MCCI Trusted Bootloader

2021-04-23

Terminology

What are we talking about?

- **Bootloader**: software that gets control after system reset and prepares the system to execute an application
- **Trusted bootloader**: bootloader that ensures that the application that is launched is not corrupt and is trusted. Not necessarily hardened against malware and physical access
- ➢ **Trusted Updating Bootloader:** trusted bootloader that can apply firmware updates to the application. Must be robust against power failures, and intentionally or unintentionally corrupted images.
- **Secure bootloader**: trusted bootloader that resists tampering and is hardened against malware and physical access. Usually involves a trusted secure element (hardware component) and often includes CPU features like ARM's TrustZone.

MCCI's bootloader is a Trusted Updating Bootloader.

Goals and non-goals

Raison d'être:

- Enable firmware update in the field without opening the door to mass hijacking of devices

Secondary goals:

- Allow some level of assertion about system integrity
- Make firmware update robust in face of normal problems like power failures and communication problems
- Keep the development cycle simple

Non-goals:

- Not defending against enemies with physical access to devices
- Not defending in depth against malware
- Not defending in the field against theft or loss of
- Not worrying about full X.509, certification revocation, etc.

Basic technology and tools

Secure Hash:

- one-way function that maps a large image into a smaller fixed-size string of bytes.
- Key attribute: it must be infeasible to change a file in such a way that the hash doesn't also change
- The hash must be easy for anyone to calculate.

Public Key Signature:

- The creator of a signature creates a key pair (secret key, public key), which are used in the following.
  - A generating function creates a byte string (signature) from an input string and the secret key
  - A checking function takes the signature, the input string, and the public key, and confirms that the signature matches the input string.
- Given a public key and an input string, it must be computationally infeasible to generate a signature.
- The public key is published; the secret key (or private key) must be kept secret in order to make signatures probative.

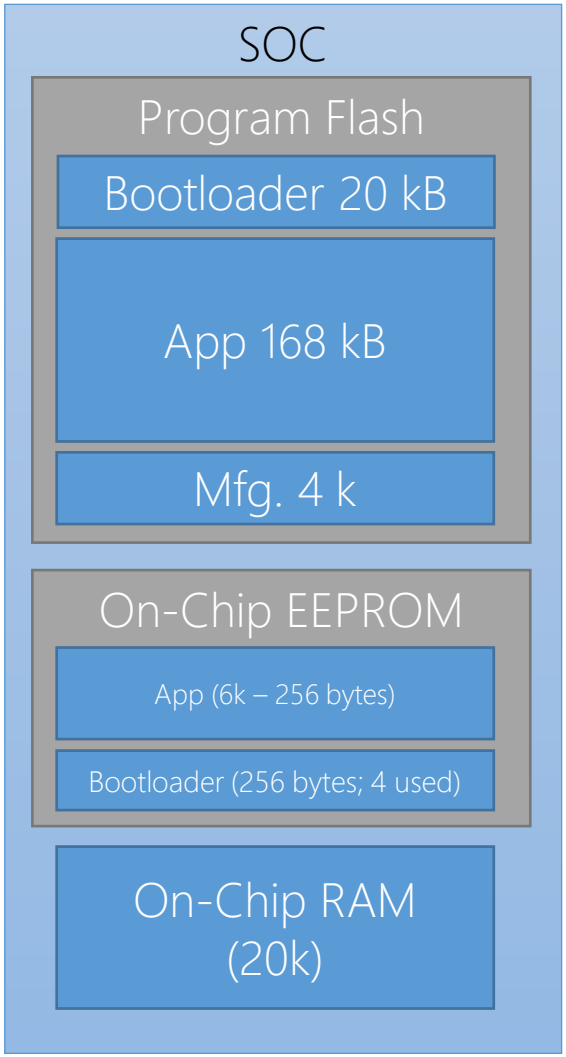
System assumptions

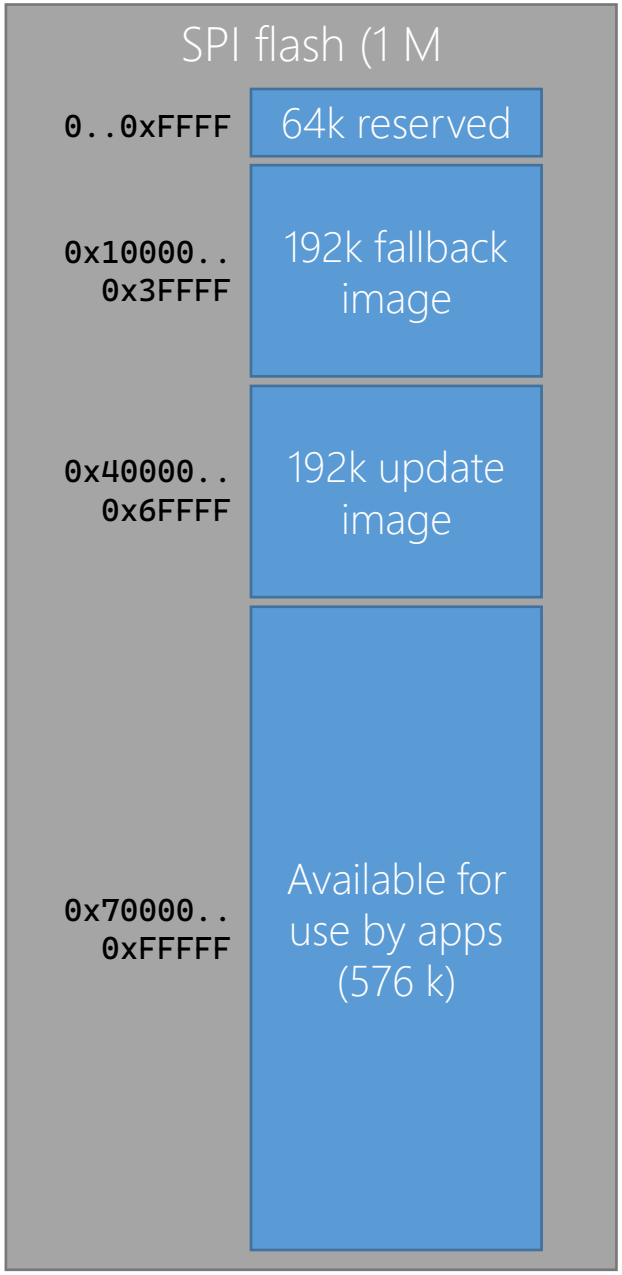
- The system is assumed to have the following components
  - Program memory: this is where the executing system program resides. It generally is non-volatile in smaller SOCs (but it need not be). On the reference device, it's implemented as Flash EPROM memory, and so we generally say "program flash" as shorthand.
  - Update staging memory: this is where incoming update images are stored for use by the bootloader. On the reference device, it's implemented by an external 1M byte Flash interfaced via a SPI bus, and so we generally say "SPI flash" as shorthand.

Reference device

- Our reference device is an STM32L0 processor: ARM Cortex M0+ with up to 192 kB of program flash, and up to 20 kB of RAM
- The bootloader doesn't try to supervise operation of the running system or protect itself. (It may optionally be write-protected, making it somewhat tamper proof.)
- With so little RAM and program Flash, it's generally not practical to stage the incoming application on-chip; we stage updates via SPI flash
- The methods used in the MCCI Bootloader can be used with systems having as little as 64 kB of program flash and 1 kB of RAM
- On the reference device, we reserve 20 kB for the boot loader, 4 kB for manufacturing data; the rest is available for application code and data. (The largest MCCI programs so far are about 90 kB).
- The bootloader only takes 10 kB on the reference device, so we could shrink the reserved area, or else put additional common features in the bootloader via an API, including support for chains of keys. ("Bob signed this image, Alice trusts Bob, and I trust Alice" in addition to the basic "Alice signed this image, and I trust Alice".

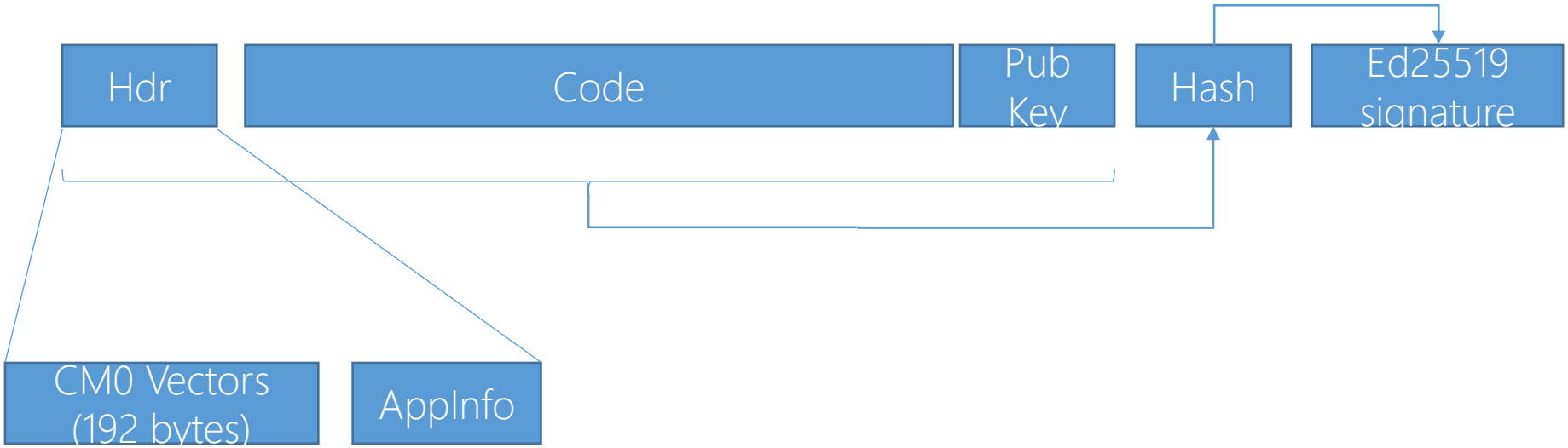
Internal & external memory





How we decide to trust an image

- Two step process
  - Calculate a SHA-512 hash over the image
  - Use the signature (and the public key) to confirm that the hash was signed by the owner of the key
  - Note that the asserted public key is included in the hash



Critical Assumption

- The bootloader assumes that images in program storage were either put there by someone with physical access, or were put there by the bootloader itself.
  - The bootloader checks the hash at each boot, to detect corruption
  - The bootloader does **not** check the signature at each boot; it assumes that this was done prior to programming the image.
- This is done to speed up boots, not because of philosophy; it would be arguably better to check signature each time.

AppInfo contents

Bytes	Name	Content	Discussion
0..3	**magic**	'MAP0', or 0x3050414d	Dedicated value indicating that this is an AppInfo block.
4..7	**size**	0x40	Size of the AppInfo block.
8..11	**targetAddress**	base address	0x08000000 for the boot loader, 0x08005000 for application images.
12..15	**imageSize**	size of image in bytes	Does not include hash and signature data.
16..19	**authSize**	0xC0	size of authentication info at end of image
20..23	**version**	version	Semantic version of app. Byte 23 is major version, 22 is minor version, 21 is patch, and 20 (if non-zero) is the pre-release indictor. Note that prior to comparing semantic versions in this format, you must decrement the LSB modulo 256, so that pre-release 0x00 is greater than any non-zero pre-release value.
24..31	**posixTime**	seconds since epoch	Normal Posix time; expressed as a 64-bit integer to avoid the year 2037 problem.
32..47	**comment**	UTF8 text, zero-padded	A comment, such as the program name.
47..63	**reserved32**	reserved, zero	Reserved for future use.

Bootloader APIs

- Since the bootloader lives in the same address space as the application, it exports APIs that may be used by the application to allow code reuse and bootloader interrogation
- These are found at runtime by the app examining the bootloader image
- APIs so far include:
  - Get the address of the update flag
  - SHA-512 hash primitives (initialize, process blocks, finish)
  - Verify64 API
- Planned for future inclusion
  - Access to other crypto APIs
  - Access to any info about the boot process (what did the bootloader decide to do & why)