How To Use Raspberry Pi Secure Boot

Raspberry Pi Ltd

2023-07-05: githash: b3a30d1-clean

Colophon

build-date: 2023-07-05
build-version: githash: b3a30d1-clean

Legal Disclaimer Notice

Document version history

Release	Date	Description
1.0	14 April 2021	Initial release
1.1	24 June 2021	Update to latest schemes
1.2	18 October 2021	Remove NDA requirements
1.3	22 October 2021	Remove USBBOOT branch note as now on master
1.4	10 November 2021	Several corrections, additional workflow description
1.5	14 February 2021	Correct the `rpiboot` path
1.6	17 January 2022	Copy edited
1.7	27 April 2022	Public release
1.8	9 August 2022	Add verification section; add turning off signed boot section; various improvements to descriptions; removed some out-of-date text
1.9	14 June 2023	Removed deprecated boot image creation instructions and provided a link to the current instructions in the `usbboot` repository

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 how to implement secure boot on devices based on Raspberry Pi 4. For an overview of the secure boot implementation, please see the *Raspberry Pi 4 Boot Security* white paper.

This white paper assumes that the Raspberry Pi running `RPIBOOT` is running Raspberry Pi OS (Linux), Bullseye version or later, and is fully up to date with the latest firmware and kernels. The secure boot system is intended for use with `buildroot` (or similar)-based OS images; using it with Raspberry Pi OS is not recommended or supported.

Guide to using signed/secure boot

This quick start guide describes how to use the Raspberry Pi Ltd supplied scripts to create a signed and secure boot system. These scripts are designed with the aim of making the entire process very easy to carry out. Note, however, that some of the operations involved in making a Raspberry Pi boot-secure are irreversible, so you should take particular care when using these instructions. This document will warn you whenever irreversible operations are about to be carried out.

We recommend that you read this whitepaper in conjunction with the online instructions at https://github.com/raspberrypi/usbboot#secure-boot. Raspberry Pi Ltd also advises checking the GitHub `usbboot` repository to ensure you have the latest instructions and bug fixes.

Prerequisites

It is assumed that the user is using a separate device, such as another Raspberry Pi, or a laptop or similar running Linux, to do all the file system collation and encryption.

Use the following command to install essential packages on this device:

```
sudo apt install pkg-config build-essential
```

The scripts use the Python cryptographic `pycryptodomex` support module, which can be installed as follows:

```
python3 -m pip install pycryptodomex
```

or

```
pip install pycryptodomex
```

Then you will also need to clone the `usbboot` repository, which contains various tools, bootloaders, and recovery files:

```
git clone https://github.com/raspberrypi/usbboot/
```

To build the `usbboot` application for Raspberry Pi OS, Ubuntu, or similar:

```
cd usbboot
sudo apt install libusb-1.0-0-dev
make
```

This document assumes, unless stated otherwise, that all commands are executed from the `usbboot` folder cloned above.

Secure boot requires the latest firmware (September 2021). This is already present in the Bullseye release of Raspberry Pi OS, but can also be downloaded from the `firmware` repository on GitHub. The required files are in the `boot` folder.

```
git clone --depth 1 --branch stable https://github.com/raspberrypi/firmware
```

NOTE

To help with debugging any issues, it can be very useful to have a Universal Serial Bus (USB) UART (universal asynchronous receiver/transmitter) adapter connected when using `rpiboot`.

Creating an RSA key pair

An RSA (Rivest–Shamir–Adleman) key pair is required to sign the electronically erasable programmable read-only memory (EEPROM) and the boot image. To create a 2048-bit RSA private key in PEM (Privacy-Enhanced Mail) format:

```
openssl genrsa 2048 > private.pem
```

You may also need a public key, which can be generated from the private key with this command:

```
openssl rsa -in private.pem -out public.pem -pubout -outform PEM
```

WARNING

Keep these keys secure. If a key is lost, then any Raspberry Pi that has been set to use that key can no longer be updated. If a key is stolen, third parties will be able to sign their images with it; this will compromise security on any device using that key.

Creating a signed boot image

The entire process for creating a signed boot image is documented on the Raspberry Pi Ltd GitHub site:

https://github.com/raspberrypi/usbboot#secure-boot

A minimal example can be found here:

https://github.com/raspberrypi/usbboot/blob/master/secure-boot-example/README.md

Once the image has been created and copied to the boot partition of the Raspberry Pi, you can secure the system.

Update the EEPROM using `usbboot`

We now need to update the EEPROM on the device with a new signed bootloader, and add any configuration changes required.

The `usbboot` repository contains a subfolder called `secure-boot-recovery`. Although frequently used to recover a system with a bad bootloader, it is also used for customising the bootloader, and for signing it for use with secure boot.

In the folder is a file, `boot.conf`, that contains all the required configuration parameters.

The most important item for secure boot is the `SIGNED_BOOT` option. When a bootloader has this set, it will look for a `boot.sig` in the boot folder and compare that against the bootloader's inbuilt key (see the next section); if they match, it will load `boot.img` into a ramdisk and use the contents to continue the boot process.

```
# Select signed-boot mode in the EEPROM. This can be used during development
# to test the signed boot image. Once secure boot is enabled via OTP this setting
# has no effect, i.e. it is always 1.
SIGNED_BOOT=1
```

This next command will apply the configuration to the bootloader, and sign it with the specified key. By default the script will use the bootloader file `pieeprom.original.bin` and the configuration file `boot.conf` from the current working folder, although both of these can be changed on the command line using `-c` and `-i` respectively.

NOTE

Use `../tools/update-pieeprom.sh -h` to display help on all the available options.

```
cd secure-boot-recovery
../tools/update-pieeprom.sh -k ../private.pem
```

We now have a configured and signed bootloader that we can transfer to the client device using `rpiboot`. So, set up your device to enable nRPIBOOT, remove EEPROM write protection (WP low), and then run the following command:

```
../rpiboot -d .
```

NOTE

`rpiboot` with the `-d` option will boot the attached device using the boot files in the specified folder, rather than the bootloader from the eMMC. In this case it is using the current folder (`.`).

Power up your device, which should now update its bootloader to the newly signed image and then boot from the `boot.img` file.

At this stage the system is not fully secure, as it is still possible, given physical access, to boot up and replace the bootloader with an unsigned version by using `rpiboot` again. The final stage in the security process is to make changes to the OTP that will ensure that *only* a signed bootloader can boot the system.

Turning off signed booting

If you find the `boot.img` file is not booting correctly, it is still possible to replace the signed bootloader with an unsigned version, because at this stage the OTP has not yet been programmed.

The easiest way of doing this is to find a spare SD card and use Raspberry Pi Imager to create a recovery SD card. Run Imager and select `Choose OS`, then `Misc Utility Images`, then `Bootloader`, then one of the three different boot options; this will usually be `SD Card Boot`. Create the SD card, then insert it into the device and reboot. This will reprogram the eMMC bootloader to factory defaults.

NOTE

If you enabled SIGNED_BOOT=1 from Raspberry Pi OS but the system fails to boot, then a file, `pieeprom.upd`, will remain on the SD card, as it is only deleted on a successful boot. Consequently, the bootloader will set up signed boot mode after you swap SD cards back. You will need to delete the file from the `boot` folder to ensure that the system does not keep setting the SIGNED_BOOT flag.

Enable secure boot mode

WARNING

Once this stage has been completed, the device is locked to images with the specified key. If you lose the key and are therefore unable to sign images with it, then the device is 'bricked', and cannot be recovered.

After verifying that the signed boot configuration is working as expected, the hash of the public key can optionally be written to the one-time programmable (OTP) memory block to move the system to full secure boot.

The current process is to edit `usbboot/secure-boot-recovery/config.txt` and add one or both of the following entries:

- `program_pubkey` If 1, write the hash of the customer's public key to OTP. The system will now be in signed boot mode. This can be used for final testing, as the system can still be recovered to a non-secure state by loading an old bootloader that does not support secure boot.
- `revoke_devkey` If 1, revoke the ROM bootloader development key in order to require secure boot mode and prevent downgrades to bootloader versions that do not support secure boot.

Now run the update while still in the secure-boot-recovery folder:

```
../rpiboot -d .
```

`recovery.bin` validates that the public key in the EEPROM verifies the signature of the embedded `bootconf.txt` file and that there are no conflicts with existing OTP values. If everything is in order then the OTP bits containing the key are written. This is a one-time-only process and cannot be undone.

The Raspberry Pi system on a chip is then locked into signed boot mode forever, and will only accept bootable images and EEPROMs where the hash of the public key matches the hash in OTP:

- The bootloader will only load OS images signed with the customer's private key.
- The EEPROM configuration file must be signed with the customer's private key.
- It is not possible to install an old version of the bootloader that does not support secure boot.
- It is not possible to use a different private key to sign the OS images.

Using the Mass Storage Gadget on a secure boot system

If secure boot is enabled, the bootloader will refuse to load *any* images that are not signed. This includes the `usbboot` mass storage device (MSD) image, so in order to use it, the `usbboot` image will need to be signed.

This can easily be done using the tools provided, as follows:

```
cd secure-boot-msd
../tools/rpi-eeprom-digest -i boot.img -o boot.sig -k ../private.pem
```

Then use the standard `rpiboot` to run:

```
../rpiboot -d .
```

Note that the requirement to have a signed `usbboot` to get into MSD mode is protection against arbitrary access via a USB cable.

WARNING

You should keep any signed `usbboot` securely stored, as a third party could use it to recover data from an otherwise secure system.

Raspberry Pi