ALP LAB

Secure Boot and Code Signing

AN-010 · E1M and E1M-X SoMs

Document Number: AN-010 **Revision:** 0.2 **Date:** June 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 OPTIGA Trust M 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.

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. A **production OPTIGA Trust M secure element** holds the signing key for fielded units; 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 the private key is generated **inside** an OPTIGA Trust M secure element (slot `0xE0F0`) and never leaves the chip; only the public half is exported to `keys/mcuboot_prod_ecdsa_p256.pub.pem`. See **Production Key Management** below.

> **Warning:** Treat the production signing key like the master key it is. For fielded units it lives inside the OPTIGA Trust M secure element and never touches a host disk — losing access to the provisioned secure element means **no future firmware can be installed** on any device whose bootloader trusts it. 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_evk_aen examples/peripheral-io/hello-world \
    --sysbuild \
    --sysbuild-config alp-sdk/zephyr/sysbuild/aen/sysbuild.conf
```

(If your `board.yaml` carries a `boot:` block, the loader emits the matching overlay at `build/alp_sysbuild.conf`, which becomes the canonical `--sysbuild-config` 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 private key inside OPTIGA Trust M** (slot `0xE0F0`). The private half is created inside the chip and never leaves the secure NVM.
2. **Export the public half** over the OPTIGA's I^2C interface and commit it as `keys/mcuboot_prod_ecdsa_p256.pub.pem`.
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. **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:

```yaml
# board.yaml
boot:
  method: mcuboot
  signing:
    algorithm: ecdsa_p256
    key_file: keys/mcuboot_prod_ecdsa_p256.pub.pem
  slots:
    primary:   { size_kib: 480 }
    secondary: { size_kib: 480 }
  swap_algorithm: scratch
```

> **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
OPTIGA Trust M secure element	Canonical store. The private key is generated **inside** the chip (slot `0xE0F0`) and never leaves the secure NVM; the release pipeline signs from a host with physical access to a provisioned device. Compromise of the dev key, build host, or signing host alone does not yield production signing power.
Air-gapped signing workstation	Where a secure element is unavailable. The signing-key file lives only on a disconnected machine; signed images are copied off via USB. Operationally heavier and weaker than the secure element.
Per-generation key rotation	Provision a second OPTIGA slot (e.g. `0xE0F1`), 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).

Table 2 Production key store

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 or grow the slot via the `slots.primary.size_kib` field in the `board.yaml` `boot:` block.
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.
Signature verification is slow (>1 s)	Acceptable on first boot. If on every wake from low-power, enable the SDK's **secure-boot cache** (`CONFIG_ALP_SECURE_BOOT_CACHE=y`).

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**: `docs/tutorials/06-secure-element-sign.md`, `examples/v2n/v2n-secure-element-sign`, header `<alp/chips/optiga_trust_m.h>`.
- **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

Table 4 Revision History