THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
Embedded Linux Conference
NORTH AMERICA

Untangling Secure Key Provisioning in U-Boot: Scalable eFuse Programming in Production

2026-05-20, Minneapolis

Harsha Vardhan V M

Kavitha Malarvizhi

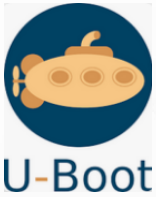
THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
NORTH AMERICA

About us: TI Processors and Open Source

Upstream FIRST!

Open Source

- ✓ Upstream FIRST mentality!
- ✓ Decades of contribution and collaboration
- ✓ Focus on long term, sustainable and quality products
- ✓ Upstream & opensource ecosystem in device architecture
- ✓ Ingrained culture to give back to the community



About Speakers

Harsha Vardhan V M

Security Software Engineer, Texas Instruments

Silicon root-of-trust configuration, cryptographic key management and lifecycle state enforcement

Kavitha Malarvizhi

Security Architect, Texas Instruments

System Security, Secure boot, Cybersecurity

Texas Instruments

Why eFuse Provisioning Matters

- eFuse (Electronic Fuse) - a single non-volatile memory bit that can be blown only once.
- The programming operation is irreversible.
- eFuses control:
  - various permanent configurations and data
  - boot configurations, security settings, MAC addresses
  - secure boot enablement, silicon identity, cryptographic keys

- Factory provisioning of thousands of devices must be reliable and repeatable

How eFuses are Programmed today (in U-boot)

- Low-level, command-driven interface
- eFuses organized as:
  - **Words** - smallest group of efuses
  - **Banks** - smallest group of fuse words
- eFuses accessed via: Bank + Word addressing
- Programming flow: ***fuse prog*** – mention exact location and value
- Supports contiguous writes only
- ! No structure, no flexible non-linear batching

Texas Instruments

U-Boot's Existing Fuse Subsystem Commands

1. fuse read
- Read fuse words from shadow cache

2. fuse readm
- Read fuse words into memory

3. fuse sense
- Read fusebox directly (bypasses shadow cache)

4. fuse cmp
- Compares the fuse value to a hex value

5. fuse override
- Modify shadow cache (non-permanent)

6. fuse prog
- Permanently program fuse words (irreversible)

Texas Instruments

The Reality of Traditional eFuse Programming

- Most teams use long, manual U-Boot *fuse prog <bank> <word> <hexval> [<hexval>…]* cmd sequences
- Works for bring-up but breaks down in production

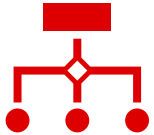
Typical pain points:

  - Difficulty handling **vendor-specific, non-linear** fuse maps
  - **Repititive, manual** cmd-by-cmd programming that doesn't scale
  - Increased risk of **typing mistakes or sequence errors**
  - Hard to integrate into **automated workflows**

Texas Instruments

The Reality of Traditional eFuse Programming

Even a small OTP layout change -> rewrite entire command sequences



Example:

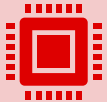
- Programming a complex non-linear OTP region.
- Run multiple fuse prog operations manually, in the right order & offsets
- One mistake can:
  - Lock a device prematurely
  - Burn the wrong lifecycle state
  - Misprogram a key region
  - Leave secure boot half-enabled

Risky and incompatible with factory automation tools

Introducing Buffer-Based Fuse Programming – fuse writebuff

Program OTP fuses using a structured buffer in memory – *fuse writebuff <addr>*

Buffer containing all necessary details for programming fuses



High-level idea:

- Construct a single buffer representing the OTP map
- Load the buffer into memory
- Let U-Boot program it
- One command, deterministic result

Fuse writebuff upstreamed into U-boot in 2025 - https://docs.u-boot.org/en/latest/usage/cmd/fuse.html

Texas Instruments

From Imperative Commands -> Data-Driven Provisioning

- ❌ Imperative model (fuse prog):
  - “How & What to program” encoded in commands
- ✅ Declarative model (fuse writebuff):
  - “What to program” defined as structured data
  - “Consistent mechanism” to apply across platforms
- Enables:
  - Batch programming
  - Strict validation before execution
  - Automation-friendly workflows

- Changed the model and redesigned the abstraction

Towards Scalable, Production-Ready eFuse Provisioning

Solves several long-standing issues

- Makes non-linear and change in OTP maps manageable
- Eliminates manual sequencing and misalignment
- No command-scripting errors
- Reduces programming time
- Easy integration with flashing automation flows
- Enables reproducible provisioning across devices

Production Considerations

- Version-control buffers
- Validate buffer boundaries
- Always verify after programming
- Lock bits written last
- Capture failures & fallback paths

TI's AM62L Processor Case Study

Step 1: Prepare the buffer and load into memory

```
U-Boot 2026.01-ti-gee3048ee0822 (Apr 09 2026 - 00:09:07 +0000)

SoC:   AM62LX SR1.1 HS-FS
Model: Texas Instruments AM62L3 Evaluation Module
```

```
=> setenv myscript 'mw 0x82000000 0 0x400; fatload mmc 1:1 0x82000000 keywrlite_ss_tool_commit_744244f_tifs_release.bin; md 0x82000000'; run myscript
636 bytes read in 32 ms (18.6 KiB/s)
82000000: 00000001 00009045 02209012 00000100  ....E..... .....
82000010: 00000000 00000000 00000000 00004a7e  ............~J..
82000020: a5a5a55a 00000000 00000000 00000000  Z...............
82000030: 00001234 5aa5a55a b002601f b7b0d97c  4...Z..Z.`..|...
82000040: a89c7dc4 7be5aad1 a184878e 2b6b632f  .}.....{..../ck+
82000050: 987d0d76 97188fa1 f2d0df60 b10c2b3e  v.}.....`...>+..
82000060: c7edc70e 3dacedc6 e0efdf9b ff3fdced  .......=......?.
82000070: 87ade97f 7d529551 00000000 00000000  ....Q.R}........
82000080: 00009ffc 5aa5a55a 78e24b38 b20ea5a7  ....Z..Z8K.x....
82000090: acfffd5a 06d38b2e 513b2af8 05f870a7  Z........*;Q.p..
820000a0: b9de9d6c 3d0d1bf3 3e06a03e 7a12e36d  l.....=>..>m..z
820000b0: 44a1c847 0de1c73f 16b5ffad b4aeae01  G..D?...........
820000c0: e007d699 80cd7428 00000000 00000000  ....(t..........
820000d0: 00005678 5aa5a55a 00000003 00000000  xV..Z..Z........
820000e0: 00000000 000062c8 a5a5a55a 00000001  .....b..Z.......
820000f0: 00000000 00000000 00008bad a5a55a5a  ............ZZ..
=>
```

Step 2: Program using fuse writebuff

```
=> fuse writebuff -y 0x82000000
Programming fuses using a structured buffer in memory starting at addr 0x82000000
NOTICE:  Writebuff version info: 0x1
=>
```


Texas Instruments

TI's AM62L Processor Case Study

- Secure boot enabled

```
U-Boot 2026.01-ti-gee3048ee0822 (Apr 09 2026 - 00:09:07 +0000)

SoC:   AM62LX SR1.1 HS-SE
Model: Texas Instruments AM62L3 Evaluation Module
```

- Fuse read user readable OTP data

```
=> fuse read 0 0 32
Reading bank 0:

Word 0x00000000: abcdef56 00000000 00000000 00000000
Word 0x00000004: 00000000 00000000 00000000 00000000
Word 0x00000008: 00000000 00000000 00000000 00000000
Word 0x0000000c: 00000000 00000000 00000000 00000000
Word 0x00000010: 00000000 00000000 00000000 00000000
Word 0x00000014: 00000000 00000000 00000000 00000000
Word 0x00000018: 00000000 00000000 00000000 00000000
Word 0x0000001c: 00000000 00000000 00000000 00000000
=>
```


Texas Instruments

Q&A

- Contact Information:
  - Harsha Vardhan V M <h-vm@ti.com>
  - Kavitha Malarvizhi <k-malarvizhi@ti.com>
- Also on IRC @ libera.chat #linux-ti

Learn more about TI products

- https://www.ti.com/linux
- https://www.ti.com/processors
- https://www.ti.com/edgeai

Why choose TI MCUs and processors?

Scalability

Our products offer scalable performance that can adapt and grow as the needs of your customers evolve.

Efficiency

We design products that extend battery life, maximize performance for every watt expended, and unlock the highest levels of system efficiency.

Affordability

We strive to make innovation accessible to all by creating cost-effective products that feature state-of-the-art technology and package designs.

Availability

Our investment in internal manufacturing capacity provides greater assurance of supply, supporting your growth for decades to come.