ALP LAB

Secure Boot and Code Signing

AN-010 · E1M and E1M-X SoMs

Document Number: AN-010 **Revision:** 0.5 **Date:** July 2026 **Status:** Preliminary

alplab.ai

1 Scope

Establish a chain of trust from the SoM's immutable boot ROM to the application image: on E1M-AEN the Alif Secure Enclave ROM hands off to MCUboot, which verifies the application's ECDSA-P256 signature against a public key compiled into the bootloader before chaining in. This AN walks the dev-key bench flow, the production-key lifecycle, and the swap-using-scratch rollback path.

The production lock-down is **destructive** — blowing the SoM's debug-disable fuse is permanent and cannot be undone, and once a production public key is baked into a fielded bootloader only images signed by the matching private key will boot. Read the **Production Key Management** section fully before locking down any module that will leave your bench.

> **Warning:** **Do not architect key management around the OPTIGA Trust M secure element.** Every E1M-AEN module populates the part, but the Alp SDK™ driver is **probe-only**: it confirms the chip ACKs on I²C and nothing more. On-chip key generation, public-key export, and secure-element signing **are not available** — those entry points return `ALP_ERR_NOSUPPORT`. The production signing key today lives on an **air-gapped signing workstation**. See **Production Key Management**.

Audience	Security engineers shipping fielded firmware that must not be replaced by attackers.
Prerequisites	An AEN EVK or E1M-AEN module, the SDK checked out, OpenSSL 3.0+, and the MCUboot `imgtool` that ships with the Zephyr build. For fielded units, an **air-gapped signing workstation** holding the production key; this AN uses the SDK's dev key only for the bench walk-through.
Outcome	Module's MCUboot bootloader accepts only firmware signed by the trusted key; unsigned or wrong-key firmware is rejected and the previous-good slot is kept.
Time	60 minutes (key generation is fast; understanding the implications is the long pole).
Source	`docs/secure-boot.md` and `docs/tutorials/10-secure-boot-signing.md` in alp-sdk.

Table 1 Scope summary

2 Step 1 — Generate Signing Keys

AEN secure boot uses ECDSA-P256 throughout. For development and bring-up, generate the MCUboot dev key once with the SDK helper (idempotent — it preserves an existing key):

```
cd alp-sdk
bash keys/generate_dev_key.sh
```

Expected output:

```
[generate_dev_key] writing keys/mcuboot_dev_ecdsa_p256.pem (chmod 600)
[generate_dev_key] Done.
```

The key is `.gitignored`; never commit it. The sysbuild profile at `zephyr/sysbuild/aen/sysbuild.conf` already references the matching public key via `SB_CONFIG_BOOT_SIGNATURE_KEY_FILE`, so it is compiled into the bootloader in Step 2.

For production, generate the key the same way but **on an air-gapped workstation**, and keep the private half there. Only the public half travels to the build host, as `keys/mcuboot_prod_ecdsa_p256.pub.pem`. See **Production Key Management** below.

> **Warning:** Treat the production signing key like the master key it is. Losing it means **no future firmware can be installed** on any device whose bootloader trusts it; leaking it means anyone can install firmware on every such device. Back it up offline, in a safe, before you sign the first production image. The dev key under `keys/` is for the bench only.

3 Step 2 — Build and Sign a Firmware Image

Sysbuild builds the MCUboot bootloader and the application together, and signs the application as part of the build:

```
west build -b alp_e1m_aen801_m55_he/ae822fa0e5597ls0/rtss_he examples/peripheral-io/hello-world \
    --sysbuild \
    -- -DSB_CONF_FILE=/abs/path/to/alp-sdk/zephyr/sysbuild/aen/sysbuild.conf
```

The sysbuild config is a CMake define passed after `--`, not a `west build` flag. The path must be **absolute**: sysbuild resolves a relative `SB_CONF_FILE` against `APP_DIR`, not the directory you ran the command from.

(If your `board.yaml` carries a `boot:` block, the loader emits the matching overlay at `build/alp_sysbuild.conf`, which becomes the canonical `SB_CONF_FILE` path.)

The build produces the bootloader at `build/mcuboot/zephyr/zephyr.bin` and the signed application at `build/zephyr/zephyr.signed.bin` (the raw `build/zephyr/zephyr.bin` is unsigned — don't flash it). The signed image is structurally:

```
+---------------------------+
| MCUboot image header      |
|   (0x200 B: version, size) |
+---------------------------+
| firmware (application)    |
+---------------------------+
| TLV trailer               |
|   ECDSA-P256 signature +  |
|   SHA-256 image hash      |
+---------------------------+
```

Flash both:

```
west flash --bin-file build/mcuboot/zephyr/zephyr.bin --domain mcuboot
west flash --bin-file build/zephyr/zephyr.signed.bin
```

MCUboot verifies the ECDSA-P256 signature in the TLV trailer **before** chaining into the application: invalid signature → the image is rejected and the previous-good slot is kept (swap-using-scratch).

4 Step 3 — Compile the Production Key into the Bootloader

There is no public-key-hash fuse to burn on AEN. The trust anchor is the public key compiled into the MCUboot bootloader. The production lifecycle:

1. **Generate the production key pair on an air-gapped workstation.** The private half stays on that machine and is never copied to a networked host.
2. **Carry the public half out** (USB, or read it off the screen) and commit it as `keys/mcuboot_prod_ecdsa_p256.pub.pem`. Only the public half leaves the air gap.
3. **Compile the bootloader against the production public key** by overriding `SB_CONFIG_BOOT_SIGNATURE_KEY_FILE` to point at that file when building MCUboot for production firmware.
4. **Sign release images on the air-gapped workstation.** Copy the unsigned image in, run `imgtool sign` against the private key, copy the signed image out.
5. **Lock the module down** by blowing the debug-disable fuse (see the per-SoM bring-up doc / **QS-** guide) so JTAG/SWD can no longer overwrite the bootloader region.

The recommended way to wire this is a top-level `boot:` block in your project's `board.yaml`; the loader emits the matching `SB_CONFIG_*` overlay automatically:

```
# board.yaml
boot:
  method: mcuboot
  signing:
    algorithm: ecdsa_p256
    key_file: keys/mcuboot_prod_ecdsa_p256.pub.pem
  swap_algorithm: scratch
```

Slot and scratch **sizes** are not `board.yaml` fields. MCUboot takes its slot geometry from the board device-tree partitions; use the `storage:` block to declare the project's own partition table. (Alp SDK™ v0.11.0 removed the `slots:`, `scratch_size_kib:` and `anti_rollback:` keys from the `boot:` block, which rejects unknown keys — a `board.yaml` still carrying them fails validation.)

> **Warning:** Once the production key is compiled in and the debug-disable fuse is blown, the module accepts **only** firmware signed by the corresponding private key, and the lock-down cannot be undone. Test the full signing flow with the dev key on an EVK **before** committing a production key to any unit that will leave your bench.

5 Step 4 — Verify

Power-cycle the module. On the UART you should see the MCUboot banner accept the image and hand off to the application:

```
*** Booting MCUboot v1.x ...
[INF] Starting bootloader
[INF] Image index: 0, Swap type: none
...
[hello] ALP SDK hello-world starting
```

Then sign with the wrong key and re-flash to confirm rejection:

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 \
    -out /tmp/fake_key.pem

imgtool sign --key /tmp/fake_key.pem --version 0.1.0 \
    --header-size 0x200 --align 8 --slot-size 0x40000 --pad-header \
    build/zephyr/zephyr.bin /tmp/zephyr.fakesigned.bin

west flash --bin-file /tmp/zephyr.fakesigned.bin
```

MCUboot output:

```
*** Booting MCUboot v1.x ...
[INF] Starting bootloader
[INF] Image index: 0, Swap type: none
[ERR] Image in the primary slot is not valid!
[ERR] Unable to find bootable image
```

MCUboot refuses to chain into the wrongly-signed image and halts. Restore a properly-signed image to recover.

6 Production Key Management

For production, the bench dev key is **not acceptable**. Recommended set-up:

Store	Notes
Air-gapped signing workstation — use this today	The practical production key store on every current SoM. The private key is generated on and never leaves a permanently disconnected machine; unsigned images are copied in and signed images copied out over removable media. Compromise of the dev key or the build host does not yield production signing power — only physical access to the workstation does. Its weakness is that the key exists as a file, so physical security, disk encryption, and an offline backup carry the whole guarantee.
Per-generation key rotation	Generate the next-generation key pair on the same air-gapped workstation, compile the bootloader with both public keys so either is accepted, roll the new key out via OTA signed by the current key, then retire the old key after one update window (typically 90 days).
OPTIGA Trust M secure element — future work	**Not usable as a key store today.** The part is populated on every E1M-AEN module and the SDK driver confirms it is alive, but the driver is probe-only: `optiga_trust_m_init()` reads the `I2C_STATE` register (I^2C address `0x30`) and returns. `optiga_trust_m_read_product_info()` and `optiga_trust_m_send_apdu()` validate their arguments and then return `ALP_ERR_NOSUPPORT`; no `OPEN_APPLICATION`, `GenKeyPair`, `CalcSign`, or ECDH is ever sent. On-chip keygen, key export, and SE signing therefore **do not work**. They land when Infineon's OPTIGA Trust M Host Library is integrated as a Zephyr module and registered as a PSA driver behind `<alp/security.h>`. Design the product so the key store can migrate later; do not depend on it now.

Table 2 Production key store

Note: The SDK ships `examples/v2n/v2n-secure-element-sign` and `examples/aen/aen-secure-element-sign` to **verify the probe contract** — they confirm the chip ACKs and that the signing entry points cleanly refuse, rather than fabricating a signature. Despite the names, they do not sign anything. `docs/tutorials/06-secure-element-sign.md` walks the same contract.

7 Troubleshooting

Symptom	Likely cause / fix
Build complains about the signing-key algorithm	AEN secure boot uses ECDSA-P256. Set `signing.algorithm: ecdsa_p256` in the `board.yaml` `boot:` block (or use the stock sysbuild profile) and a matching `key_file`.
MCUboot logs `Image in the primary slot is not valid!`	The image was signed with a key the bootloader doesn't trust. Rebuild with the `key_file` that matches `SB_CONFIG_BOOT_SIGNATURE_KEY_FILE`, then reflash a properly-signed image.
Module won't boot after compiling a production key	The bootloader was built against a **different** public key than the one signing the image. Recovery is reflashing a correctly-signed image — unless the debug-disable fuse is already blown, in which case the unit is unrecoverable.
MCUboot reports the image is too large for the slot	The signed image exceeds the slot size. Trim the firmware — a smaller image is the only in-project remedy. Slot geometry is **not** settable from `board.yaml`: MCUboot takes it from the board device-tree partitions, so growing a slot is a change to the board definition in the SDK, not a per-project field. Check `build_type: MinSizeRel` and drop unused libraries first.
Updated image boots once then reverts	The application didn't call `boot_set_confirmed()` within the documented window, so MCUboot treats the swap as a failed test and reverts to the previous slot. Confirm the new image once it is verified healthy.
`optiga_trust_m_send_apdu()`/`_read_product_info()` return `-5` (`ALP_ERR_NOSUPPORT`)	Working as designed — the driver is probe-only and does not implement APDU transport. There is no workaround; use the air-gapped signing workstation. A `-2` (`ALP_ERR_NOT_READY`) from `optiga_trust_m_init()` is the real fault signal: the chip is not ACKing at address `0x30` (not populated, held in reset, or mis-strapped).

Table 3 Common failures

8 References

- **Canonical doc**: `docs/secure-boot.md`, `docs/tutorials/10-secure-boot-signing.md`, and `docs/adr/0006-secure-boot-secure-ota.md` in alp-sdk.
- **SDK tooling**: `keys/generate_dev_key.sh`, `west build --sysbuild`, MCUboot `imgtool`, `west flash`.
- **Examples**: `examples/aen/aen-mcuboot-smoke` (MCUboot smoke test), `examples/peripheral-io/hello-world`.
- **Secure element (probe contract only)**: `docs/tutorials/06-secure-element-sign.md`, `examples/v2n/v2n-secure-element-sign`, `examples/aen/aen-secure-element-sign`, header <alp/chips/optiga_trust_m.h> (see its `@par Driver scope: [PROBE-ONLY]`).
- **Companion AN**: **AN-006** (OTA updates with signed images), **AN-007** (mproc mailbox — the M33 firmware must be signed too).
- **Vendor reference**: Alif Secure Enclave / Ensemble documentation (AEN); Infineon OPTIGA Trust M Solution Reference Manual; for V2N M33 secure boot see `docs/rzv2n-m33-secure-boot.md` (Renesas RZ/V2N TF-A BL2 chain).

9 Revision History

Revision	Changes	Date
0.1	Initial draft.	May 2026
0.2	Aligned with current alp-sdk: AEN secure boot is MCUboot + ECDSA-P256 (not RSA-PSS/boot-ROM OTP). Replaced the invented `west sign/alp-fuse` pubkey-hash flow with the real `keys/generate_dev_key.sh` dev key, sysbuild build, `imgtool` signing, OPTIGA Trust M production-key lifecycle, and `SB_CONFIG_BOOT_SIGNATURE_KEY_FILE`. Fixed example path (`examples/peripheral-io/hello-world`), board target (`alp_e1m_evk_aen`), MCUboot console output, troubleshooting, references, and canonical-source header.	June 2026
0.3	Corrected the key-store story: the OPTIGA Trust M driver is probe-only — it confirms the part ACKs and returns `ALP_ERR_NOSUPPORT` for product-info, raw APDUs, `CalcSign`, `GenKeyPair` and ECDH, so on-chip key generation, export and signing are unavailable today. The air-gapped signing workstation is now presented as the practical key store and the secure-element path as future work, with a warning against architecting product key management around it. Removed a troubleshooting row citing a Kconfig symbol that does not exist. Fixed the Alp SDK repository links (`alpDevs` → `alplabai`).	July 2026
0.4	Retensed to Alp SDK™ v0.11.0. Replaced `--sysbuild-config`, a flag that has never existed in any Zephyr — west forwarded it to CMake and the configure died with `Unknown argument --sysbuild-config` — with `-- -DSB_CONF_FILE=<absolute path>`, noting that sysbuild resolves a relative path against `APP_DIR` rather than the cwd. Removed the `slots:` key from the production `board.yaml` snippet and rewrote the "image too large for the slot" troubleshooting row, which sent readers to `slots.primary.size_kib`: v0.11.0 removed `slots:`, `scratch_size_kib:` and `anti_rollback:` from the `boot:` block, so both the snippet and the suggested remedy now fail schema validation. Slot geometry comes from the board device-tree partitions.	July 2026
0.5	Replaced the phantom board target `alp_e1m_evk_aen` in the Step 2 build command with the real fully-qualified target `alp_e1m_aen801_m55_he/ae822fa0e5597ls0/rtss_he` (E1M-AEN801 HE, Alif Ensemble E8): the old identifier resolves nowhere in the alp-sdk board tree, so the documented `west build` failed at board resolution.	July 2026

Table 4 Revision History