Raspberry Pi 4 Boot Security

Raspberry Pi Ltd

2023-06-28: githash: fc34cc2-clean

Colophon

build-date: 2023-06-28
build-version: githash: fc34cc2-clean

Legal Disclaimer Notice

TECHNICAL AND RELIABILITY DATA FOR RASPBERRY PI PRODUCTS (INCLUDING DATASHEETS) AS MODIFIED FROM TIME TO TIME ("RESOURCES") ARE PROVIDED BY RASPBERRY PI LTD ("RPL") "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW IN NO EVENT SHALL RPL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THE RESOURCES, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

RPL reserves the right to make any enhancements, improvements, corrections or any other modifications to the RESOURCES or any products described in them at any time and without further notice.

The RESOURCES are intended for skilled users with suitable levels of design knowledge. Users are solely responsible for their selection and use of the RESOURCES and any application of the products described in them. User agrees to indemnify and hold RPL harmless against all liabilities, costs, damages or other losses arising out of their use of the RESOURCES.

RPL grants users permission to use the RESOURCES solely in conjunction with the Raspberry Pi products. All other use of the RESOURCES is prohibited. No licence is granted to any other RPL or other third party intellectual property right.

HIGH RISK ACTIVITIES. Raspberry Pi products are not designed, manufactured or intended for use in hazardous environments requiring fail safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, weapons systems or safety-critical applications (including life support systems and other medical devices), in which the failure of the products could lead directly to death, personal injury or severe physical or environmental damage ("High Risk Activities"). RPL specifically disclaims any express or implied warranty of fitness for High Risk Activities and accepts no liability for use or inclusions of Raspberry Pi products in High Risk Activities.

Raspberry Pi products are provided subject to RPL's Standard Terms. RPL's provision of the RESOURCES does not expand or otherwise modify RPL's Standard Terms including but not limited to the disclaimers and warranties expressed in them.

Document version history

Release	Date	Description
1.0	30 April 2021	Initial draft
1.1	18 October 2021	Remove NDA requirements
1.2	17 December 2021	Copy edited
1.3	07 January 2022	Public release
1.4	31 May 2023	Update to latest secure boot information

Scope of document

This document applies to the following Raspberry Pi products:

Pi 0			Pi 1		Pi 2		Pi 3	Pi 4	Pi 400	CM1	CM3	CM4	Pico
0	W	H	A	B	A	B	B	All	All	All	All	All	All
								*	*			*	

Introduction

This white paper describes Raspberry Pi Ltd's approach to boot security on the Raspberry Pi 4 family of devices, based on the BCM2711 system on a chip (SoC).

Goals

Raspberry Pi Ltd's goals for boot security are as follows:

- Enable industrial customers to ensure that a Raspberry Pi 4 only runs software authorised by them.
- Ensure customers have full control of the operating system (OS) image and sign it with their own RSA private key.
- Avoid centralised code-signing servers — the OS image and signing tools are open source and are run by the customer.
- Provide a solution that runs on a standard Raspberry Pi 4 — no external hardware requirements.
- Provide a solution that supports remote software updates, e.g. kernel patches.

Notable improvements over earlier models than Raspberry Pi 4 include:

- The RSA read-only memory (ROM) keys are public, so no shared secrets can be exposed by, for example, decapping or JTAG access. The private RSA key for signing bootloader firmware is kept secure inside Raspberry Pi Ltd and is not released to customers.
- The process to bind the device to a customer's public key is public.
- A single `boot.img` that combines all the previously separate firmware and configuration files is provided for the third-stage boot, meaning more robust atomic updates and preventing a non-recoverable failure of the system.
- Configurations can be validated prior to committing to one-time programmable (OTP) memory.

Limitations

Limited hardware support is offered on the BCM2711 SoC:

- No Secure OS, e.g. TrustZone, or secure processors.
- No digital rights management (DRM) or high-bandwidth digital content protection (HDCP).
- Limited support for private keys in OTP. Hardware protection here is limited; there is the ability to store a private key in OTP, but there is no secure enclave.
- Fully secure OS boot is only available on the BCM2711 B1/C0 steppings.

Out of scope of this document

- Instructions on making and installing signed images; this is covered in a separate document.
- Encrypting the boot partition.
- Creating a security-enhanced version of Raspberry Pi OS.

Software security on Raspberry Pi 4

Signed security is only available on the B1/C0 stepping of the BCM2711. The B0 stepping does not have the RSA public keys in its ROM that would be needed for the ROM to verify the serial peripheral interface (SPI) electrically erasable programmable ROM (EEPROM) bootloader.

The stepping revision of the chip is etched on the top of the chip itself, as a subfield of the part number. Alternatively, once booted you can determine the stepping using the following command:

```
sudo busybox devmem 0xfc404000
```

This will return 0x27110010 for BCM2711B0, 0x27110011 for BCM2711B1, or 0x27110020 for BCM2711C0.

NOTE

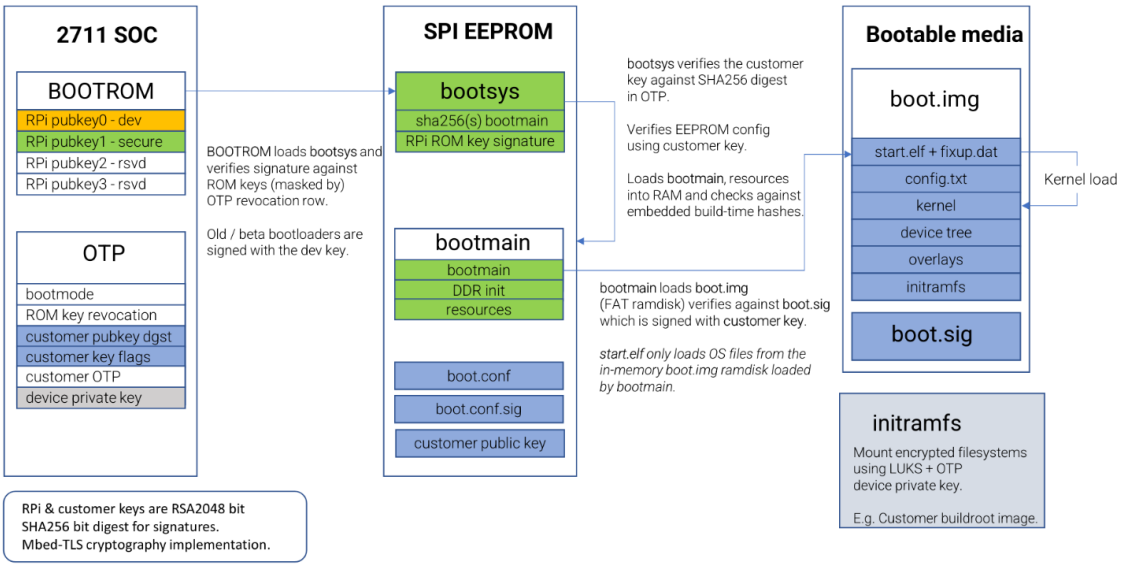
Raspberry Pi Compute Module 4 and Raspberry Pi 400 are only available with the C0 or newer stepping. Older BCM2711-based devices may or may not use the B0, B1, or C0 stepping, which may limit the applicability of the security process.

Overview of the secure boot process

- Signed boot images provide a chain of trust from the ROM to the kernel and the `initramfs` initial ramdisk, including graphics processing unit firmware and configuration data.
- The boot image files are file allocation table (FAT) disk image files containing the normal files from the boot partition as a single atomic file. These are loaded as a ramdisk loopback file system by the bootloader.
- Signed boot images can be loaded from any boot mode, e.g. network boot.
- OS updates are supported by downloading a new `boot.img` file, which is effectively an atomic update.
- Signed boot is enabled via OTP as part of the customer's usual Raspberry Pi CM4 provisioning process. Once enabled, signed boot cannot be switched off.
- EEPROM code is signed by Raspberry Pi Ltd and verified by RSA key in the ROM. There is no ability on the BCM2711 to countersign the Raspberry Pi Ltd code with a customer key.
- In secure boot mode the bootloader does check the hash of the VL805 USB hub firmware, but please note that the VL805 ROM itself does not support code signing of its firmware.

A brief description of the chain of trust

The diagram below shows the root of trust for the secure boot process.



The process is as follows:

- The device boots, and if secure boot is enabled in the OTP, the ROM in the BCM2711 will verify the second-stage bootloader against each boot ROM key in order, taking into account whether the key has been previously revoked by comparing it against the OTP revocation bits.
- If the verification succeeds then the Bootsys in SPI EEPROM is executed; if not, the boot will be halted.
- Bootsys verifies that the customer key in SPI EEPROM matches the hash of the key stored in the OTP.
- Bootsys verifies the SPI EEPROM config against the customer key.
- Bootsys now executes Bootmain.
- Bootmain does some setup, then loads the `boot.img` into a ramdisk.
- Bootmain verifies the `boot.img` against the `boot.sig` file.
- Bootmain loads the `start.elf` file from the ramdisk and runs it.
- `start.elf` loads the Linux kernel and associated device trees and overlays from the ramdisk, and the main Linux boot is started.

The individual parts of each process are covered in detail in the next sections.

ROM verification process for the second-stage bootloader

If secure boot is enabled in the OTP, then the ROM will verify the second-stage `bootloader/recovery.bin` image against the ROM keys before launching it.

- The signature only applies to the second-stage executable. There is no check for the entire EEPROM image.
- The ROM also checks the signature of `recovery.bin` when loaded via usbboot (rpiboot).
- The new bootloader that supports signed boot will be signed with the signed-boot ROM key. It is possible to revoke the ROM development key by setting an OTP bit, which effectively revokes all previous bootloader releases on that board.

Second-stage EEPROM resources

Before loading the `start.elf` file, the second-stage bootloader loads additional resources, e.g. the VLI universal serial bus

(USB) firmware. These resources are considered to be immutable, and if signed boot is enabled (via OTP) then the SHA256 hash of the resource is checked as it is read into memory. This is compared against SHA256 hashes generated at build time and compiled into the `bootcode.bin` binary. Since `bootcode.bin` has already been verified by the ROM, the compiled hashes can be trusted.

Developing images for the secure boot process

It is possible to test and develop the secure boot process without making permanent changes to the OTP on the BCM2711. This allows the testing of images to ensure they are correct before finally enabling secure boot by making permanent and irrevocable changes to the OTP.

During this development stage, a key is programmed into the EEPROM, which is then made read-only. This improves security for trivial file transfer protocol (TFTP) boot or for kiosk-type applications where a USB or Secure Digital (SD) card may be accessible. However, it would be susceptible to replacement of the EEPROM, so this must be assumed to be physically secure. In short, secure boot is suitable for environments where physical access to the board is restricted, as well as for developing signing for test purposes.

You can develop the signed boot process to this stage using the following steps:

- Customer builds and signs `boot.img` with their RSA private key.
- The EEPROM image is updated to include the customer's RSA public key.
- The EEPROM image is flashed using the Raspberry Pi CM4 provisioning process, and EEPROM write protect is enabled.

Finally securing the boot process

This extends past the development level by storing a digest of the customer key in OTP, and disables the key in ROM. The bootloader will only load `boot.img` files signed with the customer RSA key, and the ROM prevents any attacker from replacing the onboard EEPROM with one containing their code.

Even this level of signing would not protect against the replacement of the Raspberry Pi CM4 or the SoC itself in an end device. If this is a potential concern, the customer may wish to use customer OTP rows to store a decryption key to decode, for example, the `rootfs` file system.

Features include:

- Permanently enables signed boot: only signed boot images can be loaded once set up.
- Revokes the ROM 'development key': it is not possible to downgrade the SPI EEPROM bootloader to an older or non-secure version.
- An SHA256 digest of the customer's public key is written to OTP and validates the customer's RSA public key in the EEPROM.
- The bootloader EEPROM configuration must be signed with the customer's private key, which restricts which boot modes will be tried.
- If the EEPROM public key or configuration is not valid then the boot stops.

Once security flags are set during the Raspberry Pi CM4 EEPROM flashing process, they cannot be unset.

Implementation details

boot.img

The `boot.img` file is a disk image of up to 180MB of a raw block device containing a FAT file system which is mounted as a ramdisk/loopback file system. If signed boot is required then the RSA signature is verified before the bootloader mounts the ramdisk.

In secure boot mode, the bootloader only looks for `boot.img` or `boot.sig` when scanning through the currently available set of bootable media, i.e. SD, USB, NVMe, network or RPIBOOT, as defined by BOOT_ORDER. Once it finds a matching file, it reads the entire file into memory, verifies the signature, and if everything is as expected, it mounts the file contents as a ramdisk. If `boot.img` or `boot.sig` is not found, then that particular BOOT_ORDER is treated as 'firmware not found', and the bootloader moves to the next boot mode.

Scripts in the usbboot repository are provided to automate the process of creating a `.img` file from a directory of source files (`start.elf`, kernel, etc). `rpi-eeprom-config` has been updated to provide support for signing the EEPROM configuration and storing the RSA public key in the EEPROM image.

If the 'tryboot' flag is set on a reboot (`sudo reboot "0 tryboot"`), then the bootloader will search for `tryboot.img` instead of `boot.img`. This allows signed boot images to be tested before being activated.

The source for `boot.img` is a directory containing the files from the boot partition of Raspberry Pi OS. To reduce the load time, files that are not relevant for the given board (e.g. unused kernel images) can be left out. Since the boot image is just a single file, it is easy to update it to a new version.

Firmware compatibility checks

The bootloader verifies that `start.elf` supports ramdisk images before executing it. If not, a version compatibility error message is displayed, and booting stops. The firmware compatibility flags are encoded in a special section of the `.elf` file, and in future the boot image scripts will also verify this to avoid accidentally creating non-bootable image files.

Extra steps required for Raspberry Pi 4 and Raspberry Pi 400

The ROM does not support loading `recovery.bin` from the SD card in secure boot mode. This means that for Raspberry Pi 4 and 400 the nRPIBOOT programming method must be used; this requires a general-purpose input/output (GPIO) pin to be selected that when pulled low will enable nRPIBOOT mode. This is not necessary on Raspberry Pi CM4 as a GPIO pin is preprogrammed during manufacture; this is connected to a jumper on the Raspberry Pi CM4 IO board.

Several GPIO pins can be used for nRPIBOOT on Raspberry Pi 4 and 400: pins 2, 4, 5, 6, 7, and 8. There is an additional constraint in that any HAT (Hardware Attached on Top) or GPIO-connected device being used should not pull the selected pin low by default, as this will enable nRPIBOOT rather than SD card booting.

The recommended GPIO pin is GPIO8 (SPI CE0), which is not pulled low by default. Once chosen, the GPIO to be used as the boot selector is stored in the OTP, and is set using `recovery.bin`.

Please refer to the GPIO section in the BCM2711 datasheet for more details on default GPIO allocations.

More information

Performance

The overhead of verifying a minimal `boot.img` (about 9MB) is approximately 3.5 seconds, which directly influences the boot time. Most of this is the CPU overhead of the SHA256 calculation, and hence unavoidable.

Other security measures

When making a Raspberry Pi Ltd system secure other areas will need consideration. The necessary steps might include (but are not limited to):

- Change default usernames and passwords.
- Set SSH security to allow key-based access only.
- Disable sudo access.
- Disable login as 'root'.
- Remove the `pi` user from the GPIO group.
- Install firewalls.
- Improve server security, e.g. install fail2ban.
- Disable any unnecessary BOOT_MODES in the EEPROM config BOOT_ORDER, e.g. remove network boot if not required.
- Use LUKS to encrypt the root filesystem.
- Set the `eeprom_write_protect=1` flag in `config.txt` when enabling secure boot, and then pull `nWP` low to prevent OS-level security vulnerabilities from modifying the firmware, so that hacks are less likely to survive a reboot.

The details of making these changes to your operating system, should they be required, are outside the scope of this white paper, but can be found via a web search.

Answers to common questions

1. *Which cryptographic algorithms for firmware authentication are used?*

   RSA2048.

2. *Where is the root of trust stored?*

   Raspberry Pi Ltd.

3. *Which secure boot-related keys can and cannot be set by the customer?*

   The boot ROM contains four keys that are secured by Raspberry Pi Ltd and cannot be changed by the customer. Only two are currently used. All of these keys can be revoked using OTP. The customer can set a key in EEPROM that is used for signing the customer image. This key is protected by a hash in OTP.

4. *How is secure boot mode enabled and persisted in the device's configuration?*

   Flags in the BCM2711 on-board OTP.

5. *Does the chip support an anti-rollback for firmware images?*

   No; it is possible to roll back to earlier versions of **signed** software.

6. *In which type of memory is the boot ROM implemented?*

   Mask-based ROM.

7. *Which debug interfaces are available?*

   JTAG. A flag can be set in OTP to disable JTAG access to the VideoCore processor; once set, access cannot be re-enabled.

8. *What hardware crypto is supported?*

   The BCM2711 has no hardware cryptographical support.

9. *What is the type of key storage?*

   There is no dedicated key storage on the BCM2711. A customer key can be stored in OTP, but this is potentially visible to the `root` user within the signed OS image.

10. *Is there a hardware random number generator, and if so what is the entropy source for it?*

    There is a hardware RNG, but no implementation details are available.

11. *Are any Secure Execution Environments supported?*

    No.

12. *Which external or internal non-volatile memories, such as flash, does the BCM2711 support?*

    SPI EEPROM and SD/EMMC are supported by the boot ROM for loading the second stage. Secure boot disables the SD/EMMC ROM boot mode.

13. *Does the chip support execute-in-place from external non-volatile memory?*

    No.

Other

There is a buildroot-based secure boot example in the Raspberry Pi Ltd GitHub repository. It should be possible to simply copy and paste the bash commands and see secure boot working by following these instructions:

Secure boot example