Using the One-Time Programmable Memory on Raspberry Pi Single-Board Computers

Raspberry Pi Ltd

2023-02-10: githash: c65fe9c-clean

Colophon

build-date: 2023-02-10
build-version: githash: c65fe9c-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	14 Jan 2022	Initial release
1.1	26 Oct 2022	Added section on device specific private key and how to use it

Scope of document

This document applies to the following Raspberry Pi Ltd products:

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

Introduction

All Raspberry Pi single-board computers (SBCs) have an inbuilt area of one-time programmable (OTP) memory, which is actually part of the main system on a chip (SoC). As its name implies, OTP memory can be written to (i.e. a binary 0 can be changed to a 1) only once. Once a bit has been changed to 1, it can never be returned to 0. One way of looking at the OTP is to consider each bit as a fuse. Programming involves deliberately *blowing* the fuse — an irreversible process as you cannot get inside the chip to replace it!

Some of this memory is used for various items of system data, as described later, but there are also eight rows of 32 bits (256 bits in total) available for the customer to program. This can be used for any purpose the customer requires: for example, product identification, storage of cryptographic keys, etc.

This white paper assumes that the Raspberry Pi is running the Raspberry Pi operating system (OS), and is fully up to date with the latest firmware and kernels.

Layout of the OTP

The OTP is laid out in rows; each row is 32 bits wide, and there are a total of 59 rows.

NOTE

For legacy reasons, the rows are labelled from 8 to 66.

There is a tool available from the Raspberry Pi OS command line to list the content of the rows in the OTP:

```
vcgencmd otp_dump
```

On a fresh Raspberry Pi with Raspberry Pi OS many of these rows are 0, but there are some values that are preprogrammed at the factory when the boards are manufactured.

Raspberry Pi usage of OTP

Only a few locations are currently used by Raspberry Pi devices themselves.

Row	Meaning
17	Bootmode register (Raspberry Pi 1-3) Bit 1: Sets the oscillator frequency to 19.2MHz Bit 3: Enables pull-ups on the SDIO pins Bit 19: Enables GPIO bootmode Bit 20: Sets the bank to check for GPIO bootmode Bit 21: Enables booting from SD card Bit 22: Sets the bank to boot from Bit 28: Enables USB device booting Bit 29: Enables USB host booting (Ethernet and mass storage)
18	Copy of bootmode register
28	Serial number
29	One's complement of serial number
30	Revision code; also contains bits to disable overvoltage, OTP programming, and OTP reading
33	Extended board revision, depending on board: • Compute Module 4 Bit 30: 0 if Wi-Fi® module is fitted, 1 if not fitted Bit 31: 0 if EMMC module is fitted, 1 if not fitted • Raspberry Pi 400 Bits 0-7: The default keyboard country code used by `piwiz` (Raspberry Pi OS startup wizard)
36-43	Customer OTP area
45	MPG2 decode key (Raspberry Pi Zero and 1-3 only)

Row	Meaning
46	WVC1 decode key (Raspberry Pi Zero and 1-3 only)
47-54	SHA256 of RSA public key for secure boot
55	Secure-boot flags (reserved for use by the bootloader)
56-63	256 bit device-specific private key
64/65	MAC address; if set, the system will use this in preference to the automatically generated address based on the serial number. This is factory-set on Raspberry Pi 4 and later.
66	Advanced boot register (Raspberry Pi 1-3 only). Bits 0-6: GPIO for ETH_CLK output pin Bit 7: Enables ETH_CLK output Bits 8-14: GPIO for LAN_RUN output pin Bit 15: Enables LAN_RUN output Bit 24: Extends USB hub timeout parameter Bit 25: ETH_CLK frequency: 0 for 25MHz, 1 for 24MHz

Customer OTP

The customer OTP area is defined as rows 36 to 43. No other areas in the OTP, except for the device-specific private key, can be changed by the end user.

WARNING

Changing customer OTP bits from 0 to 1 is irreversible. They are called one-time programmable for a reason!

Writing customer OTP values

Reading and writing customer OTP values require the `vcmailbox` command line application. This is installed by default on Raspberry Pi OS.

Because the `vcmailbox` system is a generic way of communicating with the graphics processing unit (GPU), it is not immediately obvious how it works when using it to write to the OTP.

The general form for communicating with the GPU using OTP commands is:

```
vcmailbox <command>  [8 + number * 4] [8 + number * 4] [start_num] [number] [value]
[value] [value] ...
```

where

- `command`: 0x00038021 to program, 0x00030021 to read back
- `start_num`: the first customer row to program from 0-7 (maps to rows 36 to 43 in the overall OTP address space)
- `number`: number of rows to program
- `value`: each row value to program

The row values are 32-bit values, so an entire row is programmed for each value. This means you cannot individually address a specific byte in the row. Row values can be hexadecimal (0x…), octal (0…), or decimal (just the number).

For example, to program OTP customer rows 4, 5, and 6 to 0x11111111, 0x22222222, 0x33333333 respectively, you would use:

```
vcmailbox 0x00038021 20 20 4 3 0x11111111 0x22222222 0x33333333
```

This is programming three rows, so our calculation for the third and fourth arguments of the command is 8 + (3 * 4) = 20. This actually represents the number of bytes of `value` data plus the 8 bytes for the `start_num` and `number` at 4 bytes each. Our command will then program OTP rows 40, 41, and 42.

NOTE

The `vcmailbox` command is actually of the form `vcmailbox <command> <data to send in bytes> <data to receive in bytes> [command-specific data]`, which goes some way to explaining need for the calculation above.

Reading back customer OTP values

Values can be read back as follows (this example assumes we have used the write example in the previous section):

```
$vcmailbox 0x00030021 20 20 4 3 0 0 0
0x0000002c 0x80000000 0x00030021 0x00000014 0x80000014 0x00000000 0x00000003
0x11111111 0x22222222 0x33333333
```

The last three 32-bit values returned should be those of the three rows beginning with customer row 4 — here, the values written to the OTP by the example command in the previous section.

Locking customer OTP

In some cases it may be necessary to lock the customer OTP to prevent any further changes being made. For example, for security reasons you may wish to ensure that a product ID is never changed.

Again, this is achieved using a mailbox call, but in this case with a very specific set of parameters that are used only for this purpose:

```
vcmailbox 0x00038021 8 8 0xffffffff 0xaffe0000
```

WARNING

This operation is irreversible.

Device-specific private key

Eight rows of OTP (256 bits) are available for use as a device-specific private key. This is intended to support file-system encryption, and should be used in combination with the signed boot system to ensure key safety.

These rows can be programmed and read using similar `vcmailbox` commands to those used for managing customer OTP rows. If secure boot / file system encryption is not required, then the device private key rows can be used to store general purpose information.

- The device private key rows can only be read via the `vcmailbox` command, which requires access to `/dev/vcio`, which in turn is restricted to the `video` group on Raspberry Pi OS.
- Raspberry Pi computers do not have a hardware-protected key store. It is recommended that this feature be used in conjunction with secure boot in order to restrict access to this data.
- Raspberry Pi OS does not support an encrypted root filesystem.

See cryptsetup for more information about open source disk encryption.

Key programming script `rpi-otp-private-key`

The rpi-otp-private-key script wraps the device private key `vcmailbox` APIs in order to make it easier to read/write a key in the same format as OpenSSL.

Read the key as a 64-byte hex number:

```
rpi-otp-private-key
```

Example output:

```
f8dbc7b0a4fcfb1d706e298ac9d0485c2226ce8df7f7596ac77337bd09fbe160
```

Write a 64-byte randomly generated number to the device private key:

WARNING

This operation cannot be undone.

```
rpi-otp-private-key -w $(openssl rand -hex 32)
```

vcmailbox support

The `rpi-otp-private-key` script uses the following vcmailbox calls in much the same way as described in the customer OTP section.

To read all of the rows:

```
vcmailbox 0x00030081 40 40 0 8 0 0 0 0 0 0 0 0
```

Example output:

```
0x00000040 0x80000000 0x00030081 0x00000028 0x80000028 0x00000000 0x00000008
0xf8dbc7b0 0xa4fcfb1d 0x706e298a 0xc9d0485c 0x2226ce8d 0xf7f7596a 0xc77337bd
0x09fbe160 0x00000000
```

Write an entire the row (replace the eight trailing zeros with the key data):

```
vcmailbox 0x00038081 40 40 0 8 0 0 0 0 0 0 0 0
```

So to write the key shown in the previous example:

```
vcmailbox 0x00038081 40 40 0 8 0xf8dbc7b0 0xa4fcfb1d 0x706e298a 0xc9d0485c
0x2226ce8d 0xf7f7596a 0xc77337bd 0x09fbe160
```

Using the private key

It is envisaged that the private key will be loaded by an init-script in an `initramfs`, which in turn is used to mount a `LUKS` encrypted file system.

For this to be secure, and prevent reading of the key by a third-party software installation, it is important that the system uses the secure/signed boot procedure that is available on Raspberry Pi 4 and CM4. This ensures there is a chain of trust up to and including the `initramfs`, which ensures that only code signed to run on the system has access to the OTP. Past this stage, any further security is up to the customer — for example, ensuring SSH or similar remote access protocols are sufficiently protected.

Creating a LUKS file system

The following example set of `bash` commands will generate a random key, and create a LUKS encrypted file system based on that key using `cryptsetup`.

```
# Generate 256 bit random number for key and write it to OTP. Raspberry Pi specific
rpi-otp-private-key -w $(openssl rand -hex 32)

# Read the key. Raspberry Pi specific
rpi-otp-private-key -b > key.bin

# Commands past this point are standard cryptsetup

# Create an encrypted FS e.g. on partition 3 of an SD card
BLK_DEV=/dev/mmcblk0p3
sudo apt install cryptsetup
sudo cryptsetup luksFormat --key-file=key.bin --key-size=256 --type=luks2 ${
BLK_DEV}
```

```
# For debug
sudo cryptsetup luksDump ${BLK_DEV}

# Map the encrypted block device and create a file-system
sudo cryptsetup luksOpen  ${BLK_DEV}  encrypted-disk --key-file=./key.bin
sudo mkfs /dev/mapper/encrypted-disk
mkdir -p crypto-fs
sudo mount /dev/mapper/encrypted-disk crypto-fs
sudo sh -c 'echo "secret sauce" > crypto-fs/recipes.txt'
```

Other possible uses of the customer OTP

We have briefly mentioned product ID and cryptographic keys as examples of things you can store in the OTP. There are many other possible uses for the OTP, including:

- Company name
  You can read this back for a very basic security check.
- Custom MAC address
  This will require startup scripts (see our white paper on setting MAC addresses).
- Custom serial number
  Rather than use the inbuilt serial number, you may want your product to use your own serial number sequencing and format.

Raspberry Pi