Application Note

DA1470x Secure Boot

AN-B-093

Abstract

The DA1470x family of devices implements support for securely booting the device. This is needed for systems that run only authentic and tamper-free firmware.
This application note describes the need for secure boot feature, adds details on the procedure used by the DA1470x bootloader, and finally introduces the tools which help customers using this device feature.

Contents

Figures

Tables

1 Terms and Definitions

OTF	On-the-fly decryption
SDK	Software development kit
NVM	Nonvolatile memory
OTP	One-time programmable nonvolatile memory
ECC	Elliptic curve cryptography
ECDSA	Elliptic curve based digital signature algorithm
ROM	Read-only memory
CS	Configuration script
UART	Universal asynchronous receiver transmitter
POR	Power on reset
CPU	Central processing unit
DMA	Direct memory access
OQSPI	Octo quad serial peripheral interface
SUOTA	Software update over the air

2 References

[1] DA1470x, Datasheet, Renesas Electronics

[2] DA1470x Development Kit Schematics (mother/daughterboard), Renesas Electronics

[3] Secure Bootloader Tutorial for 69x Family of Devices, http://lpccs-docs.renesas.com/da1469x_secure_boot/running_secure_image.html,

[4] SW Update Procedure, http://lpccs-docs.renesas.com/um-b-092-da1469x_software_platform_reference/User_guides/User_guides.html#software-upgrade-over-the-air-suota

Note: References are for the latest published version, unless otherwise indicated.

3 Introduction

Security attacks on IoT devices are becoming usual because of the missing security standards for IoT devices, and the decreasing cost of attack hardware, and the availability of descriptions/guidelines on how to attack IoT devices.

Another reason is the tendency of IoT devices to be always connected to the web using technologies such as, for example, Wi-Fi or Bluetooth®. With this connectivity in place, these devices are accessible from all over the world without the need for the attacker to be next to the device and thus become an easy target for attackers. In comparison to the traditional IT for devices such as PC, laptops, or phones where firewalls, antivirus software, or content backup became standard, the market of IoT devices does typically not offer these sorts of protection mechanisms. This happens due to a missing global and binding security standard for IoT devices and the resource constraints. With more complex IoT chips and dedicated security hardware, IoT devices are starting to catch up with traditional IT infrastructure.

Also, with recent developments in the legislative area, several standards have been set to protect the IoT device itself and the assets stored inside the IoT device. One of the features that are consistently required in such security standards is a secure device initialization and a secure device update – commonly covered by a secure bootloading process.

This application note describes the concept of secure bootloading and shares details on how this is achieved on the DA1470x family of devices. Renesas provides tools as part of the SDK accompanying the DA1470x family of devices. The tools simplify the usage and configuration of the secure boot feature. See further details on how to use the Renesas tools for generating secure firmware image files in Ref. [3].

4 Secure Boot

4.1 General Concept

For IoT systems, where secure assets need protection, using a secure bootloading process is essential and a hard requirement by several security standards. Secure assets in this context could be, for example, credentials to securely sign in to web services, user data such as purchased music content, private health data, or proprietary algorithms like a custom DSP algorithm. Protecting these assets makes sure this data or IP stays safe and cannot be used by unauthorized persons. To ensure the stored secure assets cannot be compromised, a secure boot and update process is needed. This process addresses the concerns listed in Table 1.

Table 1: Secure Bootloader Targets

#	Concern	Notes
1	Image authentication	Makes sure the firmware image to be executed by the device is provided by the authorized source.
2	Image integrity	Makes sure the firmware image to be executed is complete and not tampered with by a malicious actor.
3	Image confidentiality	Allows ensuring that the firmware image and the data potentially embedded in it stays confidential. Note that this is an optional feature that may come on top of #1 and #2.

With secure booting enabled, the firmware to be executed is checked to ensure the firmware stems is from the correct source and unchanged, and ensure the intended functionality is given. This is accomplished by adding some additional information to the firmware image. This additional information is generated using data only accessible by the authorized source and verified by the bootloader at device start-up. The most common approach to ensure secure booting is asymmetric cryptography as in Figure 1.

4.2 DA1470x Booter Concept

Figure 1 shows the various phases which need to be addressed to properly implement a secure bootloader. This drawing represents the Renesas approach embedded in the secure bootloader of the DA1470x family of devices but is also representative of a typical secure bootloading approach commonly used.

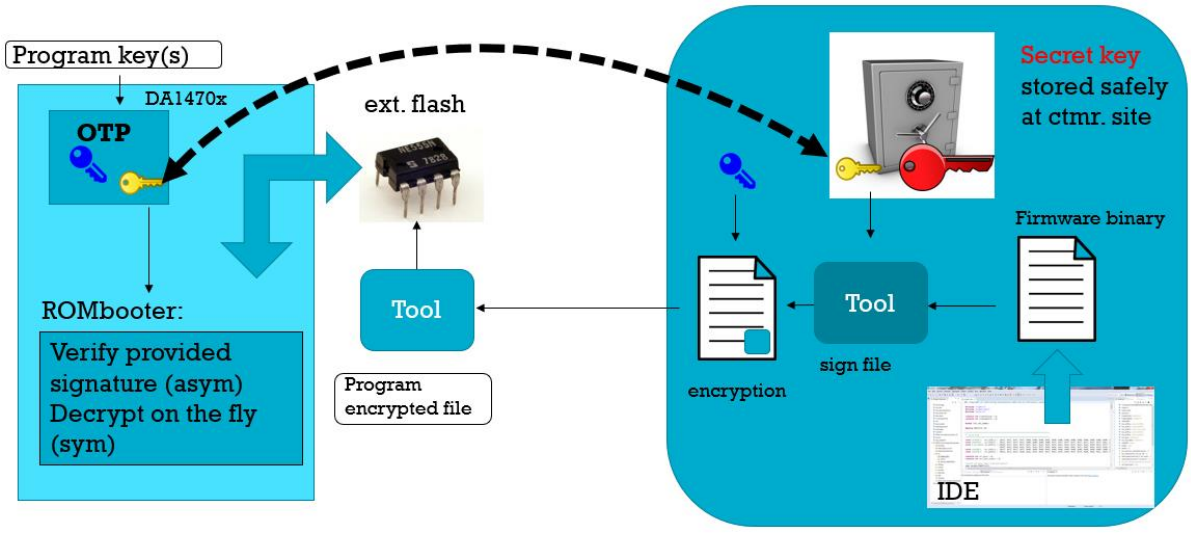

Figure 1: Asymmetric Crypto for Bootloading

The principle of operation goes back to the basics of asymmetric cryptography. Asymmetric cryptography solves the problem of sharing the keys by using two different but related keys. Good examples of this approach are cryptography based on elliptic curves (ECC) and cryptography based on RSA. In both cases two keys exist that are related to each other based on the underlying mathematics (ECC or RSA):

- The private key – is kept in a safe place by the owner of the firmware and used to *generate* a digital signature. This key should never be exposed and handed out to anyone. Leaking this key would allow an attacker to sign malicious firmware using this key
- The public key – can be freely shared and can be used to *verify* a digital signature. Given the mathematics behind this asymmetric cryptography, there is no way to derive the private key from the public key other than using a brute force attack by trying every possible key. Given the size of the keys, this will take an extremely long time thus rendering any such attempt unpromising

The actual firmware image is appended with an additional digital signature. With this digital signature as part of the firmware image, the secure bootloader can verify the authenticity and integrity of the entire binary including the firmware. For the generation of digital signatures, several approaches exist. A typical approach is using the ECDSA algorithm that generates a digital signature based on:

- A predefined elliptic curve including all needed parameters
- A private key
- A hash of the firmware
- A random number

To generate a hash function, for example, SHA-2 is used. This function generates a digest of the firmware image and makes sure that any tampering attempt is detected. The firmware image, including the digital signature, needs to get programmed to the memory where the device executes, this is XiP from an externally attached NOR flash. Besides the firmware and the signature, it is also the public key – generated from the private key used for signature generation – which needs to be stored. To make sure this public key cannot be tampered with or replaced, it is saved in a trusted memory. Trusted memory permits to read/write protect all or parts of it.

When the device starts operation (for example, after power-on), the secure bootloader starts up. For this, several approaches exist, the secure bootloader could be part of the ROM – if existing – or can be stored in internal/external NVM assuming it can be write-protected. Write-protecting the memory area where the secure bootloader was stored is essential as it prevents manipulation. When the attacker can modify the secure bootloading procedure or even bypass it, the secure assets which are protected by the secure bootloader will be compromised.

After starting up the secure bootloader initializes the device including the interface and the external memory – in case this is where code gets executed from – and verifies the digital signature by generating a hash over the complete firmware and by using the pre-programmed public key. If the digital signature was successfully verified it allows to start execution of the pre-programmed firmware (Figure 1). This firmware could be the final application, or it could be yet another bootloader that itself loads the final application using a similar mechanism as the primary bootloader does.

To address the confidentiality concern, the application firmware could optionally be encrypted before it gets programmed to the place where it gets executed from. This additional step is not always needed, and thus not considered part of the secure bootloading approach. If, for example, the firmware runs from internal NVM there is typically no need to store firmware in an encrypted format as access to this can easily be locked. For a separate external NVM device, this is a different story. In comparison to asymmetric cryptography where two different keys are used, the crypto method deployed for firmware encryption or decryption is typically based on symmetric cryptography. This is because symmetric crypto algorithms are faster and can work with minimum latency. Symmetric crypto algorithms use the same key for both encryption and decryption. A typical representative of such a symmetric crypto algorithm is AES. In case encryption is needed the firmware that needs to run on the target device needs to get encrypted before programming it to the target NVM. Programming happens during the development phase by the tools provided by Renesas (see Ref. [3]). The very same key used for encrypting needs to be programmed into the device. Protection

of this symmetric key is needed on the target device to make sure an attacker cannot steal it. The secure bootloader handles this encryption key and makes sure that the key is used for any read-access from the external NVM. To avoid additional latency, a hardware engine makes sure that fetching code from external NVM into an internal cache for execution automatically decrypts the fetched content before adding it to the cache for CPU execution.

4.3 Secure Bootloader Implementation

Renesas secure bootloader implementation follows almost completely the approach described in Section 4.1 and Section 4.2. This section gives some more DA1470x-specific details and provides links for further information. Table 2 shows some basic properties of the secure bootloader as deployed on the DA1470x family of devices.

Table 2: DA1470x Secure Bootloader Properties

#	Feature	Notes	Reference
1	Bootloader configuration	Bootloader configuration under user control, security can be enabled, disabled, or configured through entries programmed to the configuration script	4.3.1
2	Location of bootloader	ROM	
3	On-the-fly decryption	Uses AES256 HW accelerator running in CTR mode	4.3.6
4	Digital signature algorithm/hash	EdDSA using Edwards curves (Ed25519) using SHA 512 for hash generation. The crypto library is stored in ROM and executed by the M33 core.	
5	Additional features	Key revocation, rollback prevention	4.3.8, 4.3.9
6	Number of keys	8 public keys for signature verification 8 on-the-fly AES keys for on-the-fly decryption 8 AES for user data encryption/decryption	4.3.4
7	Protection features	Allows W or R/W protection of several OTP area	4.3.5

The following sections give more details on the exact operation of the secure bootloader.

4.3.1 Bootloader Configuration

The bootloader used on the DA1470x family of devices is stored as part of the ROM. After power-up and upon the occurrence of a power-on reset (POR) the bootloader starts operation by executing XiP directly from the ROM. The bootloader is executed by the M33 core and can be configured by entries in the configuration script (CS) which is stored in OTP. Adding entries is under customer control and supported by the tools provided by Renesas. Users can add entries to the CS to:

- Enable secure booting
- Indicate the location of the product header
- Allow disabling development mode
- Specify XTAL settling time
- Write {register, value} pairs, for example, to set sticky bits

Due to the nature of OTP, values written to it cannot be erased anda dedicated protection schema allows protecting the CS entries from being manipulated.

4.3.2 Secure Booting Paths

As indicated in the previous section, secure booting can be enabled by adding an entry into the CS area located in OTP. When enabled, the booting options are restricted as shown in Table 3.

Table 3: Supported Booting Paths

Booting Path	Security Disable	Security Enabled
Cached XiP from external OQSPI flash with CS in OTP	Supported	Supported
Boot from UART	Supported	

To enable secure booting, add a CS entry that sets the SECURE_BOOT bit inside the SECURE_BOOT_REG to 1. By default, this bit is set to 0. Figure 2 shows an example from the DA1470x datasheet.

0	R/W	SECURE_BOOT	Follows the respective OTP flag value. Its value is updated by the BootROM code. 1: system is a secure system supporting secure boot 0: system is not supporting secure boot	0x0

Figure 2: SECURE_BOOT

The SECURE_BOOT bit is sticky, when set to 1 it can only be reset to 0 by a POR. With that, any later write access to reset the secure bit will not be accepted by the hardware. When the booter detects that this bit is requested to be set, it knows that secure booting needs to get enabled and modifies the booter program flow to include the needed steps. What is added to the boot flow in such a case is described in Section 4.3.4.

Booting via UART is considered a development feature that allows tools to operate by temporarily downloading executable code to internal RAM. If this was possible in secure mode an attacker could download their own code to RAM and potentially extract device secrets. For this reason, the UART boot feature is not supported when secure boot is enabled. Wwhen the secure boot feature is enabled, firmware updates are only possible over the air using SUOTA (see Section 4.3.10).

NOTE
When the SECURE_BOOT bit had been detected by the ROM based booter, it expects a firmware image including a valid signature, everything else will be rejected.

4.3.3 Booter Flow

As described in the DA1470x datasheet in Section 5.6, the booter works in several phases some of which are omitted in case no secure booting is requested. If more data is needed, see Ref. [1] that contains details on the overall booting flow. Table 4 shows which booter phases are applied depending on the security mode.

Table 4: Booting Stages

Booting Stage	Security Enabled	Security Disable	Notes
Initialization	Yes	Yes	Generic booter preparation

Booting Stage	Security Enabled	Security Disable	Notes
Run config script	Yes	Yes	Locate and apply {register, value} pairs stored in CS, and wait for XTAL to settle if needed
Retrieve application code	Partly	Yes	Locate product header and handle UART booting
Device administration	Yes	Partly	Image update handling, firmware validation, handling of key revocation and rollback prevention
Load image	Yes	Yes	Set QSPI HW including setup of the OTF key and configure cache, copy IVT, and remap to start execution in cached XiP mode

After start-up, the booter initializes the device to work as expected. This includes enabling of clocks and power domains to access peripherals such as, for example, the internal OTP or the UART interface. Then the bootloader locates the CS area and applies the prestored {register, value} pairs to load the device with the trim and calibration values provided by Renesas and with the application-specific values provided by the customer. The CS section contains information about the boot flow and consequently, the bootloader might handle UART booting or try to load the product header to identify the location of the active image and possibly that of a pending update. During device administration, the bootloader reads the image header as well as verifies the digital signature, and handles key revocation or minimum FW update requests if needed.

After the successful operation, it finally makes sure the correct image gets executed. Execution happens by remapping the external OQSPI such that M33 can XiP directly from this memory using the cache as an intermediate buffer. The application starts execution from its reset handler and the IVT including the reset handler is copied to internal RAM to speed up interrupt processing.

4.3.4 OTP Storage and Layout

The OTP on the DA1470x family of devices offers 4 kB of programmable space and is used for various purposes, some of them related to secure booting. Table 5 shows the OTP layout.

Table 5: OTP Layout

Bytes	Words	Description	OTP Address
1024	256	**Configuration Script** ~100 registers write operations	0x00000C00
256	64	**OQSPI FW Decryption Keys Area – Payload** Write/read protected when secure mode enabled in CS Secure mode connects those (8 * 256-bits) keys to OQSPI Controller	0x00000B00
256	64	**User Data Encryption Keys – Payload** Write/Read protected when secure mode is enabled in CS Secure mode connects those (8 * 256-bits) keys to the AES engine	0x00000A00
32	8	**OQSPI FW Decryption Keys Area – Index** **Eight entries for eight 256-bit keys**	0x000009E0
32	8	**User Data Encryption Keys – Index** **Eight entries for eight 256-bit keys**	0x000009C0
256	64	**Signature Keys Area – Payload**	0x000008C0
32	8	**Signature Keys Area – Index**	0x000008A0
2208	552	**Customer Application Area**	0x00000000

Bytes	Words	Description	OTP Address
		Secondary bootloader, Scratchpad, binaries, and so on	

The lowest 2 kB of OTP space is reserved for customer-specific data such as, for example, a secondary bootloader. It starts at address 0 of the OTP and can be remapped to address 0 of the M33 in case the secondary bootloader needs to start after the primary ROM booter is finished.

After this customer-specific section, there are three areas where different kinds of keys can be stored. Table 6 summarizes what key areas are available and what these keys are typically used for.

Table 6: OTP Key Area

Area	Key Type	Notes
Signature key area	Asymmetric (ECC)	For digital signature verification as needed by the bootloader. Can store any other public key needed, for example, for application purposes.
User data encryption keys	Symmetric (AES)	For user AES encryption and decryption, free to use by the application
OQSPI FW on-the-fly decryption key area	Symmetric (AES256)	For on-the-fly AES256 decryption, used by the bootloader

Each one of these areas consists of two sections:

- one where the keys are stored
- another one that provides information to the SDK on whether keys were revoked

The area where the actual keys are stored provides eight slots per key type, each one with the capability to store keys up to a width of 256 bits. The size of the actual key must follow some constraints that are listed in Table 7.

The user has full control over how the key slots are filled. For example, within the signature key area, it would be possible to use two signature key slots for keys that only allow key revocation and the remaining six slots for keys that allow verifying the signature. There is no support for adding keys during the lifetime of the product. Unused key slots and their index tables should be overprogrammed with all 0 to make sure these slots cannot be misused.

Table 7: Key Width Constraints

Area	Notes
Signature key area	Keys used for booting need to be 256-bit wide and based on Edwards 25519 curve. Application-specific ECC keys can be stored according to the application needs
User data encryption keys	Key sizes depend on application needs. AES HW supports 128-, 192-, and 256-bit wide AES keys
OQSPI FW decryption key area	Only AES keys with a width of 256 bits are supported

The area where information about the revocation status per key is stored is 96 bytes in total – 32 bytes available per key area. From the 32 bytes, 4 bytes are reserved per key slot, if the complete 4 bytes are left erased (all 1) the key is valid, if the 4 bytes are programmed to all 0 the key is marked as invalid and cannot be used for booting or any other purposes. Programming these 4 bytes is under booter control and will be explained in Section 4.3.8.

The last area present in the OTP is the configuration script area which is used to store {register, value} pairs that are applied by the booter after power on. This area contains trim settings derived by

Renesas during manufacturing and can be extended by the customer by programming application-specific settings such as, for example, enabling secure boot as discussed before. Adding keys and entries to the OTP are supported by the tools which support the DA1470x family of devices.

4.3.5 Protection Features

Protection of secure assets happens partly by having a secure bootloader in place – with that only authentic software can run and access data as expected. Still, in case the CPU is running malicious code and with that access, secure assets in the system security could be compromised. This could happen, for example, when a remote attack on the protocol stack exploits vulnerabilities such as, for example, buffer overflows. To prevent such scenarios DA1470x family of devices introduces access protection capabilities for individual OTP areas. With that in place, the M33 or any other core cannot access the content of the OTP directly. Figure 3 shows what OTP areas can be protected (data from the DA1470x datasheet).

Bit field	Description
PROT_OQSPIF_KEY_READ	This bit will permanently disable CPU read capability at OTP offset 0x00000B00 and for the complete segment (see Table 83)
PROT_OQSPIF_KEY_WRITE	This bit will permanently disable ANY write capability at OTP offset 0x00000B00 and for the complete segment (see Table 83)
PROT_AES_KEY_READ	This bit will permanently disable CPU read capability at OTP offset 0x00000A00 and for the complete segment (see Table 83). The AES sections are only used by the application SW, but protecting the key area from read/write makes it secure after leaving the manufacturing facilities
PROT_AES_KEY_WRITE	This bit will permanently disable ANY write capability at OTP offset 0x00000A00 and for the complete segment. The AES sections are only used by the application SW, but protecting the key area from read/write makes it secure after leaving the manufacturing facilities
PROT_SIG_KEY_WRITE	This bit will permanently disable ANY write capability at OTP offset 0x000008C0 and for the complete segment (see Table 83). This is for protecting public keys from being written (used by ECC only)
PROT_CS_WRITE	This bit will permanently disable ANY write capability at OTP offset 0x00000C00 and for the complete segment of the CS (see Table 83)

Figure 3: OTP Protection

Using these protection bits, it is possible to write protect the area where configuration script with all the individual {register, value} pairs are stored. Write protection is needed as enabling security mechanism and protection of individual OTP areas is stored in this section. In case an attacker gets access to the CS area and would be able to overwrite the corresponding entries the secure booting might get bypassed. Read protection cannot be implemented for this OTP area because the booter as well as the SDK need to be able to read the data from this OTP area and apply it when needed.

For the two OTP areas where symmetric keys are stored, the user can disable both read or write access by setting an individual bit per region and access type. Disabling access types per OTP area can be requested through a {register, value} pair applied by the booter. The corresponding bits are tricky, when set they can never be unset until the next power-on reset. This is to make sure no CPU access can read or overwrite the key.

For the OTP area where asymmetric public keys are stored, the corresponding OTP area can be write-protected. That is because in DA1470x ECC operations are handled by the CPU which needs to access the public part of the ECC key to do the proper calculations. For the symmetric keys, however, this is a different story. For those two, hardware accelerators exist with a write-only key register. This makes sure that when the key had been programmed to this AES key register it cannot be read back by the CPU.

When one of the access protection bits had been set together with the request for secure booting, one of the channels of the generic DMA engine gets converted to a secure DMA channel and can only be used to transfer keys from OTP to the corresponding key register. The user can still configure where within the symmetric OTP key region the key should be copied from. The destination address is always the AES key register in either the generic AES engine or the AES engine which is part of the OQSPI interface. Figure 4 shows this approach and with that stays protected.

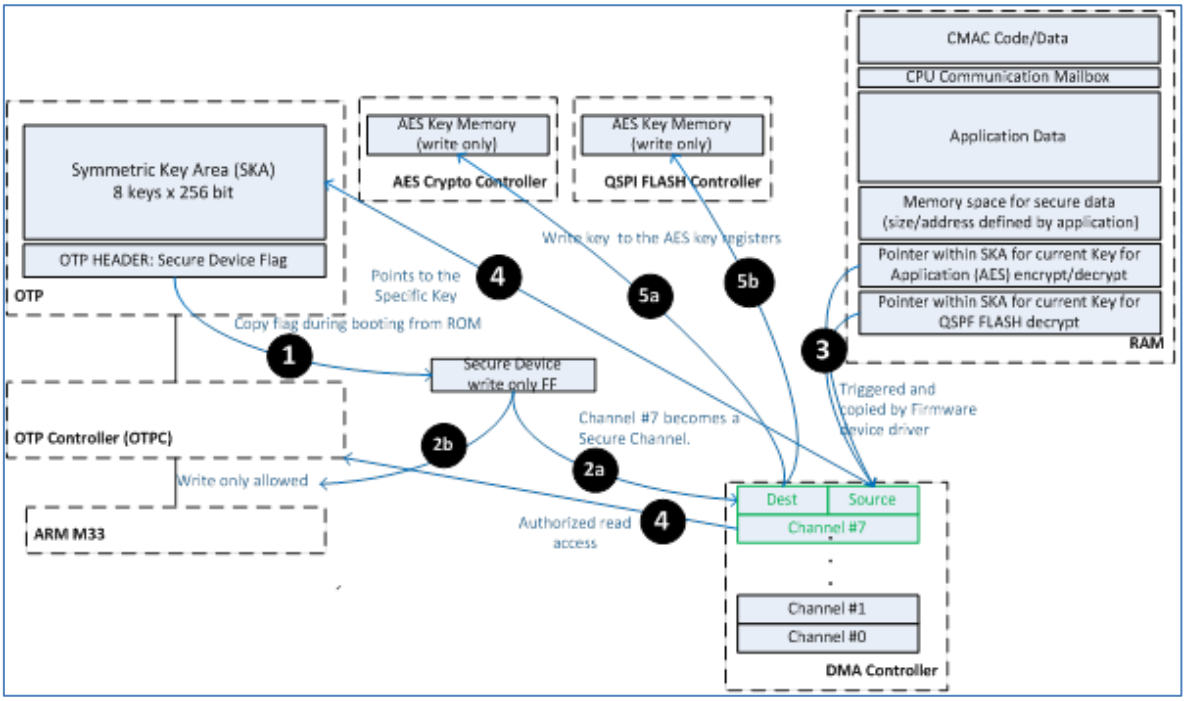

Figure 4: Secure DMA Engine

In Figure 4, the secure engine starts with the request for secure booting (step 1). This is detected by the booter when the corresponding {register, value} pair is found in the configuration script OTP area. When detected the corresponding sticky bit register gets set (SECURE_BOOT in SECURE_BOOT_REG as described in Section 4.3.2) and by that one of the DMA channels gets converted to a secure DMA channel. The booter detects the correct location of the AES key indicated in the image header by an index (step 3 and further described in Section 4.3.7) and programs the DMA channel source accordingly (step 4). The destination of this key transfer is either an AES hardware engine for on-the-fly decryption or a generic AES engine for any kind of AES-based symmetric encryption needs (step 5a/5b). When the transfer is initiated, the key is transferred secretly by the secure DMA channel from the requested OTP address to the destination key register. The transfer is always 256-bit even if the key itself is smaller.

NOTE
This DMA mechanism is not only used by the booter but also by the application in case user data needs to get encrypted and decrypted using one of the prestored AES user keys.

The sticky bit approach had been extended to also protect the individual debug interfaces. Separate protection for M33, M0, and CMAC are existing. When such a protection bit had been set it is no longer possible to start using the debug interface to communicate with the related embedded core. This feature is under user control and should be applied if no debugging capability on a secure product is needed.

4.3.6 On-the-Fly Decryption Unit

Another feature related to the secure bootloading process is on-the-fly decryption (mentioned in Table 2). Strictly speaking, it is not part of the secure bootloading procedure as it is not related to digital signature verification. On-the-fly decryption is rather a feature that makes sure the data stored in the external flash stays confidential when it is read by the M33 core. This feature allows firmware protection or any other data against reverse engineering even if an attacker has access to the physical interface between DA1470x and external nonvolatile memory. The DA1470x family of devices has added hardware support for this feature to the OQSPI interface. The area protected by on-the-fly decryption can be set by user software and can include code and data. The is no additional latency if this feature had been enabled.

Before this feature can be used by the DA1470x device the NVM data needs to be encrypted using the same key as the one stored in the OTP and used by the on-the-fly decryption unit. Renesas provides tools that help encrypt user-defined content. On-the-fly decryption uses AES256 running in CTR mode, details on the key transfer mechanism are shared in Section 4.3.5. Figure 5 shows how the OQSPI controller had been extended to add this functionality.

NOTE
For the secure bootloader available at the DA1470x family of devices on-the-fly decryption is always enabled as soon as the secure boot feature had been enabled.

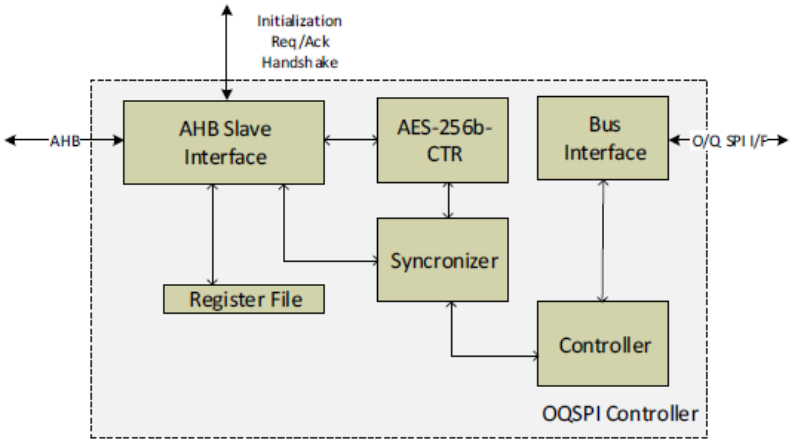

Figure 5 On-the-Fly Decryption

4.3.7 NVM Image Layout

The booter expects the external NVM to follow a predefined structure. The main members of this structure are shown in Figure 6.

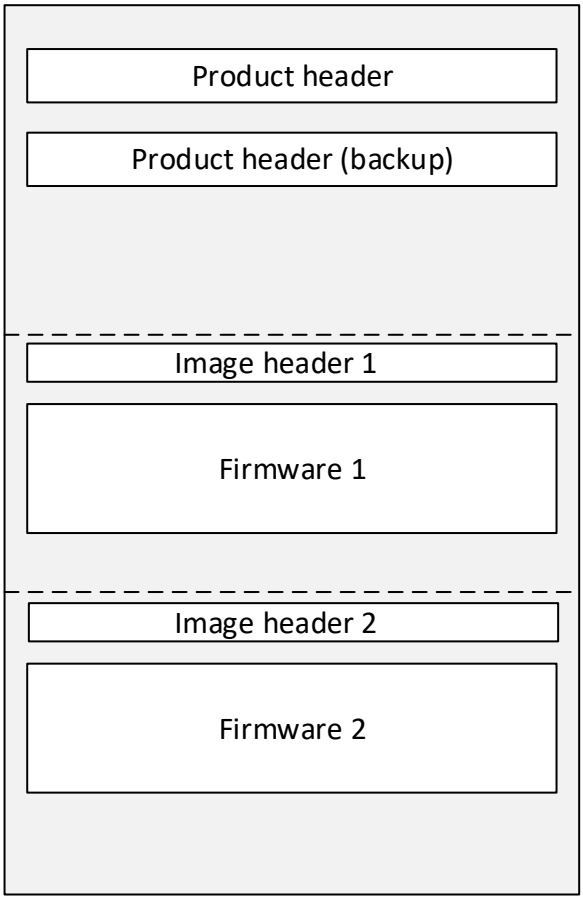

Figure 6: NVM Layout

The NVM layout is very similar for cases when the device boots securely and cases when the device does not boot securely. Initially, we have a section called “product header” which is essential for the system to boot. If it gets corrupted the system might get stuck, for this reason, an identical copy exists. The product header manages firmware updates (see additional information in Section 4.3.10) and also contains a set of settings that are unique for the externally attached NVM device and the OQSPI controller interfacing with it. This information gets extracted by the bootloader and applied to both the OQSPI interface and the OQSPI memory to be able to access stored data. This product header is protected by a CRC to make sure corrupted product headers generate an error.

Following the product header, the NVM layout contains one or more firmware images. A firmware image consists of the image header and the firmware binary itself. The image header looks as shown in Figure 7, blue fields are mandatory, and yellow ones are optional.

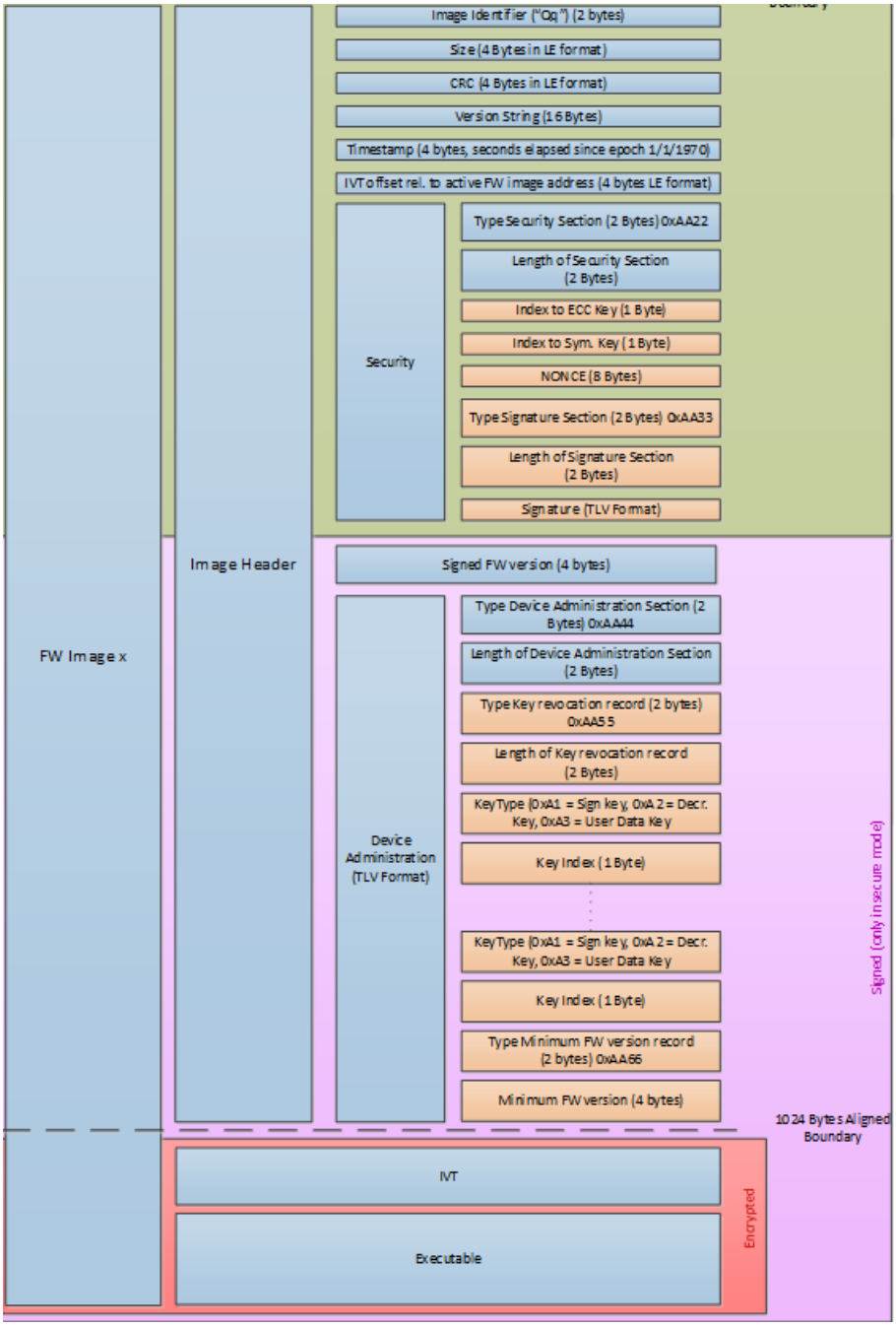

Figure 7: DA1470x Firmware Image Including Image Header

The image header contains information about the header version, firmware size, timestamp, version string, and the start of IVT and is protected by a CRC. The firmware itself is the result of the built process and can optionally get encrypted. In case the firmware needs encryption and support for secure booting, some additional data needs to be included. More details are shown in Table 8.

Table 8: Security-Related Image Header Fields

Field	Included in Signature	Notes
Security section ID	No	ID to indicate valid security section
Security section length	No	Length of the security section

Field	Included in Signature	Notes
Index to signature key	No	Index to ECC key used for digital signature
Index to OTF key	No	Index to OTF key used for firmware decryption
NONCE	No	Nonce for OTF init
Signature section ID	No	ID to indicate a valid signature section
Signature section length	No	Length of the signature section
Digital signature	No	Digital signature of the firmware binary
Firmware version	Yes	Firmware version of the signed image
Administration section ID	Yes	ID to indicate valid admin section
Administration section length	Yes	Length of the admin section
Revocation record	Yes	ID to indicate valid revocation record
Revocation length	Yes	Length of revocation record
Key type	Yes	Type of the key to be revoked
Key index	Yes	Index of the key to be revoked
Min FW version record	Yes	ID to indicate new minimum FW version
Min FW version	Yes	New minimum FW version

Renesas provides tools that allow to:

- Generate public/private key pairs
- Generate symmetric keys
- Construct the image header based on provided image and additional security-related data including the generation of a digital signature
- Encrypt the image by selecting the proper symmetric key
- Program keys and firmware image to the NVM attached to the target device
- Revoke a key
- Update the minimum supported FW version

The tools are provided as part of the SDK delivered for the DA1470x family of devices and include python scripts as well as the mkimage executable.

NOTE
The image header and firmware image need to be stored aligned to some address boundaries inside the external NVM. This is related to the operation of the cache and is taken care of by the tools.

4.3.8 Key Revocation

Key revocation is a feature that allows revoking a key that had been compromised. In such a case the key is not considered trusted and it should be removed from the list of supported keys stored inside the OTP area (see Figure 8). The booter supports the revocation of all keys except the one which was used to sign the current digital signature and decrypt the current firmware image.

Key revocation can happen if an admin field is generated and added to the image header (details on the format in Section 4.3.7). The key revocation is done with the next firmware image update (see Section 4.3.10), it cannot work without a valid firmware binary. When the booter detects a valid admin section it checks the length, key type, and index. If all provided parameters are valid the indicated key(s) is(are) marked as revoked by programming the corresponding 32-bit word – defined by the key index – to 0. For that, the additional index section inside the OTP is used. Once marked as revoked the booter will ignore any attempt to use this key for signature verification or OTF

decryption. With that, any firmware image signed and/or encrypted with a revoked key will be rejected. Figure 8 shows this approach.

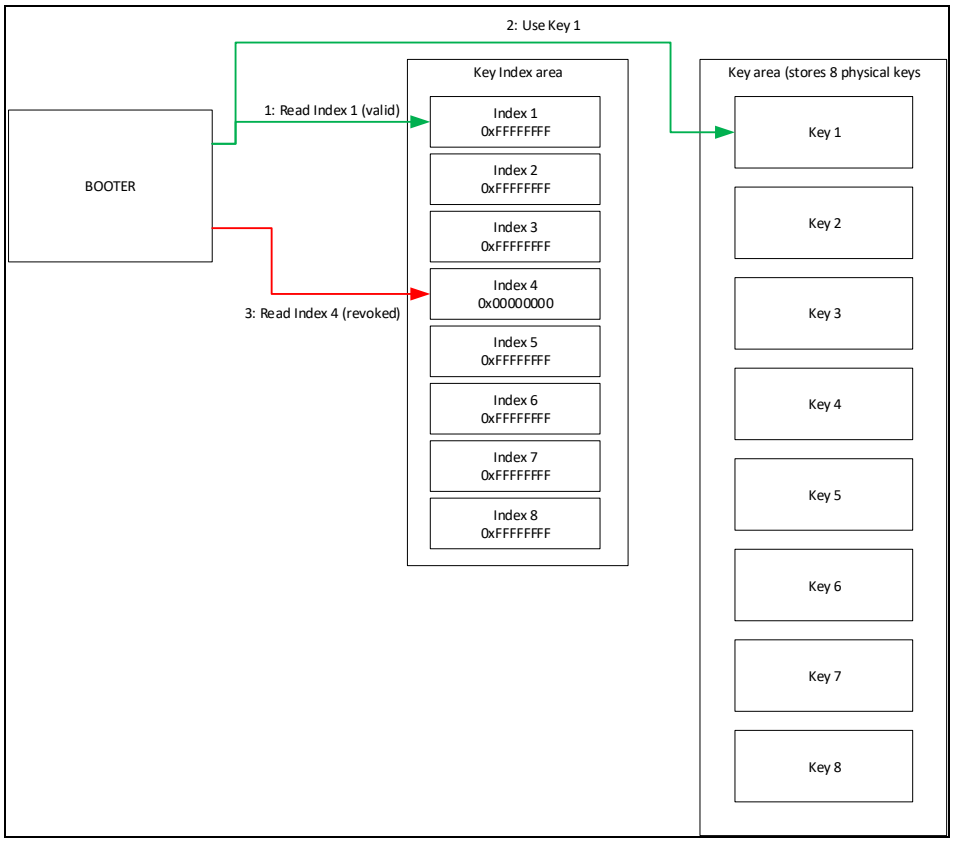

Figure 8: Key Revocation

Figure 8 shows that attempts to use key slot 4 are ignored as the key had been marked as revoked. Key slot 1, however, has not been marked as revoked and the booter is thus able to retrieve the key to proceed with the requested operation.

NOTE
The crypto adapter as provided inside the SDK looks at the index table for the user encryption key before proceeding with the requested symmetric crypto operation.

4.3.9 Rollback Prevention

Rollback prevention makes sure that firmware versions that are marked as outdated and are no longer accepted by the booter. An outdated firmware version could be one that has proven to be vulnerable to some security attacks.

The rollback prevention feature is available only for cases where security had been enabled and thus the security section is part of the image header. This feature is not enabled by default, it gets enabled when a customer defines a minimum firmware version with which the signed FW version of the next booted firmware gets compared. The signed FW version is encoded in 32 bits, included in the image header and covered by the digital signature, with that making any tampering attempts impossible (see Figure 7 for details). If the signed FW version of the image to be booted is smaller than the specified minimum firmware version booting of this firmware version gets rejected. The comparison is handled as part of the FW validation process.

Setting the minimum FW version could happen during initial mass programming or later in the field through a firmware update. To update it the new minimum firmware version needs to be included in

the administration field of the image header (as shown in Figure 7). The new minimum firmware version is protected by the digital signature. The booter parses this field and when it detects this type of record it compares the new minimum firmware version as found in the image header for the firmware image to be booted with the previous minimum firmware version stored in OTP and only accepts it if it is bigger than the previously stored minimum firmware version. If all tests were successful, the new minimum firmware version is stored at the end of the CS in the OTP. If the CS section in the OTP was write-protected using a sticky bit setting the sticky bit is deferred until the OTP writing is completed. In case multiple minimum FW versions are stored they will all be programmed to the CS – one after the other. In case the CS area in the OTP becomes full, the update won't happen and an error indication is passed to the application which can then take the appropriate actions.

At the stage when the booter compares the signed FW version of the image to be booted with the minimum firmware version as stored in OTP the booter parses through the entire list of stored minimum FW entries inside the CS area and picks the last one. Due to sanity checks when upgrading the minimum firmware version the last entry should also correspond to the maximum firmware version.

4.3.10 Firmware Update

Firmware update is another important functionality that ensures that firmware with fixed bugs or firmware with enhanced functionality (or both combined) can be securely updated. If needed the minimum supported firmware version can be updated at the very same time (see Section 4.3.9).

The SDK provided for the DA1470x family of devices supports one method for firmware updates, this can be extended by the user application if needed. Note that firmware updates in the context of this section are considered updates while the device is fully secured and provided to the field. Firmware updates during software development can be done using the debug interface or using a Renesas provided loader which programs external NVM using a UART connection with the development environment.

The SDK provided firmware update method lets the update service which is part of the provided SDK handle the actual firmware update procedure. The essential steps of this update process are described in this section, but further details can be found in a user guide that can be found following the link provided in Ref. [4].

The FW update method supported by the SDK version supporting the DA1470x family of devices uses Bluetooth® LE functionality for any FW update. Renesas had developed a dedicated Bluetooth® LE service, called SUOTA, that is in charge to handle firmware updates. The SUOTA server is built into the application firmware and the SUOTA client runs on a device interfacing with the user. The client can run on mobile OS as well as on a Windows PC environment and initiates the FW update procedure. When a Bluetooth® LE connection is established between the client and server a defined SUOTA protocol makes sure the relevant data is properly transferred. The Bluetooth® LE connection is expected to use a secured transport layer by using a secure connection. In case a secure image is transferred, the data that is exchanged using the SUOTA protocol is the image header as well as the complete firmware binary. The transmitted data needs to be signed and encrypted using prestored keys such that the secure bootloader on the DA1470x family of devices can verify this using the prestored keys. A checksum provided by the initiator at the end of the transfer between the SUOTA client and the SUOTA server makes sure the data was received completely. When the SUOTA transfer is completed the SUOTA server is in charge to adapt the product header. The product header – besides other data – contains two address pointers as shown in Table 9.

Table 9: Product Header Details

Product Header Field	Notes
Flash Programmed Identifier	ID to indicate valid product header
Active FW image address	Start address of the image header of the currently active image, relative to address 0

Product Header Field	Notes
Upgrade FW image address	Start address of the image header of the upgrade active image, relative to address 0 (if available, otherwise equal to Active FW image address)

After the initial programming during mass programming, both pointers point to the same image as shown in Figure 9.

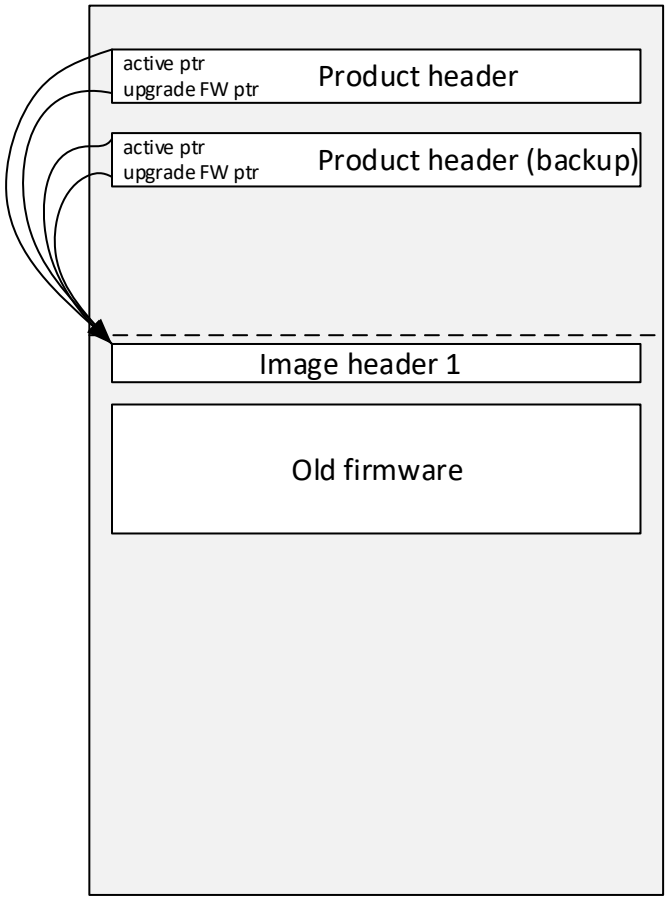

Figure 9: Product Header – No Pending Update

In case a firmware update was completely received the SUOTA server is in the charger to update the upgrade FW image pointer to point to the start of the image header from the update image. As the product header is available twice (see Figure 6) the SUOTA service has to update both before issuing a reset to get the ROM-based secure bootloader starting. Figure 10 shows where both image pointers direct to after the SUOTA programming and after the secure bootloader managed the upgrade.

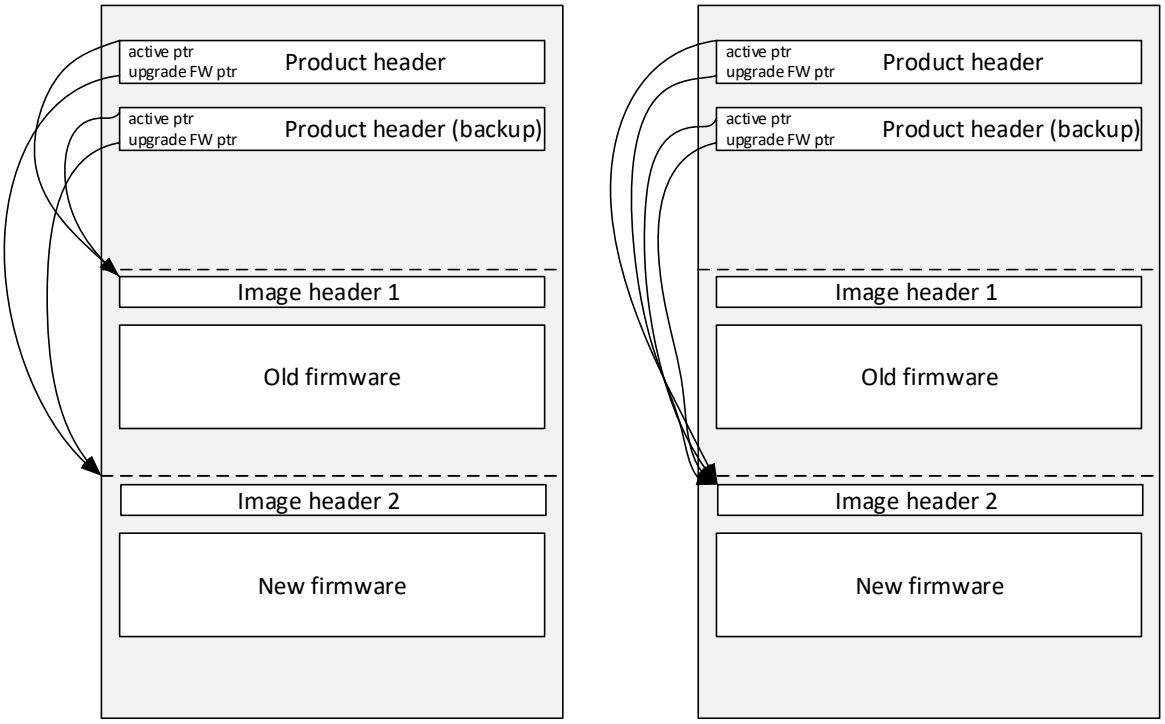

Figure 10: Product Header – Pending Update

When the ROM-based secure bootloader starts up it looks for the product header by an entry in the CS area. It then looks at both FW pointers which are part of the product header (see Table 9). If both point to the same image no FW update is pending. If both point to different images the bootloader starts verification of the pending update image pointed to by the upgrade FW pointer. The procedure used to verify the update image is exactly as described in the booter flow (see Section 4.3.3).
The booter does not differentiate between booting an active or booting an update image. If the update image fails verification the booter reruns firmware validation on the active image and executes this one if successfully verified. The booter does not update (erase and reprogram) the product headers as this is too risky, for example, for cases where the battery is near the end of its operating life. Product header update is handled by the SDK at start-up.

NOTE
The booter always looks at the original product header and if it is correct proceeds accordingly. The backup product header is only used if the original one was corrupted. Corruption is detected if the mandatory ID is missing or if the CRC is not correct.

4.3.11 Error Handling

In case an error occurs, the bootloader will issue a reset which restarts the booter operation. Errors can occur due to various reasons, some of which are listed below:

- Invalid entry in CS
- Signature of both active and upgrade image invalid
- Configuration of QSPI controller failed
- Revoked key had been used for signature or OTF decrypt
- Invalid image header ID
- Missing security section in case secure booting was enabled

- FW version below minimum firmware version
- Update of the minimum firmware version failed
- Key index invalid

When the booter successfully finishes image verification and before it passes control over to the booted application, it clears the RAM area used by the booter to make sure no sensitive information is remaining. There is one exception to this: in case the booter eventually boots the application some data is passed at a defined address which allows the booted application to see what sort of booting occurred (UART, ACTIVE, UPDATE) and whether or not OTP writing (in the context of minimum FW update) was successful.

5 Revision History

Revision	Date	Description
1.1	13-Mar-2023	Editorial changes.
1.0	30-May-2022	Initial version.

Status definitions

Status	Definition
DRAFT	The content of this document is under review and subject to formal approval, which may result in modifications or additions.
APPROVED or unmarked	The content of this document has been approved for publication.

RoHS Compliance

Renesas Electronics complies to European Directive 2001/95/EC and from 2 January 2013 onwards to European Directive 2011/65/EU concerning Restriction of Hazardous Substances (RoHS/RoHS2). RoHS certificates from our suppliers are available on request.

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit www.renesas.com/contact-us/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.