RZ/G2 Trusted Execution Environment

Porting Guide

Introduction

This document is intended to give users an understanding of the Trusted Execution Environment provided for RZ/G2 Group (hereinafter referred to as “TEE for RZ/G2”) and to serve as a reference for developing software for systems that use TEE for RZ/G2.

This document is intended for developers implementing the TEE for RZ/G2 using Yocto build environment provided for the RZ/G2 Group.

Target Device

RZ/G2E

RZ/G2M

RZ/G2N

RZ/G2H

Contents

1. Overview

TEE for RZ/G2 is an isolated execution environment that is implemented by Arm® TrustZone® supported by RZ/G2 Group and software utilizing that TrustZone®. This isolated execution environment guarantees code and data loaded inside to be protected with respect to confidentiality and integrity.

1.1 Functions

This section describes the software and its functions related to TEE for RZ/G2.

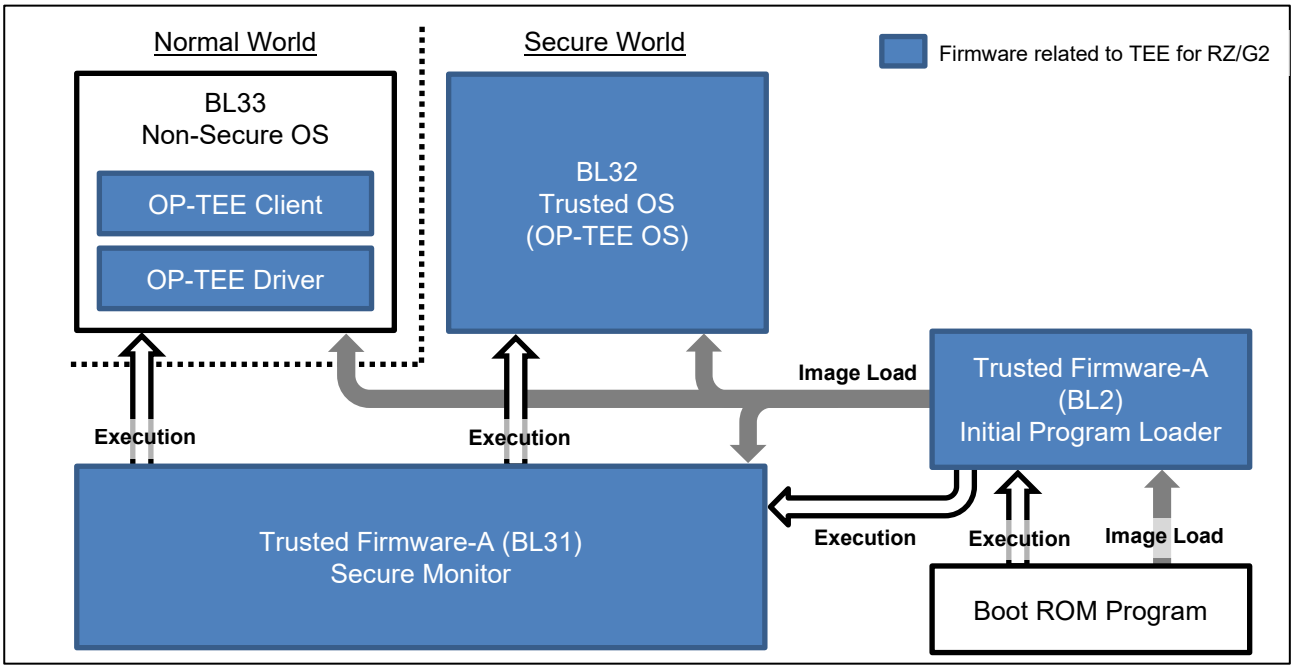

Figure 1. Software related to TEE for RZ/G2

(a) Trusted Firmware-A (BL2)

Trusted Firmware-A (BL2) is software for loading Trusted Firmware-A (BL31) and the firmware images to be booted after that. BL2 boots after being loaded into RAM from the boot device ROM by the boot ROM program. After booting, BL2 loads the firmware images from ROM to RAM after initializing hardware such as peripherals and setting security.

— Secure Boot
TEE for RZ/G2 supports Secure Boot that detects tampering with the firmware images loaded by BL2. If Secure Boot is implemented, the firmware images are signed and encrypted before being stored in boot device ROM. Secure Boot detects tampering by decrypting and verifying this firmware images. For the implementation of Secure Boot, refer to “5. How to Implement Secure Boot”.

(b) OP-TEE OS

Trusted Firmware-A (BL31) is the Secure World software including the Secure Monitor, various ARM interface standards such as the Power State Coordination Interface (PSCI).

— Secure Monitor
The Secure Monitor manages the switches between the Secure World and the Normal World. When a SMC, FIQ and IRQ are generated, the Exception Handler decides to need to switch the world. If it needs to switch the world, the Secure Monitor saves register data and restore register for the next world.

— PSCI
PSCI is the interface from the Normal World software to firmware implementing power management use-cases, Secondary CPU Boot, CPU Hotplug, CPU Idle and System Shutdown/Reset/Suspend.

(c) OP-TEE OS

OP-TEE OS is a Trusted OS which is running in the Secure World. OP-TEE OS provides "TEE Internal API" defined by the GlobalPlatform TEE Standard to a Trusted Application that accesses secure resources.

- Trusted Application (TA)
  Applications running in OP-TEE OS are called TA. TA is a passive type of application. TA receives and executes the request command from Client Application (CA). And return the results to CA.
- Client Application (CA)
  Applications running in the Non-Trusted OS are called CA. CA makes use of the TEE Client API to access the secure resources provided by TA.

(d) OP-TEE Driver

OP-TEE Driver for Linux OS allows communication between Linux OS and OP-TEE OS.

(e) OP-TEE Client

OP-TEE Client consists of the TEE Client library and TEE supplicant.
TEE Client library is a library that contains APIs defined by the GlobalPlatform TEE Standard. This library is used by CA that are executed on Non-Trusted OS to communicate with the OP-TEE OS and TA.
TEE Supplicant operates miscellaneous features of OP-TEE OS in the Secure World, such as file system access and loading TA.

1.2 References

1.2.1 Standard Documents

The following table shows the standard documents related to TEE for RZ/G2.

Table 1-1 Standard Documents

No	Issue	Title	Edition
1	GlobalPlatform	TEE Client API Specification	1.0
2	GlobalPlatform	TEE Internal Core API Specification	1.1

1.2.2 Related Documents

The following table shows the documents related to TEE for RZ/G2.

Table 1-2 Related Documents

No	Issue	Title
1	Renesas Electronics	RZ/G Verified Linux Package for 64bit kernel Release Note
2	Renesas Electronics	Linux Interface Specification Yocto recipe Start-Up Guide
3	Renesas Electronics	RZ/G2 Reference Boards Start-up Guide
4	Renesas Electronics	RZ/G Series, 2nd Generation User's Manual: Hardware LSIs for Rich Graphics Applications
5	Renesas Electronics	RZ/G Series, 2nd Generation User's Manual: Hardware Additional Document for Security
6	Renesas Electronics	RZ/G2 Trusted Execution Environment Start-Up Guide

1.2.3 Related Original Software

The following table shows the original software related to TEE for RZ/G2.

Table 1-3 Related Original Software

No	Software	Title and URL
1	Trusted Firmware-A	Secure Monitor https://github.com/ARM-software/arm-trusted-firmware
2	OP-TEE OS	Trusted side of the TEE https://github.com/OP-TEE/optee_os
3	OP-TEE Driver	Normal World driver https://git.kernel.org/pub/scm/linux/kernel/git/cip/linux-cip.git Branch: linux-4.19.y-cip Source Directory: drivers/tee
4	OP-TEE Client	Normal World Client side of the TEE https://github.com/OP-TEE/optee_client
5	OP-TEE Test	OP-TEE Test suite https://github.com/OP-TEE/optee_test
6	Security Module	RZ/G Security Module https://github.com/renesas-rz/rzg_security-module
7	Flash Writer	RZ/G2 Flash Writer https://github.com/renesas-rz/rzg2_flash_writer

1.3 Related Packages

The following table shows the packages related to TEE for RZ/G2.

Table 1-4 Related Packages

No	Package	Explanation
1	RZ/G2 Secure IP Package	This is a package provided by Renesas and is required to implement Secure Boot. For more information, refer to "How to Implement Secure Boot".

1.4 Licenses

The following table shows the licenses of software related to TEE for RZ/G2.

Table 1-5 Licenses

No	Software		Licenses
1	Trusted Firmware-A		BSD-3-Clause
2	OP-TEE OS		BSD-2-Clause or BSD-3-Clause
3	OP-TEE Driver		GPLv2
4	OP-TEE Client		BSD-2-Clause
5	OP-TEE Test	Client Application	GPLv2
		Trusted Application	BSD-2-Clause
6	Security Module		BSD-3-Clause
7	Flash Writer		BSD-3-Clause

1.5 Terminology

The following table shows the terminology related to this document.

Table 1-4 Terminology

No	Term	Explanation
1	PSCI	Power State Coordination Interface. It defines a Standard Interface for power management that can be used by OS vendors for supervisory software working at different levels of privilege on an ARM device.
2	Secure World	It is one of the security states that defined ARMv8-A architecture. When in this state, the CPU can access both the Secure and Non-Secure space.
3	Normal World	It is one of the security states that defined ARMv8-A architecture. When in this state, the CPU can access only Non-Secure space.
4	SMC	Secure Monitor Call. An ARM assembler instruction that causes an exception that is taken synchronously into EL3.
5	RPMB	Replay Protected Memory Block
6	Exception Levels (EL0/EL1/EL3)	The ARMv8-A architecture defines a set of Exception levels EL0 to EL3 where: If ELn is the Exception level, increased values of n indicate increased software execution privilege. Execution at EL0 is called unprivileged execution. EL1 provides support for virtualization of Non-Secure operation. EL3 provides support for switching between to Security states, Secure state and Non-Secure state.

2. Operating Environment

2.1 Build Environment

The recommended environment for build is the same as the RZ/G2 Linux BSP. For details, refer to “Related Documents No.2”.

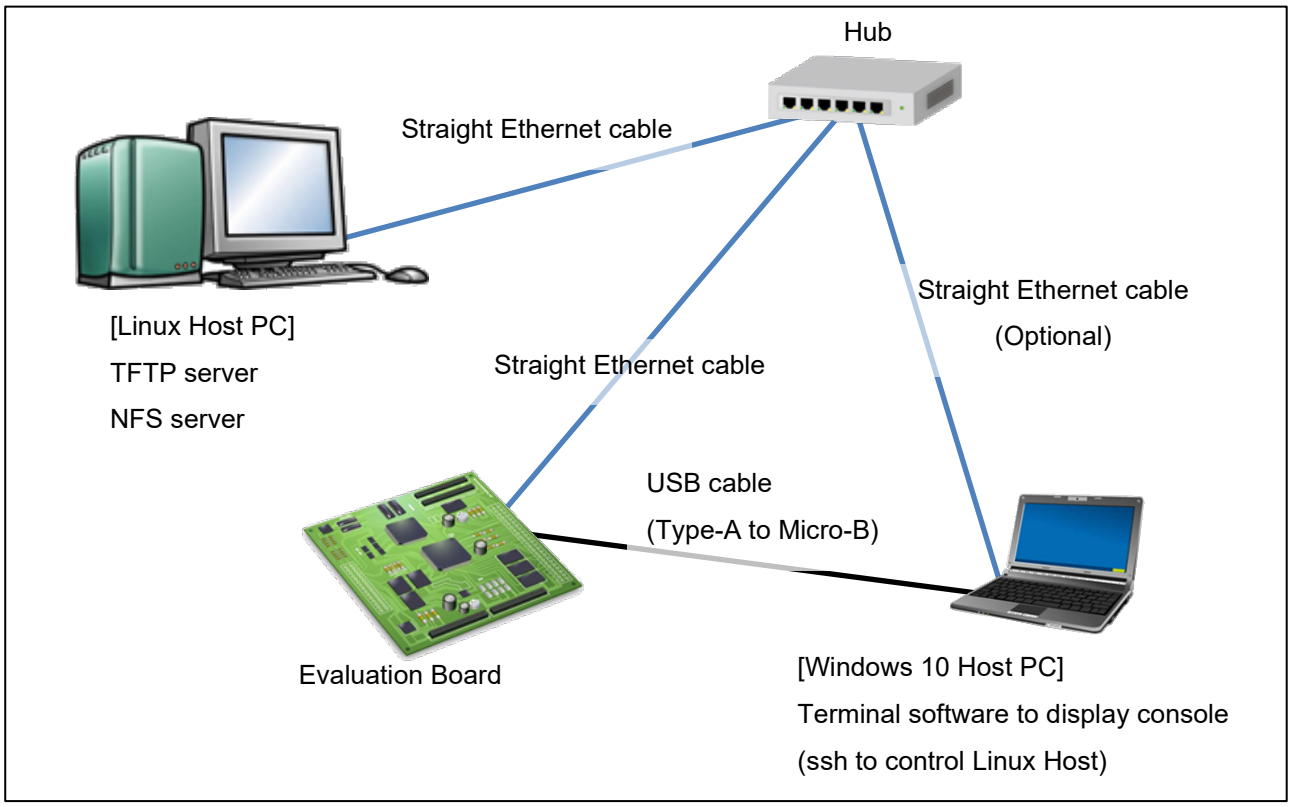

Figure 2-1 Recommended Environment

2.2 Module Configuration

This section shows the software relationship and configurations.

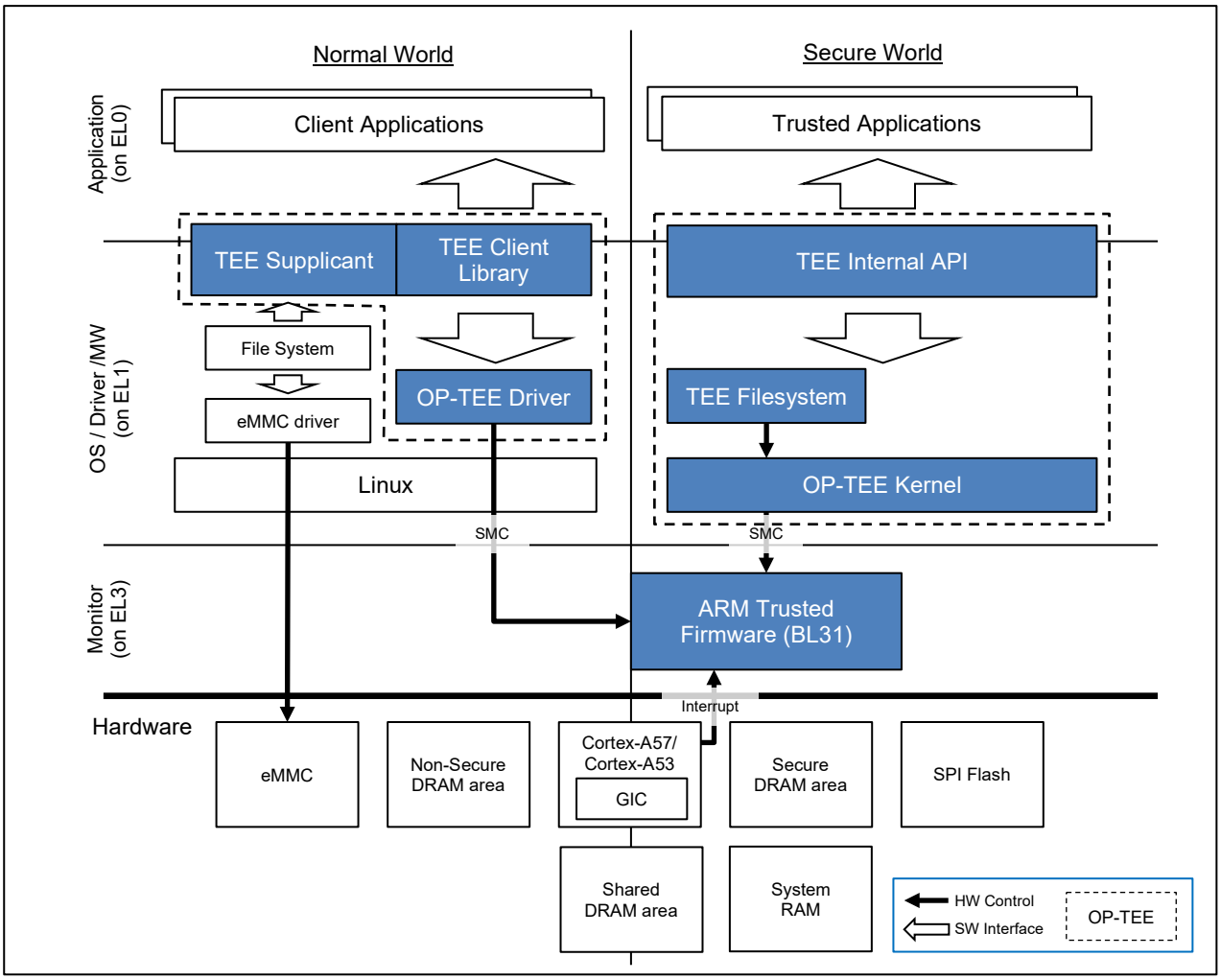

Figure 2-2 Software relationship

3. External Interface

3.1 Software API

This section describes the software API of defined in TEE for RZ/G2.

3.1.1 OP-TEE OS

- TEE Internal API
  For details on the specification for TEE Internal API, refer to “Standard Documents No.2”.
  The specification of TEE Internal API provided by TEE for RZ/G2 are based on the implementation of the original software because there is no change from the source code of “Related Original Software No.1”.
  All APIs are declared in the following header files.

  [${WORK}/build/tmp/work/<work-sub-directories>/optee-os/<Properties-of-yocto-environment>/git/lib/libutee/include/tee_api.h]

3.1.2 OP-TEE Client

- TEE Client API
  For details on the specification for TEE Client API, refer to “Standard Documents No.1”.
  The specification of TEE Client API provided by TEE for RZ/G2 are based on the implementation of the original software because there is no change from the source code of “Related Original Software No.4”.
  All APIs are declared in the following header files.

  [${WORK}/build/tmp/work/<work-sub-directories>/optee-client/<Properties-of-yocto-environment>/git/out/export/usr/include/tee_client_api.h]

3.2 Definitions

This section shows the definitions defined in TEE for RZ/G2.

3.2.1 OP-TEE OS

- TEE Internal API
  For details on the specification for TEE Internal API, refer to “Standard Documents No.2”.
  All definitions are declared in the following header files.

  [${WORK}/build/tmp/work/<work-sub-directories>/optee-os/<Properties-of-yocto-environment>/git/lib/libutee/include/tee_api_defines.h]

3.2.2 OP-TEE Client

- TEE Client API
  For details on the specification for TEE Client API, refer to “Standard Documents No.1”.
  All definitions are declared in the following header files.

  [${WORK}/build/tmp/work/<work-sub-directories>/optee-client/<Properties-of-yocto-environment>/git/out/export/usr/include/tee_client_api.h]

3.3 Structures

This section shows the structures defined in TEE for RZ/G2.

3.3.1 OP-TEE OS

- TEE Internal API
  For details on the specification for TEE Internal API, refer to “Standard Documents No.2”.
  All structures are declared in the following header files.

  [${WORK}/build/tmp/work/<work-sub-directories>/optee-os/<Properties-of-yocto-environment>/git/lib/libutee/include/tee_api_types.h]

3.3.2 OP-TEE Client

- TEE Client API
  For details on the specification for TEE Client API, refer to “Standard Documents No.1”.
  All structures are declared in the following header files.

  [${WORK}/build/tmp/work/<work-sub-directories>/optee-client/<Properties-of-yocto-environment>/git/out/export/usr/include/tee_client_api.h]

4. Implementation

4.1 Directory Configuration

This section shows the directory configuration of the software related to the TEE for RZ/G2.

4.1.1 Trusted Firmware-A

Trusted Firmware-A added the directory for RZ/G2 Group to the directory configuration of "Related Original Software No.1".

In Yocto build environment, the source code of Trusted Firmware-A is stored in the following path.

[${WORK}/build/tmp/work/<work-sub-directories>/arm-trusted-firmware/<Properties-of-yocto-environment>/git/]

The source code for RZ/G2 Group added to Trusted Firmware-A is stored in the following directory.

```
git
├── drivers
│   └── renesas
│       └── rzg
│           ├── auth
│           ├── board
│           ├── ddr
│           ├── emmc
│           ├── io
│           ├── pfc
│           ├── qos
│           ├── rom
│           └── watchdog
├── plat
│   └── renesas
│       └── rzg
│           └── include
└── tools
    └── renesas
        ├── rzg_layout_create
        └── rzg_security_tools
            ├── encrypt_fw
            │   ├── include
            │   └── src
            ├── fiptool
            │   └── src
            └── sign_fw
                ├── include
                └── src
```

Figure 4-1 Trusted Firmware-A directory configuration

4.1.2 OP-TEE OS

OP-TEE OS added the directory for RZ/G2 Group to the directory configuration of the “Related Original Software No.2”.

In Yocto build environment, the source code of OP-TEE OS is stored in the following path.

[${WORK}/build/tmp/work/<work-sub-directories>/optee-os/<Properties-of-yocto-environment>/git/]

The source code for RZ/G2 Group added to OP-TEE OS is stored in the following directory.

```
git
└── core
    └── arch
        └── arm
            └── plat-rzg
                ├── driver
                └── include
```

Figure 4-2 OP-TEE OS directory configuration

4.1.3 OP-TEE Client

OP-TEE Client does not modify the original source code of "Related Original Software No.4".

In Yocto build environment, the source code of OP-TEE Client is stored in the following path.

[${WORK}/build/tmp/work/<work-sub-directories>/optee-client/<Properties-of-yocto-environment>/git/]

4.1.4 OP-TEE Driver

OP-TEE Driver does not modify the original source code of "Related Original Software No.3".

In Yocto build environment, the source code of OP-TEE Client is stored in the following path.

[${WORK}/build/tmp/work-shared/<work-sub-directories>/kernel-source/drivers/tee]

4.2 Build Instructions

To build software related to TEE for RZ/G2 uses Yocto build environment provided for the RZ/G2 Group. For the Build Instructions, refer to “Related Documents No.6”.

4.3 Build Options

This section shows the build options related to TEE for RZ/G2.

4.3.1 Trusted Firmware-A

- SPD
  Set string is “opteed” or “none”. The string “opteed” specifies that OP-TEE OS will start. The string “none” specifies that OP-TEE OS will not start. If this option is not set, the string is set “opteed" internally.

4.3.2 OP-TEE OS

4.3.2.1 Secure Storage

- CFG_REE_FS
  Set value is “y” or “n”. The “y” specifies to enable REE Filesystem. If this option is not set, the value is set “n” internally.

- CFG_RPMB_FS
  Set value is “y” or “n”. The “y” specifies to enable RPMB Filesystem. If this option is not set, the value is set “n” internally.

  Note: This Secure Storage utilizes the Replay Protected Memory Block (RPMB) partition of the MMC/SD device. The MMC/SD controller driver must support access to the RPMB.

- CFG_RPMB_WRITE_KEY
  Set value is “y” or “n”. The “y” specifies to enable RPMB security key programming. If this option is not set, the value is set “n” internally.

4.3.2.2 Cryptography features

- CFG_CRYPTO_WITH_CE
  Set value is “y” or “n”. The “y” specifies to enable ARMv8 Cryptography Extension. If this option is not set, the value is set “n” internally.

- CFG_RZG_SEC_IP_DRV
  Set value is “y” or “n”. The “y” specifies to enable the Secure IP driver provided for RZ/G2 Group. The Secure IP driver is a driver for using the on chip Trusted Secure IP included with RZ/G2 Group. If this option is not set, the value is set “n” internally.

  Note: If this option is set to “y”, then CFG_RZG_SEC_LIB_DIR must be set.
  If this option is set to “y”, then Secure Boot provided by TEE for RZ/G2 must be implemented. For the implementation of Secure Boot, refer to “5. How to Implement Secure Boot”.

- CFG_RZG_SEC_LIB_DIR
  Set the path to the directory where the Secure IP library is stored. This option is only referenced when CFG_RZG_SEC_IP_DRV is set to “y”.

- CFG_RZG_SEC_IP_RNG
  Set value is “y” or “n”. The “y” specifies to enable the random number generator by the Secure IP driver. The "n" specifies to enable the default random number generator implemented in the OP-TEE OS. If this option is not set, the value is set “n” internally.

  Note: If this option is set to “y”, then CFG_RZG_SEC_IP_DRV must be set to “y”.

4.3.3 OP-TEE Client

- RPMB_EMU
  OP-TEE Client also has an emulation mode which implements a virtual RPMB device for test purposes. Set value is “1” or “0”. “0” specifies to access to a virtual RPMB device. “1” specifies to access to a real RPMB device.

4.4 How to set build options

This section shows how to set build options related to TEE for RZ/G2.

4.4.1 Trusted Firmware-A

The following is an example of adding or modifying the build options for Trusted Firmware-A.

On ${WORK}/meta-rzg2/recipes-bsp/arm-trusted-firmware/arm-trusted-firmware_git.bb

```
ATFW_OPT_r8a774c0 = “LSI=G2E … SPD="none" XXXXX=YY"
ATFW_OPT_r8a774a1 = “LSI=G2M … SPD="none" XXXXX=YY"
ATFW_OPT_r8a774b1 = “LSI=G2N … SPD="none" XXXXX=YY"
ATFW_OPT_r8a774e1 = “LSI=G2H … SPD="none" XXXXX=YY"
```

4.4.2 OP-TEE OS

The following is an example of adding or modifying the build options for OP-TEE OS.

On ${WORK}/meta-rzg2/recipes-bsp/optee/optee-os_git.bb

```
do_compile() {
    oe_runmake PLATFORM=${PLATFORM} … CFG_ARM64_core=y XXXXX=YY
}
```

4.4.3 OP-TEE Client

The following is an example of adding or modifying the build options for OP-TEE Client.

On ${WORK}/meta-rzg2/recipes-bsp/optee/optee-client_git.bb

```
EXTRA_OEMAKE = "RPMB_EMU=0 XXXXX=YY"
```

4.5 How to customize

This section describes the security features implemented in TEE for RZ/G2. Refer to the following site for the implementation of the original source code of the software related to TEE for RZ/G2.

Trusted Firmware-A Documentation:

https://trustedfirmware-a.readthedocs.io/en/latest/

OP-TEE Documentation:

https://optee.readthedocs.io/en/latest/index.html

4.5.1 Security access protection setting

The Security access protection setting determines whether to protect the access from Normal world to SRAM, SDRAM, and IPs. This is implemented by the TrustZone® and the peripheral module Life Cycle.

The Security access protection settings are implemented in the following source code and are executed by the Initial Program Loader.

[${WORK}/build/tmp/work/<work-sub-directories>/arm-trusted-firmware/<Properties-of-yocto-environment>/git/plat/renesas/rcar/bl2_secure_setting.c]

For the peripheral module Life Cycle, refer to the “Related Documents No.5”.

4.5.2 Hardware Unique Key

The Hardware Unique Key (hereinafter referred to as "HUK") is a key required to be implemented in OP-TEE OS. The HUK could for example be used when deriving keys used in secure storage etc.

The HUK is implemented in the following source code.

[${WORK}/build/tmp/work/<work-sub-directories>/optee-os/<Properties-of-yocto-environment>/git/core/arch/arm/plat-rzg/tee_common_otp.c]

4.5.3 Hardware Crypto IP

4.5.3.1 Cryptographic function

RZ/G2 Group supports ARMv8 Cryptography Extension. The Cryptographic function implementation by ARMv8 Cryptography Extensions does not modify the original source code of "Related Original Software No.2".

4.5.3.2 Random Number Generation

RZ/G2 Group supports the Hardware Random Number Generation using Trusted Secure IP. The Hardware Random Number Generation using Trusted Secure IP is enabled by implementing Secure Boot provided by TEE for RZ/G2.

The Hardware Random Number Generation is implemented in the following source code.

[${WORK}/build/tmp/work/<work-sub-directories>/optee-os/<Properties-of-yocto-environment>/git/core/arch/arm/plat-rzg/rzg_rng.c]

4.5.4 Secure Storage

The Secure Storage provided by OP-TEE OS does not modify the original source code of the "Related Original Software No.2".

4.5.5 Trusted Application Private/Public Keypair

The Trusted Application Private/Public Keypair is the key pair that OP-TEE OS uses to validate the Trusted Application. This key pair does not modify the original source code of "Related Original Software No.2".

4.5.6 Secure Boot

For Secure Boot provided by TEE for RZ/G2, refer to "How to Implement Secure Boot".

5. How to Implement Secure Boot

5.1 Functions

TEE for RZ/G2 supports Secure Boot using the on chip Trusted Secure IP (hereinafter referred to as “TSIP”) included with RZ/G2 Group processor. The signed and encrypted data stored in the non-volatile memory is decrypted and verified using TSIP to check for tampering.

Secure Boot using TSIP is a trigger to enable the cryptography functions* provided by TSIP. In Secure Boot is not implemented system, the cryptography functions provided by TSIP cannot be used.

Note: In TEE for RZ/G2, the cryptography functions provided by TSIP is the Hardware Random Number Generation implemented in OP-TEE OS.

The Secure IP library for accessing TSIP is required to implement Secure Boot. This library is referenced in build of Security Module and OP-TEE OS. The Secure IP library is included in the Secure IP Package. For inquiries regarding the provision of RZ/G2 Secure IP Package, please contact Renesas Electronics distributor or contact us.

5.1.1 Secure Boot

Secure Boot sequence is shown below.

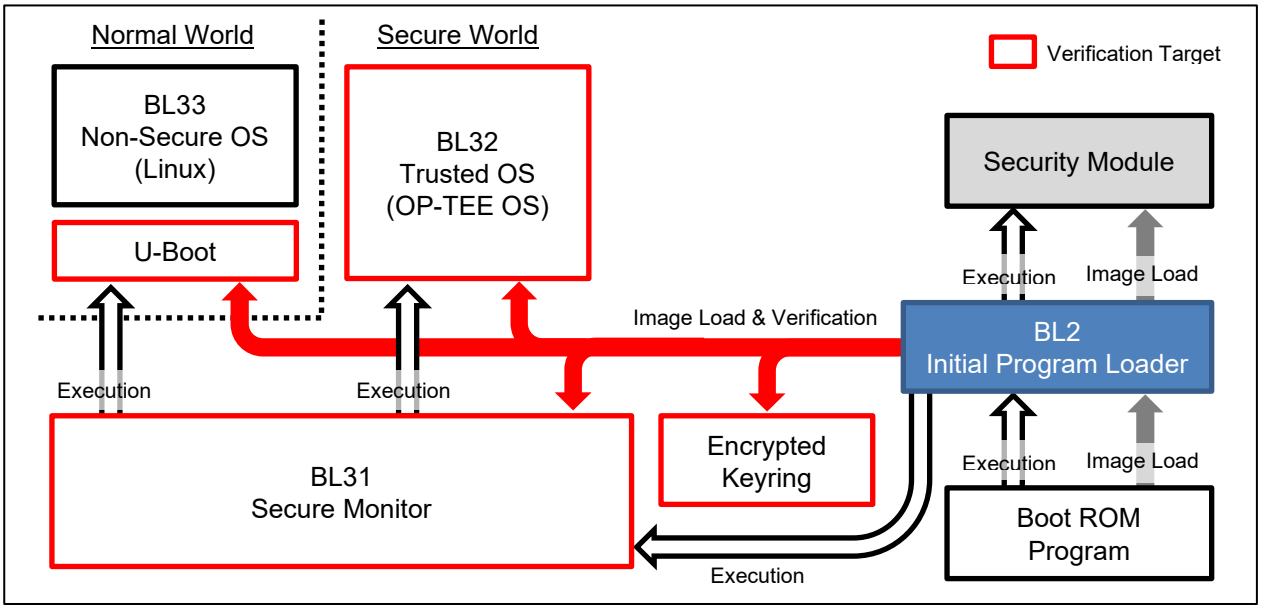

Figure 5-1 Secure Boot Sequence

BL31, BL32, U-Boot, and Keyring are verified by Secure Boot. These data are signed and encrypted before being stored in non-volatile memory. During Secure Boot, BL31, BL32 and U-Boot are decrypted and validated, and placed in RAM. The Keyring is verified and placed in RAM.

Verification and decryption of firmware images by Secure Boot is done in the Security Module. If Security Module failed validation, BL2 aborts the boot sequence.

(a) Security Module

Security Module is software for using TSIP included with RZ/G2 Group processor. Security Module includes Secure IP library for decryption and verification using TSIP. Secure IP library is a library for accessing TSIP.

Security Module functions are:

- Verification of Keyring
- Decryption and Verification of User Data
- Re-Encryption of Keyring
- Re-Encryption of User Data

Security Module uses "Verification of Keyring" and "Decryption and Verification of User Data" for Secure Boot.

(b) Encrypted Keyring

Keyring is a bunch of Session Keys used to bring data prepared in the external environment to the user product environment. Encrypted Keyring is the data that Keyring is encrypted with a device-specific key. Encrypted Keyring can only be accessed by TSIP and are never decrypted into RAM.

Encrypted Keyring is used to securely bring in data from the outside, such as Provisioning of the User Data and Firmware Update.

5.1.2 Provisioning

To implement Secure Boot environment on the user product environment, Encrypted Keyring and Encrypted User Data must be stored in non-volatile memory. The process from preparing Keyring and User Data in the external environment to encrypting the data with the device-specific key in the user product environment is called Provisioning.

The Provisioning sequence is shown below.

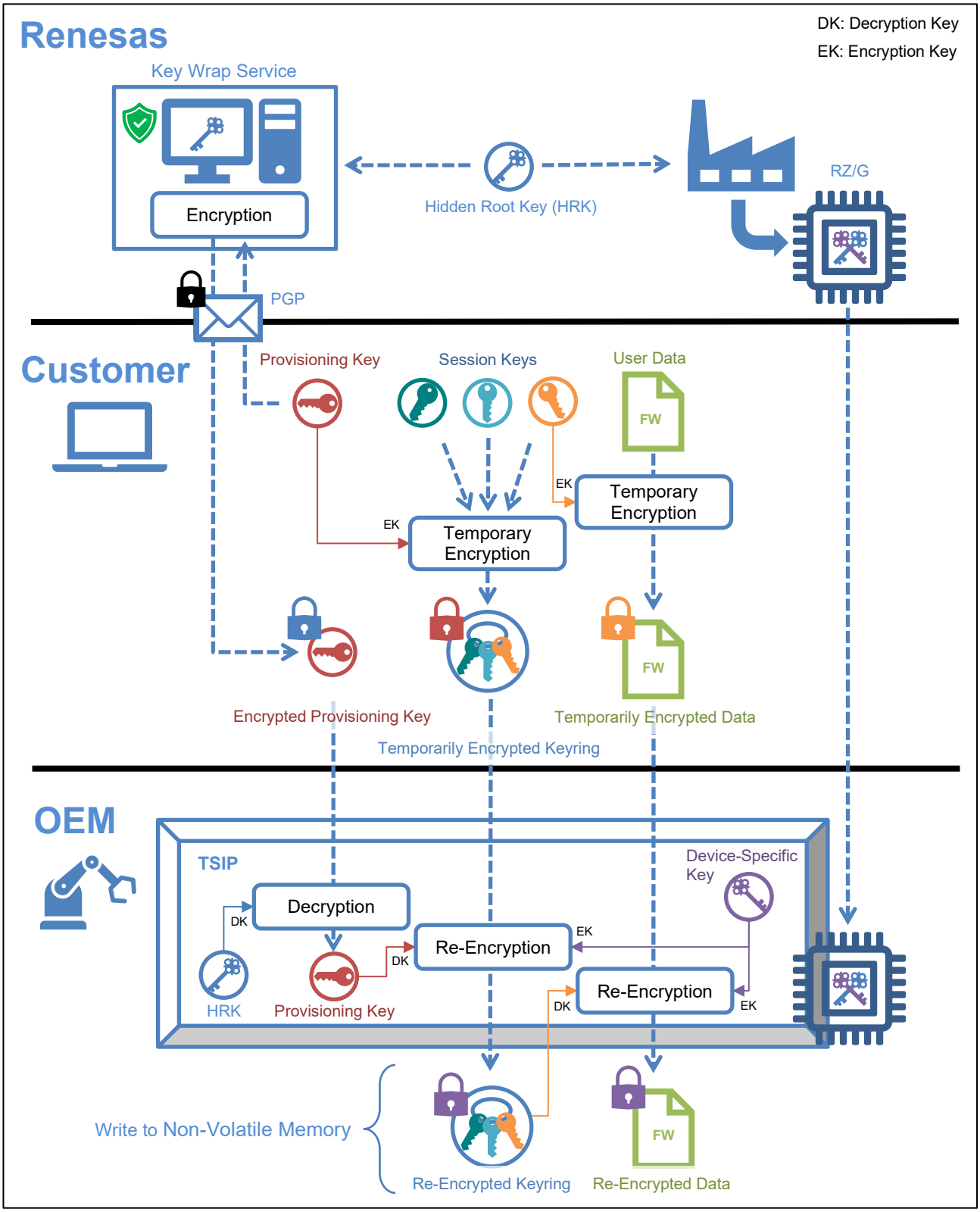

Figure 5-2 Provisioning Sequence

The keys used for Provisioning are shown below.

Table 5-1 Keys used for the Provisioning

Key	Explanation	Creator
Hidden Root Key (hereinafter referred to as HRK)	Key managed within Renesas, and this key is not provided to user. The HRK exists only inside the Key Wrap Service and TSIP. Used to encrypt and decrypt Provisioning Key.	Renesas
Device-Specific Key	Unique key that exists only inside TSIP. All data verified by Secure Boot is encrypted with this key.	
Provisioning Key	Key for temporarily encrypting Keyring. By encrypting Keyring with this key prevents the session key from leaking between the external environment and the user product environment.	User
Encrypted Provisioning Key	Provisioning Key encrypted with the Hidden Root Key. It can be encrypted using the Key Wrap Service provided by Renesas. Used to re-encrypt the temporarily encrypted Keyring in the user product environment.	
Session Keys	Keys used to temporarily encrypt User Data and Keys prepared in the external environment. By encrypting the data with this key prevents leakage and tampering the data between the external environment and the user product environment.	
Keyring	A bunch of Session Keys.	
Temporarily Encrypted Keyring	Keyring encrypted with Provisioning Key.	
Re-Encrypted Keyring	Keyring re-encrypted with Device-Specific Key.	

The Provisioning process performed by the customer is shown below.

Step 1 Process on the build environment

1. Generation of Keyring
2. Temporary encryption of Keyring
3. Temporary encryption of User Data

Step 2 Process on the target environment

4. Re-Encryption of Keyring
5. Re-Encryption of User Data

The following is an overview of each step in the Provisioning process.

5.1.2.1 Generation of Keyring

"Generation of Keyring" is the process to be executed in the external environment.

Keyring is a bunch of Session Keys. Session Keys is used to temporarily encrypt the User Data and Keys prepared in the external environment.

This process generates Session Keys and Keyring in the external environment.

Note: In Yocto build environment provided by TEE for RZ/G2, Keyring and Session Keys are generated in the local directory of the user build environment. Please be careful not to leak this Keyring and Session Keys to the outside.

Format of Keyring is shown below.

Table 5-2 Format of Keyring

Session Keys		**Algorithm**	**Size (Byte)**
Reserved		(fixed 0)	32
For Secure Boot	Temporary Encryption Key for User Data	AES-128	16
		IV0	16
	Temporary Verification Key for Signature of User Data*	PublicKey(n)	256
		PublicKey(0^15 Padding || e || 0^96 Padding)	16
Reserved		-	272
For Secure Software Update	Temporary Encryption Key for Keyring	AES-128	16
	MAC Key for Keyring	AES-128	16
Reserved		-	32

Note: Private key that is pair of this key (temporary signing key for signature of user data) is used for signing User Data in the Provisioning process.

The following shows how to use Session Keys included in Keyring.

— for Secure Boot
This key is the session key to securely bring User Data for Secure Boot into the user product environment.
This key is used to re-encrypt the Temporarily Encrypted User Data.

— for Secure Software Update
This key is the session key used to securely update Keyring in the user product environment.
This key is used to re-encrypt Temporarily Encrypted Keyring for the purpose of updating Keyring after Provisioning.

5.1.2.2 Temporary encryption of Keyring

"Temporary encryption of Keyring" is the process to be executed in the external environment.

This process encrypts Keyring using Provisioning Key. Encrypting Keyring prevents the session key from leaking between the external environment and the user product environment.

Provisioning Key is length of 256 bits, which is a concatenation of the two keys. Format of Provisioning Key is shown below.

Table 5-3 Format of Provisioning Key

Key		Algorithm	Size (Byte)
for Provisioning	Temporary Encryption Key for Keyring	AES-128	16
	MAC Key for Keyring	AES-128	16

Encrypted Provisioning Key is required to re-encrypt Temporarily Encrypted Keyring in the user product environment. For information about how to encrypt the provisioning key, refer to “5.6. Key Wrap Service”.

5.1.2.3 Temporary encryption of User Data

"Temporary encryption of User Data" is the process to be executed in the external environment.

This process encrypts User Data for Secure Boot using the Temporary Encryption Key for User Data included in Keyring. User Data is signed with a private key that is paired with Temporary Verification Key for Signature of User Data before it is encrypted.

Encrypting User Data prevents the data from leaking between the external environment and the user product environment.

5.1.2.4 Re-encryption of Keyring

"Re-encryption of Keyring" is the process to be executed in the user product environment.

This process re-encrypts Temporarily Encrypted Keyring using Device-Specific Key by TSIP. Re-encryption of Keyring requires Encrypted Provisioning Key to decrypt Temporarily Encrypted Keyring.

Re-Encrypted Keyring is stored in non-volatile memory and is used for "Re-encryption of User Data" and Secure Boot. Re-Encrypted Keyring cannot be used on other devices because it is encrypted using Device-Specific Key.

Note: After Re-encryption of Keyring is complete, Temporarily Encrypted Keyring and Encrypted Provisioning Key must be deleted from the user product environment.

5.1.2.5 Re-encryption of User Data

"Re-encryption of User Data" is the process to be executed in the user product environment.

This process re-encrypts Temporarily Encrypted User Data using Device-Specific Key by TSIP. Re-encryption of User Data requires Re-Encrypted Keyring to decrypt Temporarily Encrypted User Data.

Re-Encrypted User Data is stored in non-volatile memory and decrypted and validated during Secure Boot. Re-Encrypted User Data cannot be decrypted and verified on other devices because it is encrypted using Device-Specific Key.

Note: After Re-encryption of User Data is complete, Temporarily Encrypted User Data must be deleted from the user product environment.

5.2 Build Instructions

To build software related to TEE for RZ/G2 uses Yocto build environment provided for the RZ/G2 Group. For the Build Instructions, refer to “Related Documents No.6”.

5.3 Build Options

This section shows the build options related to implement Secure Boot.

5.3.1 Trusted Firmware-A

- RZG2_SECURE_BOOT
  Set value is “1” or “0”. “1” specifies to enable Secure Boot. If this option is not set, the value is set “0” internally.

5.3.2 Security Module

- LSI
  Set string is the RZ/G device in the user product. The RZ/G device is selected from "G2E", "G2M", "G2N", "G2H". This is mandatory option. If not set it then the build error occurs.

- SEC_LIB_DIR
  Set the path to the directory where the Secure IP library is stored. This is mandatory option. If not set it then the build error occurs.

5.3.3 Flash Writer

- SEC_PRV
  Set string is “ENABLE” or “DISABLE”. The “ENABLE” specifies to enable Provisioning for Secure Boot. If this option is not set, the string is set “DISABLE” internally.

- SEC_PRV_KEY_ENC
  Set the file path for Encrypted Provisioning Key. This option is only referenced when SEC_PRV is set to “y”.

5.4 How to set build options

This section shows how to set build options related to implement Secure Boot.

5.4.1 Trusted Firmware-A

The following is an example of adding or modifying the build options for Trusted Firmware-A.

On ${WORK}/meta-rzg2/recipes-bsp/arm-trusted-firmware/arm-trusted-firmware_git.bbappend

```
ATFW_OPT_append += “RZG2_SECURE_BOOT=1"
```

5.4.2 Security Module

The following is an example of adding or modifying the build options for Security Module.

On ${WORK}/meta-rzg2/recipes-bsp/security/secmod_1.0.bb

```
LSI_OPT_r8a774c0 = "LSI=G2E"
LSI_OPT_r8a774a1 = "LSI=G2M"
LSI_OPT_r8a774b1 = "LSI=G2N"
LSI_OPT_r8a774e1 = "LSI=G2H"
LSI_OPT_append += " SEC_LIB_DIR=${SYMLINK_NATIVE_SEC_LIB_DIR}"
```

5.4.3 Flash Writer

The following is an example of adding or modifying the build options for Flash Writer.

On ${WORK}/meta-rzg2/recipes-bsp/flash-writer/flash-writer_1.02.bbappend

```
EXTRA_OEMAKE_append += "SEC_PRV=y SEC_PRV_KEY_ENC =${SEC_PRV_KEY_ENC}"
```

5.5 Provisioning Environment

This section describes the Provisioning environment provided to implement Secure Boot. This Provisioning Environment is included in Yocto build environment.

This section assumes that Yocto build environment has been built according to “Related Documents No.6”.

The correspondence between the Provisioning process and the tools provided to RZ/G2 Group is shown below.

Table 5-5 Correspondence table of Provisioning and tools

No	Provisioning	Tools
1	Generation of Keyring	• Provisioning Tool
2	Temporary encryption of Keyring	• Provisioning Tool
3	Temporary encryption of User Data	• Encryption Tool
4	Re-Encryption Keyring	• Packaging Tool • Flash Writer
5	Re-Encryption User Data	• Packaging Tool • Flash Writer

5.5.1 Prior confirmation

Before running the tool in the Provisioning environment, confirm the following:

Step 1 Enable Secure Boot

Confirm that the "RZG2_SECURE_BOOT" option is "ENABLE".

On ${WORK}/meta-rzg2/include/rzg2-security-config.inc

```
RZG2_SECURE_BOOT  = 'ENABLE'
```

Step 2 Set the directory path

Confirm the path settings of the directory that stores the generated keys and Secure IP library. Change this directory path according to user's build environment.

On ${WORK}/meta-rzg2/recipes-bsp/security/secprv-native_1.0.bb

```
DIRPATH_SEC_STORAGE = "${HOME}/.secprv"

DIRPATH_GEN_KEY_ROOT = "${DIRPATH_SEC_STORAGE}/${MACHINE}/key"
DIRPATH_SEC_LIB_ROOT = "${DIRPATH_SEC_STORAGE}/${MACHINE}/lib"
```

Note: The generated key file and Secure IP libraries are saved in this directory. Please be careful when managing this directory.

Step 3 Save Secure IP library

Confirm that the Secure IP library is saved as follows.

```
${HOME}/.secprv
├─ ek874
│   └─ lib
│       ├─ libr_secure_ip_lib_g2e.a
│       └─ libr_secure_ip_lib_g2e.a.X.X.X
│
├─ hihope-rzg2m
│   └─ lib
│       ├─ libr_secure_ip_lib_g2m.a
│       └─ libr_secure_ip_lib_g2m.a.X.X.X
│
├─ hihope-rzg2n
│   └─ lib
│       ├─ libr_secure_ip_lib_g2n.a
│       └─ libr_secure_ip_lib_g2n.a.X.X.X
│
└─ hihope-rzg2h
    └─ lib
        ├─ libr_secure_ip_lib_g2h.a
        └─ libr_secure_ip_lib_g2h.a.X.X.X
```

Figure 5-3 Example of saving Secure IP library

5.5.2 Provisioning Tool

Provisioning Tool is a tool that runs in Yocto build environment.

Provisioning Tool provides the following features:

- Generation of Keyring
- Configure Build Path

Provisioning Tool consists of the recipe files and scripts. The configuration of Provisioning Tool is shown below.

```
${WORK}/meta-rzg2/recipe-bsp/security/
├── secprv-native_1.0.bb
│
└── secprv-1.0
    └── tool
        ├── config.sh
        ├── genkey.sh
        ├── keyring.sh
        ├── sec_keygen.sh
        ├── utility.sh
        └── wrapkey.sh
```

Figure 5-4 Provisioning Tool

5.5.2.1 Generation of Keyring

The procedure for Generation of Keyring is shown below.

Step 1 Execute command

Execute the following command from the terminal.

```
$ bitbake secprv-native -c newkey -f
```

After executing the command, the following directories and files will be created in the directory defined in the recipe file.

```
${HOME}/.secprv/${MACHINE}
└── key
    ├── 0.0.0
    │   ├── BCF1-Key.bin
    │   ├── BCF2-Key.bin
    │   ├── E-Key.bin
    │   ├── Keyring.bin
    │   ├── Keyring_Enc.bin
    │   ├── SBP-Key.pem
    │   ├── SBS-Key.pem
    │   ├── SS_UP1-Key.bin
    │   └── SS_UP2-Key.bin
    ├── 1.0.0
    │       ...
    ├── 2.0.0
    │       ...
    └── Provisioning
        └── ProvisioningKey.bin
```

Figure 5-5 Example of generated keys

A directory created with a version number such as "0.0.0" or "1.0.0" stores generated Keyring and Session Keys. A new directory is created each time the command in Step 1 is executed. This directory also stores Encrypted Keyring that is temporarily encrypted using Provisioning Key.

“Provisioning” directory stores generated Provisioning Key. However, Provisioning Key is only generated if it does not exist. If Provisioning Key already exists, it will not be generated by executing the command in Step 1.

Note: To generate new Provisioning Key, delete all directories below the “key” directory before executing the command in Step 1. Temporarily Encrypted Keyring is encrypted with Provisioning Key. Therefore, if Provisioning Key changes, Encrypted Keyring must be removed.

The following table shows the keys generated by Provisioning Tool.

Table 5-4 List of generated key files

Key Type		**File Name**
For Secure Boot	Temporary Encryption Key for User Data	E-Key.bin
	Temporary Verification Key for Signature of User Data	SBP-Key.pem
	Temporary Signing Key for User Data	SBS-Key.pem
For Secure Software Update	Temporary Encryption Key for Keyring	SS_UP1-Key.bin
	MAC Key for Keyring	SS_UP2-Key.bin
Reserved		BCF1-Key.bin
		BCF2-Key.bin
Keyring		Keyring.bin
Temporarily Encrypted Keyring		Keyring_Enc.bin
Provisioning Key		ProvisioningKey.bin

Encrypted Provisioning Key is required to re-encrypt Temporarily Encrypted Keyring in the user product environment. For information about how to encrypt Provisioning Key, refer to “5.6. Key Wrap Service”.

Encrypted Provisioning Key must be renamed to “ProvisioningKey_Enc.bin” and saved in the “Provisioning” directory before executing the next step “Configure Build Path”.

5.5.2.2 Configure Build Path

In "Configure Build Path", the path to the directory where the generated key and the secure IP library is stored is configured to Yocto build environment.

The procedure for Configure Build Path is shown below.

Step 1 Specify the Keyring version

From Keyrings generated by the commands in section 5.5.2.1, select Keyring used for Secure Boot. Define the name (version number) of the directory where the selected Keyring is stored in the following recipe file.

On ${WORK}/meta-rzg2/recipes-bsp/security/secprv-native_1.0.bb

```
DIR_V_MAJOR = '0'  # 0, 1, 2, 3, …
DIR_V_MINOR = '0'
DIR_V_TRACE = '0'
DIR_VERSION = "${DIR_V_MAJOR}.${DIR_V_MINOR}.${DIR_V_TRACE}"
```

Step 2 Execute command

Execute the following command from the terminal.

```
$ bitbake secprv-native -c install -f
```

After executing the command, A symbolic link to the directory containing the keys and Secure IP library is created in Yocto build environment. In Yocto build environment, refer to the following symbolic links when referencing keys and Secure IP library.

On ${WORK}/meta-rzg2/include/rzg2-security-config.inc

```
DIRPATH_SEC_DATADIR_NATIVE  = "${STAGING_DATADIR_NATIVE}/.secure"
SYMLINK_NATIVE_BOOT_KEY_DIR = "${DIRPATH_SEC_DATADIR_NATIVE}/keyring"
SYMLINK_NATIVE_PROV_KEY_DIR = "${DIRPATH_SEC_DATADIR_NATIVE}/provkey"

DIRPATH_SEC_LIBDIR_NATIVE   = "${STAGING_LIBDIR_NATIVE}/.secure"
SYMLINK_NATIVE_SEC_LIB_DIR  = "${DIRPATH_SEC_LIBDIR_NATIVE}/library"
```

5.5.3 Encryption Tool

Encryption tools are used to temporarily encrypt User Data. Encryption Tool is created as a Trusted Firmware-A tool.

Encryption Tool is stored in the following path.

[${WORK}/build/tmp/work/<work-sub-directories>/arm-trusted-firmware/<Properties-of-yocto-environment>/git/tools/renesas/rzg_security_tools/]

The configuration of Encryption Tool is shown below.

```
rzg_security_tools
├── encrypt_fw
│   ├── include
│   │   ├── cmd_opt.h
│   │   ├── debug.h
│   │   └── encrypt.h
│   ├── Makefile
│   └── src
│       ├── cmd_opt.c
│       ├── encrypt.c
│       └── main.c
└── sign_fw
    ├── include
    │   ├── cmd_opt.h
    │   ├── debug.h
    │   └── sign.h
    ├── Makefile
    └── src
        ├── cmd_opt.c
        ├── main.c
        └── sign.c
```

Figure 5-6 Encryption Tool

The following is an example of build the Encryption Tool in Yocto build environment.

On ${WORK}/meta-rzg2/recipes-bsp/arm-trusted-firmware/arm-trusted-firmware_git.bbappend

```
do_compile_append() {
    oe_runmake -C tools/renesas/rzg_security_tools/sign_fw clean
    oe_runmake -C tools/renesas/rzg_security_tools/sign_fw

    oe_runmake -C tools/renesas/rzg_security_tools/encrypt_fw clean
    oe_runmake -C tools/renesas/rzg_security_tools/encrypt_fw
}
```

The following program is created by building the Encryption Tool.

- encrypt_fw
  encrypt_fw is a encryption tool. This tool reads the data in the input file and saves the encrypted data in the output file. For temporary encryption, the data in the input file must be signed using sign_fw before it can be encrypted.
- sign_fw
  sign_fw is a signing tool. This tool reads the data in the input file and saves the signed data in the output file.

The following is an example of temporary encryption using the Encryption Tool.

On ${WORK}/meta-rzg2/recipes-bsp/arm-trusted-firmware/arm-trusted-firmware_git.bbappend

```
temp_encrypt() {
    ./tools/renesas/rzg_security_tools/sign_fw/sign_fw --key-alg rsa --hash-alg sha256 --align 16 \
        --key ${FILE_PATH_SIG_KEY} --in ${FILE_PATH_IN} --out ${FILE_PATH_SIG}

    ./tools/renesas/rzg_security_tools/encrypt_fw/encrypt_fw --key-alg cbc --nonce ${STR_ENC_IV0} \
        --key ${STR_ENC_KEY} --in ${FILE_PATH_SIG} --out ${FILE_PATH_OUT}
}
```

- FILE_PATH_IN
  — The path of the file from which the data to be temporarily encrypted is read.
- FILE_PATH_OUT
  The path to the file where the temporarily encrypted data will be stored.
- FILE_PATH_SIG
  — The path to the file where the signed data will be stored.
- FILE_PATH_SIG_KEY
  — The path of the private key used for signing. Specify the path of the “SBS-Key.pem” file generated by the Provisioning Tool.
- STR_ENC_ KEY
  — Specifies the value of the key used for encryption in hexadecimal text format. Use the top 16 bytes of the “E-Key.bin” file generated by the Provisioning Tool.
- STR_ENC_IV0
  — Specifies the value of the initialization vector used for encryption in hexadecimal text format. Use the lower 16 bytes of the “E-Key.bin” file generated by the Provisioning Tool.

5.5.4 Packaging Tool

The Packaging Tool is for packaging multiple firmwares that are written to flash memory. The packaged file is called the Firmware Image Package (hereinafter referred to as FIP).

The format of FIP implemented by TEE for RZ/G2 has modified from the specification defined by Trusted Firmware-A. Therefore, the Packaging Tool provided by TEE for RZ/G2 is a customized tool of fiptool included in the source code of Trusted Firmware-A. Refer to the Trusted Firmware-A documentation for the original specifications of the FIP format.

Trusted Firmware-A (BL2) provided by TEE for RZ/G2 does not support loading firmware from FIP. However, for compatibility with upcoming devices, the firmware is implemented in FIP format between the build environment and the user product environment. In TEE for RZ/G2, FIP brought into the user product environment are disassembled into individual firmware and written to flash memory.

Packaging Tool is stored in the following path.

[${WORK}/build/tmp/work/<work-sub-directories>/arm-trusted-firmware/<Properties-of-yocto-environment>/git/tools/renesas/rzg_security_tools/]

The configuration of Packaging Tool is shown below.

```
rzg_security_tools
└── fiptool
    ├── Makefile
    └── src
        ├── fiptool.c
        ├── fiptool.h
        ├── fiptool_platform.h
        ├── rzg_firmware_image_package.h
        ├── tbbr_config.c
        └── tbbr_config.h
```

Figure 5-7 Packaging Tool

The following is an example of build the Encryption Tool in Yocto build environment.

On ${WORK}/meta-rzg2/recipes-bsp/arm-trusted-firmware/arm-trusted-firmware_git.bbappend

```
do_compile_append() {
    oe_runmake -C tools/renesas/rzg_security_tools/fiptool clean
    oe_runmake -C tools/renesas/rzg_security_tools/fiptool
}
```

The following program is created by building the Packaging Tool.

- fiptool_fw_ipl
  This tool packages the firmware required to load BL31 and later firmware when the user product boots. For example, Trusted Firmware-A(BL2), Security Module, etc.
- fiptool_keyring
  This tool packages Temporary Encrypted Keyring. Temporarily Encrypted Keyring packaged by this tool can be re-encrypted by the Flash Writer described in section 5.5.5.
- fiptool_boot_fw
  This tool packages Temporarily Encrypted User Data. Temporarily Encrypted User Data packaged by this tool can be re-encrypted by the Flash Writer described in section 5.5.5.

The following is an example of packaging using the Packaging Tool.

On ${WORK}/meta-rzg2/recipes-bsp/arm-trusted-firmware/arm-trusted-firmware_git.bbappend

```
do_deploy_append() {
    ./tools/renesas/rzg_security_tools/fiptool/fiptool_fw_ipl create --align 16 \
        --tb-fw-cert  ${S}/tools/renesas/rzg_layout_create/bootparam_sa0.bin \
        --soc-fw-cert ${S}/tools/renesas/rzg_layout_create/cert_header_sa6.bin \
        --tb-fw      ${DEPLOYDIR}/bl2-${MACHINE}.bin \
        --sec-mod   ${DEPLOY_DIR_IMAGE}/sec_module-${MACHINE}.bin \
        ./tools/renesas/rzg_security_tools/fiptool/fip_fw_ipl.bin

    ./tools/renesas/rzg_security_tools/fiptool/fiptool_keyring create --align 16 \
        --key-ring   ${SYMLINK_NATIVE_BOOT_KEY_DIR}/Keyring_Enc.bin \
        ./tools/renesas/rzg_security_tools/fiptool/fip_keyring.bin

    ./tools/renesas/rzg_security_tools/fiptool/fiptool_boot_fw create --align 16 \
        --soc-fw    ${DEPLOYDIR}/bl31-${MACHINE}_Enc.bin   \
        --tos-fw     ${DEPLOYDIR}/tee-${MACHINE}_Enc.bin    \
        --nt-fw      ${DEPLOYDIR}/u-boot-${MACHINE}_Enc.bin \
        ./tools/renesas/rzg_security_tools/fiptool/fip_boot_fw.bin
}
```

In the default environment of TEE for RZ/G2, each tool packages the following firmware.

fiptool_fw_ipl:

- bootparam_sa0.bin
- cert_header_sa6.bin
- bl2-${MACHINE}.bin
- sec_module-${MACHINE}.bin

fiptool_keyring:

- Keyring_Enc.bin

fiptool_boot_fw:

- bl31-${MACHINE}_Enc.bin
- tee-${MACHINE}_Enc.bin
- u-boot-${MACHINE}_Enc.bin

The tool used to package the firmware is defined in the following file.

[${WORK}/build/tmp/work/<work-sub-directories>/arm-trusted-firmware/<Properties-of-yocto-environment>/git/tools/renesas/rzg_security_tools/fiptool/src/tbbr_config.c]

The Packaging Tool creates multiple FIPs. By combining these FIPs into one, they can be written together by the Flash Writer. When combining FIPs, it is necessary to combine them in the order of FIPs created by the following tools.

1. fiptool_fw_ipl > 2. fiptool_keyring > 3. fiptool_boot_fw

The following is an example of combining FIP files.

On ${WORK}/meta-rzg2/recipes-bsp/arm-trusted-firmware/arm-trusted-firmware_git.bbappend

```
do_deploy_append() {

    cat ./tools/renesas/rzg_security_tools/fiptool/fip_fw_ipl.bin > \
        ./tools/renesas/rzg_security_tools/fiptool/fips-${MACHINE}.bin
    cat ./tools/renesas/rzg_security_tools/fiptool/fip_keyring.bin >> \
        ./tools/renesas/rzg_security_tools/fiptool/fips-${MACHINE}.bin
    cat ./tools/renesas/rzg_security_tools/fiptool/fip_boot_fw.bin >> \
        ./tools/renesas/rzg_security_tools/fiptool/fips-${MACHINE}.bin
}
```

When writing FIPs individually using the flash writer, it is necessary to write the FIPs in the same order as when combining them.

5.5.5 Flash Writer

The Flash Writer is used to write the firmware to the flash memory mounted on the RZ/G2 evaluation board.

In TEE for RZ/G2, the Flash Writer provided by RZ/G2 Linux BSP is customized for Provisioning as follows.

Add the following functions.

- Loading FIP
- Re-encryption of Keyring
- Re-encryption of User Data

The Flash Writer implements these functions in the following files.

[${WORK}/build/tmp/work/<work-sub-directories>/flash-writer/<Properties-of-yocto-environment>/git/fiploader.c]

For information about to how to load FIP using the flash writer, refer to “Related Documents No.6”.

5.5.5.1 Loading FIP

This is a function to load the FIP packaged in Yocto build environment into the user product environment using serial communication. FIP brought into the user product environment are disassembled into individual firmware and written to flash memory.

5.5.5.2 Re-encryption of Keyring

This is a function to re-encrypt Temporarily Encrypted Keyring packaged by fiptool_keyring. Re-Encrypted Keyring is written to flash memory.

The Flash Writer calls Security Module for Re-encryption of Keyring. Therefore, Security Module packaged by fiptool_fw_ipl must be preloaded and written to flash memory in advance.

5.5.5.3 Re-encryption of User Data

This is a function to re-encrypt Temporarily Encrypted User Data packaged by fiptool_boot_fw. Re-Encrypted User Data is written to flash memory.

The Flash Writer calls Security Module for Re-encryption of User Data. Also, Re-Encrypted Keyring is required for Re-encryption of User Data. Therefore, Security Module and Re-Encrypted Keyring must be written to flash memory in advance.

5.6 Key Wrap Service

Provisioning Key (ProvisioningKey.bin) and Encrypted Provisioning Key (ProvisioningKey_Enc.bin) included in Secure IP Package is sample keys. Provisioning Key and Encrypted Provisioning Key for mass-produced products must be created in the customer environment.

For information about how to generate Provisioning Key, refer to “5.5.2. Provisioning Tool”. Encrypted Provisioning Key is created by using the Key Wrap Service.

The following shows the procedure for creating Encrypted Provisioning Key.

1. The customer sends Provisioning Key to Key Wrap Service.
2. Key Wrap Service encrypts the received Provisioning Key with the Hidden Root Key.
3. Key Wrap Service sends Encrypted Provisioning Key to the customer.

Sending and receiving keys to and from the Key Wrap Service are performed with PGP encryption.

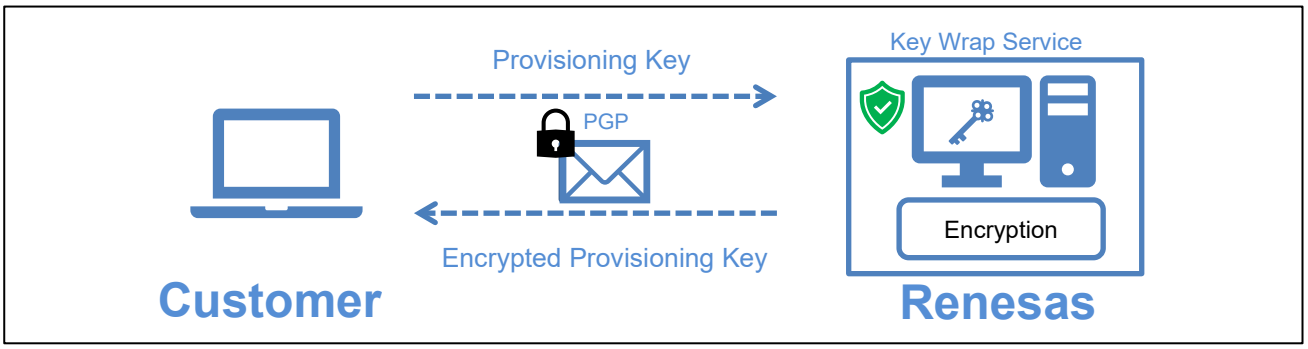

Figure 5-8 Key Wrap Service

Provisioning Key for evaluation can also be created by Key Wrap Service. In this case, the Key Wrap Service sends the customer Provisioning Key and Encrypted Provisioning Key.

Key Wrap Service is provided on following URL. For detail, refer to FAQ on Key Wrap Service site.

https://dlm.renesas.com/

5.7 Security Module

Security Module is software for using TSIP included with RZ/G2 Group processor. By accessing the TSIP, Security Module provides features related to Secure Boot.

In TEE for RZ/G2, Security Module is called from the Flash Writer and Trusted Firmware-A(BL2). The Flash Writer calls Security Module for Re-encryption of Keyring and Re-encryption of User Data. Trusted Firmware-A(BL2) calls Security Module for Verification of Keyring and Decryption and Verification of User Data.

The following table shows the Security Module functions.

Table 5-5 Security Module Functions

No	Functions	Explanation
1	Initialization of TSIP	Reset the TSIP status.
2	Re-encryption of Keyring	Re-encrypt Temporarily Encrypted Keyring.
3	Re-encryption of User Data	Re-encrypt Temporarily Encrypted User Data.
4	Verification of Keyring	Verify Re-Encrypted Keyring.
5	Decryption and Verification of User Data	Decrypt and verify Re-Encrypted User Data.

5.7.1 Directory Configuration

In Yocto build environment, the source code of Security Module is stored in the following path.

[${WORK}/build/tmp/work/<work-sub-directories>/secmod/<Properties-of-yocto-environment>/git]

The directory configuration of Security Module is shown below.

```
git
├── LICENSE.txt
├── makefile
├── README.md
├── sec_module.c
├── sec_module.h
├── sec_module.ld.S
└── tsip
    ├── common
    │   └── TSIP_Common_Define.h
    ├── core
    │   ├── TSIP_Core_API.h
    │   ├── TSIP_Core_Boot_API.c
    │   ├── TSIP_Core_Init_API.c
    │   ├── TSIP_Core_KeyRingVerify_API.c
    │   ├── TSIP_Core_Local_API.c
    │   ├── TSIP_Core_Local_API.h
    │   ├── TSIP_Core_Prepare_API.c
    │   ├── TSIP_Core_Proc_API.c
    │   └── TSIP_Core_Proc_API.h
    ├── proc
    │   └── TSIP_Procedure.h
    ├── R_TSIP_Boot_Lib.h
    ├── stub
    │   ├── TSIP_Driver.h
    │   ├── TSIP_Driver_nonos.c
    │   ├── TSIP_Stub_API.h
    │   └── TSIP_Stub_API_nonos.c
    └── wrapper
        ├── TSIP_Wrapper_Boot_API.c
        ├── TSIP_Wrapper_Boot_API.h
        ├── TSIP_Wrapper_Init_API.c
        ├── TSIP_Wrapper_Init_API.h
        ├── TSIP_Wrapper_KeyRingVerify_API.c
        ├── TSIP_Wrapper_Local_API.c
        ├── TSIP_Wrapper_Local_API.h
        ├── TSIP_Wrapper_Prepare_API.c
        └── TSIP_Wrapper_Prepare_API.h
```

Figure 5-9 Security Module

5.7.2 External Interface

This section describes the external interface of Security Module.

The external interface of Security Module is declared in the following header files.

[${WORK}/build/tmp/work/<work-sub-directories>/secmod/<Properties-of-yocto-environment>/git/sec_module.h]

Security Module has a shared area with external software in the area of 0x200 bytes from the load address. The Entry point of Security Module is placed immediately after this shared area.

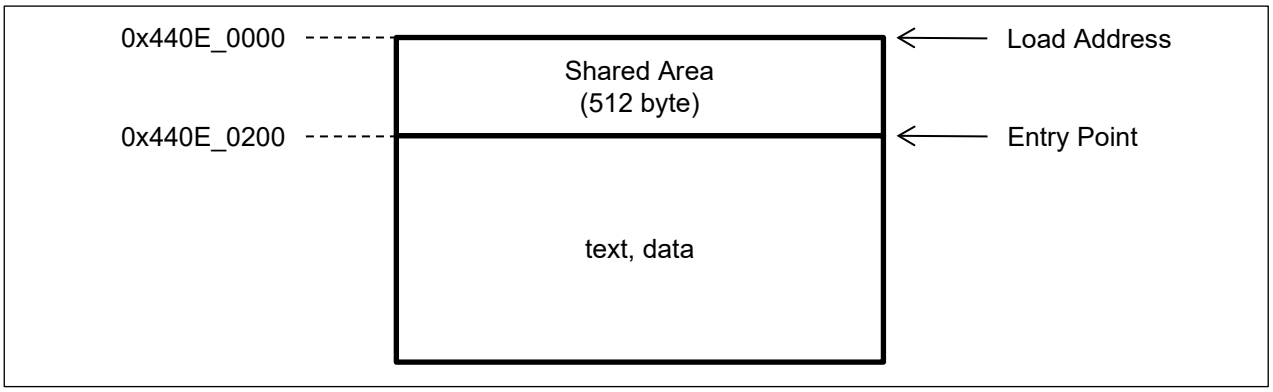

Figure 5-10 Security Module Memory Map

The Security Module defines commands for each function. To call Security Module, external software stores the parameters in the shared area according to the command, and then jumps to the entry point.

5.7.2.1 Commands

This section shows the commands defined by Security Module.

(a) CMD_RESET_INIT

CMD_RESET_INIT	Security Module
Initialization of TSIP	

Parameter	st_sec_module_arg_t structure format cmd: CMD_RESET_INIT len: sizeof(st_reset_init_t) prm: st_reset_init_t structure variable
Return Value	• SEC_MODULE_RET_OK: Success • SEC_MODULE_RET_ERROR_INIT: Hardware error • SEC_MODULE_RET_ERROR_PARAMETER: Invalid parameter • SEC_MODULE_RET_ERROR_FAIL: Initialization failure Resource Conflict
Description	This command executes the "Initialization of TSIP". This command must be executed before the commands described below. For the execution procedure of this command, refer to the command described below.

(b) CMD_INJECT_KEY

CMD_INJECT_KEY	Security Module
Re-encryption of Keyring	

Parameter

st_sec_module_arg_t structure format
- cmd: CMD_INJECT_KEY
- len: sizeof(st_inject_key_t)
- prm: st_inject_key_t structure variable

Return Value

- SEC_MODULE_RET_OK:
  Success
- SEC_MODULE_RET_ERROR_PARAMETER:
  Invalid parameter
- SEC_MODULE_RET_ERROR_FAIL:
  Resource Conflict
  Illegal command execution procedure
  Abnormal Temporary Encrypted Keyring or Provisioning Key

Describe

This command executes the "Re-Encryption of Keyring".
Execute this command after the "Initialization of TSIP". Even if this command is executed repeatedly, execute it from the “initialization of TSIP”.

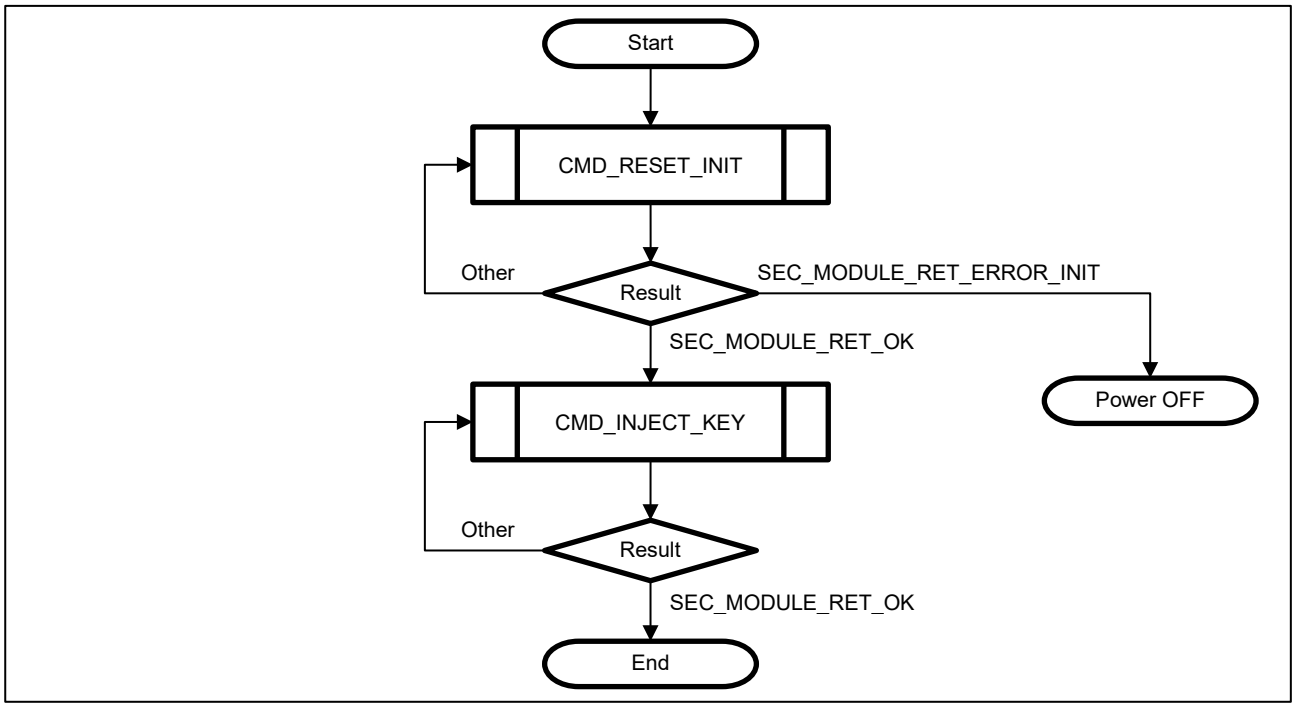

Figure 5-11 Re-encryption of Keyring

(c) CMD_RE_ENC_DATA

CMD_RE_ENC_DATA	Security Module
Re-encryption of User Data	

Parameter

st_sec_module_arg_t structure format
- cmd: CMD_RE_ENC_DATA
- len: sizeof(st_re_enc_data_t)
- prm: st_re_enc_data_t structure variable

Return Value

- SEC_MODULE_RET_OK:
  Success
- SEC_MODULE_RET_ERROR_PARAMETER:
  Invalid parameter
- SEC_MODULE_RET_ERROR_FAIL:
  Resource Conflict
  Illegal command execution procedure
  Abnormal Temporary Encrypted User Data
  Abnormal Re-Encrypted Keyring

Description

This command executes the "Re-Encryption of User Data".
Execute this command after the "Initialization of TSIP". Even if this command is executed repeatedly, execute it from the “initialization of TSIP”.
All User Data to be verified by Secure Boot, must be re-encrypted all at once. Therefore, set the information for All User Data to be verified by Secure Boot to the parameters of this command.

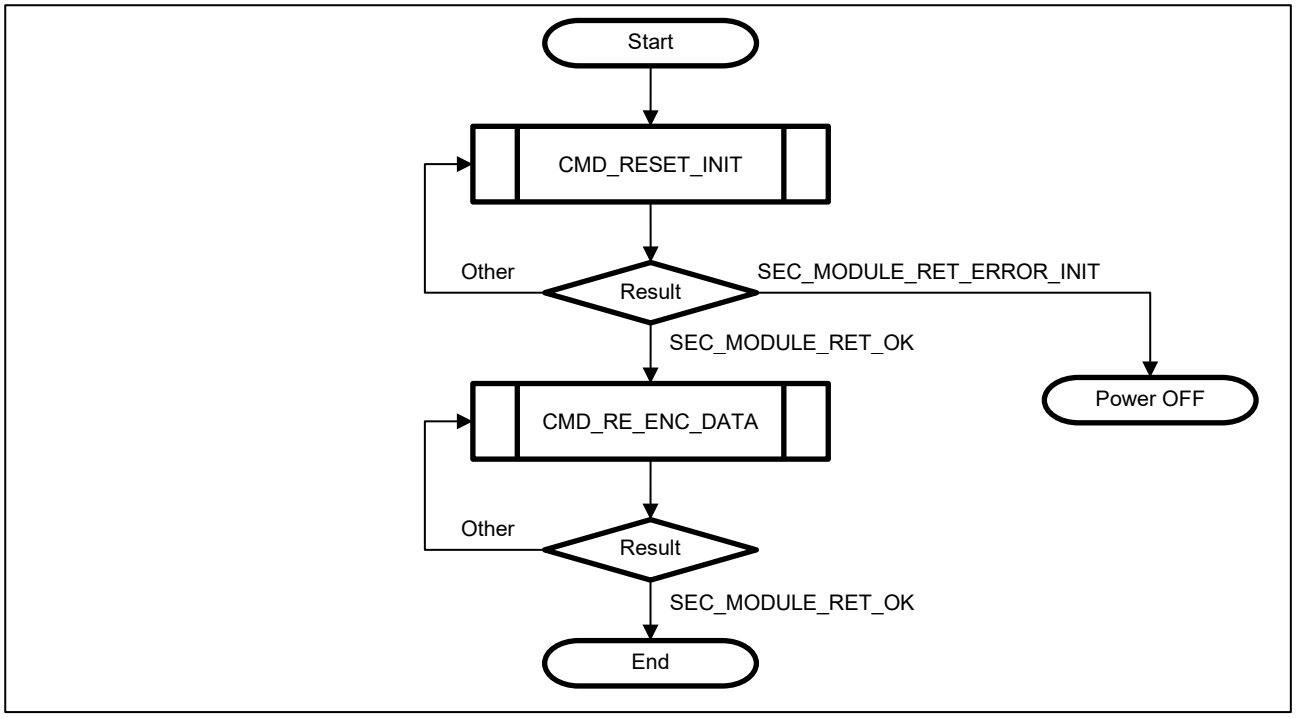

Figure 5-12 Re-encryption of User Data

(d) CMD_VERIFY_DATA

CMD_VERIFY_DATA	Security Module
Verification of Keyring / Decryption and Verification of User Data	

Parameter

st_sec_module_arg_t structure format
- cmd: CMD_VERIFY_DATA
- len: sizeof(st_verify_data_t)
- prm: st_verify_data_t structure variable

Return Value

- SEC_MODULE_RET_OK:
  Success
- SEC_MODULE_RET_ERROR_PARAMETER:
  Invalid parameter
- SEC_MODULE_RET_ERROR_VERIFICATION:
  Failed to verify Re-Encrypted Keyring or Re-Encrypted User Data
- SEC_MODULE_RET_ERROR_FAIL:
  Resource Conflict
  Illegal command execution procedure

Description

This command executes the " Verification of Keyring" and "Decryption and Verification of User Data".
Execute this command after the "Initialization of TSIP". Even if this command is executed repeatedly, execute it from the "initialization of TSIP".
To complete Secure Boot, all User Data encrypted by the CMD_RE_ENC_DATA command must be verified using this command.

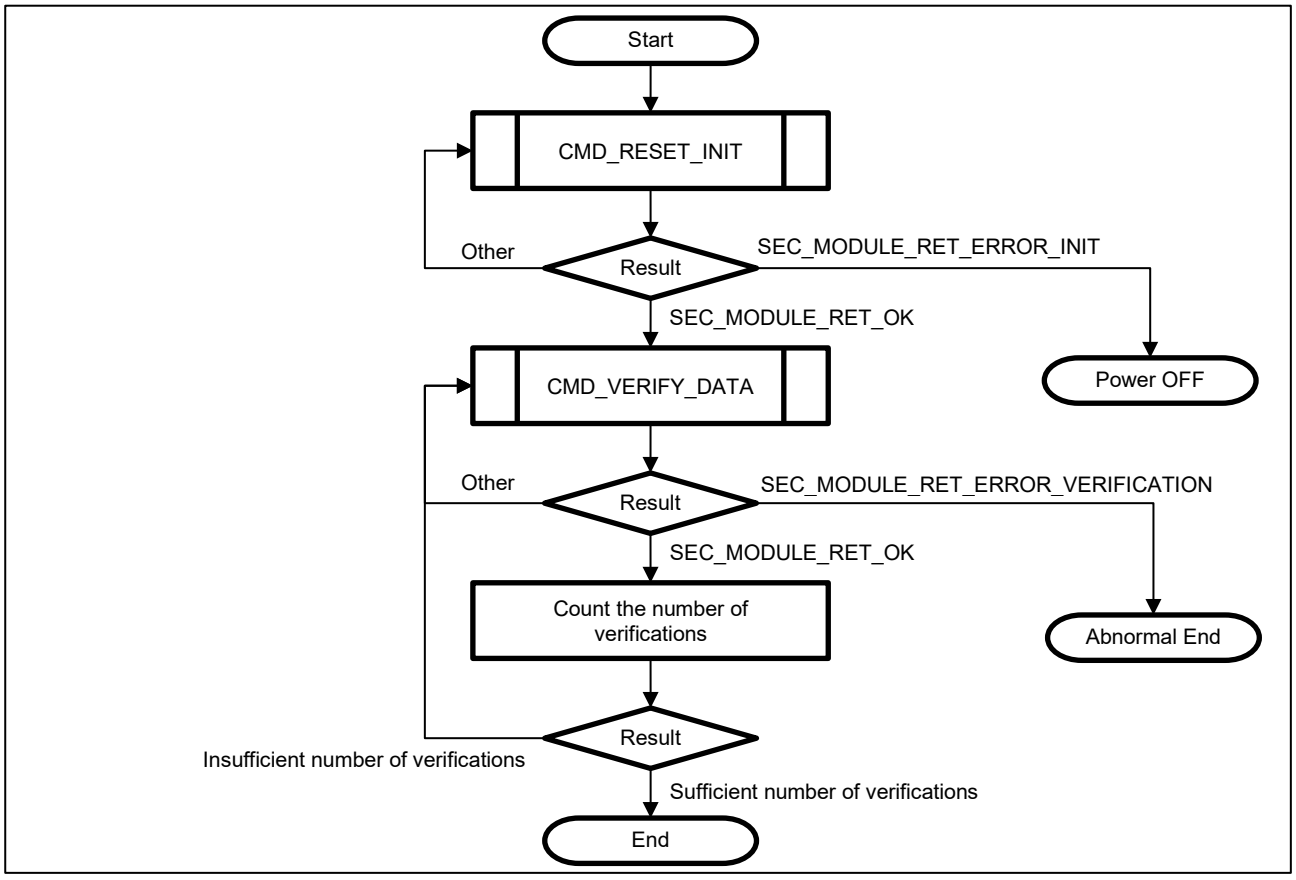

Figure 5-13 Verification of Encrypted Data

5.7.2.2 Structures

This section shows the structures defined by Security Module.

(a) st_reset_init_t

Format

```
typedef struct {
  uint64_t inst_area;
} st_reset_init_t;
```

Member

Member Name	IN/OUT	Description
inst_area	IN/OUT	Address of the area where the Re-Encrypted Keyring is placed. 1296 bytes required.

Description

This is a structure of parameters required to execute the CMD_RESET_INIT command.

The usage of the area specified in "inst_area" differs depending on the commands executed after this command. The usage of this area is shown below.

- Execute the CMD_INJECT_KEY command
  Re-Encrypted Keyring is output to the area specified in "inst_area". Read Re-Encrypted Keyring from the area specified in "inst_area" and store it in the non-volatile memory.
- Execute other than the CMD_INJECT_KEY command
  Re-Encryption Keyring is read from the area specified in "inst_area". This Re-Encrypted Keyring is used to decrypt and verify User Data. Therefore, Re-Encrypted Keyring must be placed in this area before executing the command.

(b) st_inject_key_t

Format

```
typedef struct {
  uint64_t key_ring;
  uint64_t prov_key;
} st_inject_key_t;
```

Member

Member Name	IN/OUT	Description
key_ring	IN	Address of the area where Temporarily Encrypted Keyring is stored
prov_key	IN	Address of the area where Encrypted Provisioning Key is stored

Description

This is a structure of parameters required to execute the CMD_INJECT_KEY command.

Temporarily Encrypted Keyring is read from the area specified in "key_ring" and re-encrypted with Device-Specific Key. Re-Encrypted Keyring is output to the area specified in "inst_area" of the CMD_RESET_INIT command parameter.

(c) st_re_enc_data_t

Format

```
typedef struct {
  int num;
  struct {
    uint64_t src;
    uint64_t len;
    uint64_t dst;
  } list[16];
} st_re_enc_data_t;
```

Member

Member Name	IN/OUT	Description
num	IN	Number of User Data to be re-encrypted.
list[n].src	IN	Address of the area where Temporarily Encrypted User Data is stored.
list[n].len	IN	Size of Temporarily Encrypted User Data.
list[n].dst	OUT	Address of the area where Re-Encrypted User Data is output.

Description

This is a structure of parameters required to execute the CMD_RE_ENC_DATA command.

Temporarily Encrypted User Data is read from the area specified in "list[n].src" and re-encrypted with Device-Specific Key. Re-Encrypted User Data is output to the area specified in "list[n].dst".

The maximum number of User Data that can be re-encrypted is 16. Set the number of User Data to be re-encrypted in "num", and set the parameters of User Data to be re-encrypted in order from the beginning of the List array. The order of User Data set in the List array must be the same as the order of User Data to be verified by Secure Boot.

The size of the area specified in "list[n].dst" is as follows.

- Size of “list[0].dst” = “list[0].len” + 64 bytes
- Size of “list[1-15].dst” = “list[1-15].len” + 16 bytes

(d) st_verify_data_t

Format

```
typedef struct {
  uint64_t src;
  uint64_t len;
  uint64_t dst;
  uint64_t heap;
} st_verify_data_t;
```

Member

Member Name	IN/OUT	Description
src	IN	Address of the area where Re-Encrypted User Data is stored.
len	IN	Size of Re-Encrypted User Data.
dst	OUT	Address of the area where decrypted Data is output.
heap	OUT	Address of heap area used for decryption / verification of Re-Encrypted User Data.

Description

This is a structure of parameters required to execute the CMD_VERIFY_DATA command.

Re-Encrypted User Data is read from the area specified in “src”, decrypted and verified. The decrypted User Data is output to the area specified in “dst”.

The size of the area specified in “dst” and “heap” are as follows.

- Size of “dst” = “len” - 320 bytes
- Size of “heap” = “len” - 272 bytes

(e) st_sec_module_arg_t

Format

```
typedef struct {
  uint64_t cmd;
  uint64_t len;
  union {
    st_reset_init_t reset_init;
    st_inject_key_t inject_key;
    st_re_enc_data_t re_enc_data;
    st_verify_data_t verify_data;
  } prm;
} st_sec_module_arg_t;
```

Member

Member Name	IN/OUT	Description
cmd	IN	Command number.
len	IN	Size of the command structure.
prm	IN	Area where the command structure is stored.

Description

This is a structure of parameters stored in the shared area of Security Module. This structure stores parameters depending on the function executed in Security Module.

The following table shows the parameters to set according to the command.

Table 5-6 Command parameters

Functions	Parameter		
	cmd	len	prm
Initialization of TSIP	CMD_RESET_INIT	sizeof(st_reset_init_t)	st_reset_init_t structure
Re-encryption of Keyring	CMD_INJECT_KEY	sizeof(st_inject_key_t)	st_inject_key_t structure
Re-encryption of User Data	CMD_RE_ENC_DATA	sizeof(st_re_enc_data_t)	st_re_enc_data_t structure
Verification of Keyring / Decryption and Verification of User Data	CMD_VERIFY_DATA	sizeof(st_verify_data_t)	st_verify_data_t structure

5.7.3 Execution Example

The following is an execution example when calling Security Module from external software.

```
/* Calculate the entry point address */
void* shared_area = SEC_MODULE_BASE;
fp_sec_module_api_t ep = SEC_MODULE_BASE + SEC_MODULE_SHARED_SIZE;

/* Set the command to be executed in the shared area */
st_sec_module_arg_t *args = (st_sec_module_arg_t *)shared_area;
args->cmd = CMD_RESET_INIT;
args->len = sizeof(st_reset_init_t);

/* Set the parameters for initialization command */
st_reset_init_t *reset_init = &(args->prm);
reset_init->inst_area = SEC_KEYRING_BASE;

/* Call the entry point of Security Module */
if(SEC_MODULE_RET_OK != ep())
{
   ERROR("Security Module initialization failed");
}
```

Figure 5-14 Example of CMD_RESET_INIT command

- SEC_MODULE_BASE
  — Address of the area where Security Module is loaded.
- SEC_MODULE_SHARED_SIZE
  — Size of the shared area.
- SEC_KEYRING_BASE
  — Address of the area where Re-Encrypted Keyring is loaded.

In TEE for RZ/G2, Security Module is called from the following files.

- Trusted Firmware-A (BL2)
  [${WORK}/build/tmp/work/<work-sub-directories>/arm-trusted-firmware/<Properties-of-yocto-environment>/git/drivers/renesas/rzg/auth/auth_mod.c]
- Flash Writer
  [${WORK}/build/tmp/work/<work-sub-directories>/flash-writer/<Properties-of-yocto-environment>/git/fiploader.c]

6. Memory Map

This chapter describes the memory map when Secure Boot is implemented. For the default memory map of RZ/G2 Linux BSP, refer to “Related Documents No.2”.

Secure Boot implementation adds Security Module and Re-Encrypted Keyring to the default memory map. The address where Security Module and Re-Encrypted Keyring are located is the same for RZ/G2E|G2M|G2N|G2H devices. Trusted Firmware-A(BL31), OP-TEE OS and U-Boot are re-encrypted before being placed in flash memory.

Decryption and validation by Secure Boot use the heap area (Secure Boot Heap). The size of this heap area depends on the size of Re-Encrypted User Data. Therefore, this heap area must be allocated according to the size of the largest Re-Encrypted User Data. For more information on the size of the heap area, refer to “5.7.2.2. Structures”.

The following is a memory map of TEE for RZ/G2 built on the RZ/G2E System Evaluation Board EK874 using RZ/G2 Linux BSP.

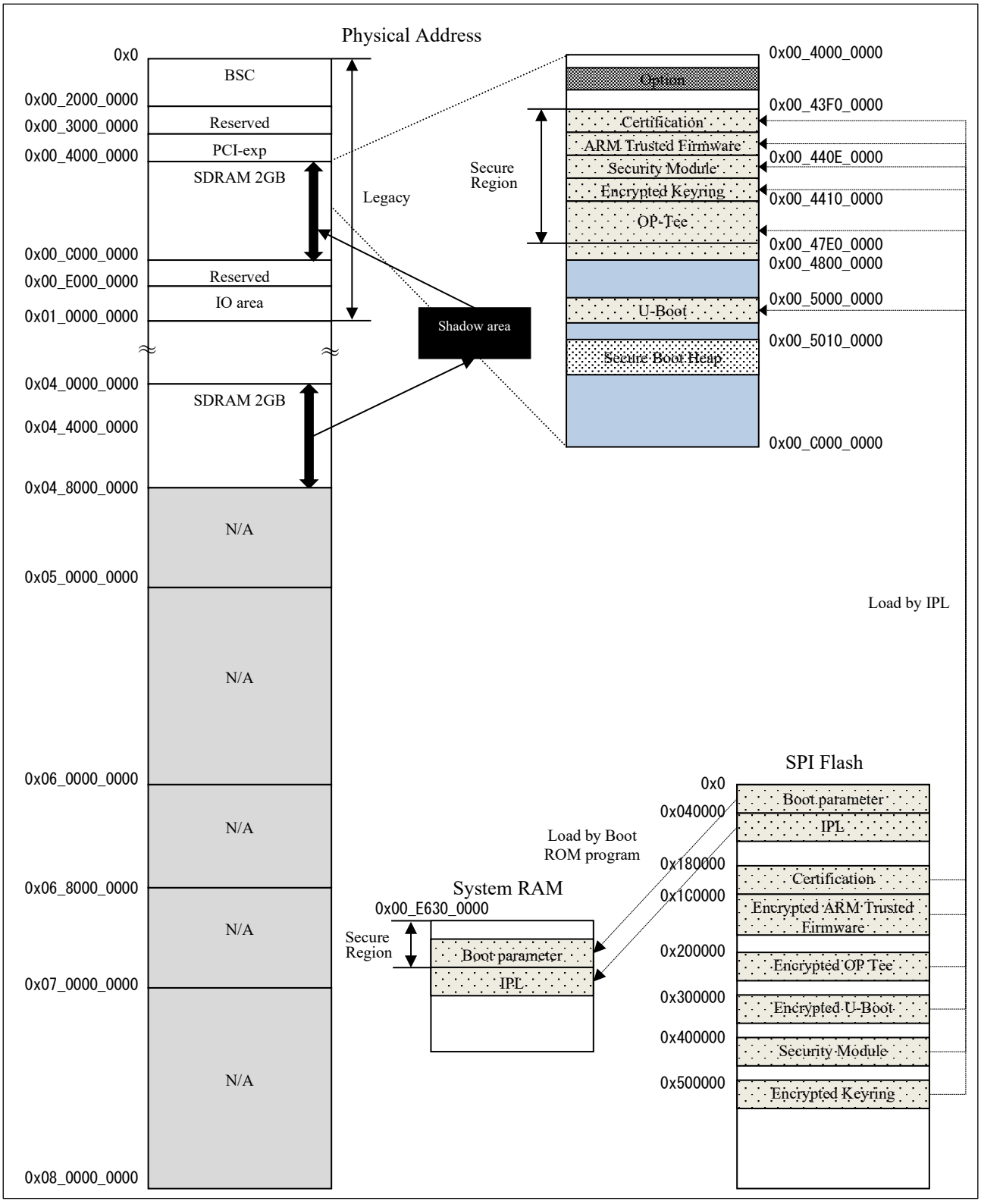

Figure 6-1 RZ/G2E System Evaluation Board EK874 memory map (Boot)

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Mar. 31, 2021	-	First Release

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)
   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on
   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state
   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins
   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals
   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin
   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses
   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products
   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

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

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/.