complete-data-flow.md
File

Pack_Controller_Communications_Wor...
File

Next_Steps_Implementation.md
File

Pairing_Protocol_Documentation.md
File

this is parallel work on modbatt modules but really on web4, and by extension on jetson/sensors/memory/etc.. docs may not be fully synced/aligned, still wip. review and give me your thoughts

Thought for 24s

Got it—read through all four docs. Here's a tight review with concrete upgrades and a short action checklist.

What's solid

- **End-to-end layering is crisp.** Cell → Module → Pack → App data path, rates, and error handling are spelled out and map cleanly to responsibilities.
- **Three-tier trust is coherent.** Binding (identity), Pairing (context), Validation (attestation) is consistent across the Pack Controller gateway narrative and the protocol specs.
- **Selective encryption is pragmatic.** Critical ops use pairing key; routine telemetry can be exchange-key or clear—good performance/complexity trade.
- **Roadmap is phased sensibly.** Binding first, then pairing, then validations + CAN integration, with concrete test coverage notes.

Gaps & upgrades (high-leverage)

1. **CAN FD + Extended ID details**

- You signal encryption in Extended ID bits (17–19); specify **exact flag mapping** and **require EFF** on *both* busses. Also lock in **FD vs classic** at each tier (modules bus currently 250 kbps classic per the data-flow doc). If encrypted payloads ride on CAN FD only, say so and define fallback.
- Define **fragmentation & reassembly** for certificates/validations that exceed 8/64 bytes, with sequence + MIC per frame.

2. **Crypto profiles & capability mapping**
    - Lock algorithms per role (e.g., **ECC-P256 for binding**, **AES-256-GCM for pairing/exchange**, **HKDF-SHA256** for derivation) and map **who can do what** on real silicon (WB55 vs ATmega). Likely: asymmetric ops on **Pack Controller**, symmetric only on **Module**; make that explicit and route binding-level signatures accordingly.
    - Add **KIDs (key IDs)**, **versioning**, and **alg identifiers** to all structures so rotation doesn't break old nodes.
3. **Certificate and message schemas**
    - The pairing cert is well described—now freeze a **TLV/CBOR schema** with sizes, endianness, and integrity fields (**hash of clear-metadata already mentioned—good**). Include **nonce/sequence**, **created_at/valid_from/expires_at** (you already sketch this) and **policy digests** for lightweight verification.
    - Define **on-wire CAN frames** for:
        - Announce, Witness, Handshakes (L3/L4),
        - Paired Command (L5),
        - Validation (L6/L7).
          Each needs: MsgID, EFF flag, FD flag, DLC, payload layout, optional signature tag.
4. **Latency & throughput budget with crypto**
    - You quote ~500 ms cell→UI; add a **before/after table** with expected overhead for:
        - L4 paired handshake,
        - L5 command,
        - typical encrypted telemetry burst.
          Use realistic MCU cycles + bus speeds to keep surprises out of the field.

5. **Escalation paths**
    - Great that escalation exists; define **automatic triggers** (e.g., temp delta > X, sequence gaps, replay suspicion) that push from Exchange → Pairing → Binding validations, and how nodes **rate-limit** to avoid storms.
6. **Zero-knowledge guardrails**
    - You assert the auth controller can’t reconstruct pairing keys—good. Add a **proof sketch** in the spec: which materials are stored where, and why that’s insufficient to derive the final key (with HKDF inputs/labels). Also specify **secure erase** of exchange keys on completion.
7. **Failure & recovery invariants**
    - You cover key loss and expiry; also define:
        - **Factory reset** sequences (what persists vs wiped),
        - **Clock skew** handling for expirations,
        - **Partial pairing** timeouts (one party completes, other doesn’t).
8. **Windows app parity**
    - App needs to parse **EFF/FD flags**, the **encryption-level bits**, and verify **L5 ACK signatures**. Add a decoder module spec + minimal UI hooks for **Validation events** stream.

Web4 + Jetson/sensors/memory tie-ins (coherence engine fit)

- Treat **sensors (Jetson, cameras, IMUs)** as **bound entities** with their own LCTs; use **pairing certs** to authorize streams per context (drive, diagnostics, training). This lets the coherence engine pull **Validation Level 2/4 events** as “witness” signals for provenance of observations.
- Push **tile-trust + motion summaries** as low-value Exchange-level telemetry; escalate to Pairing-level when a tile becomes focus (higher value, needs integrity). Aligns compute cost with informational value. (Connects directly to our tiling plan.)
- Memory plane: store **pairing cert hashes + validation receipts** alongside embeddings so every learned artifact carries provenance hooks (your ATP/traceability story starts here).

Quick wins to implement next (concrete)

1. **Lock crypto profile & IDs** (ECC-P256, AES-GCM, HKDF-SHA256) + key IDs and version fields across Binding/Pairing/Validation.
2. **Define CAN FD framing** for 6 message types (Announce, Witness, Paired Handshake, Paired Command, Pairing Validation, Binding Validation) with exact byte layouts.
3. **Add fragmentation spec** (seq#, total, CRC/MIC per chunk) for payloads > DLC.
4. **Capability map**: who performs asymmetric ops vs symmetric on each MCU; update flows accordingly.
5. **Latency budget sheet** with target ceilings per operation; re-tune telemetry rates if needed.
6. **Test harness**: simulated node pair + golden vectors (certs, frames, signatures) to run through Binding→Pairing→L5 command loop and verify end-to-end.

If you want, I can package this into a spec addendum (CBOR/TLV schemas + CAN frame tables) as a downloadable MD next.

Sources

please do!

Here's the spec addendum with CBOR/TLV schemas and CAN frame definitions: spec_addendum_cbor_tlv_can.md

says file not found. can you re-generate?

Here's the regenerated file: spec_addendum_cbor_tlv_can.md got it! will review with claude and fold it in with everything. you're awesome as always :)

Glad it hit the mark :)

That one should drop straight into your doc set without too much massaging, and it gives Claude some nice clean hooks for the next round of integration.