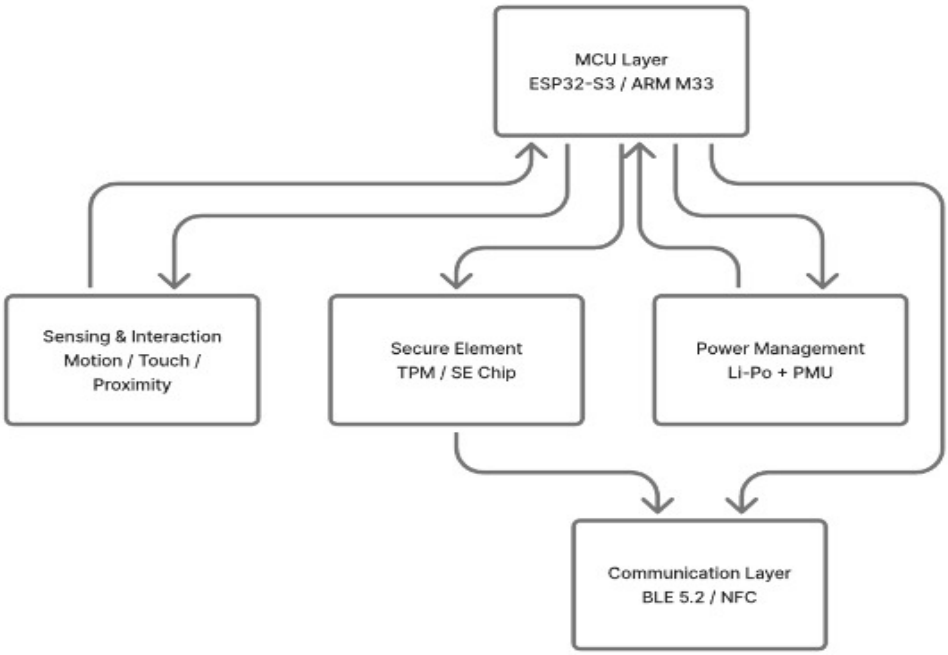

Fig.5 – Device setup architecture (Made using Figma)

2) CONNECTIVITY LAYER

Design principles

- Minimal attack surface; authenticated, encrypted channels; ephemeral session keys; fallback/hybrid crypto for compatibility.

Primary transports

- **BLE 5.2 (GATT profile)** — main interactive channel; uses encrypted characteristics carrying encapsulated PQ payloads.
- **NFC** — tap-to-provision & quick auth; short-range initialization channel.
- **Optional Wi-Fi (Gateway)** — device may connect via mobile app acting as gateway; never put Wi-Fi on keychain unless necessary.

Handshake / session protocol (high level)

1. **Initiate:** Device advertises; phone scans and opens pairing GATT.
2. **Hybrid Key Establishment:** Phone sends an ephemeral classical ECDH public; device encapsulates Kyber KEM ciphertext (or encaps from phone) → both sides KDF(hybrid) → session symmetric key.
3. **Mutual Authentication:** Device signs challenge via SE with PQ signature; phone verifies (or cloud attests).

4. **Session:** AES-GCM (or XChaCha20-Poly1305) with session key for subsequent messages.
5. **Close:** Session key destroyed; only session metadata stored (no keys).

Message transport choices

- **Small events / unlocks:** BLE characteristic writes (low latency).
- **Telemetry / logs batching:** app buffers and forwards to cloud (mobile uploads over HTTPS).
- **OTA:** Delivered via app gateway or push via cloud → signed image validated in SE/bootloader.

Security controls

- BLE pairing restricted to authenticated PQ handshake.
- Fail-open policy: never; fallback is manual override via multi-factor (NFC + PIN on app).
- Rate limiting, replay protection via nonces + sequence numbers.
- Certificate or attestation from cloud to validate device authenticity.

3) DATA STORAGE LAYER (Hybrid SQL / NoSQL design)

Principle: Use SQL for structured, transactional records (user/device bindings, policies), and NoSQL for time-series/audit logs and large unstructured blobs (firmware snapshots, telemetry). All sensitive fields are encrypted at rest (keys managed by KMS).

A. SQL (Relational) — core authoritative data

Purpose: User management, device registry, policy, key metadata (public only), transactional operations.

Recommended DB: PostgreSQL (supports JSONB for semi-structured), deployed with encryption at rest (cloud provider KMS).

Schema (core tables — simplified)

```
users (
  user_id UUID PRIMARY KEY,
  name TEXT,
  phone TEXT UNIQUE,
  email TEXT,
  created_at TIMESTAMP
)
```

```
devices (
  device_id UUID PRIMARY KEY,
  user_id UUID REFERENCES users(user_id),
  device_type TEXT,
  device_model TEXT,
  public_pq_key BYTEA,        -- public KEM key
  se_certificate BYTEA,
  provisioning_date TIMESTAMP,
  status TEXT,                -- active, revoked
  last_seen TIMESTAMP
)

policies (
  policy_id UUID PRIMARY KEY,
  device_id UUID REFERENCES devices(device_id),
  policy_json JSONB,
  effective_from TIMESTAMP,
  effective_to TIMESTAMP
)
```

Transactions: Use ACID transactions for policy change, device revoke, provisioning.

B. NoSQL (Audit / Telemetry / Time-series)

Purpose: High-volume logs, telemetry, usage traces, replayable events.

Recommended DBs:

- **Time-series:** InfluxDB or TimescaleDB (Timescale extends Postgres — can keep in same cluster).
- **Document store:** MongoDB or DynamoDB for arbitrary event payloads.

Example document model (NoSQL telemetry)

```
{
  device_id: "...",
```

```
  timestamp: "2025-11-18T10:24:00Z",
  event_type: "unlock_attempt",
  success: true,
  latency_ms: 480,
  meta: { rssi: -64, session_kdf: "hybrid-v1" }
}
```

C. Object Storage (Blobs)

- **Use case:** Firmware images (signed), device backups, audit snapshots.
- **Store:** Encrypted objects in S3 (or equivalent), signed manifests in SQL.

D. Key Management

- **KMS (cloud):** Manage server-side encryption keys, rotate with policies.
- **Device keys:** Private PQ keys stay in SE — never exported. Certificates/public keys stored in SQL (public only).
- **Symmetric keys for backups:** Encrypted with KMS; access logged.

4) APPLICATION LAYER (Mobile App, Cloud Services, Admin)

Components

- **Mobile App (iOS/Android):** provisioning, pairing UI, notifications, second factor, OTA gateway.
- **Cloud Backend (Microservices):** Auth service, Device Manager, Policy Manager, Telemetry ingest, OTA distribution, Audit/Alerting, Key rotation orchestrator.
- **Admin Console / Dashboard:** For enterprise: device fleet view, revoke, push policies.
- **Firmware & OTA Service:** Stores signed firmware artifacts; provides manifest and diff updates.

API Surface (example endpoints)

- POST /api/v1/provision — register device (public_pq_key, se_cert)
- POST /api/v1/auth/challenge — cloud issues attestation challenge for device verification
- POST /api/v1/telemetry — app uploads batched telemetry (HTTPS + JWT)

- GET /api/v1/ota/latest?model=... — retrieve latest signed firmware manifest
- POST /api/v1/policy/{device_id} — push policy update

Message flows

- **Provision:** Mobile app initiates provisioning → device generates PQ keys in SE → app sends public key to cloud via secure channel → cloud records device.
- **Auth:** Phone ←→ device perform hybrid handshake. Optionally cloud attests device state (challenge signed by SE).
- **OTA:** Cloud stores PQ-signed firmware → app downloads, forwards to device → device verifies signature via SE and bootloader.

Authentication & Authorization

- **Mobile app auth:** OAuth2 / OIDC with MFA for user access.
- **API auth:** Mutual TLS between microservices + short-lived JWTs signed by cloud KMS.
- **Admin:** RBAC controls, audit logging for every sensitive action.

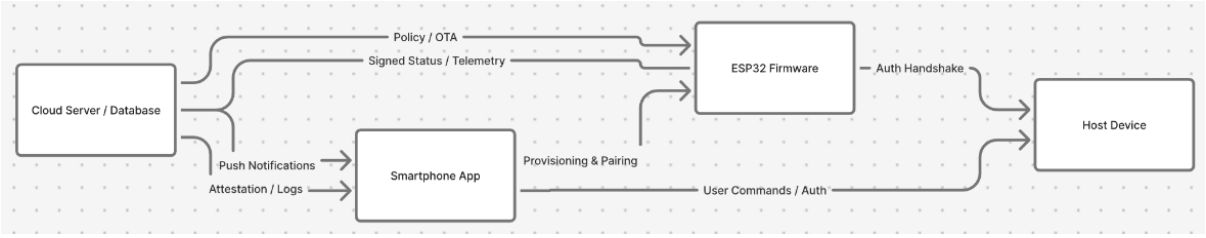

Fig.6 –Application layer (Made using Figma)

7) SIMPLE ARCHITECTURE DIAGRAM

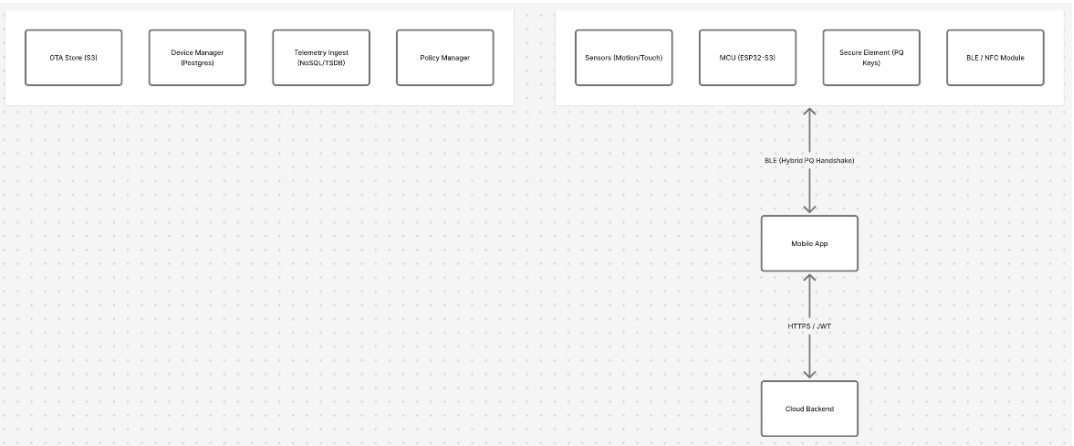

Fig.7 – Made using Figma