Trust官员 Firmware

Trusted Boot
Agenda

• Concept of trusted boot
• Bootloader in TF-M
• Firmware upgrade
• Alternatives for upgrade
• Alternatives for crypto
• Plans
• Q&A
What is trusted environment?

An integrated execution environment (HW + SW) which can protect valuable assets against extraction:

- Sensitive user data
- Crypto keys
- Firmware itself, etc.
Introduction to trusted bootloader concept

What?
SWG whose aim is to verify the origin and integrity of other SW components which run on the target system.

- Bootloader runs as soon as system is released from reset prior any other SW. In case of successful authentication it passes execution to the runtime firmware.

Why?
One wants to ensure that only a certain set of SW, without any external modification, can run on a particular device.

- Device contains sensitive assets which could be extracted with the usage of malicious SW.

How?
Device contains immutable SW and data, which can be used for authentication:

- Integrity of SW:
  - Checking hash value
- Origin of SW:
  - Checking digital signature
Considerations at selection of bootloader

Secure boot requirements

PSA spec defines boot and firmware update requirements:

- Support for firmware upgrade
- Support for chain-of-trust
- Support for NIST or NSA approved cryptographic algorithm: SHA2, RSA, ECDSA, HMAC, KDF
- Etc.

Device constraints

Device constraints mandate `yet-another` bootloader:

- Usually less than 1 MB flash memory for code
- Usually less than 256 KB RAM for data
- Usage of cryptographic accelerator HW component
- Computing power
- No MMU, no memory virtualization
- Power failure awareness
- Etc.
Bootloader in TF-M

MCUBoot is utilized to act as BL2 in TF-M:

- Open source project with Apache 2.0 licensing
- Low memory footprint; designed for 32 bit microcontrollers
- Running from flash (currently XIP)
- Several secure boot features are supported for firmware authentication: SHA256, RSA-2048, (ECDSA)
- Usage of 3rd party libraries for cryptographic operations: mbedTLS, (TinyCrypt)
- Firmware update with image swapping
- Power failure resistant upgrade
- Fallback mechanism to stable version
First bootloader release

MCUBoot integrated within TF-M repository:

Customized to be OS agnostic
Currently SHA256 and RSA-2048 are supported
SPE and NSPE are concatenated to a single binary blob
Hash and digital signature tooling and runtime check
Software Upgrade prototype as proof of concept:
  • Emulating flash interface and behaviour over code SRAM
System constraints:
  • No support for image size that does not fit in available RAM
  • CoT reduced to verify SPE and NSPE in the same go
## Chain of trust

- **Immutable BL1 code**
  - Hash
  - Check integrity
  - Public key
    - SIP/OEM Root of Trust Public Key (ROTPK)

- **BL2 image**
  - Check signature
  - OEM/Developer Public Key(s)
  - Check signature

- **SPE image**
  - Check signature
  - Public image signing keys can be rotated

- **NSPE image**
  - Check signature

Use CA public key if supporting certificate revocation
## Boot process

<table>
<thead>
<tr>
<th>Stage</th>
<th>BL1</th>
<th>BL2</th>
<th>SPE</th>
<th>NSPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSA (not mandatory)</td>
<td>Immutable boot code in ROM</td>
<td>Boot code in eFlash</td>
<td>Secure runtime firmware</td>
<td>RTOS &amp; Application</td>
</tr>
<tr>
<td>TF-M</td>
<td>TBD</td>
<td>BL2 MCUBoot</td>
<td>Core SPM Secure services</td>
<td>RTOS &amp; Application</td>
</tr>
</tbody>
</table>

Combined hash and signature

Verify
Basic operation and memory layout

CPU released from reset

BL2 - Bootloader started

Initialize phase

Is there aborted swap?

yes

Finalize aborted swap

no

Is new SW in SLOT_1

yes

Authenticate SW in SLOT_1

no

Erase SLOT_1

Valid SW?

no

Pass execution to SPE in SLOT_0

yes

Swap images between SLOT_0 and SLOT_1

Scratch area

TLV(SHA, DS)

Non-secure firmware

Secure firmware

Header

Trailer(Swap status)

TLV(SHA, DS)

Non-secure firmware

Secure firmware

Header

BL2 - Bootloader

0xXXXX... Used during image swapping

SLOT_1 Placeholder for new image

SLOT_0 Active image

Currently not updatable 0x0000...
Image swapping

- Code linked to Slot_0 memory space
- Divided into rounds
- Scratch-sized data is moved in one go
- Status info saved after each round
- Power failure safe

**Active image Slot_0**

- Slot_0_Sector_0
- Slot_0_Sector_1
- Slot_1_Sector_2
- Slot_1_Sector_3
- Slot_1_Sector_N

**New image Slot_1**

- Slot_1_Sector_0
- Slot_1_Sector_1
- Slot_0_Sector_2
- Slot_0_Sector_3
- Slot_0_Sector_N

**Scratch area**

1.) Erase scratch
2.) Copy Slot_1 to scratch
3.) Copy scratch to Slot_0
4.) Copy Slot_1 to scratch
5.) Erase Slot_0

Save swap status info
Firmware upgrade

- Upgrade is a task of runtime FW
- Potentially split between NSPE and SPE
- XIP images
Image fallback

- Store previous image
- Health-check new image with BIST
- Self confirmation
- Reboot in case of failure
- Revert back stable image
- Set rollback after confirmation
Design constraints

Header size - VTOR alignment:
- Device dependent 512-1024 bytes

Image slot’s layout must be aligned

Scratch area size:
- Flash memory wear-out
- At least as the largest block size

Real image size smaller than image slot:
- Image header, TLV, swap status info, etc.

No recovery option, if both images are faulty
## Common threats

<table>
<thead>
<tr>
<th>Threat</th>
<th>Mitigation</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malicious firmware sent to device</td>
<td>Signed firmware images</td>
<td>Yes</td>
</tr>
<tr>
<td>Downgrade to old vulnerable version</td>
<td>Version or fallback counters check</td>
<td>Not yet*</td>
</tr>
<tr>
<td>Persistent malware(rootkits)</td>
<td>Immutable boot code and data(BL1)</td>
<td>Not yet</td>
</tr>
<tr>
<td>Remote bricking of the device</td>
<td>Backup image</td>
<td>Yes</td>
</tr>
<tr>
<td>Attacker gets signing key</td>
<td>Key revocation support</td>
<td>Not yet*</td>
</tr>
</tbody>
</table>

*: Planned to be addressed in 2018
# Alternatives to image swapping

## Position independent code

**Pros:**
- Reduced P/E cycle leads to longer lifetime
- Reduced BL complexity and code footprint
- Reduced boot-up time (no swapping)

**Cons:**
- Might lead bigger firmware code footprint
- Some compiler switches are not compatible with PIC code
- Some C lib (Microlib) cannot be compiled to be PIC
- Other constraints when compiling code to be PIC

## Dual image build

**Pros:**
- Reduced P/E cycle leads to longer lifetime
- Reduced BL complexity and code footprint
- Reduced boot-up time (no swapping)

**Cons:**
- More complex build process
- Extra logic in update client
Alternatives to image swapping

**Execute from RAM**

**Pros:**
- Reduced P/E cycle leads to longer lifetime
- Faster firmware execution
- Reduced BL complexity and code footprint

**Cons:**
- Usually infeasible: less RAM than ROM

**Off-chip storage**

**Pros:**
- Reduced P/E cycle leads to longer lifetime
- Reduced BL complexity and code footprint

**Cons:**
- Might be a security risk: when to verify signature?
- Might require image encryption, increased code footprint (include AES) and boot-up time
Alternatives to image swapping

Overwrite

Pros:
• Reduced P/E cycle leads to longer lifetime
• No need for scratch space
• Reduced BL complexity and code footprint

Cons:
• Risk of bricking the device because no revert possible
MCUBoot as PIC code

Experiment to compile PIC code:

- RO and RW position independent (--ropi, --rwpi)
- Vector table and IRQ handlers must be in RAM
- IRQ handling unavailable until vectors and handlers relocated to RAM
- Image size increased:
  - 29KB -> 38KB; More std. C lib was compiled-in
- Limitations on source code:
  - Constant pointer cannot be used
  - CMSE armclang flag is not compatible with ROPI
  - Microlib cannot be compiled to be position independent
# Comparison of crypto algorithms

**RSA**
- Big key size: up to 15KB
- 128 bit level of security: RSA-3072
- ROM size (mbedTLS): ~14KB
- RAM usage (mbedTLS): ~7KB
- Key generation: slower
- Signature generation: slower
- Signature verification time: faster

**ECC**
- Small key size: up to 512 bits
- 128 bit level of security: ECC-256
- ROM size (mbedTLS):
- RAM usage (mbedTLS): ~13KB
- Key generation: faster
- Signature generation: faster
- Signature verification: slower

Moving from RSA to ECC
"Speed up asymmetric crypto"

Signature verification with RSA or ECC is time consuming
Symmetric crypto can spare clock cycles

Replace asym. crypto with symmetric: HMAC, CMAC, etc:

- Previously verified images (upgrade time) can get a MAC, generated based on Hardware Unique Key (HUK)
- At boot time this MAC is verified instead of original signature
- Boot time can be significantly reduced
Alternatives for crypto libraries

HW accelerator:
- Improved performance / reduced code footprint

CryptoCell-312:
- Symmetric and asymmetric crypto
- Runtime library: use mbedTLS API
- Boot library:
  - Signature verification
  - X509 certificate parsing
  - Image verification and optional decryption
- Asset provisioning to OTP memory
- Rollback counters
Bootloader plans

PSA compliance:
- Anti-rollback protection
- Create interface between SPE and bootloaders
- Add support of multiple chains of trust and might be certificates

Explore possibilities to make BL2 updatable
Integrate crypto HW accelerator (CC312) with BL2
How to get involved

TF-A and TF-M master codebases

- [https://git.trustedfirmware.org/](https://git.trustedfirmware.org/)

TF-M Team @ Connect HKG18

- Abhishek Pandit
- Ashutosh Singh
- Tamas Ban
- Miklos Balint

Get in touch

- Come round LITE hacking room between 3-4 pm Wednesday
- Schedule a meeting via [hkg18.pathable.com](http://hkg18.pathable.com)

More info on [developer.arm.com](http://developer.arm.com)
Thank You!
Danke!
Merci!
谢谢!
ありがとう!
Gracias!
Kiitos!
감사합니다
धन्यवाद
Supported platforms

MCUBoot with TF-M can run on:

- In simulator environment (FVP) on PC.
- MPS2 development board with AN521 (Castor) FPGA image
- MPS2 development board with AN519 (M23) FPGA image
- Musca_A porting is in progress
Scratch area

TLV(SHA, DS)
Non-secure firmware

Secure firmware
Header
Trailer(Swap status)

TLV(SHA, DS)
Non-secure firmware

Secure firmware
Header

BL2 - Bootloader

SLOT_1
Placeholder for new image

SLOT_0
Active image

0xXXXX... Used during image swapping

Currently not updatable
0x0000...
1. Download:
   - firmware
   - manifest

2. Authentication:
   - send manifest

3. Provision new image

4. Reboot

5. Authenticate and swap images, start image in slot_0