ALP LAB

Over-the-Air Firmware Updates

AN-006 · E1M and E1M-X SoMs

Document Number: AN-006 **Revision:** 0.3 **Date:** July 2026 **Status:** Preliminary

alplab.ai

1 Scope

Set up signed Over-the-Air (OTA) firmware updates for fielded SoMs. Two flows are supported by the Alp SDK™:

- **Yocto / Linux** targets (Cortex-A55 on V2N family) use **Mender** for full-image updates with A/B partition rollback. This is the shipping in-field path.
- **Zephyr** targets (Cortex-M on AEN, and the M33 on V2N) use **MCUboot** for swap-using-scratch A/B images with ECDSA-P256 signing. As of v0.4 MCUboot provides signed boot plus automatic rollback, but there is **no in-field OTA client** for Zephyr yet (deferred to v1.1); deliver signed images by physical reflash or via a Linux companion until then.

Audience	Engineers and DevOps teams managing fielded SoM fleets.
Prerequisites	Functional Wi-Fi or Ethernet network (see **AN-002**), secure-boot keys generated (see **AN-010**), Mender server or local CDN reachable.
Outcome	End-to-end OTA: build artefact, sign, upload to update server, device pulls and installs, automatic rollback on failed health-check.
Time	60 minutes (server setup is the long pole).
Source	`docs/tutorials/12-mender-ota.md`, `docs/ota.md`, `docs/ota-device-contract.md` in alp-sdk.

Table 1 Scope summary

2 Hardware Setup

No carrier-side wiring beyond the standard network connection. The on-module memory layout (eMMC + LPDDR4X on V2N; MRAM + optional OSPI on AEN) provides the A/B partition layout the OTA flows expect.

> **Note:** For the V2N family, the bootloader (U-Boot) owns A/B partition selection and automatic rollback. The GD32 bridge MCU is updated through its own independent firmware-update path (see `docs/ota-device-contract.md`) and is decoupled from the main-system OTA, so a failed bridge update cannot brick the application core, and vice versa.

3 Software Walkthrough

3.1 Yocto / Linux (V2N A55)

Enable Mender declaratively via the top-level `ota:` block in your project's `board.yaml` (the orchestrator emits the matching `MENDER_*` weak-assignments plus `INHERIT += "mender-full"` into the slice's `local.conf`), then build the Mender-aware image:

```
# Build the Mender-aware Alp image (produces the signed .mender artefact):
MACHINE=e1m-v2n101-a55 bitbake alp-image-edge
# Output: tmp/deploy/images/e1m-v2n101-a55/alp-image-edge-e1m-v2n101-a55.mender
# Upload the .mender artefact via the Mender UI's Releases tab,
# then create a Deployment targeting your device group.
```

3.2 Zephyr (AEN / V2N M-class)

Drive MCUboot from the `boot:` block in `board.yaml` (the orchestrator emits the `SB_CONFIG_BOOTLOADER_MCUBOOT`, `SB_CONFIG_MCUBOOT_SIGNATURE_TYPE_ECDSA_P256`, and `SB_CONFIG_BOOT_SIGNATURE_KEY_FILE` sysbuild options). The underlying build is a sysbuild MCUboot child image that signs the slot0 app at build time – there is no separate `west sign` step:

```
# Build a signed image via sysbuild (MCUboot child image + signed slot0 app):
west build -p always --sysbuild \
```

```
    -b alp_e1m_aen801_m55_he/ae822fa0e5597ls0/rtss_he \
    examples/aen/aen-mcuboot-smoke -d build/mcuboot-smoke -- \
    -DSB_CONFIG_BOOT_SIGNATURE_KEY_FILE="<abs>/keys/mcuboot_dev_ecdsa_p256.pem"
# Signed slot0 app: build/mcuboot-smoke/aen-mcuboot-smoke/zephyr/zephyr.signed.bin
```

There is no in-field OTA client header (`<alp/ota.h>`) for Zephyr yet. Until v1.1, deliver the signed image by physical reflash (J-Link / OpenOCD) or via a Linux companion coordinating through the bridge. The declarative `boot:` + `ota:` blocks are exercised by `examples/connectivity/iot-fleet-ota` (the v0.6 reference, `native_sim`-verified; HiL gates on a staged Mender server).

3.3 Helper-MCU firmware: CC3501E radio co-processor (AEN)

The CC3501E is updated over the Alif→ CC3501E SPI bridge, independently of the application core. **The full OTA cold-swap cycle is proven on E8 silicon:** the image streams over the bridge (`OTA_BEGIN` → RAM-staged `OTA_WRITE` → `OTA_FINISH`, which does one flash burst into `psa_fwu_install`), reaching STAGED; the CC3501E then issues its **own** `psa_fwu_request_reboot()`, the bridge link drops for roughly two seconds, and BL2/MCUboot swaps the pending slot to primary. The swapped image self-accepts and persists across a true cold power-on with no rollback.

> **Warning:** **The OTA payload's signed version must exceed the running primary.** The CC3501E enforces monotonic anti-rollback: a downgrade is refused at `psa_fwu` install and the session ends in the error state. This is the single most common cause of a "streams cleanly but nothing changes" result.

A signed prebuilt ships at `firmware/cc3501e/prebuilt/cc3501e-v0.2.0.bin`, with a detached ECDSA-P256/SHA-256 signature (`.sig`) and a SHA-256 manifest (`.sha256`). The AEN SoM presets wire it as `helper_firmware.cc3501e_otp`; its `flash_args` stay unset because they are bench-specific rather than SoM properties.

> **Note:** **Unjamming a stuck slot — `cc3501e_ota_promote()` (`OTA_PROMOTE`, opcode `0x46`, protocol v4).** A STAGED image survives a reset, but the device's RAM session state resets to `IDLE`, and a fresh session is rejected while the slot is occupied — so `cc3501e_ota_finish()` becomes unreachable and the slot cannot be freed. `cc3501e_ota_promote()` breaks that deadlock: it arms the same deferred swap-reboot `FINISH` uses, for an image already committed to STAGED. If nothing is pending, the reboot is a clean no-op. It returns `ALP_ERR_NOT_READY` against a CC3501E whose **running** firmware predates the opcode, so a device in the field must already carry v4 firmware for this escape hatch to exist.

3.4 Helper-MCU firmware: GD32 bridge (V2N)

The GD32 bridge is updated through Path A — an application bootloader plus A/B slots, over the opcode range `0xF0–0xF6` (`OTA_BEGIN`, `WRITE_CHUNK`, `VERIFY`, `COMMIT`, `ROLLBACK`, `GET_STATE`, `ABORT`). It is **safe by default**: the flash path is armed only in a `-DBRIDGE_OTA_PARTITIONED` build. A stock build answers `STATUS_NOSUPPORT` across the whole range and touches no flash, so the unpartitioned image cannot brick itself.

Current hardening a customer should design around:

Behaviour	What it means for you
Slot and image validation	A slot id that is not A or B is rejected outright rather than silently resolving to slot A. A CRC-valid but truncated or vector-less image is rejected at `COMMIT` **and** again before the bootloader jumps — CRC integrity alone is treated as necessary but not sufficient.
Newest-first boot records, with fallback	The bootloader orders both A/B metadata records newest-first and tries each in turn. A newer record that fails validation does **not** suppress an older bootable one, so a bad update cannot strand the part in recovery while a good slot sits unused.
Host gates OTA on protocol minor >= 6	The chunk wire format changed incompatibly in v0.6 (an explicit length byte after the offset) in a way a major-only handshake cannot detect. Call `gd32g553_ota_supported()` first: it is true only if the bridge advertised minor >= 6 at init. `gd32g553_ota_begin()` and `_write_chunk()` refuse an older peer with `ALP_ERR_NOSUPPORT` **before** touching flash, so a current host cannot corrupt a pre-v0.6 bridge.
`OTA_BEGIN` is non-blocking	`BEGIN` no longer erases the slot synchronously — that stalled the SPI reply long enough to hang the host. It arms a background erase pumped from `bridge_hw_tick` (one page-region per main-loop tick) and acks immediately, leaving the state machine `BUSY`. **Poll `gd32g553_ota_get_state()` until it reports `READY` before streaming the first chunk**; a chunk sent while `BUSY` is rejected.

Table 2 GD32 bridge OTA Path-A behaviour

> **Note:** **Scope of the GD32 silicon claim.** Boot-select, dual-bank erase, and the background-erase path are silicon-validated on the GD32, as is the full stream → verify → commit → boot-new-slot → rollback cycle over the 25 MHz link. That validation covers the **bridge firmware**. It is not a statement about the `gd32g553` chip driver as a whole, which remains `partial` with `hil_silicon: untested` in the SDK's chip manifest. First flash of a partitioned part additionally needs a factory metadata record written at `0x08008000`, and a bricked part is recovered with a bench SWD probe — there is no host-driven SWD reflash on this hardware revision.

4 The Update Audit Log and Its Assurance Tiers

`<alp/update_log.h>` gives one portable, append-only, hash-chained record of what firmware was installed and whether it was accepted. `alp_update_log_verify()` walks the chain and reports mutation, truncation, rollback, or reorder. The log never wraps: when the backing store fills, `alp_update_log_append()` returns `ALP_ERR_NOMEM` and the existing chain stays intact and verifiable, because wrapping would erase audit history.

The strength of that record is **not the same on every SoM**. Query it — never assume it:

```
#include <alp/update_log.h>

alp_update_log_t *log = alp_update_log_open();
if (alp_update_log_assurance(log) == ALP_UPDATE_LOG_HW_ENFORCED) {
    /* Log writer lives behind a hardware boundary. */
} else {
    /* ALP_UPDATE_LOG_SW_TAMPER_EVIDENT -- app-cooperative. */
}
```

Tier	What it actually guarantees
`ALP_UPDATE_LOG_SW_TAMPER_EVIDENT`	Every target. Hash chain anchored to a monotonic counter; `alp_update_log_verify()` **detects** out-of-band mutation, truncation, rollback, and reorder. It is tamper-**evident**, not tamper-proof: code that can write the backing partition can rebuild the store and its counter consistently and **forge a coherent history**. With `CONFIG_ALP_SDK_UPDATE_LOG_PERSIST` plus an `alp_ulog_partition` fixed partition the log survives reboot and update; without it, entries live in RAM and vanish on reboot. **Persistence does not raise the assurance level.**
`ALP_UPDATE_LOG_HW_ENFORCED`	Only where a trusted owner exists. The application core is a **client**; the log writer and its store sit behind a hardware boundary the app cannot rewrite. Routes: TF-M (`CONFIG_ALP_SDK_UPDATE_LOG_TFM`, PSA Protected Storage) and Alif E4/E8 AEN (a trusted M55 owner plus an SE/firewall-locked MRAM partition). If the trusted owner or the isolation proof is absent, `alp_update_log_open()` **silently falls through to the software tier** — unless `CONFIG_ALP_SDK_UPDATE_LOG_REQUIRE_HW_ENFORCED` is set, which makes it fail closed instead.

Table 3 Update-log assurance tiers

Warning: **Scope the `HW_ENFORCED` guarantee precisely: it is app-immutable, not reflash-immutable.** The application cannot rewind or rewrite the log — the owner holds it and its sequence counter inside the protected region. It does **not** survive an attacker who can reflash the part. A reflash-proof guarantee needs a monotonic anchor stored outside the rewindable MRAM, and current E8 SE firmware exposes no runtime-writable one (no SE NV-counter mailbox, no runtime OTP-write path, no per-log MCUboot security counter). If your threat model includes an adversary with physical reflash access, this log is **not** your control.

Note: Prefer `alp_update_log_append_boot()` over `alp_update_log_append()` for update-result entries. It sources the version, image hash, and verification status from the platform's authenticated boot-metadata provider instead of accepting them from app code, so firmware cannot log a forged-but-well-formed entry. Platforms with no provider return `ALP_ERR_NOSUPPORT` — which is the signal to fix provisioning, not to fall back to the app-supplied path. Both boot-metadata entry points are marked `[ABI-EXPERIMENTAL]`: the surface may change until the hardware backend is silicon-proven.

5 Expected Output

On the Yocto / Mender path, watch the device pick up a deployment:

```
$ journalctl -u mender-client -f
... Update available
... Downloading update
... Installing update to the inactive A/B rootfs slot
... Rebooting to apply update
... (after reboot) Running version: alp-image-edge-...
... Update committed
```

If the new image boots cleanly and the post-reboot health check passes, the swap is committed. If the device does not confirm within the inventory interval, the bootloader reverts to the previous partition automatically on the next reset. On the Zephyr / MCUboot path the equivalent confirm is the image calling MCUboot's `boot_set_confirmed()` within its health-check window; an image that never confirms is rolled back by MCUboot on the next reboot.

6 Troubleshooting

- **Signature verification fails**: the artefact was signed with a different key than the ECDSA-P256 verify key provisioned into the device. On Mender this anchor is the verify key written into the rootfs (`/etc/mender/`); on MCUboot it is the public key compiled into the bootloader. Use the same key pair from your **AN-010** secure-boot provisioning.

- **Health-check times out**: the device never confirmed the new image. On Zephyr, the firmware must call MCUboot's `boot_set_confirmed()` within its health-check window (typically 30 s after boot). On Yocto, the device must check in within `MENDER_INVENTORY_INTERVAL` (default 60 s) or the bootloader rolls back after `MENDER_RETRY_POLL` (default 5 min).
- **Download stuck at 0 %**: server URL unreachable, or TLS pinning mismatch. Test with `curl` from the device or host first.
- **Rollback loop**: the new image keeps failing health-check, so MCUboot keeps reverting. Pin the version on the server until the bug is fixed.

7 References

- **Tutorial**: `docs/tutorials/12-mender-ota.md` in alp-sdk.
- **OTA architecture**: `docs/ota.md`, `docs/ota-device-contract.md`.
- **Examples**: `examples/connectivity/iot-fleet-ota/` (declarative `boot: + ota:` reference, `native_sim`-verified), `examples/connectivity/firmware-update-log/` (update audit log; its README carries the assurance-tier scoping), `examples/aen/aen-mcuboot-smoke/` (SES → MCUboot → slot0 secure-boot bench test).
- **SDK API**: `<alp/update_log.h>` (firmware-update audit log + `alp_update_log_assurance()`), `<alp/iot.h>` (transport), `<alp/security.h>` (signature verification). There is no `<alp/ota.h>` Zephyr OTA client header yet.
- **Helper-MCU OTA**: `<alp/chips/cc3501e/ota.h>` and `<alp/protocol/cc3501e.h>` (CC3501E bridge OTA, incl. `OTA_PROMOTE 0x46`); `<alp/chips/gd32g553.h>` (GD32 bridge Path-A OTA opcodes) and `firmware/gd32-bridge/src/bootloader/DESIGN.md`.
- **Companion AN**: **AN-010** (secure boot and code signing — the key pair that signs every artefact here).

8 Revision History

Revision	Changes	Date
0.1	Initial draft.	May 2026
0.2	Corrected to current SDK: removed non-existent `<alp/ota.h>`/`alp_ota_confirm()`/`CONFIG_ALP_OTA_GRACE_SECONDS`; clarified that Zephyr in-field OTA is deferred to v1.1 (MCUboot signed boot + `boot_set_confirmed()` rollback only); aligned Yocto build to `bitbake alp-image-edge` (MACHINE `e1m-v2n101-a55`) and Zephyr build to the sysbuild MCUboot flow; fixed example path to `examples/connectivity/iot-fleet-ota` and added `firmware-update-log/aen-mcuboot-smoke`; pointed API references at `<alp/update_log.h>`, `<alp/iot.h>`, `<alp/security.h>`.	June 2026
0.3	Added the update-log assurance tiers with the load-bearing scoping that `HW_ENFORCED` is app-immutable, not reflash-immutable, and that the SW tier detects but cannot prevent forgery. Added the CC3501E OTA state (full cold-swap proven on E8, forward-version-only, `OTA_PROMOTE` 0x46, signed v0.2.0 prebuilt) and the GD32 bridge Path-A hardening, with the silicon claim scoped to boot-select, dual-bank erase and background erase — bridge firmware, not the gd32g553 chip driver. Corrected cross-references: secure boot is AN-010, not AN-008. Fixed the Alp SDK repository links (`alpDevs` → `alplabai`).	July 2026

Table 4 Revision History