Renesas RA Family

Installing and Utilizing the Cryptographic User Keys using SCE9

Introduction

Cryptography is important because it provides the tools to implement solutions for authenticity, confidentiality, and integrity, which are vital aspects of any security solution. In modern cryptographic systems, the security of the system no longer depends on the secrecy of the algorithm used but rather on the secrecy of the keys. Renesas RA Family cryptographic key installation provides several options for minimizing the exposure of the user keys and providing optimal production and field key management support.

This application project explains these key installation methods and provides examples for the currently available methods supported via the Renesas Flexible Software Package.

An AES plaintext key installation example is provided for the Renesas RA Family RA6M4 MCU group.

Required Resources

Development tools and software

- The e^2 studio ISDE v2020-10 or greater
- Renesas Flexible Software Package (FSP) v2.2.0 or later
- SEGGER J-link® USB driver

The above three software components: the FSP, J-Link USB drivers and e2 studio are bundled in a downloadable platform installer available on the FSP webpage at renesas.com/ra/fsp.

Hardware

- EK-RA6M4, Evaluation Kit for RA6M4 MCU Group (http://www.renesas.com/ra/ek-ra6m4)
- Workstation running Windows® 10 and Tera Term console, or similar application
- Two USB device cables (type-A male to micro-B male)

Prerequisites and Intended Audience

This application note assumes you have some experience with the Renesas e^2 studio IDE and Arm® TrustZone® based development models with e^2 studio. In addition, the application note assumes that you have some knowledge of RA Family MCU security features. See chapter 49, Security Features in the *Renesas RA6M4 Group MCU User's Manual: Hardware* for background knowledge preparation for the cryptographic key installation.

The intended audience are product developers, product manufacturers, product support, or end users who are involved with any stage of the MCU user key management of the RA Family MCUs with Arm TrustZone.

Contents

1. Root of Trust and its Protection

1.1 What is Root of Trust

Roots of trust are highly-reliable hardware, firmware, and software components that perform specific, critical security functions (https://csrc.nist.gov/projects/hardware-roots-of-trust). In an IoT system, a root of trust typically consists of identity and cryptographic keys rooted in the hardware of a device. It establishes a unique, immutable, and unclonable identity to authorize a device in the IoT network.

- Secure boot is part of the services provided in the Root of Trust in many security systems. Authentication of the application utilizes Public Key Encryption. The associated keys are part of the Root of Trust of the system.
- Device Identity, which consists of Device Private Key and Device Certificate, is part of the Root of Trust for many IoT devices.

1.2 Protecting the Root of Trust

From the above Root of Trust discussion, we can realize that leakage of the cryptographic user keys can bring the secure system into a risky state. Protection of the Root of Trust involves key accessibility within the cryptographic boundary only and keys that are unclonable. Root of Trust should be locked from read and write access from unauthorized parties.

Renesas user key management system can provide all the above desired protection. In addition, Renesas user key installation services provide several options from which user can pick and choose the installation methods which fits their existing architecture.

2. Introduction to SCE9 and Associated Keys

Next generation RA Family MCUs expand Renesas' strong security offerings with the new SCE9 secure crypto engine and authenticated Device Lifecycle Management (DLM). This application project focuses on the security feature brought by SEC9. For the security features brought by DLM, please reference the application note Renesas Device Lifecycle Management Key Installation for details.

2.1 Secure Crypto Engine (SCE9)

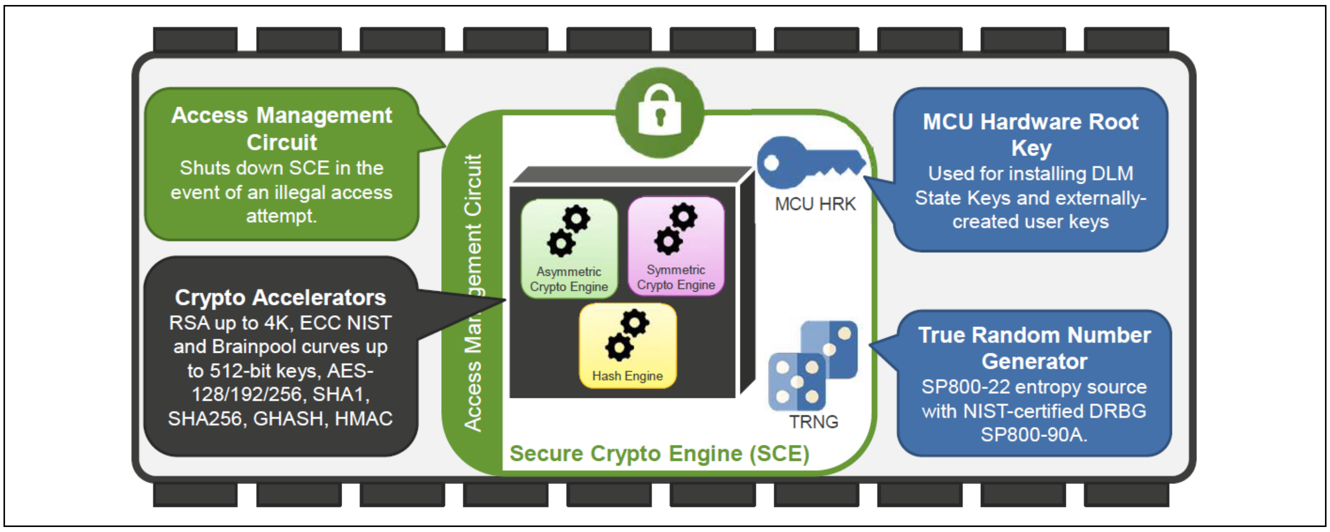

Access Management Circuit
Shuts down SCE in the event of an illegal access attempt.
Crypto Accelerators
RSA up to 4K, ECC NIST and Brainpool curves up to 512-bit keys, AES-128/192/256, SHA1, SHA256, GHASH, HMAC
Access Management Circuit
Asymmetric Crypto Engine
Symmetric Crypto Engine
Hash Engine
MCU HRK
TRNG
Secure Crypto Engine (SCE)
MCU Hardware Root Key
Used for installing DLM State Keys and externally-created user keys
True Random Number Generator
SP800-22 entropy source with NIST-certified DRBG SP800-90A.

Figure 1. Secure Crypto Engine 9

The Secure Crypto Engine 9 is an isolated subsystem within the MCU.

- The crypto engine contains hardware accelerators for both symmetric and asymmetric cryptographic algorithms, as well as various hashes and message authentication codes.
- It also contains a True Random Number Generator, providing an entropy source for the cryptographic operations.
- The Secure Crypto Engine is protected by an Access Management Circuit, which shuts down the crypto engine in the event of an illegal external access attempt.

2.2 SCE9 Associated Keys

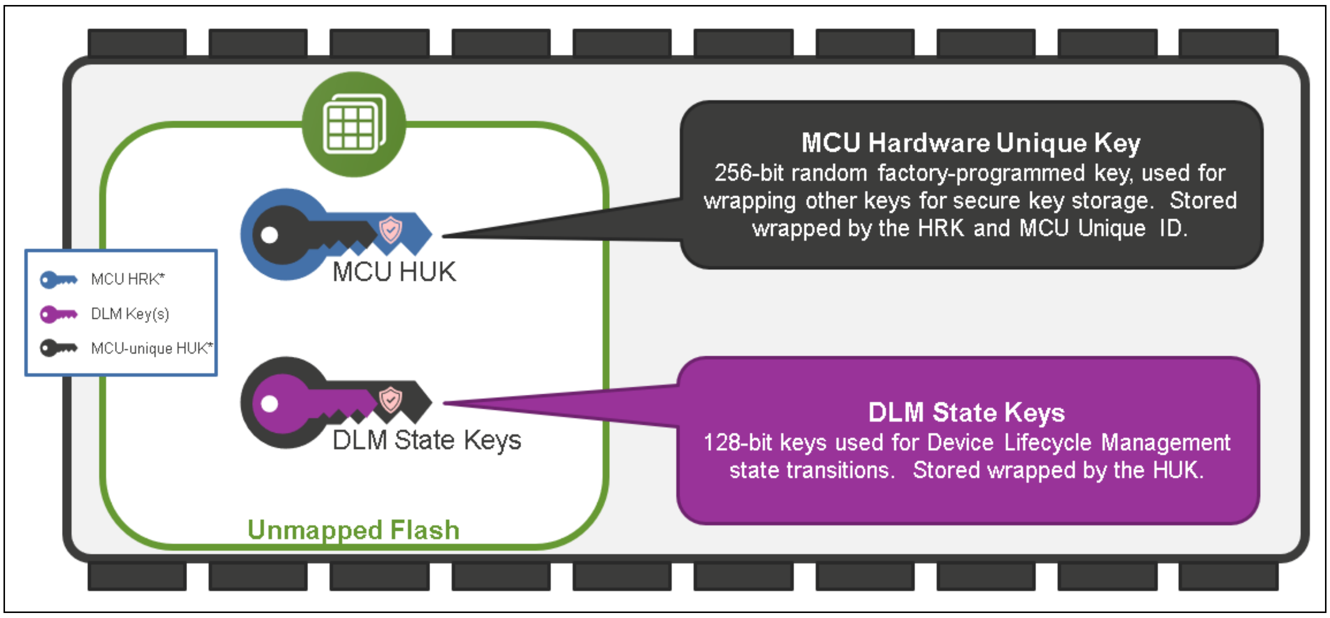

MCU Hardware Unique Key
256-bit random factory-programmed key, used for wrapping other keys for secure key storage. Stored wrapped by the HRK and MCU Unique ID.
MCU HUK
MCU HRK*
DLM Key(s)
MCU-unique HUK*
DLM State Keys
DLM State Keys
128-bit keys used for Device Lifecycle Management state transitions. Stored wrapped by the HUK.
Unmapped Flash

Figure 2. Security Keys

RA Family MCUs with the SCE9 crypto engine is associated with two new types of keys compared with SCE7.

- The first is an MCU-unique Hardware Unique Key (HUK), a 256-bit random key that is preprogrammed in the Renesas factory.
  - This key is stored in unmapped flash, assessable only by the SCE9, not to application code. It is further protected by being stored not in plaintext, but rather wrapped by the HRK (Hardware Root Key) and MCU unique ID.
  - The SCE9 can access this HUK to perform user key wrapping.
  - Since the HUK is stored in an MCU-uniquely wrapped format, even if an attacker was able to extract the stored key, another MCU won't be able to use it.
- The second type of keys are associated with the Renesas Device Lifecycle Management (DLM) system.
  - For the installation and usage of the DLM Keys, please reference application note Renesas Device Lifecycle Management Key Installation.

The HRK and HUK are used in the user key Installation process.

3. Cryptographic User Key Installation

The table below summarizes the key types that can be installed into Renesas RA MCUs with the SCE9 Secure Crypto Engine, for example, RA6M4 and RA4M3. Installed keys will be stored wrapped by the MCU's HUK.

Table 1 Supported Key Types

Lifecycle Transition Keys	SECDBG_KEY, NONSECDBG_KEY, RMA_KEY
AES	AES-128, AES-192, AES-256
RSA	RSA-1024, RSA-2048, RSA-3072, RSA-4096 (Public and Private)
ECC	NIST P-192, P-224, P-256, and P-384 Brainpool P256r1, P384r1, and P512r1 (Public and Private)
HMAC	HMAC-SHA224, HMAC-SHA256

3.1 Plaintext User Key Installation Features

Plaintext user key refers to the fact that the user keys can be provided in plaintext format to SCE9. When the plaintext key is installed, the SCE9 wraps the plaintext key with HUK and provides the wrapped key outside SCE9 for storage.

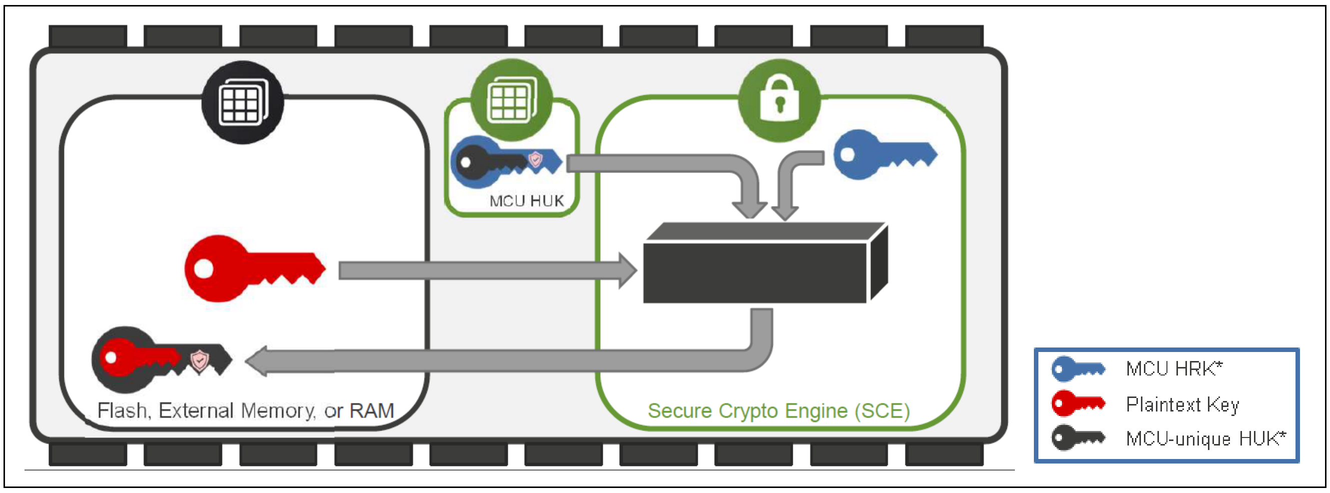


Figure 3. Plaintext Key Installation

This plaintext key installation process gives all security control of the keys to the product developer, which enables the developer to benefit from any existing secure key provisioning infrastructure. However, we do not recommend long-term storage of plaintext keys on the MCU. Therefore, the RA Family MCUs have the capability to install and securely store a plaintext key in wrapped format by wrapping the key with the MCU HUK.

Plaintext key installation is supported with FSP 2.0.0 or later. How to get the plaintext user key into the MCU RAM or flash in preparation for installation is out of scope for this application project. Product developers can use their existing infrastructure to interface to the MCU based on their specific environment.

Note: This plaintext key installation procedure is recommended to be performed in a secure environment.

3.2 Key Wrapping with SCE9

Key wrapping with SCE9 involves encryption using the MAC of the user key and MCU unique ID encrypted with the HUK. The encryption aspect provides confidentiality of the key. Wrapping with MAC code adds integrity and authenticity. Finally, wrapping with the MCU HUK adds clone protection.

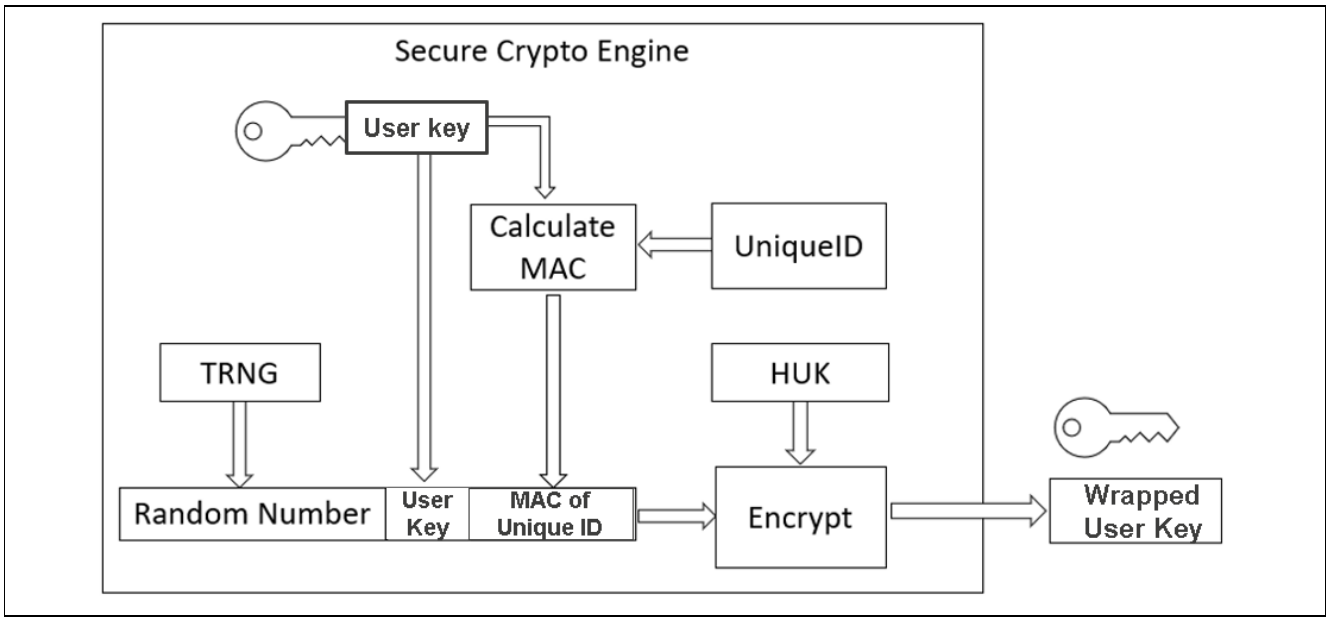


Figure 4. Key Wrapping with SCE9

3.3 Advantages of Key Wrapping

3.3.1 Advantages of Key Wrapping over Key Encryption

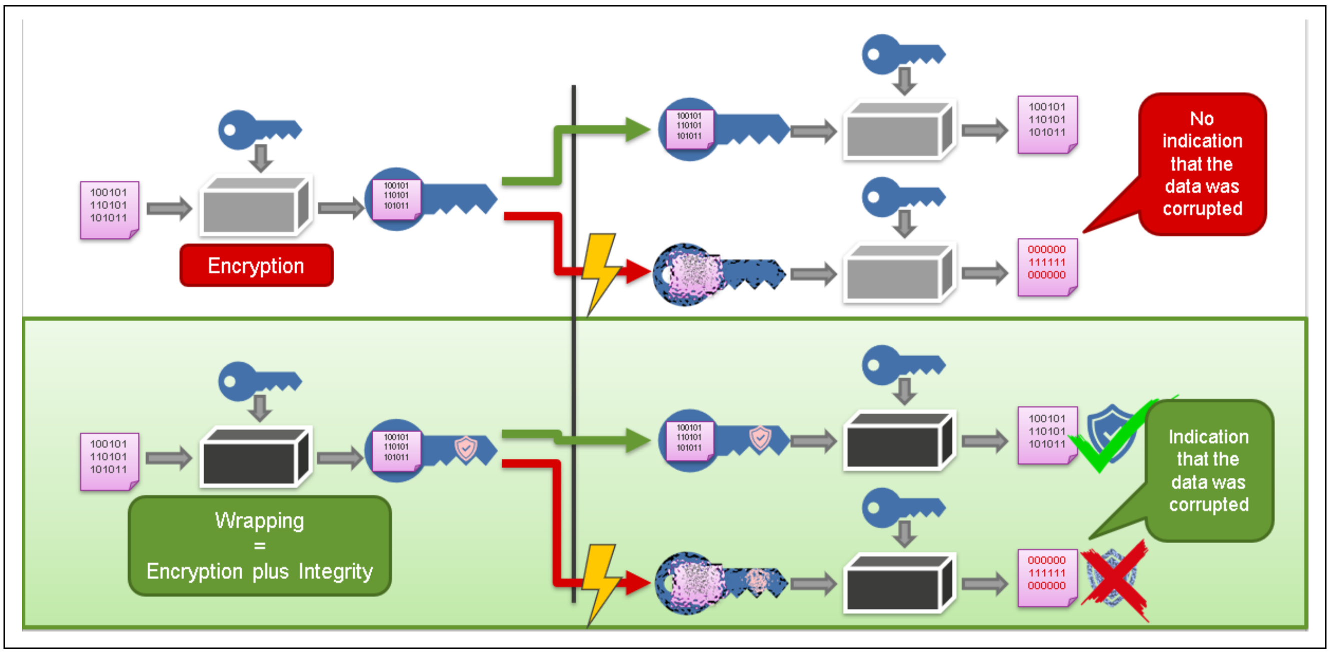

100101
110101
101011
Encryption
No indication that the data was corrupted
000000
111111
000000
Wrapping
=
Encryption plus Integrity
Indication that the data was corrupted

Figure 5. Key Wrapping vs. Key Encryption

It is important to understand the difference between wrapping and encrypting for secure asset storage. We will use symmetric encryption here to demonstrate.

- When data is encrypted and sent to another recipient, if that recipient has the same key, they can decrypt the data. This results in a confidential exchange of information. However, what if there was a problem with the transmission of the encrypted data? If the recipient unknowingly receives corrupted information, the decryption algorithm will generate garbage data, with no indication that the original data has been corrupted.
- Wrapping solves this problem for us by adding an integrity checking mechanism (as shown in Figure 4) to the encrypted output.

3.3.2 Advantages of Key Wrapping using MCU HUK

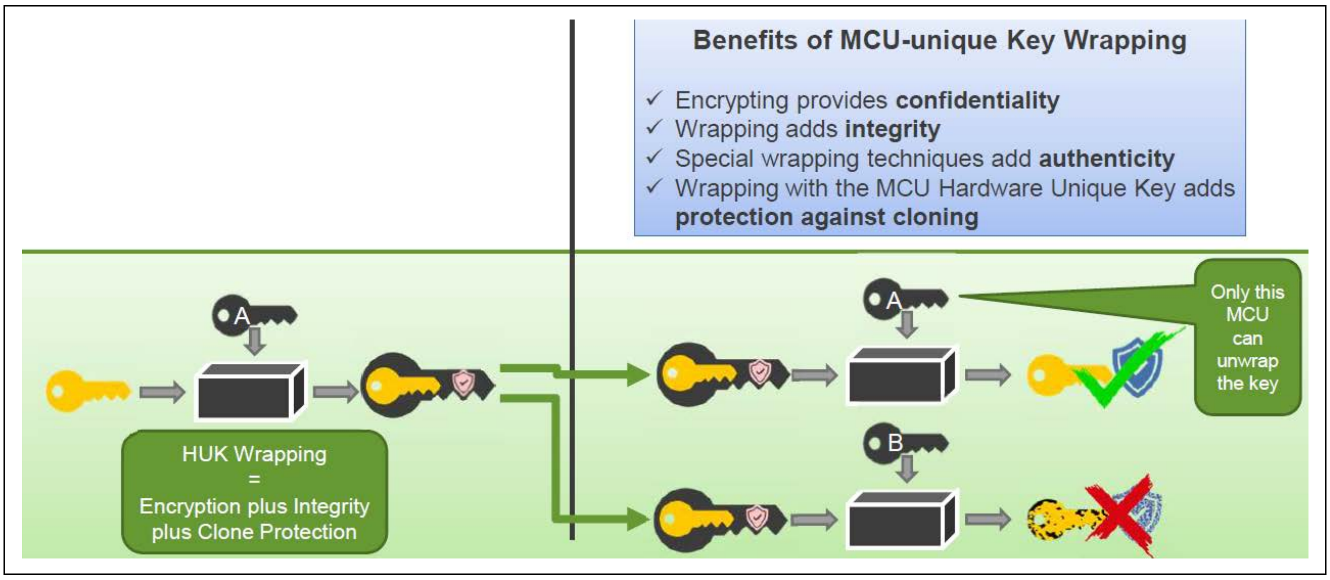

Benefits of MCU-unique Key Wrapping
✓ Encrypting provides **confidentiality**
✓ Wrapping adds **integrity**
✓ Special wrapping techniques add **authenticity**
✓ Wrapping with the MCU Hardware Unique Key adds **protection against cloning**
A
B
HUK Wrapping
=
Encryption plus Integrity plus Clone Protection
Only this MCU can unwrap the key

Figure 6. Key Wrapping using the HUK

Using the MCU Hardware Unique Key to wrap the stored keys adds another protection feature – clone protection.

- If the wrapped key is transmitted or copied to another MCU, that MCU's HUK will not be able to unwrap nor decrypt the information, maintaining the security of the key.
- MCU-wrapped keys can only be unwrapped by the MCU that wrapped them
  - The MCU's HUK is used as part of the wrapping algorithm
  - Since the HUK is unique, no other MCU can unwrap the key
- Benefits
  - Wrapped keys can be stored in non-secure memory
  - Even if the entire MCU contents are copied onto another device, the keys cannot be utilized nor exposed

3.4 Plaintext User Key Installation Use Cases

This section summaries several common use cases for key installation.

Case 1: Plaintext Key Installation During Production Provisioning/Programming

In this case, user keys are injected to the MCU based on customer's existing or preferred method. The injected plaintext key is then installed by MCU application-level code using the Renesas RA Family FSP. This use case enables installation of pre-generated keys, which should be performed in a secure environment. Solutions for this use case are supported by FSP 2.0.0 or later. The FSP APIs used are demonstrated in the example project included in this application project.

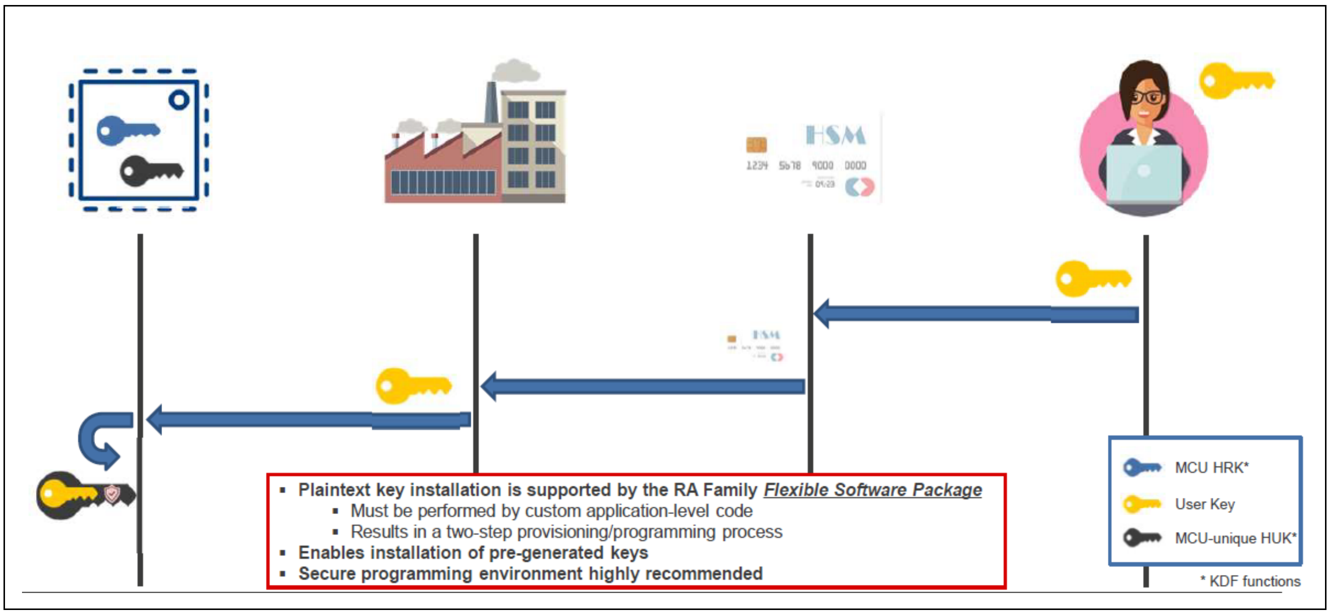

HSM
Plaintext key installation is supported by the RA Family Flexible Software Package
Must be performed by custom application-level code
Results in a two-step provisioning/programming process
Enables installation of pre-generated keys
Secure programming environment highly recommended
MCU HRK*
User Key
MCU-unique HUK*
* KDF functions

Figure 7. Plaintext Key Installation During Production

Case 2: Plaintext Key Installation Over Secure Communication Path

It is possible to provide a secure communication path for plaintext key installation. In this use case, the plaintext key is securely transmitted and injected to the MCU. The MCU secure application software then installs the plaintext key, storing the key in wrapped format. Solutions to support this use case are dependent on the communication path implementation. Customers can leverage the MCU operations provided for Case 1 to implement this solution.

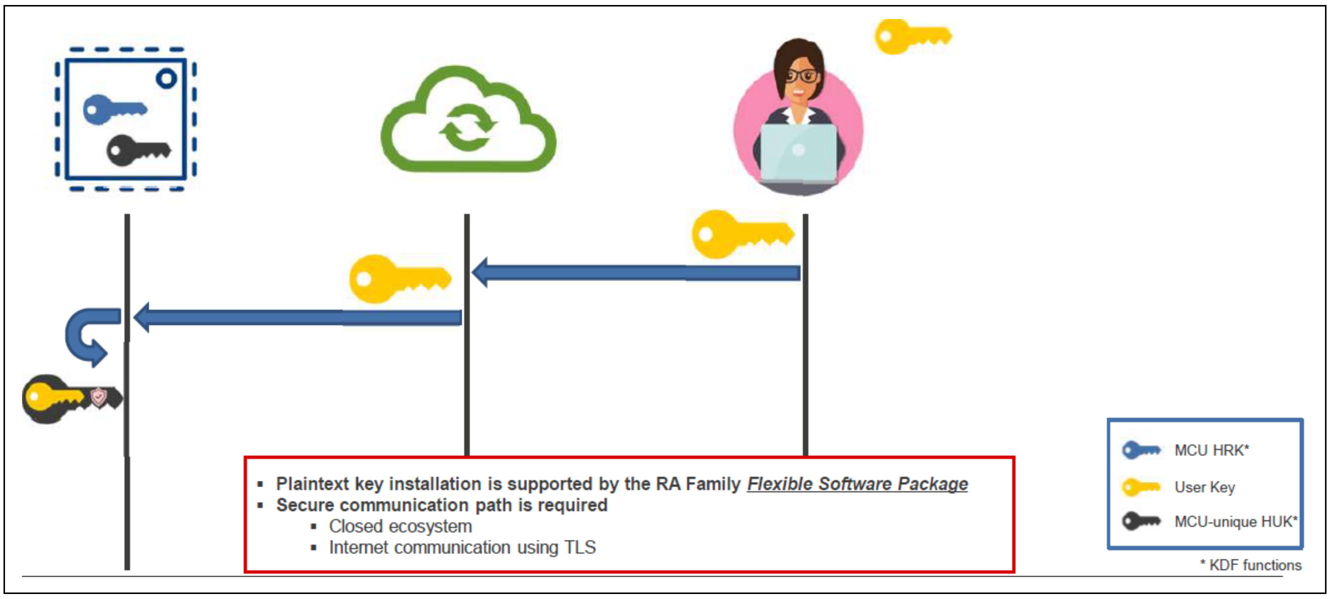

Plaintext key installation is supported by the RA Family Flexible Software Package
Secure communication path is required
Closed ecosystem
Internet communication using TLS
MCU HRK*
User Key
MCU-unique HUK*
* KDF functions

Figure 8. Plaintext Key Installation Over Secure Communication Path

Comparing Key Installation and MCU Key Generation

The following table summarizes the use case comparison between Key Injection and MCU Key Generation:

Table 2 Use Case Comparison with MCU Generated Keys

	Plaintext Key Injection	**MCU Key Generation (Wrapped Key)**
Mass Production	Provides scalability, Faster	Provides scalability, Slower
Secure Environment	Recommended	Not required
Device Identity	Supported	Supported

4. Example Project with AES User Key Handling

The hardware features of SCE9 are accesed via the FSP driver `r_sce`. For most application development, developers can use the middleware `PSA Crypto` layer to interface with the SCE9. However, some SCE9 functionality does not map to PSA Crypto APIs; therefore, `r_sce` key installation related APIs must be used directly.

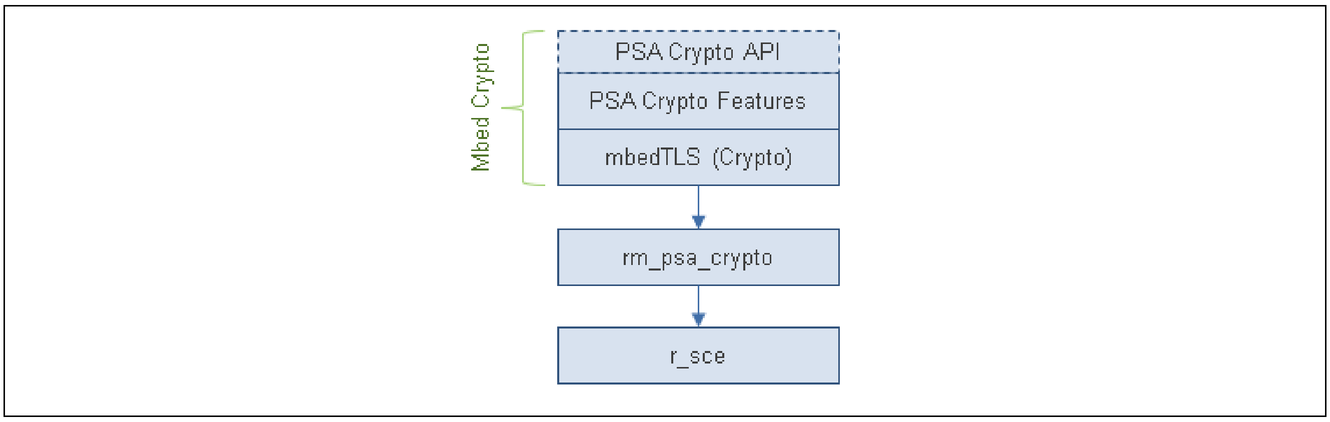


Figure 9. Crypto stacks

Using `PSA Crypto` with TrustZone needs some special handling compared with other drivers. Unlike other FSP drivers, the `PSA Crypto` module cannot be added as a Non-Secure-Callable module. The reason for this is that to achieve the security objective of controlling access to protected keys, both the `PSA Crypto` code as well as the keys must be placed in the secure region. The `PSA Crypto` API requires access to the keys directly during initialization and later via a key handle. Therefore, the `PSA Crypto` module should reside in the secure region.

To provide services to the Non-Secure region, you need to create application-specific user defined Non-Secure Callable (NSC) APIs in the secure region. Proper security considerations can be implemented in the Non-Secure Callable API to limit access to the NSC APIs.

The need for the Non- Secure region accessing cryptographic service in the secure region varies from application to application. You need to adjust the Non-Secure Callable API provided in this example project based on your specific application. It is not advised to use the example as-is for a real-world secure application.

Below is the high-level software block diagram of the example project provided in this application project.

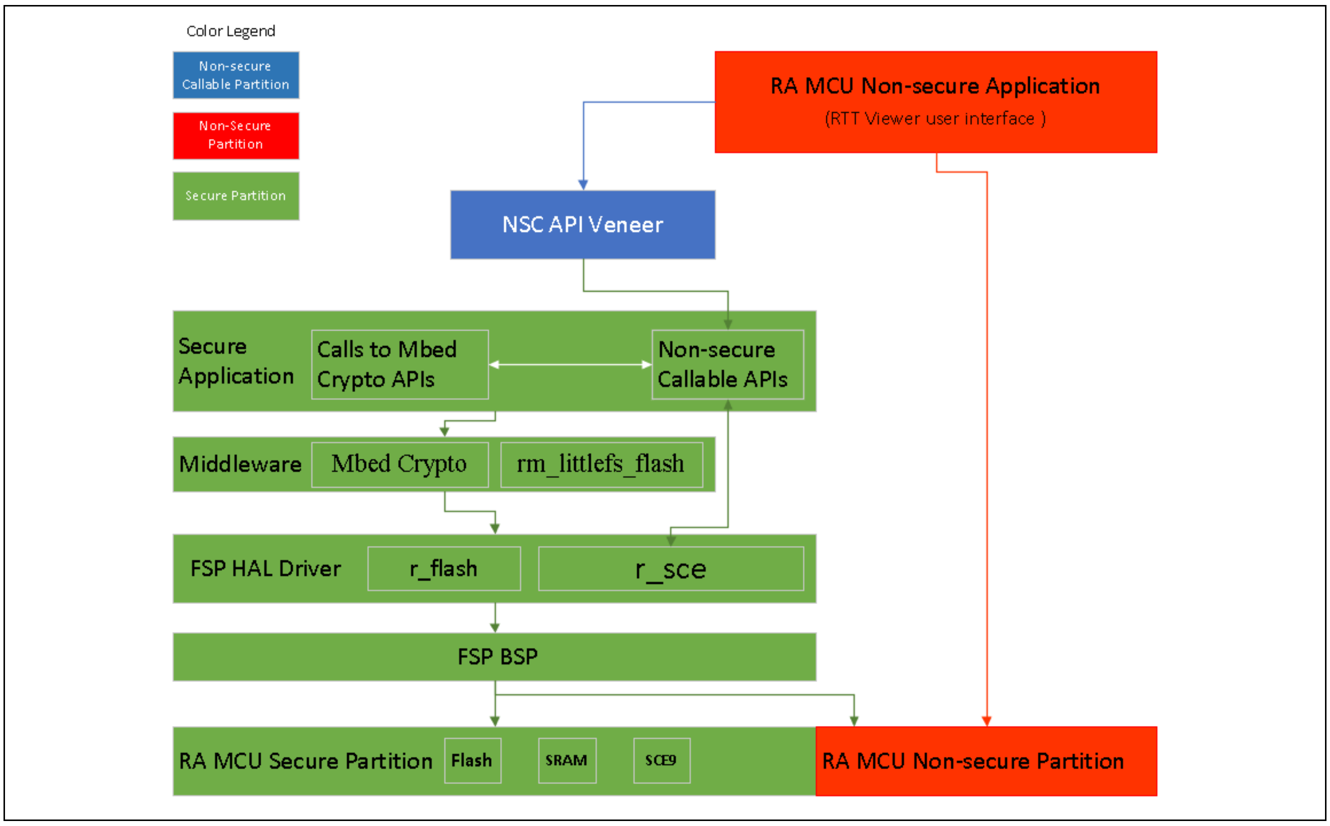


Figure 10. Software Block Diagram

The Non-Secure Callable APIs are defined in `aes_functions.h` file. These APIs are explained below:

- **`BSP_CMSE_NONSECURE_ENTRY bool`** `init_lfs(void);`
  Initializes the LittleFS system: formatted and mounted.
- **`BSP_CMSE_NONSECURE_ENTRY bool`** `create_aes_key(aes_creation_args_t *args);`
  Allows the Non-Secure project to initiate new AES key creation by installing a 256-bit AES plaintext key (generated as a random number using the MCU’s TRNG) as a wrapped key. Once the plaintext user key is injected into the MCU, the SCE9 driver is used to convert the plaintext key into wrapped key format by wrapping the plaintext key using the HUK. The plaintext key will be erased immediately after the conversion. The wrapped AES key is further imported into the PSA key storage system and stored in the data flash for user application usage.
- **`BSP_CMSE_NONSECURE_ENTRY bool`** `destroy_the_key(void);`
  Removes the key from the key management system.
- **`BSP_CMSE_NONSECURE_ENTRY bool`**
  `encryption_operation(non_secure_encryption_args_t *args);`
  Encrypts user data using the created AES key.
- **`BSP_CMSE_NONSECURE_ENTRY bool`**
  `decryption_operation(non_secure_decryption_args_t *args);`
  Decrypts user data using the created AES key.

4.1 Import and Compile the Example Project

Follow the FSP User's Manual section**,** *Importing an Existing Project into e^2 studio* to import the Secure and Non-Secure Projects into the workspace and compile in the order instructed below.

1. Expand the secure project `installing_utilizing_user_keys_ra6m4_s`, and double-click the `configuration.xml` to launch the configurator. Click **Generate Project Content** and then build the Secure project. Project should build with no errors.
   Note that there are third party software warnings.
2. Expand the non-secure project `installing_utilizing_user_keys_ra6m4_ns`, double-click the **`configuration.xml`** to launch the configurator. Click **Generate Project Content** and then build the Non-Secure project.

4.2 Setting up the Hardware

Establish the following connections:

- EK-RA6M4 jumper setting: J6 closed, J9 open. Other jumpers keep out-of-box setting.
- USB cable connected between J10 and development PC to provide power and debugging capability using the on-board debugger

Initialize the MCU using Renesas Device Partition Manager

This step is optional but recommended. Prior to downloaing the example application, we recommend initializing the device to SSD state. Flash content not permanently locked down will be erased during this process. This is particularly helpful if the device was previously used in NSECSD state or have certain flash block locked up temporarily.

Note: You need to power cycle the board prior to working with **Renesas Device Partition Manager** after a debug session if using J-Link as connection interface.

Open the **Renesas Device Partition Manager**. With the e^2 studio IDE, click the **Run** tab and then select **Renesas Device Partition Manager**.

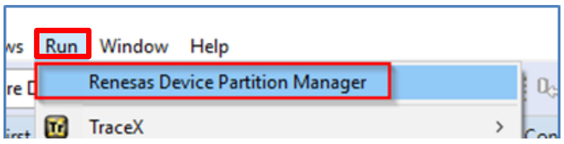
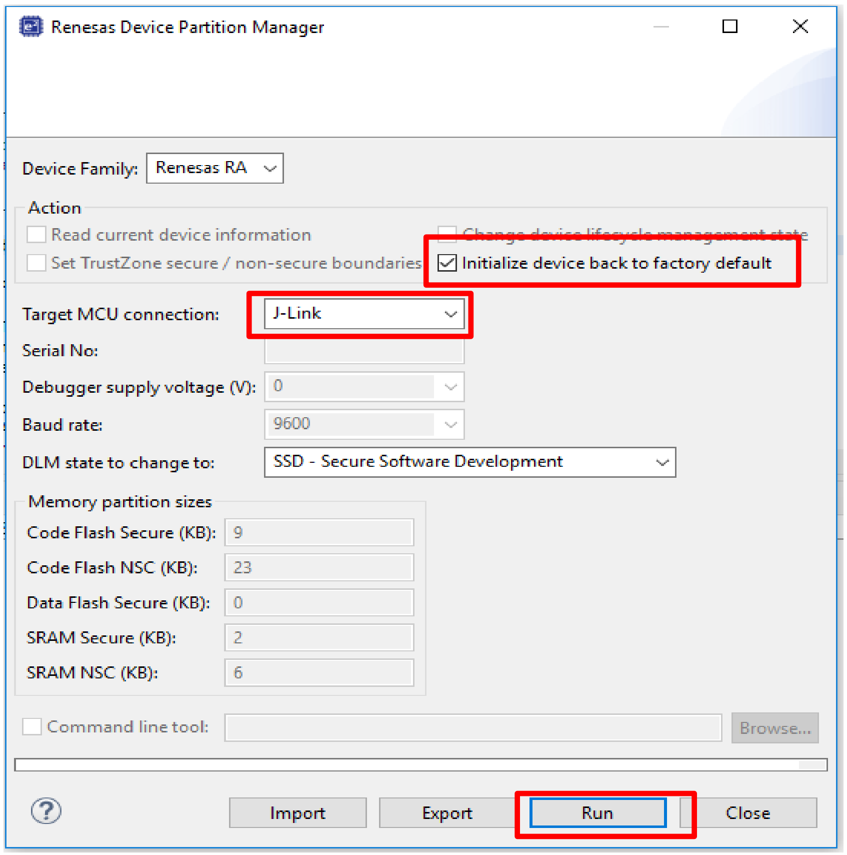


Figure 11. Open Renesas Device Partition Manager

Next, check **Initialize device back to factory default**, choose **J-Link** as the connection method, and then click **Run**.



Device Family: Renesas RA

Action
- [ ] Read current device information
- [ ] Set TrustZone secure / non-secure boundaries
- [ ] Change device lifecycle management state
- [x] Initialize device back to factory default

Target MCU connection: J-Link
Serial No:
Debugger supply voltage (V): 0
Baud rate: 9600
DLM state to change to: SSD - Secure Software Development

Memory partition sizes
Code Flash Secure (KB): 9
Code Flash NSC (KB): 23
Data Flash Secure (KB): 0
SRAM Secure (KB): 2
SRAM NSC (KB): 6

- [ ] Command line tool:

Figure 12. Initialize RA6M4 using Renesas Device Partition Manager

4.3 Running the Example Project

To run the application, right-click on `installing_utilizing_user_keys_ra6m4_ns` and select **Debug As > Renesas GDB Hardware Debugging**.

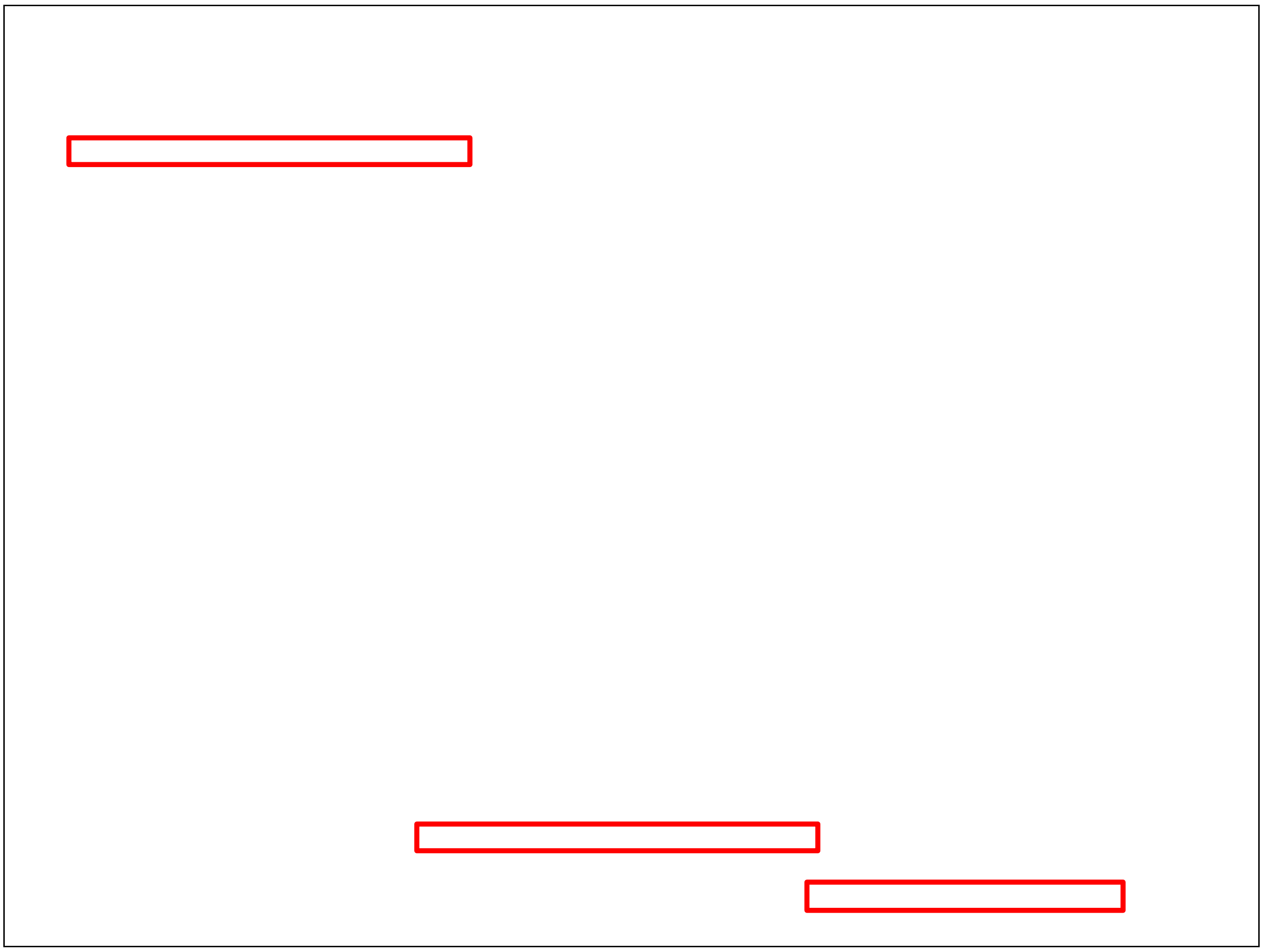
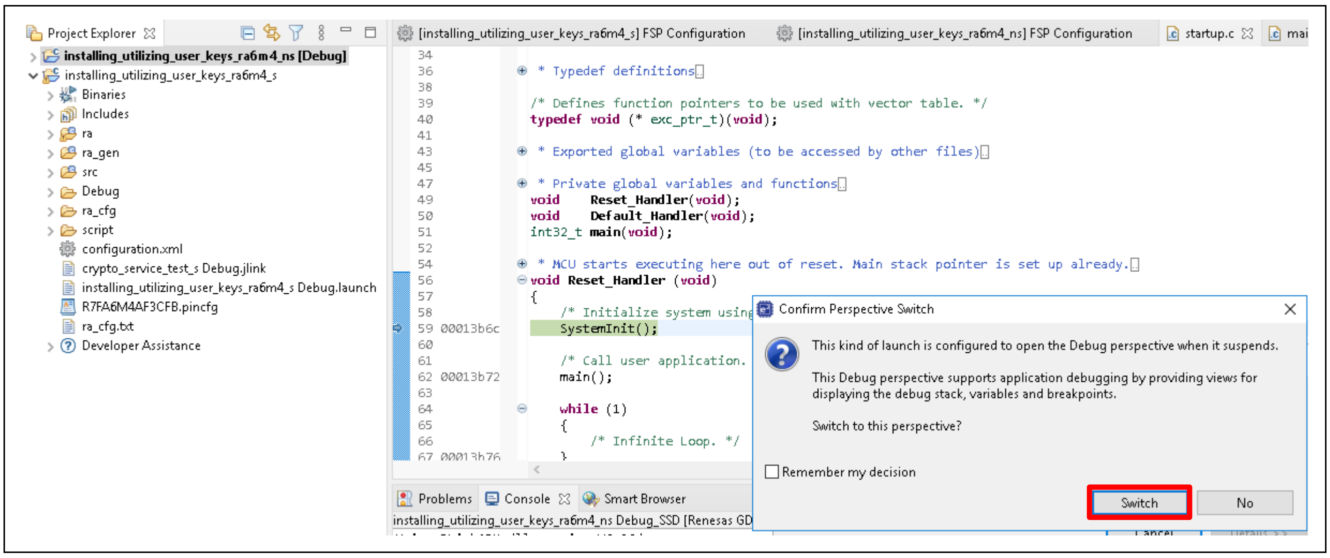


Figure 13. Running the Application

Note that prior to the application execution, the IDAU regions will be set up to assume the values via the debugger interaction with the MCU bootloader.

Both the secure and non-secure projects are now loaded, and the debugger should be paused in the `Reset_Handler()` at the `SystemInit()` call in the Secure project.



Project Explorer
- installing_utilizing_user_keys_ra6m4_ns [Debug]
- installing_utilizing_user_keys_ra6m4_s
  - Binaries
  - Includes
  - ra
  - ra_gen
  - src
  - Debug
  - ra_cfg
  - script
  - configuration.xml
  - crypto_service_test_s Debug.jlink
  - installing_utilizing_user_keys_ra6m4_s Debug.launch
  - R7FA6M4AF3CFB.pincfg
  - ra_cfg.txt
  - Developer Assistance

```
 * Typedef definitions

/* Defines function pointers to be used with vector table. */
typedef void (* exc_ptr_t)(void);

 * Exported global variables (to be accessed by other files)

 * Private global variables and functions
void    Reset_Handler(void);
void    Default_Handler(void);
int32_t main(void);

 * MCU starts executing here out of reset. Main stack pointer is set up already.
void Reset_Handler (void)
{
    /* Initialize system using
00013b6c    SystemInit();

    /* Call user application.
00013b72    main();

    while (1)
    {
        /* Infinite Loop. */
00013b76    }
```

installing_utilizing_user_keys_ra6m4_ns Debug_SSD [Renesas GD

Confirm Perspective Switch

This kind of launch is configured to open the Debug perspective when it suspends.

This Debug perspective supports application debugging by providing views for displaying the debug stack, variables and breakpoints.

Switch to this perspective?

☐ Remember my decision

[installing_utilizing_user_keys_ra6m4_s] FSP Configuration
[installing_utilizing_user_keys_ra6m4_ns] FSP Configuration
startup.c
Problems
Console
Smart Browser
Switch
No

Figure 14. Secure Project Reset Handler

Click **Switch** if the **Confirm Perspective Switch** window pops up. Click twice to run the project.

Next, launch **J-Link RTT Viewer** V6.86 or later.

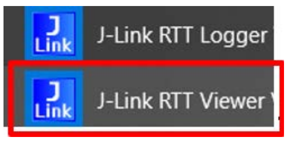
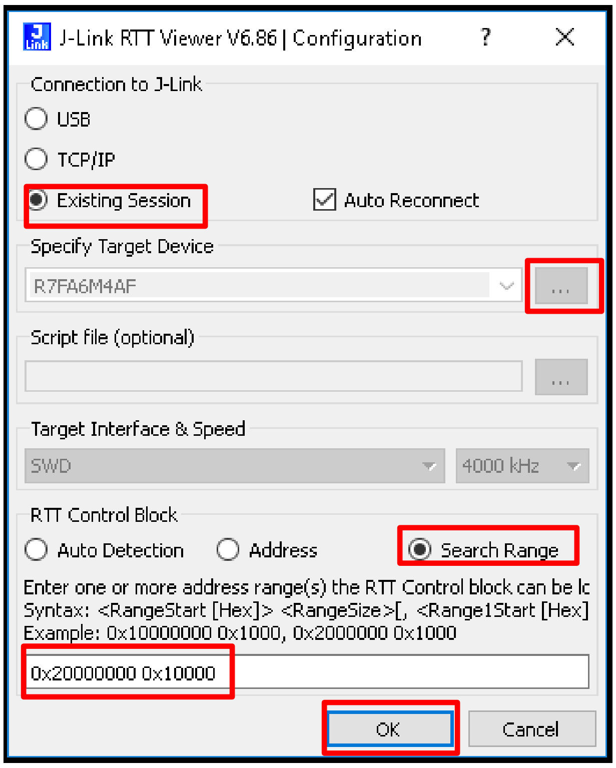


Figure 15. Launch J-Link RTT Viewer

Select **Existing Session** as connection type. Click on the ... button and scroll down to **Renesas** to find the correct device **R7FA6M4AF**. Also set up the RTT Control Block to **Search Range**. Set the search range to 0x20000000 0x10000 and then click **OK** to start the RTT Viewer.

Note: This Search Size 0x10000 is based on this example applicatio project, if your application uses RTT viewer in Non-Secure region and there is a large secure binary, user needs to increase the Search Size to cover the Non-Secure project SRAM regions.

If the host PC has more than one J-Link debugger connected to the PC, set the **Serial No** (by default **Serial No** is set to 0).

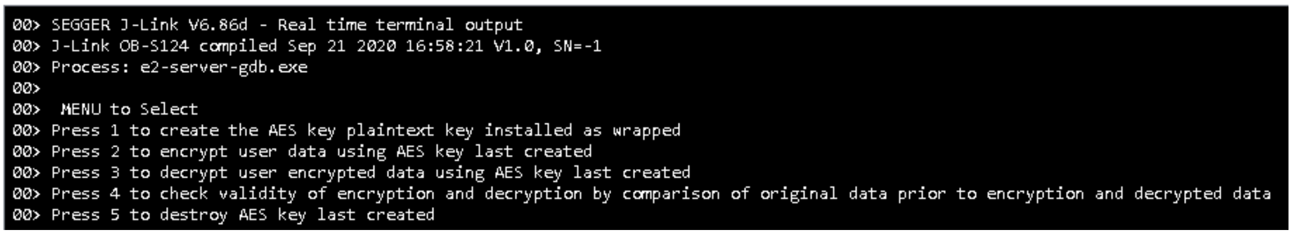


Enter one or more address range(s) the RTT Control block can be lc
Syntax: <RangeStart [Hex]> <RangeSize>[, <Range1Start [Hex]
Example: 0x10000000 0x1000, 0x2000000 0x1000
0x20000000 0x10000

Figure 16. Launch SEGGER RTT Viewer

Click **OK** and follow the steps below to walkthrough the functions provided by the system:

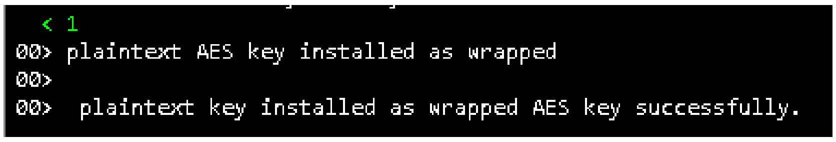
Step 1: The main menu items are printed on the RTT Viewer Terminal 0.



```
00> SEGGER J-Link V6.86d - Real time terminal output
00> J-Link OB-S124 compiled Sep 21 2020 16:58:21 V1.0, SN=-1
00> Process: e2-server-gdb.exe
00>
00>  MENU to Select
00> Press 1 to create the AES key plaintext key installed as wrapped
00> Press 2 to encrypt user data using AES key last created
00> Press 3 to decrypt user encrypted data using AES key last created
00> Press 4 to check validity of encryption and decryption by comparison of original data prior to encryption and decrypted data
00> Press 5 to destroy AES key last created
```

Figure 17. Main Menu

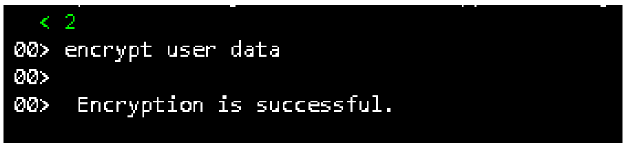
Input **1** to create Wrapped Key from Plaintext key input. The AES plaintext key is generated by using PSA API `psa_generate_random`.



```
  < 1
00> plaintext AES key installed as wrapped
00>
00>  plaintext key installed as wrapped AES key successfully.
```

Figure 18. Creating the Wrapped Key from plaintext input

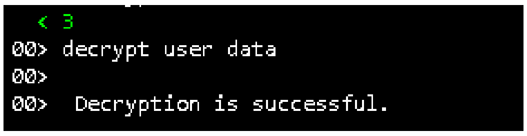
Input **2** to encrypt the user data using the generated wrapped AES key.



```
  < 2
00> encrypt user data
00>
00>  Encryption is successful.
```

Figure 19. Encrypt user data using the Wrapped AES key

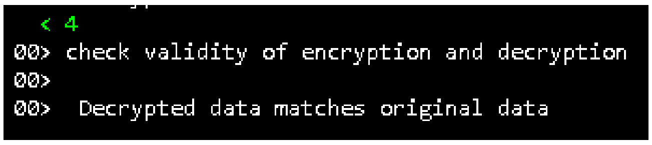
Input **3** to decrypt the encrypted data.



```
  < 3
00> decrypt user data
00>
00>  Decryption is successful.
```

Figure 20. Decrypt the Encrypted Data using the Same AES Key

Input **4** to compare decrypted user data with original unencrypted data.

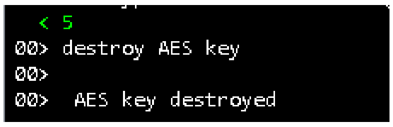


```
  < 4
00> check validity of encryption and decryption
00>
00>  Decrypted data matches original data
```

Figure 21. Check whether the decrypted data matches the original

Input **5** to remove the AES key from system.

Note: No TrustZone secure fault is generated in this case.



```
  < 5
00> destroy AES key
00>
00>  AES key destroyed
```

Figure 22. Destroy the created AES Key

5. Appendix

5.1 Glossary

Term	Meaning
HSM	A **hardware security module** (**HSM**) is a physical computing device that safeguards and manages digital keys, performs encryption and decryption functions for digital signatures, strong authentication, and other cryptographic functions.
HRK	**H**ardware **R**oot **K**ey is a secret key resides in the SEC9 module which is used to wrap the HUK for secure storage.
Unique ID	A **Unique ID**entification value, unique to each individual RA Family MCU, that is stored inside the MCU. The unique ID is used by the SCE9 when it wraps a key.
MAC	**M**essage **A**uthentication **C**ode is a short piece of information used to authenticate a message to confirm that the message came from the stated sender (its authenticity) and has not been changed. The MAC value protects both a message's data integrity as well as its authenticity, by allowing verifiers (who also possess the secret key) to detect any changes to the message content.

6. References

1. Renesas Device Lifecycle Management Key Installation

7. Website and Support

Visit the following URLs to learn about the RA family of microcontrollers, download tools and documentation, and get support.

EK-RA6M4 Resources	renesas.com/ra/ek-ra6m4
RA Product Information	renesas.com/ra
Flexible Software Package (FSP)	renesas.com/ra/fsp
RA Product Support Forum	renesas.com/ra/forum
Renesas Support	renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Dec.2.2020	-	First release document