

Renesas RA Family

Application Design using RA8 Series MCU Decryption on the Fly for OSPI

Introduction

The RA8 MCU has the Octal Serial Peripheral Interface (OSPI). This is the OSPI_B version of the OSPI peripheral module on the RA8 MCUs. The Decryption On-The-Fly (DOTF) peripheral on the RA8 MCUs enables secure external storage of application code or data on the OSPI memory. The information can be stored on the OSPI memory via an independent mechanism, with the decryption key provisioned on the MCU using the appropriate key injection method. Alternatively, the MCU can internally generate a key and write encrypted information to the OSPI for secure storage and later usage. The primary advantage to using DOTF is that code execution and data reading of the external information is performed at about full speed with seamless background decryption.

This application project provides guidelines on how to use the DOTF with the RA8 MCU Renesas Secure IP (RSIP) in Compatibility Mode and Protected Mode. Refer to the Renesas RA Family Security Engine Operational Modes AN (R11AN0498) and Renesas RA Secure Key Injection application project (R11AN0496) to understand these two operational modes and how to use them with the MCU.

The example projects included in this application project use the EK-RA8M1 and EK-RA8P1 evaluation kits. The procedure and application described are applicable to other RA8 MCUs that support the DOTF feature.

For the Renesas Secure IP (RSIP) Compatibility Mode, runtime-encrypted data is stored and decrypted using DOTF. For the RSIP Protected Mode, a securely injected DOTF key is used.

Furthermore, this application project demonstrates how the DOTF feature operates in a dual-core environment using the RA8P1 MCU.

Target Devices

- RA8M1
- RA8D1
- RA8P1
- RA8M2 (*)
- RA8D2 (*)
- RA8T2 (*)

(*) These devices will be supported starting from FSP v6.2.0 and later.

Required Resources

Software and development tools

- e² studio IDE v2025-04.1
- Renesas Flexible Software Package (FSP) v6.0.0

The links to download the above software are available at <https://github.com/renesas/fsp>.

- Renesas Flash Programmer (RFP) v3.20
<https://www.renesas.com/us/en/products/software-tools/tools/programmer/renesas-flash-programmer-programming-gui.html>
- Renesas Security Key Management Tool v1.10
<https://www.renesas.com/software-tool/security-key-management-tool>
- Gpg4win
<http://www.gpg4win.org/>

Hardware

- EK-RA8M1, Evaluation Kit for RA8M1 MCU Group ([renesas.com/ra/ek-ra8m1](https://www.renesas.com/ra/ek-ra8m1))
- EK-RA8P1, Evaluation Kit for RA8P1 MCU Group ([renesas.com/ek-ra8p1](https://www.renesas.com/ek-ra8p1))
- Workstation running Windows® 10 and the Tera Term console or similar application
- One USB device cable (type-A male to micro-B male)
- One USB device cable (type-C male to type-C male) (for the EK-RA8P1 board)

Prerequisites and Intended Audience

This application project assumes that the user has experience using the Renesas e² studio IDE. In addition, knowledge of Renesas RA key injection methods, the Secure Key Management Tool (SKMT), the Renesas Flash Programmer (RFP) and the RSIP operational modes is required prior to evaluating the RA8 DOTF system. Furthermore, the user should also have knowledge of developing dual-core projects when working with DOTF on dual-core devices. The reference section has information on the available Application Projects and User Manuals to gain this knowledge. General knowledge of cryptographic algorithms is highly desired.

Contents

Introduction	1
1. RA8 MCU Decryption on the Fly	5
1.1 DOTF Architecture	5
1.2 DOTF Features	6
1.3 Example Operational Flow	7
1.4 DOTF Usage Notes	8
1.4.1 Endianness of DOTF Operation	8
1.4.2 Specific Data Handling Performing Runtime Encryption with DOTF	8
1.4.3 Setting the Initialization Vector (IV) for DOTF Operation	9
1.4.4 Usage of the AES-CTR	9
1.4.5 Use the RSIP and Key Injection in Matching Mode	9
1.5 Configuring DOTF Operation using FSP	10
1.6 Allocating Data to the OSPI Area	10
1.7 Using Multiple DOTF Keys	11
1.8 Reset the OSPI Device	11
2. Example Implementation: Using DOTF with RSIP Compatibility Mode	11
2.1 Creating the Application with RSIP Compatibility Mode	12
2.2 Encrypt the OSPI Data at Runtime	16
2.3 Allocating Plaintext Data to the OSPI Area	17
2.3.1 Allocation on RA8M1	17
2.3.2 Allocation on RA8P1	21
2.4 Running the Example Application	22
2.4.1 Set up the Hardware and Import the Application	22
2.4.2 Launch the Debug Session and Observe the Demonstration	26
3. Example Implementation: Using DOTF with RSIP Protected Mode	29
3.1 Tools Used in the DOTF Design with RSIP Protected Mode	29
3.2 Creating the Wrapped DOTF Key	30
3.3 Configure the Application Project with RSIP in Protected Mode	33
3.4 Update the Linker Script	35
3.5 Allocating Code to the DOTF Destination Area	36
3.6 Import and Build the RSIP Protected Mode Example Project	37
3.6.1 Encrypt the DOTF Destination Area Using the SKMT CLI	37
3.6.2 Encrypt the DOTF Destination Area using SKMT GUI	39
3.7 Running the Example Application	41
3.7.2 Launch the Debug Session using the SKMT CLI Generated Images	44
3.7.3 Launch the Debug Session using the SKMT GUI Encryption Result	48

4.	Guidelines for DOTF Production Support.....	49
5.	Appendix	51
5.1	Linker Script used in the Compatibility Mode Example Project with Arm Compiler (AC6).....	51
5.2	Update the Linker Script for the Protected Mode Example Project.....	51
6.	Known Issues and Troubleshooting	52
6.1	Data Swapping in DDR-OPI Mode (EK-RA8P1)	52
6.2	J-Link Script Failure.....	53
7.	References	54
8.	Website and Support	55
	Revision History	56

1. RA8 MCU Decryption on the Fly

This section introduces the architecture of the RA8 DOTF peripheral, its features, the use cases, and the example operational flow. Some general usage notes are also provided as a reference when designing an application using DOTF.

1.1 DOTF Architecture

The following block diagram describes the interactions between the DOTF peripheral and the MCU bus system and other supporting security peripherals.

- RSIP supports both DOTF key injection and key generation
- The dedicated AES-CTR decryption engine performs the decryption for the DOTF operation
- The DOTF controller manages the DOTF operation

The following are descriptions of the five numbered legends (1) through (5). These are the key operations when designing an application with DOTF enabled.

- (1) Encrypted data read operations that go through the AES CTR engine for decryption
- (2) Plaintext data read operations that bypass the AES CTR engine
- (3) Decrypted data read operations following AES CTR decryption
- (4) Data write operations that bypass the DOTF operation
- (5) XSPI I/O register interface read/write operations bypassing the DOTF operation

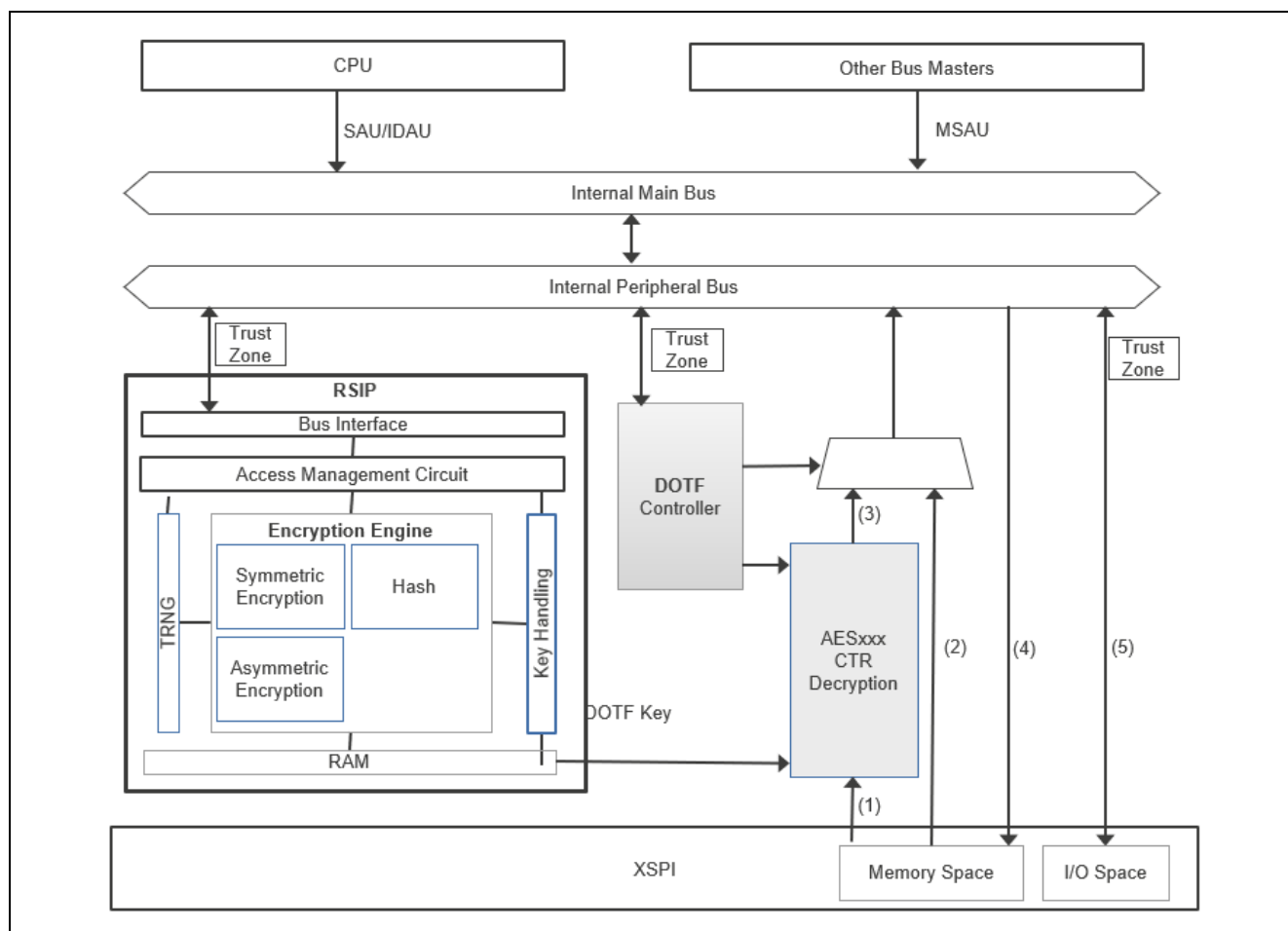


Figure 1. RA8 Decrypt on the Fly

On the RA8 Cortex-M85 Devices, the OSPI area starts at 0x80000000. The OSPI peripheral interfaces with external OctaFlash and/or OctaRAM chip(s) can perform data I/O operations. This is the OSPI_B version of

the OSPI peripheral module for the RA family. When both OctaFlash and OctaRAM devices are interfaced, they must be connected to dedicated chip-select lines. The devices cannot share a single chip-select line.

On both the EK-RA8M1 and EK-RA8P1 evaluation kits, an OctaFlash is connected on Unit 0, Slave 1 starting at 0x90000000, with a supported address range of 256MB. In this application project, we will use this channel and the on-board OSPI to demonstrate the DOTF operation.

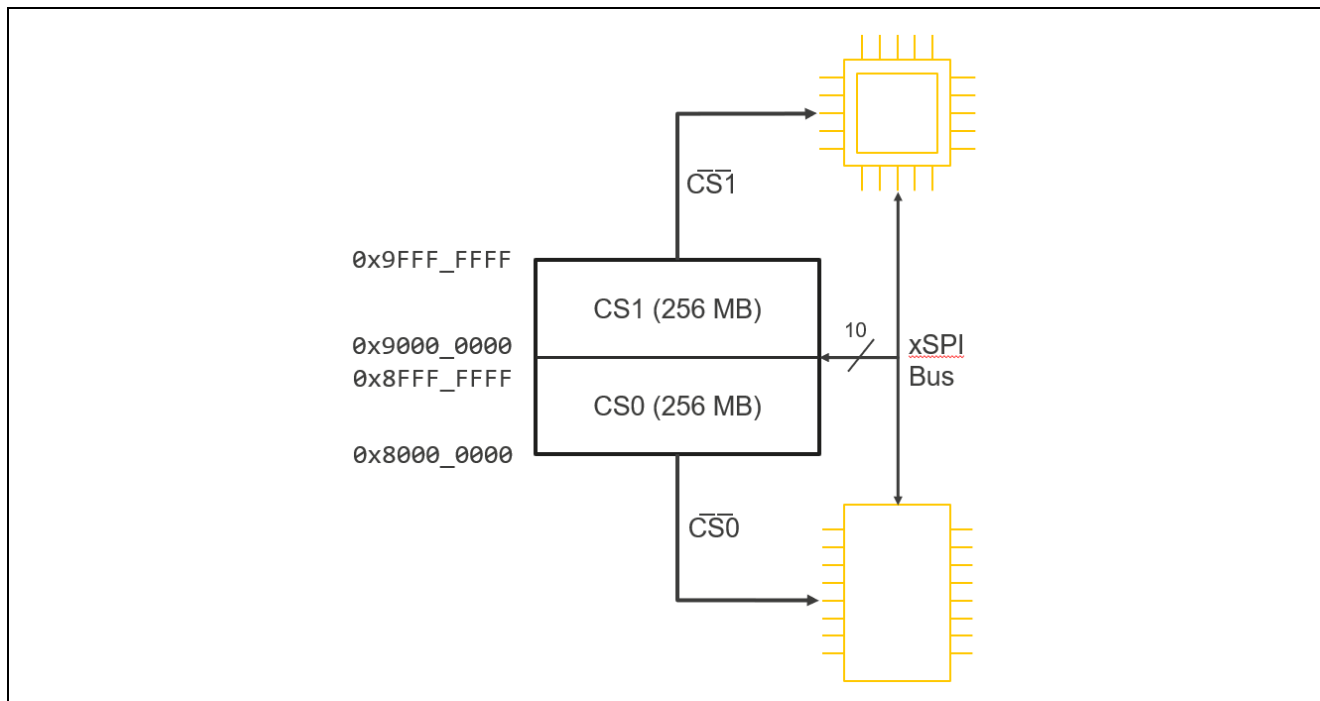


Figure 2. OSPI Memory Space

1.2 DOTF Features

The key DOTF features include the following:

- DOTF is supported with both RSIP Protected Mode and Compatibility Mode. To learn about the RSIP Protected Mode and Compatibility Mode, please refer to the Renesas RA Family Security Engine Operation Modes application note (R11AN0498). It is important to note that the key storage type must match the RSIP mode.
- DOTF supports confidential external code and/or data. A mix of plaintext and encrypted code/data is allowed.
- The external storage data can be pre-stored with a known key or stored at run-time with a generated key. Any previously injected or internally generated key can be used as the DOTF key.
- Code execution using the external storage and data read from the external storage are transparent to the application using the DOTF feature.
- DOTF uses the AES-CTR cryptographic algorithm AES128-CTR, AES192-CTR, and AES256-CTR.
- Any range of the valid OSPI area can be defined as a region to be decrypted by the DOTF. Multiple DOTF destination regions can be configured with distinct DOTF keys.

The following are some major use cases where DOTF can be used. All use cases are supported under both Protected Mode and Compatibility Mode, but the mechanisms for injecting any pre-shared keys will differ.

• Pre-program code or data with a known key

OEM may have sensitive content that needs to be protected in the OSPI area. The sensitive code or data can be pre-encrypted and programmed in the OSPI area prior to deliver to the end customer for application development. This is demonstrated using the RSIP Protected Mode in this application project.

• Store data at run-time with a generated key

The application may generate sensitive data that needs to be stored to the OSPI area encrypted. For example, private patient information may be collected at run time and stored encrypted in the OSPI area. This is demonstrated using the RSIP Compatibility Mode in this application project.

OSPI writes to the specified DOTF address range are not automatically encrypted. Application code must encrypt the data prior to writing it using the DOTF decryption key.

1.3 Example Operational Flow

For the first use case mentioned above, the following flow using DOTF with RSIP Protected Mode is demonstrated in this application project. In this example, the OSPI data encryption is performed by using the SKMT tool.

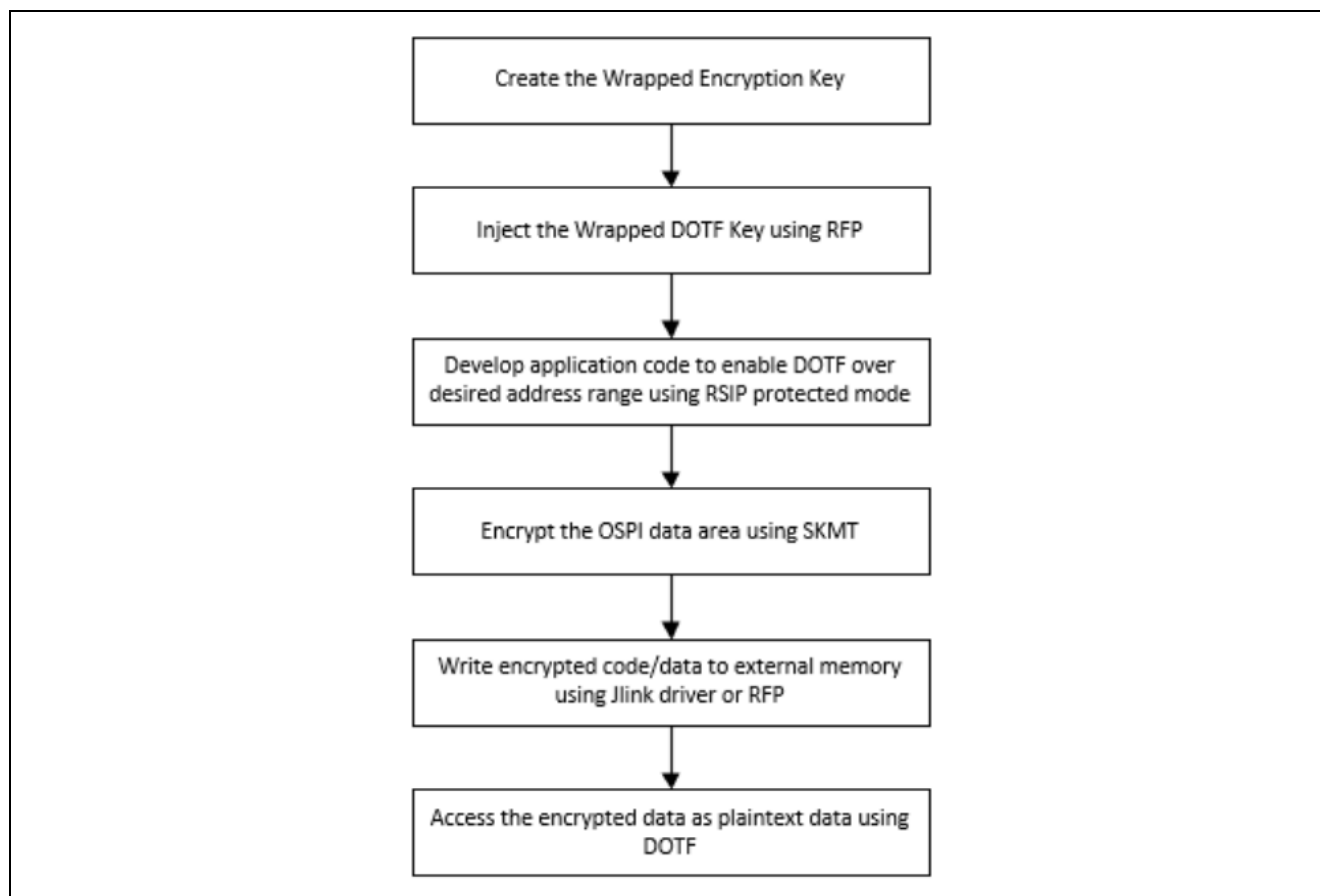


Figure 3. Example DOTF Operational Flow using RSIP Protected Mode

For runtime data encryption support, refer to the following Compatibility Mode operational flow. The DOTF RSIP Compatibility Mode example project included in this application project demonstrates this flow. In this example, the OSPI data encryption is performed at runtime using an application generated plaintext DOTF key.

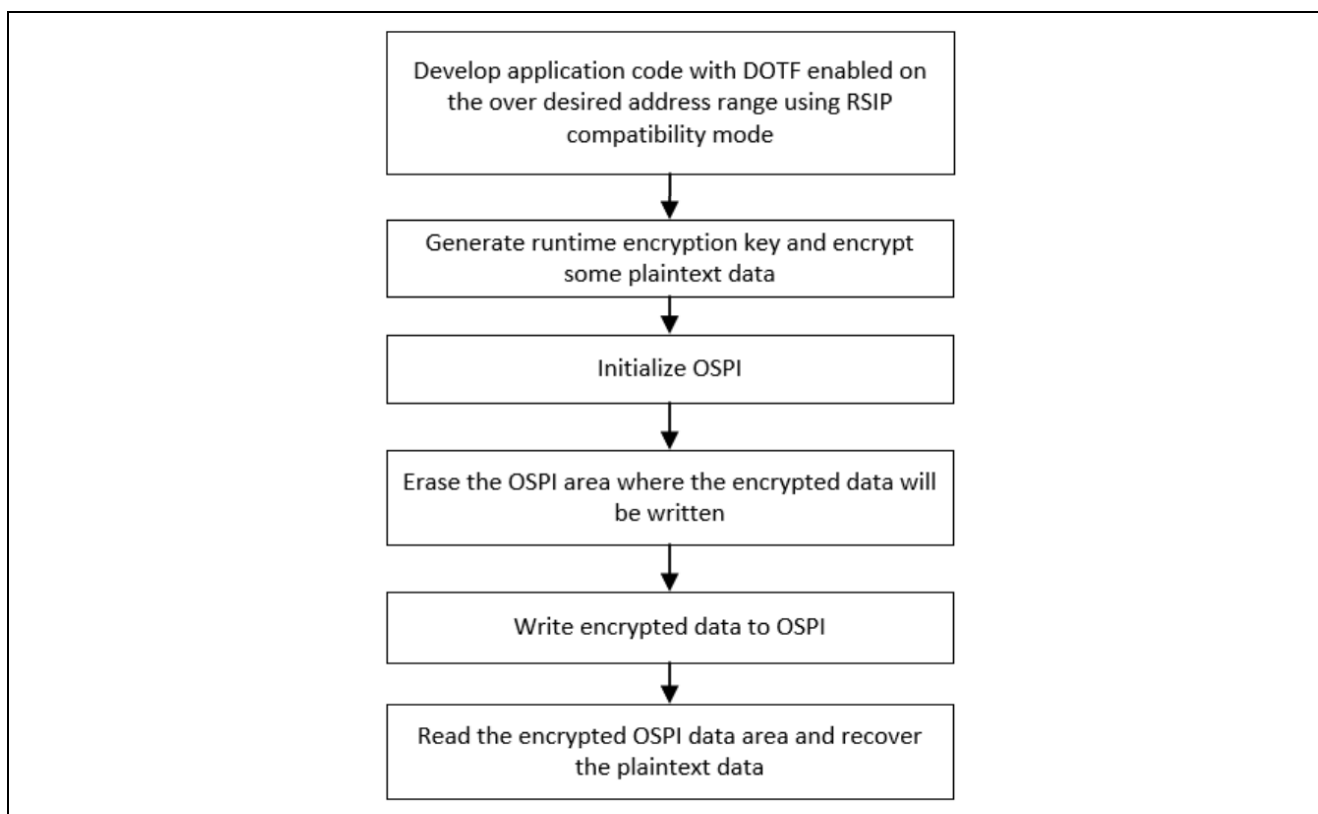


Figure 4. Runtime DOTF Key Generation and Data Encryption using RSIP Compatibility Mode

1.4 DOTF Usage Notes

When designing applications with DOTF, please be aware of the following usage notes.

1.4.1 Endianness of DOTF Operation

The RA8 MCU operates in little endian mode while SKMT and DOTF operate in big endian. It is recommended to provide the DOTF key and the Initialization Vector (IV) in byte format. The SKMT and DOTF operations will automatically use them in big-endian format. Refer to the example project for demonstrations on how to set up the IV and DOTF Key.

1.4.2 Specific Data Handling Performing Runtime Encryption with DOTF

When encrypting data for use with DOTF at runtime using application code, the byte order of each 16-byte block must be reversed prior to and after the AES-CTR encryption. This is not needed when using SKMT to encrypt the OSPI data because this operation is handled by SKMT. Refer to the following flow chart for a summary of the major steps when performing Runtime Encryption using DOTF.

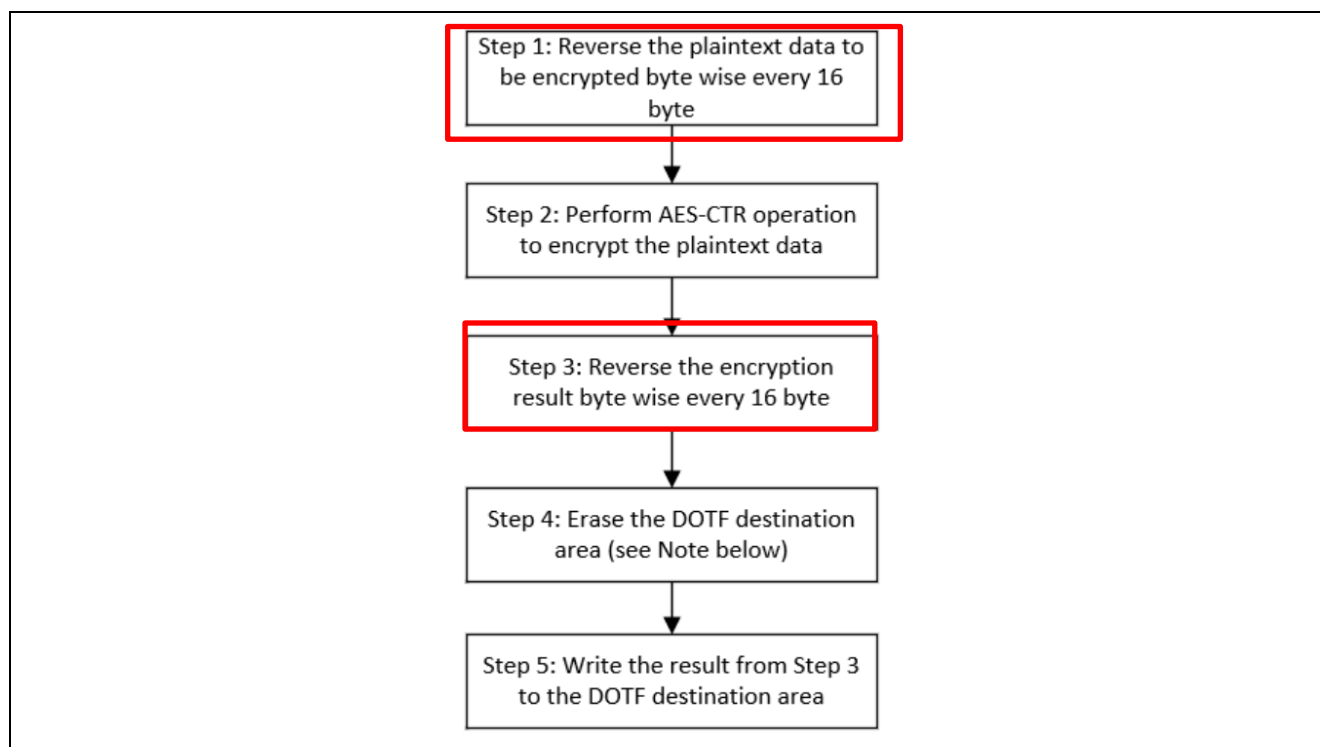


Figure 5. Runtime Encryption Operational Flow

1.4.3 Setting the Initialization Vector (IV) for DOTF Operation

For the DOTF AES-CTR implementation, the IV comprises a nonce and the counter. The first 100 bits of the IV are used as the nonce for AES-CTR. The most significant 28 bit of the DOTF destination address of the data to be encrypted are used as the initial counter. The destination address can be omitted in the encryption command. If the destination address is omitted, the start address of the encryption range is used in place of the destination address.

Counter [127:0] = {IV[127:28], DOTF Destination Address[31:4]}

where DOTF Destination Address is the memory mapped address of the encrypted data.

1.4.4 Usage of the AES-CTR

For the RA8 Series MCU DOTF usage, the information must be AES-encrypted using CTR mode. The configurable key length is 128, 192, 256 bits. No special keys are required – any previously injected or internally generated AES key of the configured length can be used.

The AES-CTR was selected for its fast performance and flexibility in handling data of any size without padding. The AES-CTR offers confidentiality for customer data storage. Using DOTF with authentication is impractical in that it would require a much more complicated system to store the authentication tags. Such a system requires additional storage space, making it impossible to decrypt the data on a 16-byte boundary (which is the specification DOTF supports).

When using the DOTF system, users need to be aware that nonce reuse is a vulnerability for CTR mode. Optimally, use a different nonce for every device.

1.4.5 Use the RSIP and Key Injection in Matching Mode

When using the RA8 series MCU DOTF functionality with key injections, ensure that the secure key storage (wrapping) type matches the security engine mode being used by the application. For example, if the application uses the security engine in Protected Mode, ensure that the DOTF keys are securely injected using the factory boot firmware serial interface or are generated with the security engine operating in Protected Mode.

1.5 Configuring DOTF Operation using FSP

When using the EK-RA8M1 or EK-RA8P1 and FSP to develop applications with OSPI, the OSPI hardware configurations can use the default FSP settings. For DOTF, based on the use cases and the RSIP operational modes, the following are the key configurations that may need to be updated.

With an e2studio project, open the smart configurator and add the OSPI module using the **Stacks** tab via **New Stack > Storage > OSPI Flash (r_ospi_b)**. By default, DOTF is disabled. It can be enabled in **Protected Mode** or **Compatibility Mode** as shown in Figure 6.

Note that FSP uses `r_ospi_b` for the RA8 OSPI driver to differentiate with the OSPI driver for the RA6 MCU series.

Property	Value
▼ Common	
> Memory-mapping Support	
Parameter Checking	Default (BSP)
DMAC Support	Disable
Autocalibration Support	Disable
DOTF Support	Disable
▼ Module g_ospi0 OSPI Flash (r_ospi_b)	Enable (Protected Mode)
> General	Enable (Compatibility Mode)
> Defaults	Disable

Figure 6. Select the DOTF Operation Mode

Open the **Properties** setting of the `r_ospi_b` stack. The following are the key configurations for the **DOTF**.

Table 1 Configuration Properties for DOTF Operation

	Default Setting	Description	Additional Comments
Name	<code>g_ospi_dotf</code>	DOTF Configuration name.	Name must be a valid C symbol
AES Key	<code>g_ospi_dotf_key</code>	Name of Key variable.	Name must be a valid C symbol
AES IV	<code>g_ospi_dotf_iv</code>	Name of IV variable	Name must be a valid C symbol
AES Key Length	128	Select AES key length. Options are:128, 192, 256	none
Key Format	Plaintext	Plaintext or Wrapped	Choose Plaintext if Compatibility Mode plaintext key is used. Choose Wrapped if Compatibility Mode or Protected Mode wrapped key is used.
Decryption start address	0x90000000	OSPI decryption start address	Value must be an integer between 0x80000000 and 0x9FFFFFFF
Decryption end address	0x90001FFF	OSPI decryption end address	Value must be an integer between 0x80000000 and 0x9FFFFFFF

1.6 Allocating Data to the OSPI Area

When the DOTF region is defined in the FSP OSPI stack, the OSPI area will be separated into encrypted and plaintext data regions. When designing an application using the DOTF with both encrypted and plaintext data, the application needs to be aware of the DOTF region and take this into consideration in the design

process. Refer to section 2.3 and section 3.5 for how the example projects included in this application note allocate data to the OSPI area.

1.7 Using Multiple DOTF Keys

FSP API `R_OSPI_B_DOTF_Configure` can be used to set up multiple DOTF regions with multiple DOTF keys each targeting a specific DOTF region at run-time. The application code can configure the `ospi_b_dotf_cfg` structure to change the DOTF address range and DOTF key at runtime.

```
/* This structure is used to hold all the DOTF related configuration. */
typedef struct st_ospi_b_dotf_cfg
{
    ospi_b_dotf_aes_key_type_t key_type;
    ospi_b_dotf_key_format_t   format;
    uint32_t                   * p_start_addr;
    uint32_t                   * p_end_addr;
    uint32_t                   * p_key;
    uint32_t                   * p_iv;
} ospi_b_dotf_cfg_t;
/* OSPI DOTF AES Type. */
typedef enum e_ospi_b_dotf_aes_key_type
{
    OSPI_B_DOTF_AES_KEY_TYPE_128 = 0U,
    OSPI_B_DOTF_AES_KEY_TYPE_192 = 1U,
    OSPI_B_DOTF_AES_KEY_TYPE_256 = 2U
} ospi_b_dotf_aes_key_type_t;

fsp_err_t R_OSPI_B_DOTF_Configure (spi_flash_ctrl_t * const p_ctrl, ospi_b_dotf_cfg_t *
const p_dotf_cfg)
```

1.8 Reset the OSPI Device

For the OSPI device on EK-RA8M1 and EK-RA8P1, if the device was entered into 8D-8D-8D mode prior to the initialization routine, the OSPI device needs a Reset to be successfully initialized. This is handled in the example projects in the `R_BSP_WarmStart` function using the `BSP_WARM_START_POST_C` event.

2. Example Implementation: Using DOTF with RSIP Compatibility Mode

This section explains the establishment of runtime encrypted data and decryption using DOTF with RSIP operating in Compatibility Mode.

For the RA8M1, the GCC compiler is used for the Compatibility Mode example project, while the LLVM compiler is used for Protected Mode example project.

For the RA8P1, the LLVM compiler is used for both the Compatibility Mode and Protected Mode example projects.

Data Cache is also enabled to achieve better system performance.

In addition, starting from FSP v6.0.0 with e2studio v2025-04.1, users can configure and define memory partitions directly through the **Solution Project**, as shown in Figure 7. This enhancement also enables more effective and intuitive memory region management by allowing users to edit the “`solution.xml`” file in the **Solution Project**, as shown in Figure 8.

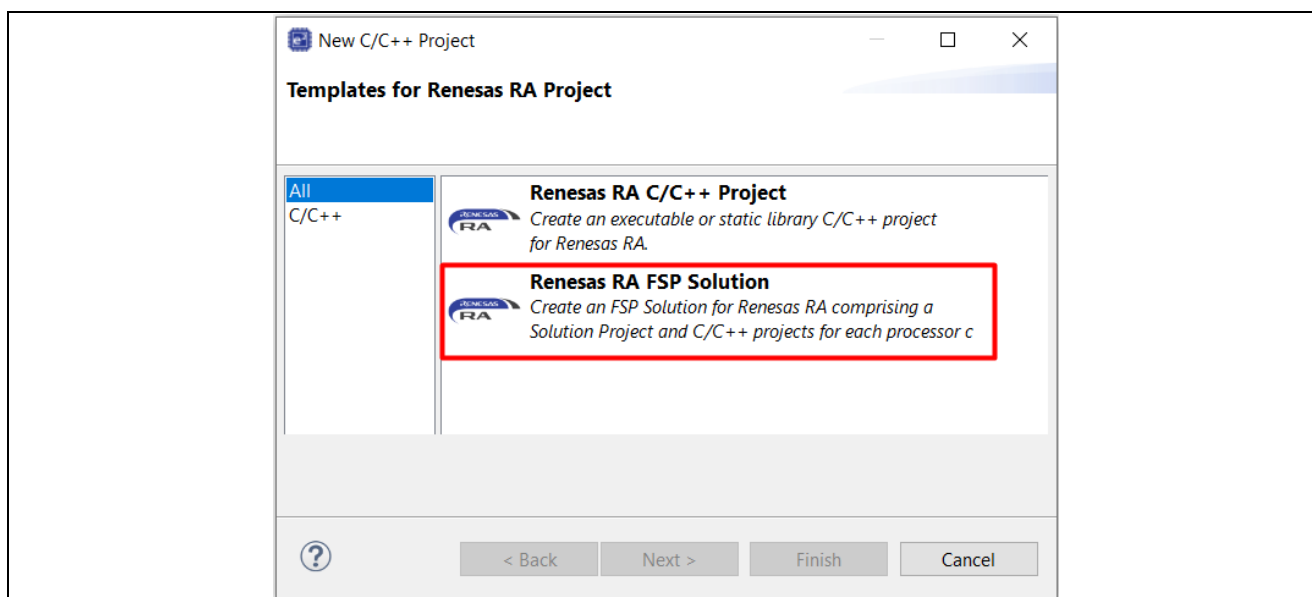


Figure 7. Create the Renesas RA FSP Solution

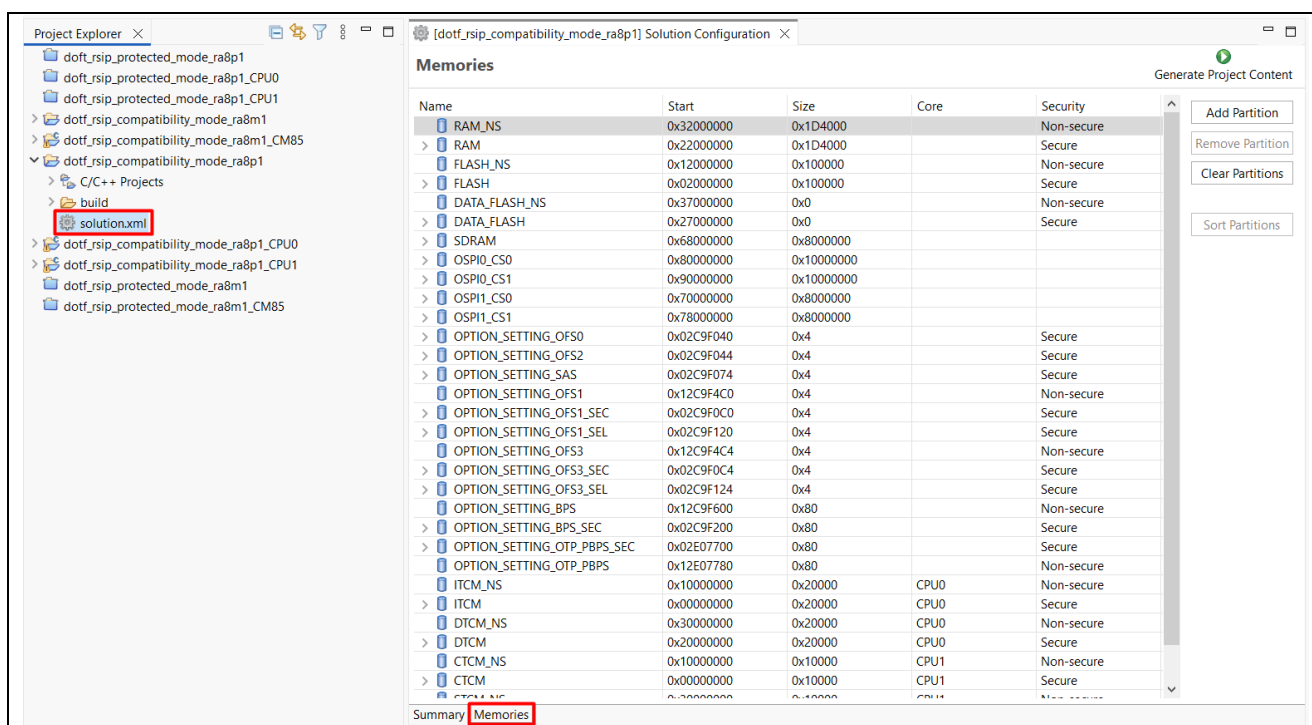


Figure 8. Memory Partition in Solution Project

2.1 Creating the Application with RSIP Compatibility Mode

When choosing Compatibility Mode based on the OSPI Common Property (refer to Figure 6), we have the option of using **Plaintext** DOTF key or **Wrapped** DOTF key. In this example project, a run-time generated Wrapped AES128 key is used as the DOTF key.

This example project uses the **MbedTLS (Crypto only)** module and the PSA Certified Crypto API for the runtime DOTF key generation and plaintext data encryption. The **MbedTLS (Crypto only)** module can be added using the **Stacks** tab via **New Stack > Security > MbedTLS (Crypto Only)**.

The **Wrapped AES128** key is generated using the PSA Certified Crypto API: `psa_generate_key`. This API returns the key handle of the Plaintext key. The `psa_export_key` PSA Certified Crypto API is used to

generate the RAW key data pointed to by the `encryption_key` buffer as the DOTF key. The application project provides the IV of the AES CTR algorithms in the buffer named `encryption_iv`.

In this example, the DOTF decryption range for the RA8M1 is from 0x90000000 to 0x90000FFF.

For the RA8P1, the DOTF decryption ranges are defined separately for each core:

- CPU0: 0x90000000 to 0x90001FFF
- CPU1: 0x90004000 to 0x90005FFF

Figure 9 is the OSPI memory layout for both the DOTF RSIP Compatibility Mode and the DOTF RSIP Protected Mode example projects on RA8M1.

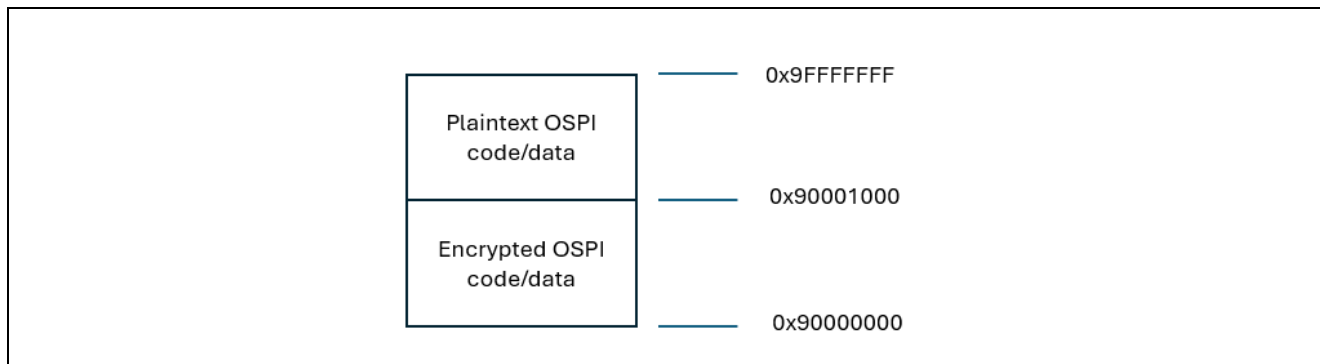


Figure 9. OSPI Memory Map of the Example Projects on RA8M1

Figure 10 is the OSPI memory layout for both the DOTF RSIP Compatibility Mode and the DOTF RSIP Protected Mode example projects on RA8P1.

To validation purposes, the DOTF decryption regions on RA8P1 were expanded by 0x1000 bytes compared to the regions defined on RA8M1.

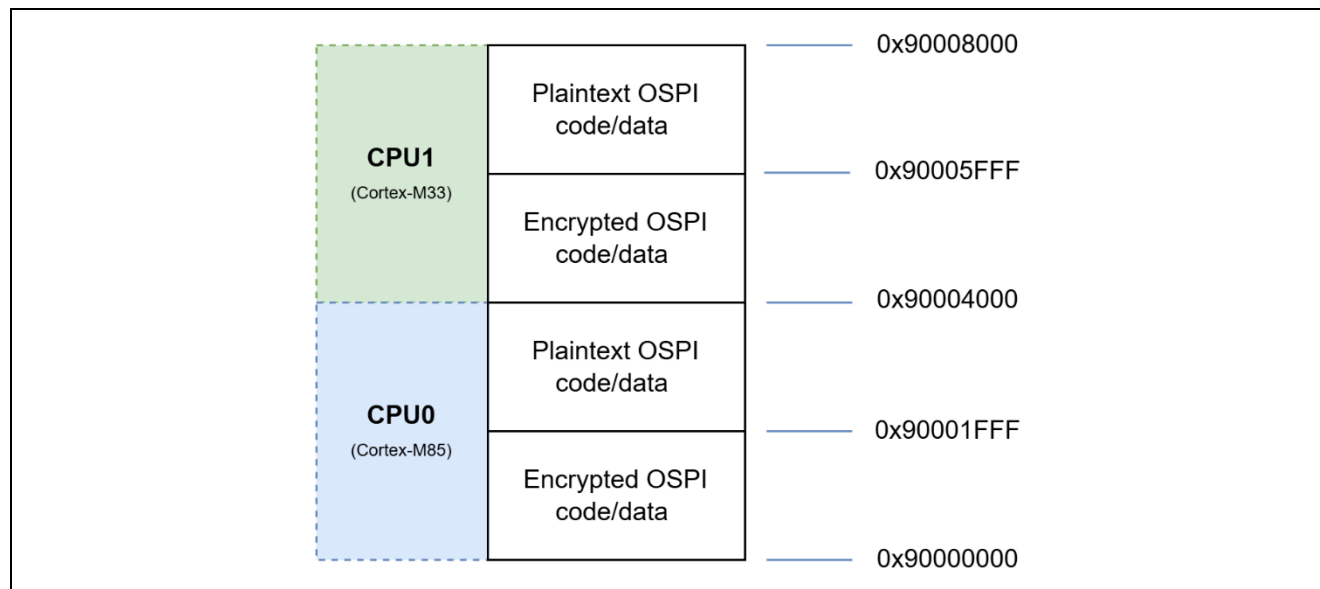


Figure 10. OSPI Memory Map of the Example Projects on RA8P1

Figure 11 is the key DOTF configurations for the Compatibility Mode example project on RA8M1.

g_ospi_b OSPI (r_ospi_b)		
Settings	Property	Value
API Info	▼ Common	
	> Memory-mapping Support	
	Parameter Checking	Default (BSP)
	DMAC Support	Enable
	Autocalibration Support	Disable
	DOTF Support	Enable (Compatibility Mode)
	Row Addressing Support	Disable
	▼ Module g_ospi_b OSPI (r_ospi_b)	
	> General	
	> Command Sets	
	> Timing Settings	
	> XiP Mode	
	▼ DOTF	
	Name	g_ospi_dotf
	AES Key	encryption_key
	AES IV	encryption_iv
	AES Key Length	128
	Key Format	Wrapped
	Decryption start address	0x90000000
	Decryption end address	0x90000FFF

Figure 11. Configure the DOTF with Compatibility Mode on RA8M1

Figure 12 and Figure 13 show the key DOTF configurations for the Compatibility Mode example project on RA8P1 (CPU0 and CPU1).

g_ospi_b OSPI (r_ospi_b)		
Settings	Property	Value
API Info	▼ Common	
	> Memory-mapping Support	
	Parameter Checking	Default (BSP)
	DMAC Support	Enable
	Autocalibration Support	Disable
	DOTF Support	Enable (Compatibility Mode)
	Row Addressing Support	Disable
	▼ Module g_ospi_b OSPI (r_ospi_b)	
	> General	
	> Command Sets	
	> Timing Settings	
	> XiP Mode	
	▼ DOTF	
	Name	g_ospi_dotf
	AES Key	encryption_key
	AES IV	encryption_iv
	AES Key Length	128
	Key Format	Wrapped
	Decryption start address	0x90000000
	Decryption end address	0x90001FFF

Figure 12. Configure the DOTF with Compatibility Mode on RA8P1 (CPU0)

g_ospi_b OSPI (r_ospi_b)		
Settings	Property	Value
API Info	▼ Common	
	> Memory-mapping Support	
	Parameter Checking	Default (BSP)
	DMAC Support	Enable
	Autocalibration Support	Disable
	DOTF Support	Enable (Compatibility Mode)
	Row Addressing Support	Disable
	▼ Module g_ospi_b OSPI (r_ospi_b)	
	> General	
	> Command Sets	
	> Timing Settings	
	> XiP Mode	
	▼ DOTF	
	Name	g_ospi_dottf
	AES Key	encryption_key
	AES IV	encryption_iv
	AES Key Length	128
	Key Format	Wrapped
	Decryption start address	0x90004000
	Decryption end address	0x90005FFF

Figure 13. Configure the DOTF with Compatibility Mode on RA8P1 (CPU1)

Figure 14 and Figure 15Figure 16 provide an overview of the software components used in the example application projects on RA8M1 and RA8P1.

Selected software components
Board Support Package Common Files
I/O Port
Arm CMSIS Version 6 - Core (M)
Board support package for R7FA8M1AHECBD
Board support package for RA8M1
Board support package for RA8M1 - FSP Data
Board support package for RA8M1 - Events
Board support package for RA8M1 - Linker
RA8M1-EK Board Support Files
Arm PSA Crypto Implementation
Direct Memory Access Controller
Octa Serial Peripheral Interface Flash
Secure Cryptography Engine on RA8 (RSIP-E51A) Compatibility Mode
SCI UART
MbedCrypto H/W Acceleration

Figure 14. Software Components Used for Runtime Encryption and DOTF Support on RA8M1

Selected software components	
Board Support Package Common Files	v6.0.0
I/O Port	v6.0.0
Arm CMSIS Version 6 - Core (M)	v6.1.0+fsp.6.0.0
Board support package for R7KA8P1KFLCAC	v6.0.0
Board support package for RA8P1	v6.0.0
Board support package for RA8P1 - FSP Data	v6.0.0
Board support package for RA8P1 - Events	v6.0.0
Board support package for RA8P1 - Linker	v6.0.0
RA8P1-EK Board Support Files	v6.0.0
Arm PSA Crypto Implementation	v3.6.2+renesas.2.fsp.6.0.0
Direct Memory Access Controller	v6.0.0
Octa Serial Peripheral Interface Flash	v6.0.0
Secure Cryptography Engine on RA8 (RSIP-E50D) Compatibility Mode	v6.0.0
SCI UART	v6.0.0
MbedCrypto H/W Acceleration	v6.0.0

Figure 15. Software Components Used for Runtime Encryption and DOTF Support on RA8P1

The `r_sci_b_uart` stack is used for the J-link virtual console to communicate with a PC terminal (eg. Tera Term). The virtual console can be used to print the system status information, for example, error messages or time usage information.

2.2 Encrypt the OSPI Data at Runtime

Refer to the Figure 4 for the general flow of the example runtime OSPI data encryption flow. The following are some key considerations in the implementation.

- In this example implementation, the MbedTLS Crypto module is configured with AES128 CTR encryption enabled. The Asymmetric algorithms like ECC and RSA are disabled to reduce flash and SRAM usage.
- Set up the AES-CTR algorithm IV. As explained in section 1.4.3, the last 28 bit of the IV is the counter which is initialized with the first 28 bit of the destination address of the DOTF operation.

In this example project, the destination address is set to 0x90000000 for RA8M1 and RA8P1 (CPU0), while RA8P1 (CPU1) uses 0x90004000 as the destination address. So, the first 28-bit of the destination address will be the initial value of the counter (which is the last 28 bit of the IV).

The example IV used in this application project is:

```
uint8_t encryption_iv[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x09, 0x00, 0x00, 0x00};
```

In this example project, any nonce (which is the first 100 bits of the IV) would work so long as the initial counter (which is the last 28 bits of the IV) matches the destination address.

- Refer to Figure 5 for the major steps used for the runtime encryption.
- The byte-wise reverse of every 16-byte block is implemented in the following function (`\src\app_runtime_encryption.c`)

```
static void reverse_every_16_bytes(volatile uint8_t *array, size_t length);
```
- Note *: If DOTF is enabled, then when this area is viewed from the e2studio Memory window, the DOTF peripheral will automatically “decrypt” whatever data is read from the OSPI. After erasure, this area will not show as 0xFFs, but rather as “decrypted” 0xFFs.

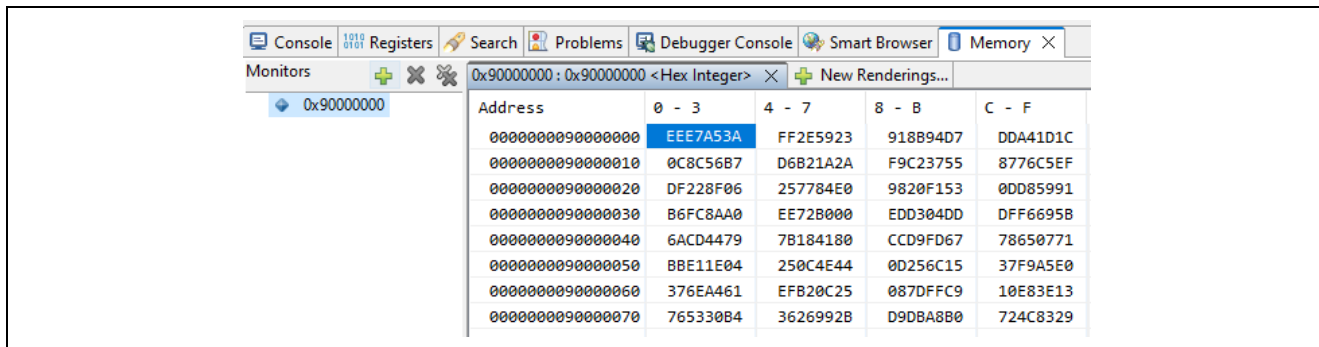


Figure 16. Memory View of the DOTF Area After Erasing (DOTF Enabled)

In this example runtime encryption implementation, the plaintext data to be encrypted is defined in array `plaintext_data(\src\app_runtime_operations.c)`. After this array is encrypted following the procedure described in Figure 5, it is written to start of the DOTF destination area at 0x90000000 for RA8M1 and RA8P1 (CPU0), while for RA8P1 (CPU1) is written to 0x90004000.

2.3 Allocating Plaintext Data to the OSPI Area

This example project includes a simple evaluation of the OSPI area access time from the encrypted region and the plaintext region.

By using the **Solution Project**, users can easily manage the memory partitions through a graphical interface, as introduced in section 2.

2.3.1 Allocation on RA8M1

Based on Figure 9, users can define memory partitions for both the encrypted region and the plaintext region in the OSPI area, as shown in Figure 17.

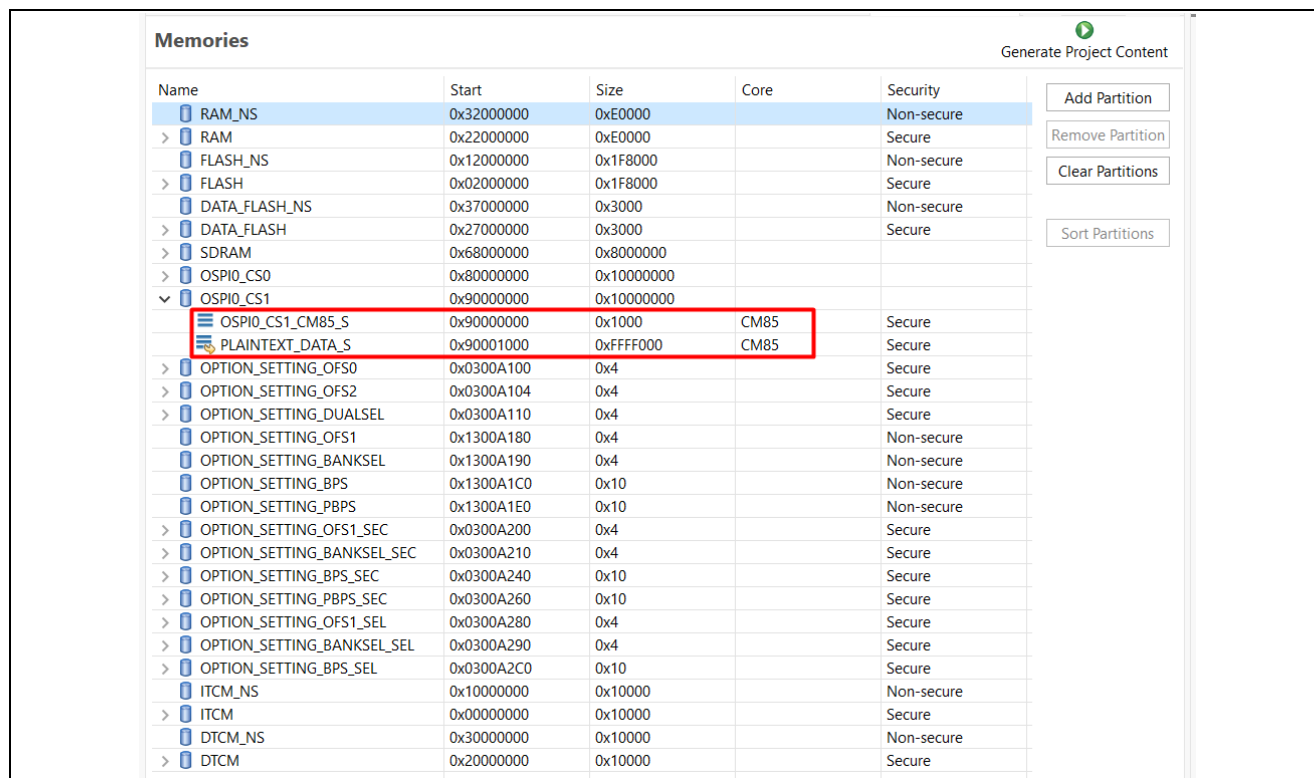


Figure 17. Memory Partition: OSPI Area on RA8M1

After that, click **Generate Project Context** in the **Solution Project**, and then build the project. The plaintext region will be generated in `Debug/fsp_gen.ld` file.

```

815 {
816     __option_setting_bankssel_sel_reg$$Base = .;
817     KEEP(*(.option_setting_bankssel_sel))
818     __option_setting_bankssel_sel_reg$$Limit = .;
819 }> OPTION_SETTING_BANKSEL_SEL
820 .option_setting_bankssel_sel.endof (READONLY) :
821 {
822     __ddsc_OPTION_SETTING_BANKSEL_SEL_END = .;
823
824 }> OPTION_SETTING_BANKSEL_SEL
825
826 /***** OPTION_SETTING_BPS_SEL memory section allocations *****/
827 .option_setting_bps_sel.startof (READONLY) :
828 {
829     __ddsc_OPTION_SETTING_BPS_SEL_START = .;
830
831 }> OPTION_SETTING_BPS_SEL
832 /* BPS Register Select */
833 __option_setting_bps_sel_reg$$ (READONLY) :
834 {
835     __option_setting_bps_sel_reg$$Base = .;
836     KEEP(*(.option_setting_bps_sel))
837     __option_setting_bps_sel_reg$$Limit = .;
838 }> OPTION_SETTING_BPS_SEL
839 .option_setting_bps_sel.endof (READONLY) :
840 {
841     __ddsc_OPTION_SETTING_BPS_SEL_END = .;
842
843 }> OPTION_SETTING_BPS_SEL
844
845 /* User region PLAINTEXT_DATA_S */
846 __plaintext_data_s$$ (NOLOAD) :
847 {
848     KEEP(*(.plaintext_data_s))
849 }> PLAINTEXT_DATA_S
850
851 }

```

Figure 18. Plaintext region defined in the linker script

In addition, all macros related to the start addresses and sizes of memory regions are generated automatically in Debug/bsp_linker_info.h file.

```

1  /* UNCRUSTIFY-OFF */
2  #ifndef BSP_LINKER_H
3  #define BSP_LINKER_H
4
5  /* Macro definitions
6  *****/
7
8  /* Solution Definitions *****/
9
10 #define BSP_PARTITION_RAM_CPU0_S_START (0x22000000)
11 #define BSP_PARTITION_RAM_CPU0_S_SIZE (0xE0000)
12 #define BSP_PARTITION_RAM_CPU0_C_START (0x220E0000)
13 #define BSP_PARTITION_RAM_CPU0_C_SIZE (0x0)
14 #define BSP_PARTITION_FLASH_CPU0_S_START (0x02000000)
15 #define BSP_PARTITION_FLASH_CPU0_S_SIZE (0x1F8000)
16 #define BSP_PARTITION_FLASH_CPU0_C_START (0x021F8000)
17 #define BSP_PARTITION_FLASH_CPU0_C_SIZE (0x0)
18 #define BSP_PARTITION_DATA_FLASH_CPU0_S_START (0x27000000)
19 #define BSP_PARTITION_DATA_FLASH_CPU0_S_SIZE (0x3000)
20 #define BSP_PARTITION_SDRAM_CPU0_S_START (0x68000000)
21 #define BSP_PARTITION_SDRAM_CPU0_S_SIZE (0x8000000)
22 #define BSP_PARTITION_OSPI0_CS0_CPU0_S_START (0x80000000)
23 #define BSP_PARTITION_OSPI0_CS0_CPU0_S_SIZE (0x10000000)
24 #define BSP_PARTITION_OSPI0_CS1_CPU0_S_START (0x90000000)
25 #define BSP_PARTITION_OSPI0_CS1_CPU0_S_SIZE (0x1000)
26 #define BSP_PARTITION_PLAINTEXT_DATA_S_START (0x90001000)
27 #define BSP_PARTITION_PLAINTEXT_DATA_S_SIZE (0xFFFF000)
28 #define BSP_PARTITION_ITCM_CPU0_S_START (0x00000000)
29 #define BSP_PARTITION_ITCM_CPU0_S_SIZE (0x10000)
30 #define BSP_PARTITION_DTCM_CPU0_S_START (0x20000000)
31 #define BSP_PARTITION_DTCM_CPU0_S_SIZE (0x10000)
32

```

Figure 19. Automatically generated memory region macros in bsp_linker_info.h file

Figure 20 is the plaintext data allocated to the section `.plaintext_data_s`.

```
/* writes to memory-mapped OSPI region restricted to 64-bit accesses */
uint8_t __attribute__((aligned(8))) plaintext_data[PLAINTEXT_DATA_SIZE] BSP_PLACE_IN_SECTION(".plaintext_data_s") = {
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
    0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F,
    0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F,
    0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F,
    0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C, 0x4D, 0x4E, 0x4F,
    0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, 0x5A, 0x5B, 0x5C, 0x5D, 0x5E, 0x5F,
    0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, 0x6A, 0x6B, 0x6C, 0x6D, 0x6E, 0x6F,
    0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79, 0x7A, 0x7B, 0x7C, 0x7D, 0x7E, 0x7F,
    0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89, 0x8A, 0x8B, 0x8C, 0x8D, 0x8E, 0x8F,
    0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98, 0x99, 0x9A, 0x9B, 0x9C, 0x9D, 0x9E, 0x9F,
    0xA0, 0xA1, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6, 0xA7, 0xA8, 0xA9, 0xAA, 0xAB, 0xAC, 0xAD, 0xAE, 0xAF,
    0xB0, 0xB1, 0xB2, 0xB3, 0xB4, 0xB5, 0xB6, 0xB7, 0xB8, 0xB9, 0xBA, 0xBB, 0xBC, 0xBD, 0xBE, 0xBF,
    0xC0, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7, 0xC8, 0xC9, 0xCA, 0xCB, 0xCC, 0xCD, 0xCE, 0xCF,
    0xD0, 0xD1, 0xD2, 0xD3, 0xD4, 0xD5, 0xD6, 0xD7, 0xD8, 0xD9, 0xDA, 0xDB, 0xDC, 0xDD, 0xDE, 0xDF,
    0xE0, 0xE1, 0xE2, 0xE3, 0xE4, 0xE5, 0xE6, 0xE7, 0xE8, 0xE9, 0xEA, 0xEB, 0xEC, 0xED, 0xEE, 0xEF,
    0xF0, 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7, 0xF8, 0xF9, 0xFA, 0xFB, 0xFC, 0xFD, 0xFE, 0xFF,
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
    0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F,
    0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F,
    0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F,
    0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C, 0x4D, 0x4E, 0x4F,
    0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, 0x5A, 0x5B, 0x5C, 0x5D, 0x5E, 0x5F,
    0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, 0x6A, 0x6B, 0x6C, 0x6D, 0x6E, 0x6F,
    0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79, 0x7A, 0x7B, 0x7C, 0x7D, 0x7E, 0x7F,
    0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89, 0x8A, 0x8B, 0x8C, 0x8D, 0x8E, 0x8F,
    0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98, 0x99, 0x9A, 0x9B, 0x9C, 0x9D, 0x9E, 0x9F,
    0xA0, 0xA1, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6, 0xA7, 0xA8, 0xA9, 0xAA, 0xAB, 0xAC, 0xAD, 0xAE, 0xAF,
    0xB0, 0xB1, 0xB2, 0xB3, 0xB4, 0xB5, 0xB6, 0xB7, 0xB8, 0xB9, 0xBA, 0xBB, 0xBC, 0xBD, 0xBE, 0xBF,
    0xC0, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7, 0xC8, 0xC9, 0xCA, 0xCB, 0xCC, 0xCD, 0xCE, 0xCF,
    0xD0, 0xD1, 0xD2, 0xD3, 0xD4, 0xD5, 0xD6, 0xD7, 0xD8, 0xD9, 0xDA, 0xDB, 0xDC, 0xDD, 0xDE, 0xDF,
    0xE0, 0xE1, 0xE2, 0xE3, 0xE4, 0xE5, 0xE6, 0xE7, 0xE8, 0xE9, 0xEA, 0xEB, 0xEC, 0xED, 0xEE, 0xEF,
    0xF0, 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7, 0xF8, 0xF9, 0xFA, 0xFB, 0xFC, 0xFD, 0xFE, 0xFF,
};
```

Figure 20. Plaintext OSPI Data Buffer

When the project is compiled, this plaintext data array is allocated to 0x90001000, which is the start of the plaintext data region.

If the `.plaintext_data_s` section is marked with `NOLOAD` attribute in the linker script (see Figure 18), then plaintext data array—declared with `BSP_PLACE_IN_SECTION(".plaintext_data_s")`—will lose its initialized values at runtime. To fix this, remove `NOLOAD` attribute from the linker script for the `plaintext_data_s` section.

Users need to copy the `fsp_gen.ld` file located in `Debug` folder into `script` folder, then rename it to a custom name (`fsp_gen_app.ld`) and remove `NOLOAD` attribute for `plaintext_data_s` section in that file, as shown in Figure 21.

In addition, users also need to copy the `fsp.ld` file and rename it to `fsp_app.ld`. This file only includes the entire content of the `fsp_gen_app.ld`.



Figure 21. Updated Memory Section “plaintext_data_s”

The updated linker script is named as `fsp_app.ld` and is configured to be used by the example project on the project **Properties** page.

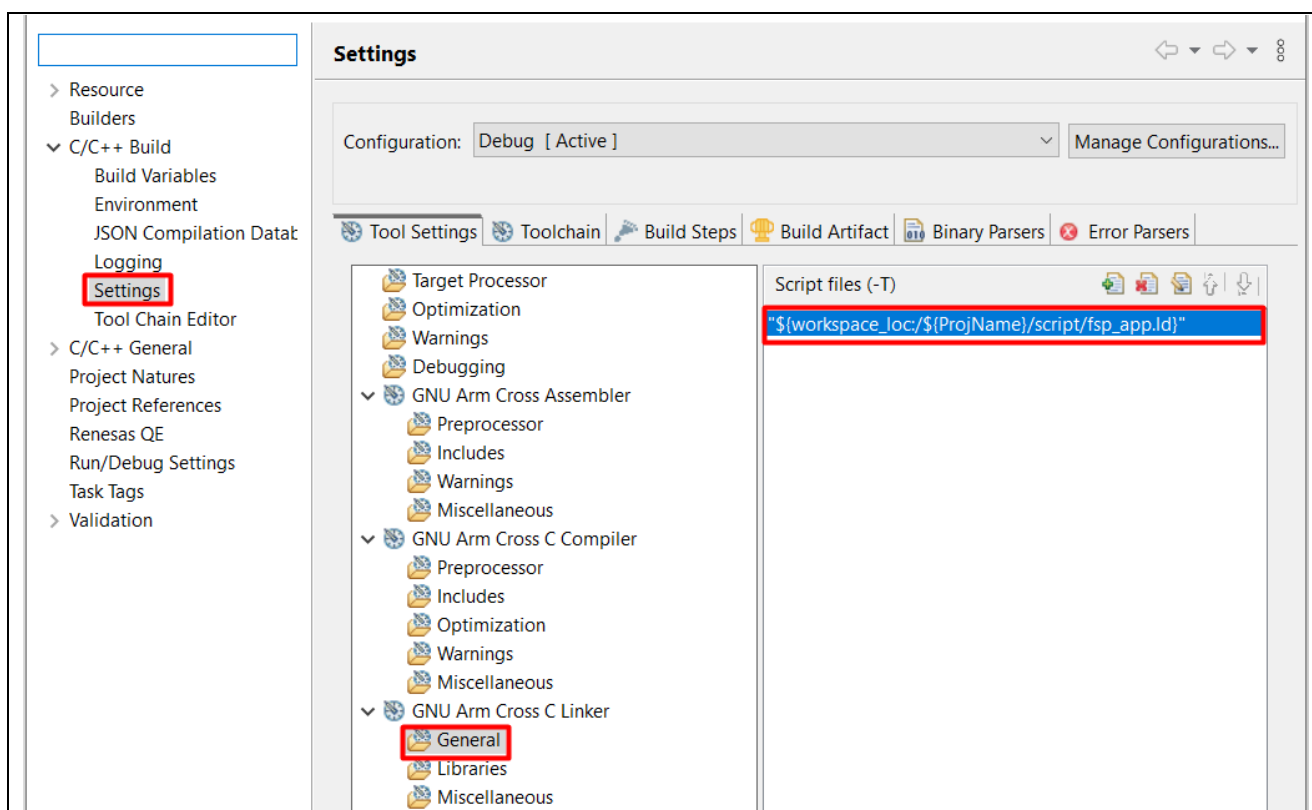


Figure 22. Configure to Use the Custom Linker Script

Note: The application code needs to guarantee that it does not write encrypted data outside the DOTF destination area. The application code should not write plaintext data to the DOTF destination area either. These rules must be followed for the DOTF application to operate correctly.

2.3.2 Allocation on RA8P1

Based on Figure 10, users can define memory partitions for both the encrypted and the plaintext regions in the OSPI area, as shown in Figure 23.

Memories					Generate Project Content	
Name	Start	Size	Core	Security		
RAM_NS	0x32000000	0x1D4000		Non-secure		
> RAM	0x22000000	0x1D4000		Secure		
FLASH_NS	0x12000000	0x100000		Non-secure		
> FLASH	0x02000000	0x100000		Secure		
DATA_FLASH_NS	0x37000000	0x0		Non-secure		
> DATA_FLASH	0x27000000	0x0		Secure		
> SDRAM	0x68000000	0x8000000				
> OSPI0_CS0	0x80000000	0x10000000				
▼ OSPI0_CS1	0x90000000	0x10000000				
OSPI0_CS1_CPU0_S	0x90000000	0x2000	CPU0	Secure		
PLAINTEXT_CPU0_S	0x90002000	0x2000	CPU0	Secure		
OSPI0_CS1_CPU1_S	0x90004000	0x2000	CPU1	Secure		
PLAINTEXT_CPU1_S	0x90006000	0x2000	CPU1	Secure		
> OSPI1_CS0	0x70000000	0x8000000				
> OSPI1_CS1	0x78000000	0x8000000				
OPTION_SETTING_OFS0	0x02C9F040	0x4		Secure		
OPTION_SETTING_OFS2	0x02C9F044	0x4		Secure		
OPTION_SETTING_SAS	0x02C9F074	0x4		Secure		
OPTION_SETTING_OFS1	0x12C9F4C0	0x4		Non-secure		
OPTION_SETTING_OFS1_SEC	0x02C9F0C0	0x4		Secure		
OPTION_SETTING_OFS1_SEL	0x02C9F120	0x4		Secure		
OPTION_SETTING_OFS3	0x12C9F4C4	0x4		Non-secure		
OPTION_SETTING_OFS3_SEC	0x02C9F0C4	0x4		Secure		
OPTION_SETTING_OFS3_SEL	0x02C9F124	0x4		Secure		
OPTION_SETTING_BPS	0x12C9F600	0x80		Non-secure		
OPTION_SETTING_BPS_SEC	0x02C9F200	0x80		Secure		
OPTION_SETTING_OTP_PBPS_SEC	0x02E07700	0x80		Secure		
OPTION_SETTING_OTP_PBPS	0x12E07780	0x80		Non-secure		
ITCM_NS	0x10000000	0x20000	CPU0	Non-secure		
> ITCM	0x00000000	0x20000	CPU0	Secure		
DTCM_NS	0x30000000	0x20000	CPU0	Non-secure		

Figure 23. Memory Partition: OSPI Area on RA8P1

After that, click **Generate Project Context** in the **Solution Project**, and then build the project. Similar to the RA8M1 (see Figure 18), the plaintext regions for CPU0 and CPU1 will also be generated in the `Debug/fsp_gen.ld` file.

Unlike the RA8M1, the plaintext data buffer is not allocated at compile time. Instead, it is loaded during runtime. The reason for this will be explained in detail in section 6.

2.4 Running the Example Application

2.4.1 Set up the Hardware and Import the Application

- For the EK-RA8M1, connect J10 to the Development PC using a USB Type-A male to Micro-B male cable.
- For the EK-RA8P1, connect J10 to the Development PC using a USB Type-A male to Type-C male cable.

This connection provides power, debug, and virtual COM port interface to the board with the default jumper settings.

Next, follow section 2.4.1.1 to Initialize the MCU, then power cycle the board (EK-RA8M1 or EK-RA8P1), and then erase the OSPI follow section 2.4.1.2 and power cycle the board. After this, the hardware is ready to run the DOTF applications.

2.4.1.1 Initialize the MCU

For a smooth evaluation, it is recommended to initialize the device using the RDPM or RFP prior to running this example project.

Open the Renesas Device Partition Manager (RDPM):

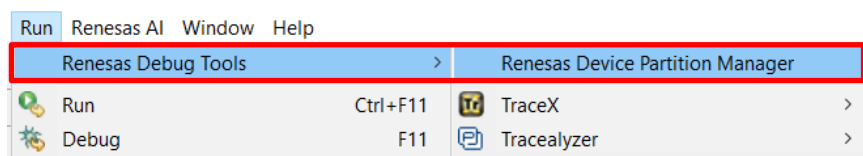


Figure 24. Open the Renesas Device Partition Manager

Next, check **Initialize device**, choose the connection method, then click **Run**.

For EK-RA8M1, choose either SCI or SWD as the connection method if the default jumpers are in place (Refer to the EK-RA8M1 User's Manual for the default jumper setting).

For EK-RA8P1, choose SWD as the connection method if the default jumpers are in place (Refer to the EK-RA8P1 User's Manual for the default jumper setting).

For a custom PCB board, the Connection Type should be selected based on the Boot Mode interfaces available.

In addition, for the EK-RA8P1, to enable the external flash functionality, ensure that the third switch on SW4 (on the board) is switched to the **OFF** position.

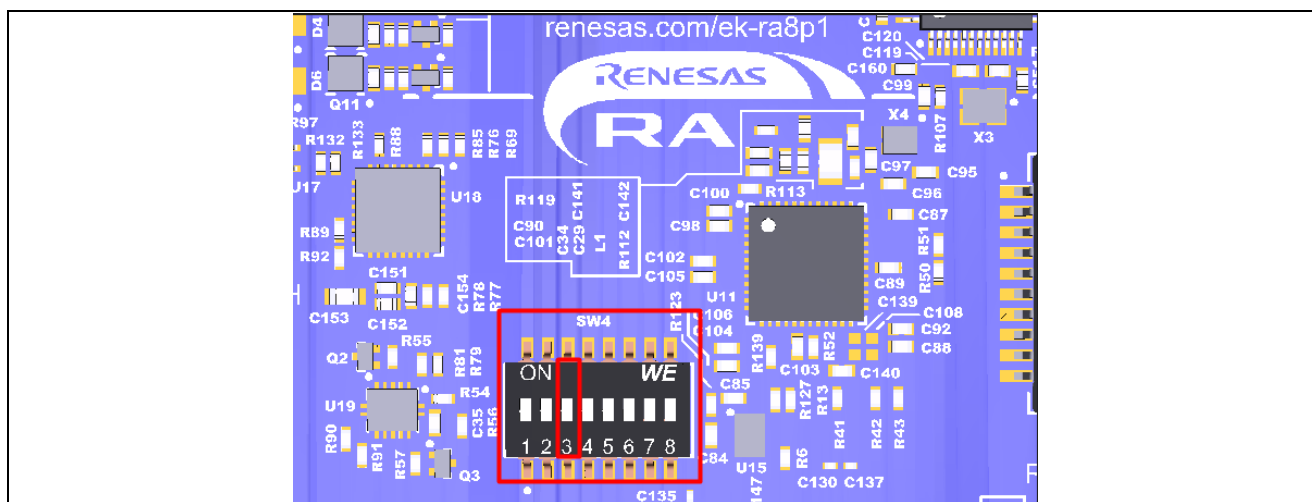


Figure 25. Turn off the third switch on SW4

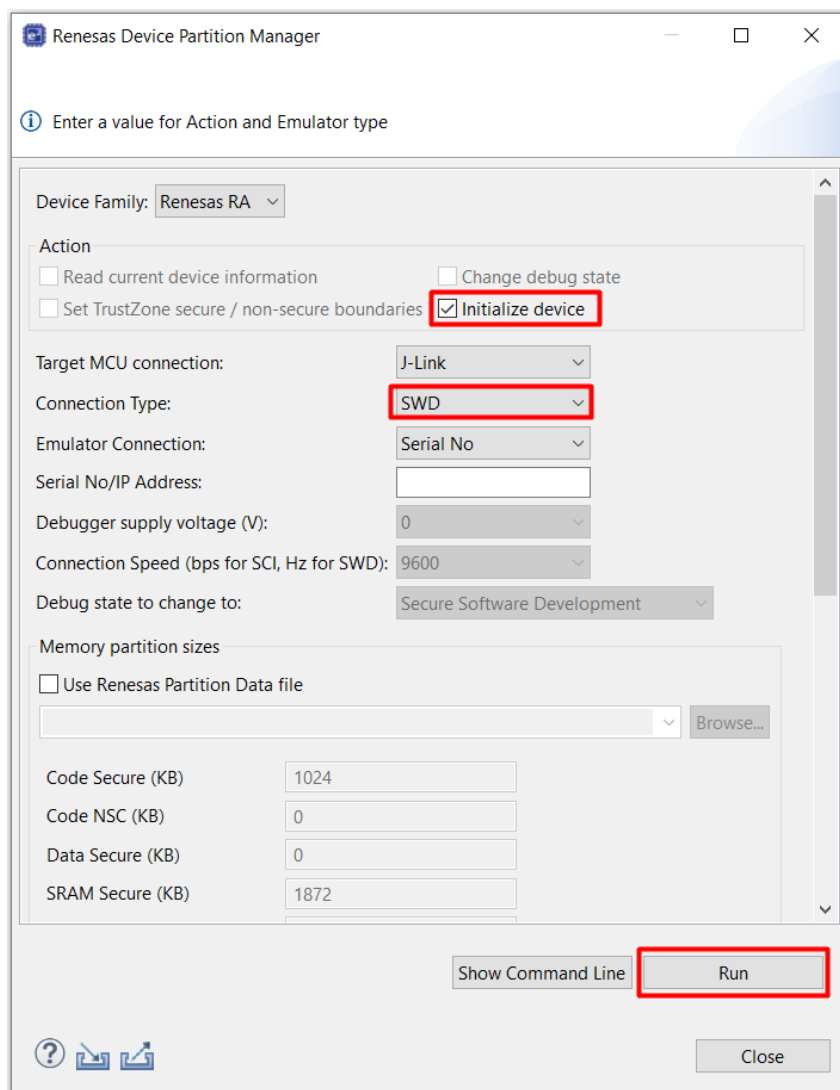


Figure 26. Initialize the MCU using Renesas Device Partition Manager

Ensure the following output is achieved and power cycle the board (EK-RA8M1 or EK-RA8P1).

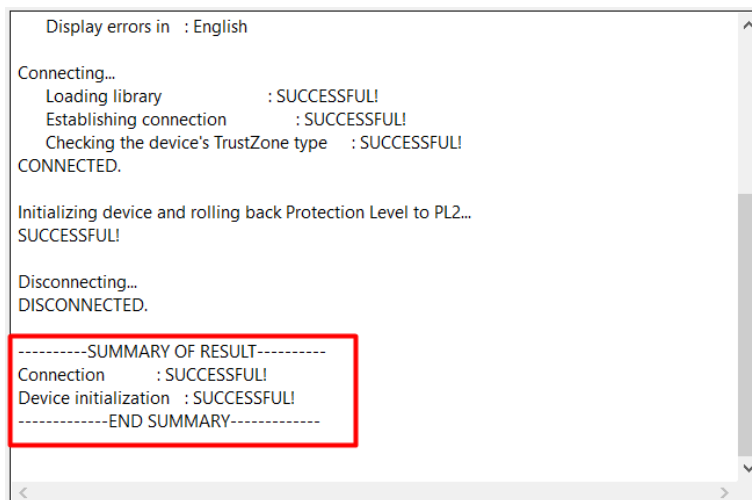


Figure 27. MCU Initialization Successful

2.4.1.2 Erase the OSPI

Additionally, perform these actions to erase the OSPI and then power cycle the board.

- For EK-RA8M1, unzip the `jlink_scripts.zip`, then double-click the `\jlink_scripts\ek_ra8m1\erase_ospi_8kB.bat` to erase the first 8kB of the OSPI area. Ensure the following output is achieved.

```
Reset: Reset device via AIRCR.SYSRESETREQ.
ResetTarget() end - Took 56.1ms
Erasing selected range...
J-Link: Flash download: Total time needed: 3.519s (Prepare: 0.299s, Compare: 0.000s, Erase: 3.071s, Program: 0.000s, Verify: 0.000s, Restore: 0.148s)
J-Link: Flash download:
Flash sectors within Range [0x90000000 - 0x90002000] deleted.
Erasing done.
J-Link>rx 100
Reset delay: 100 ms
Reset type NORMAL: Resets core & peripherals via SYSRESETREQ & VECTRESET bit.
ResetTarget() start
Reset: ARMv8M core with Security Extension enabled detected. Switch to secure domain.
Reset: Halt core after reset via DEMCR.VC_CORERESET.
Reset: Reset device via AIRCR.SYSRESETREQ.
ResetTarget() end - Took 58.4ms
J-Link>g
Memory map 'after startup completion point' is active
J-Link>r
Reset delay: 0 ms
Reset type NORMAL: Resets core & peripherals via SYSRESETREQ & VECTRESET bit.
Memory map 'before startup completion point' is active
ResetTarget() start
Reset: ARMv8M core with Security Extension enabled detected. Switch to secure domain.
Reset: Halt core after reset via DEMCR.VC_CORERESET.
Reset: Reset device via AIRCR.SYSRESETREQ.
ResetTarget() end - Took 60.4ms
Script processing completed.
```

Figure 28. Erase 8kB OSPI Using the Script

- For EK-RA8P1, unzip the `jlink_scripts.zip`, then double-click the `\jlink_scripts\ek_ra8p1\erase_ospi_16kB_CPU0.bat` script to erase the first 16kB of the OSPI area for CPU0, as shown in Figure 29.
- For CPU1, use `\jlink_scripts\ek_ra8p1\erase_ospi_16kB_CPU1.bat` script to erase the corresponding OSPI area, as shown in Figure 30.

Ensure the same expected output is observed. If the J-Link Script does not work in the user's environment, users can refer to section 6.2 for troubleshooting.

```
Reset: Reset device via AIRCR.SYSRESETREQ.
ResetTarget() end - Took 57.1ms
Device specific reset executed.
Erasing selected range...
J-Link: Flash download: Total time needed: 0.645s (Prepare: 0.247s, Compare: 0.000s, Erase: 0.287s, Program: 0.000s, Verify: 0.000s, Restore: 0.110s)
J-Link: Flash download:
Flash sectors within Range [0x90000000 - 0x90004000] deleted.
Erasing done.
J-Link>rx 100
Reset delay: 100 ms
ResetTarget() start
Reset: ARMv8M core with Security Extension enabled detected. Switch to secure domain.
Reset: Halt core after reset via DEMCR.VC_CORERESET.
Reset: Reset device via AIRCR.SYSRESETREQ.
ResetTarget() end - Took 58.0ms
Device specific reset executed.
J-Link>g
Memory map 'after startup completion point' is active
J-Link>r
Reset delay: 0 ms
Memory map 'before startup completion point' is active
ResetTarget() start
Reset: ARMv8M core with Security Extension enabled detected. Switch to secure domain.
Reset: Halt core after reset via DEMCR.VC_CORERESET.
Reset: Reset device via AIRCR.SYSRESETREQ.
ResetTarget() end - Took 57.3ms
Device specific reset executed.
Script processing completed.
```

Figure 29. Erase 16kB OSPI for CPU0 using the Script


```

ResetTarget() start
Secondary core selected. Reset is skipped.
ResetTarget() end - Took 261us
Device specific reset executed.
Erasing selected range...
J-Link: Flash download: Total time needed: 0.631s (Prepare: 0.221s, Compare: 0.000s, Erase: 0.320s, Program: 0.000s, Verify: 0.000s, Restore: 0.089s)
J-Link: Flash download:
Flash sectors within Range [0x90004000 - 0x90008000] deleted.
Erasing done.
J-Link>rx 100
Reset delay: 100 ms
ResetTarget() start
Secondary core selected. Reset is skipped.
ResetTarget() end - Took 559us
Device specific reset executed.
J-Link>g
Memory map 'after startup completion point' is active
J-Link>r
Reset delay: 0 ms
Memory map 'before startup completion point' is active
ResetTarget() start
Secondary core selected. Reset is skipped.
ResetTarget() end - Took 621us
Device specific reset executed.
Script processing completed.

```

Figure 30. Erase 16kB OSPI for CPU1 using the Script

- **Must Power cycle the EK-RA8M1 or EK-RA8P1 before the next step.**

For other OSPI applications that use more than 8kB or 16 kB OSPI area, execute the `erase_entire_ospi.bat` to erase the entire OSPI area. Erasing the entire OSPI device memory takes several minutes.

Next follow the “Importing an Existing Project into e² studio” section in the FSP User’s Manual to import the Compatibility Mode project:

- `dotf_rsip_compatibility_mode_ek_ra8m1.zip` for EK-RA8M1, or
- `dotf_rsip_compatibility_mode_ek_ra8p1.zip` for EK-RA8P1.

After importing the project, users need to follow these steps:

1. Open the RA Configurator

- Double-click on `configuration.xml` located in the project:
 - `dotf_rsip_compatibility_mode_ra8m1_CM85` for EK-RA8M1.
 - `dotf_rsip_compatibility_mode_ra8p1_CPU0`, `dotf_rsip_compatibility_mode_ra8p1_CPU1` for EK-RA8P1.
- Click **Generate Project Content** to apply the configuration settings.

Note: For the EK-RA8P1, users must **Generate Project Content** and **Build** the CPU0 project first before opening the `configuration.xml` of the CPU1 project in RA Configurator. Otherwise, a Smart Bundle Error will occur.

2. Build the solution project

- Right-click on the solution project:
 - `dotf_rsip_compatibility_mode_ra8m1` for EK-RA8M1.
 - `dotf_rsip_compatibility_mode_ra8p1` for EK-RA8P1.
- Select **Build Project**. This command will build all projects within the solution project.

Note: There are warnings generated from the third-party libraries, which can be ignored.

2.4.2 Launch the Debug Session and Observe the Demonstration

Next, launch the e2studio Debug session. Once the `Reset_Handler` is hit, launch **Tera Term** and select the enumerated COM port (JLink CDC UART Port).

Note: The COM number may be different for your setup.

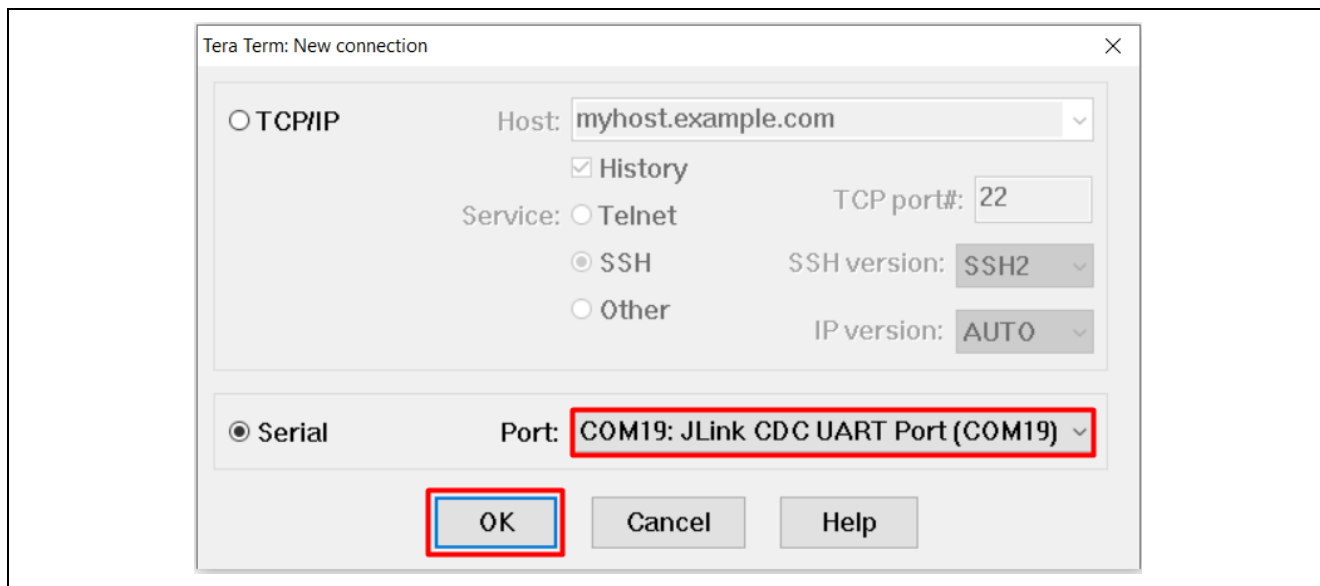


Figure 31. Select the JLink Console Connection

Once the COM port is open, navigate to the **Setup** tab and select **Serial port**.

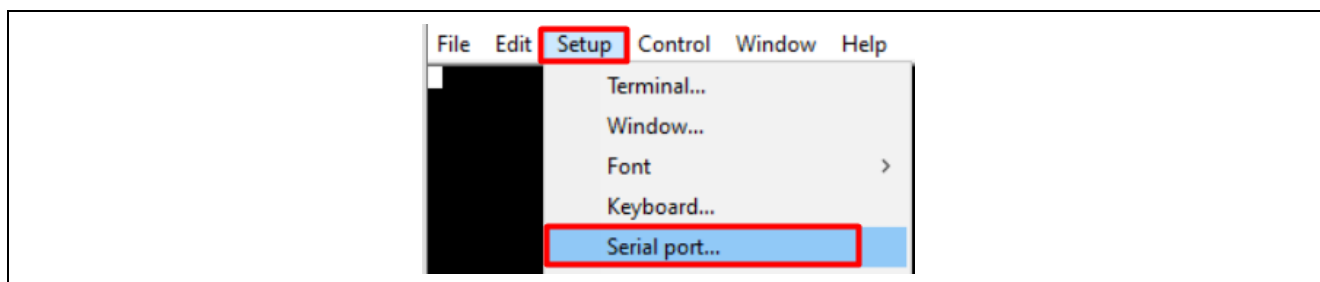


Figure 32. Open the “Serial port” interface

Update the **Speed** to **115200** and click **New setting** to commit the update.

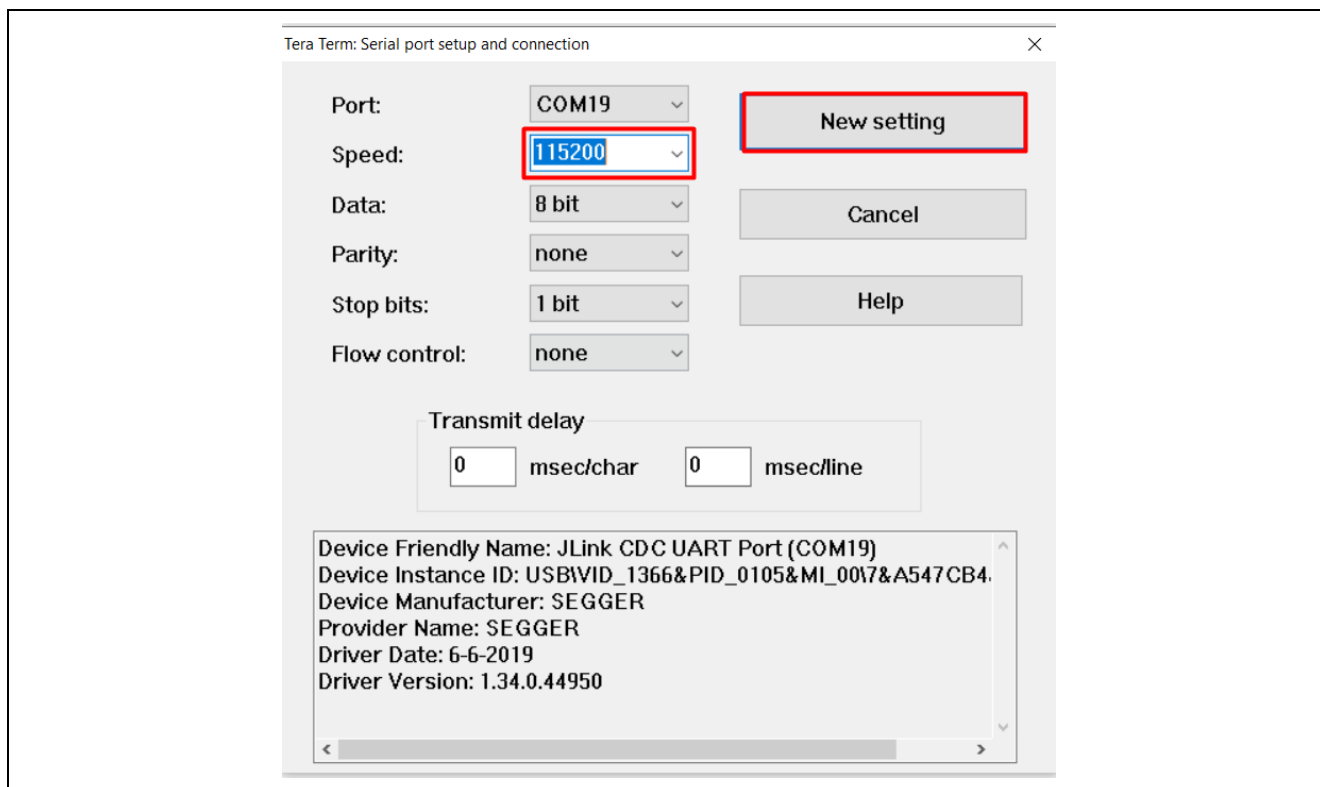


Figure 33. Configure the Tera Term

Resume the Debug session. Similar output as shown in Figure 34 should be observed in the Tera Term terminal. Time used to access the DOTF area is faster than the time used to access the plaintext OSPI area. The blue LED should be blinking after the evaluation is done.



Figure 34. Access Plaintext and DOTF Decrypted Data in Compatibility Mode on RA8M1

For EK-RA8P1, to run the application, right-click on `dotf_rsip_compatibility_mode_ra8p1_CPU0` and select **Debug As > Debug Configurations**, then select the **Launch Group > dotf_rsip_compatibility_mode_ra8p1_CPU1 Debug_Multicore Launch Group** to debug the dual core.

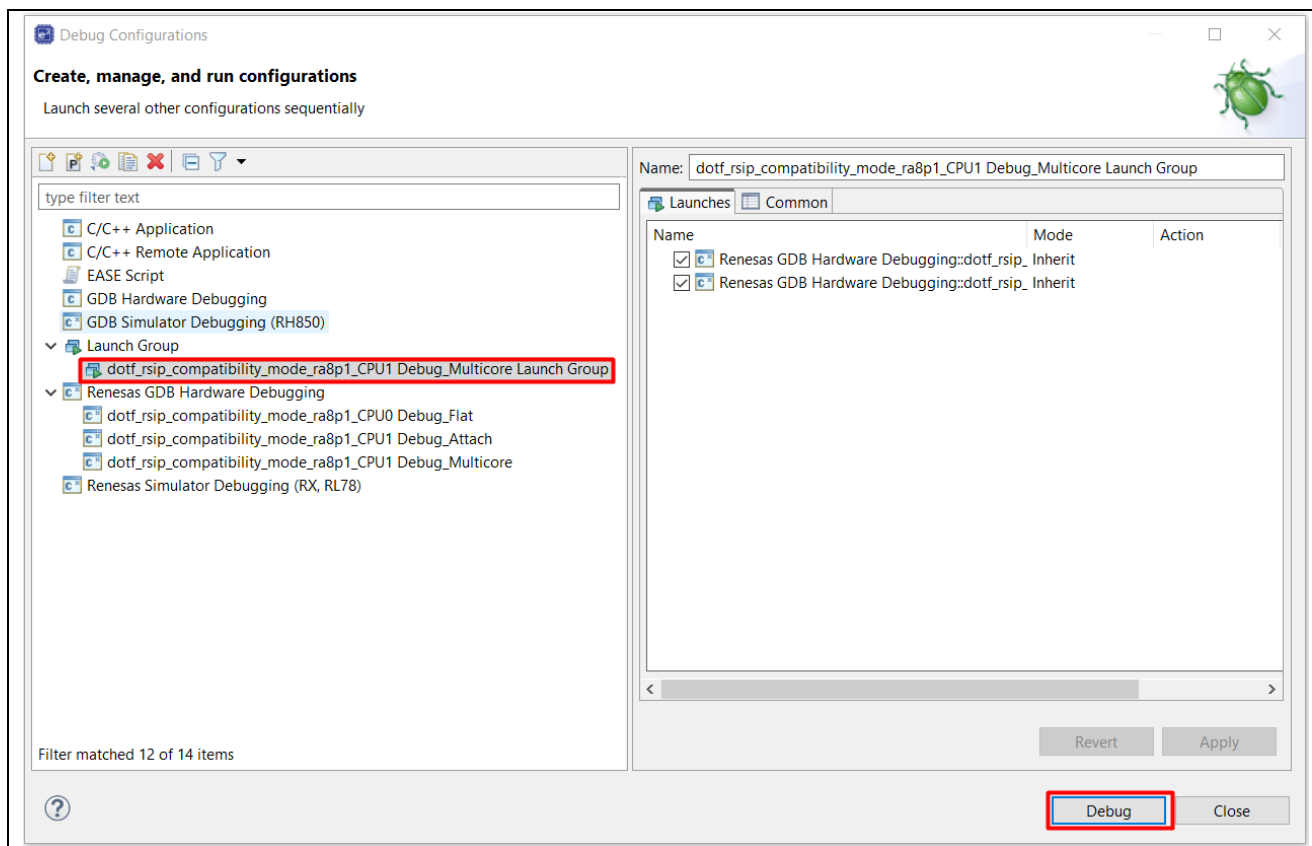


Figure 35. Debug the dual core

Click  twice to run the project and observe the expected output, as shown in Figure 36.



Figure 36. Access Plaintext and DOTF Decrypted Data in Compatibility Mode on RA8P1 (CPU0)

The blue LED should blink five times, then click  again to start CPU1. The green LED should be blinking after the evaluation is done, as shown in Figure 37.

```
Compatibility Mode: Time used to access the plaintext OSPI data without using DOTF: 55 microseconds;
Compatibility Mode: Time used to access the encrypted OSPI data using DOTF: 69 microseconds;
Compatibility Mode: Time used to access the plaintext OSPI data without using DOTF: 55 microseconds;
Compatibility Mode: Time used to access the encrypted OSPI data using DOTF: 69 microseconds;
Compatibility Mode: Time used to access the plaintext OSPI data without using DOTF: 55 microseconds;
Compatibility Mode: Time used to access the encrypted OSPI data using DOTF: 69 microseconds;
Compatibility Mode: Time used to access the plaintext OSPI data without using DOTF: 55 microseconds;
Compatibility Mode: Time used to access the encrypted OSPI data using DOTF: 69 microseconds;
Compatibility Mode: Time used to access the plaintext OSPI data without using DOTF: 55 microseconds;
Compatibility Mode: Time used to access the encrypted OSPI data using DOTF: 69 microseconds;
Compatibility Mode: Time used to access the plaintext OSPI data without using DOTF: 55 microseconds;
Compatibility Mode: Time used to access the encrypted OSPI data using DOTF: 69 microseconds;
DOTF RSIP Compatibility Mode evaluation finished successfully on CPU1.
```

Figure 37. Access Plaintext and DOTF Decrypted Data in Compatibility Mode on RA8P1 (CPU1)

Note: Users need to be aware of the LLVM toolchain optimizations. In the Compatibility Mode project on RA8P1, a buffer is used to copy data from DOTF and non-DOTF areas for timing measurement purposes. If the buffer is not used afterward, the LLVM toolchain may optimize it out, which can lead to inaccurate measurement results. To prevent this, users can apply the `__attribute__((used))` qualifier to the buffer declaration.

```
static uint8_t BSP_ALIGN_VARIABLE(8) ospi_read_data[ENCRYPTED_DATA_SIZE] __attribute__((used)) = {RESET_VALUE};
```

Figure 38. Preventing LLVM Toolchain Optimization

Furthermore, when Data Cache is enabled, encrypted data accessed through DOTF can appear faster than plaintext because the DOTF block uses a one-line cache, 128B buffering, and a tightly coupled AES data path to deliver burst-aligned cache line fills efficiently. In contrast, bypassing DOTF results in narrower, fragmented OSPI fetches that take more cycles. With Data Cache disabled, these buffering benefits disappear, and the AES decryption overhead becomes directly visible.

3. Example Implementation: Using DOTF with RSIP Protected Mode

This section describes the establishment of a DOTF application using RSIP Protected Mode. A wrapped AES128 key is generated and injected as the DOTF key. SKMT is used to encrypt the function allocated to the OSPI area. The LLVM embedded toolchain for Arm version 18.1.3 is used in this example project. Data Cache is also enabled to achieve better system performance.

3.1 Tools Used in the DOTF Design with RSIP Protected Mode

There are three tools used in the DOTF design with RSIP Protected Mode besides the IDE.

- **Gpg4win**

This tool is used in the process of the Wrapped UFPK. It is used to establish a PGP encrypted communication channel between user and the Renesas Key Wrap server. Using this tool, the user can generate a user PGP key pair, perform key exchange with the Renesas DLM server, and assist the reception of the W-UFPK.

- **Renesas Security Key Management Tool (SKMT)**

This tool is used to encrypt the plaintext data or code as an .srec file which can be included in the e2studio Debug configuration and programmed to the MCU using the J-Link driver. In this Application Note, two ways of encrypting the DOTF region code are demonstrated. Both methods use the SKMT tool to encrypt the DOTF region code, the difference is whether the Graphic User Interface is used (as demonstrated in section 3.6) or the Command Line Interface (CLI) is used (as demonstrated in section 3.7).

- **Renesas Flash Programmer (RFP)**

This tool is used to inject the Wrapped DOTF key through the MCU boot interfaces (SCI UART, USB or SWD/JTAG). RFP is used as a demonstration for the general operation of secure key injection.

3.2 Creating the Wrapped DOTF Key

A wrapped AES128 key is generated as the DOTF key. The process of generating the wrapped DOTF key uses the same procedure as wrapping a user key. This can be achieved following the procedure below.

For the convenience of evaluating the DOTF operation, the wrapped DOTF Key that matches the included example project are included in the application project: the DOTF_AES_128_RA8M1.rkey, DOTF_AES_128_RA8P1_CPU0.rkey, DOTF_AES_128_RA8P1_CPU1.rkey. These keys can be used in section 3.6.2.2. If the included wrapped DOTF key is used, there is no need to generate another wrapped DOTF key and this entire section can be skipped.

1. Generate the Wrapped User Factory Programming Key (refer to section Wrapping the User Factory Programming Key Using the Renesas Key Wrap Service in R11AN0785)
2. Generate the Wrapped AES128 bit key for the RSIP protected mode.
 - 1) Select security engine:

RA Family, RSIP-E51A Security Functions and Protected Mode for RA8M1

RA Family, RSIP-E50D Security Functions and Protected Mode for RA8P1

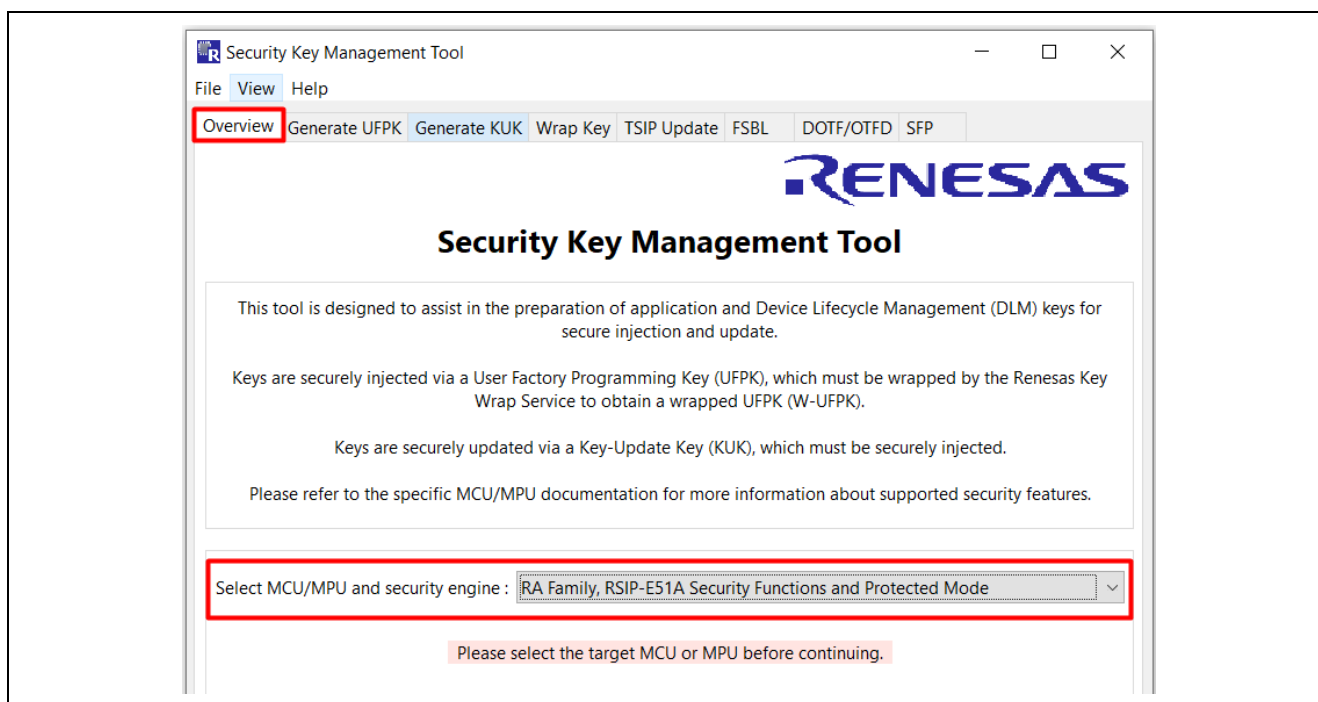


Figure 39. Select RSIP Protected Mode

- 2) Under the **Wrap Key** tab, select **AES-128** bits as the **Key Type**. Browse for the **UFPK** and **W-UFPK**, select **Use specified value** and name the wrapped key
(eg. DOTF_AES_128_RA8M1.rkey, DOTF_AES_128_RA8P1_CPU0.rkey, DOTF_AES_128_RA8P1_CPU1.rkey)

The screenshot shows the 'Wrap Key' tab in the Renesas RA MCU configuration tool. The 'Key Type' section has 'AES' selected with '128 bits'. The 'Wrapping Key' section has 'UFPK' selected, with 'UFPK File' and 'W-UFPK File' paths. The 'IV' section has 'Use specified value' selected with a hex value. The 'Output' section has 'Format' set to 'RFP' and 'File' path.

Figure 40. Select AES-128 and Provide WUFPK

- 3) Next, provide **Key Data** and then click **Generate File**. The wrapped AES key is now generated.

Overview Generate UFPK Generate KUK **Wrap Key** TSIP Update FSBL DOTF/OTFD SFP

Keys must be wrapped by the UFPK for secure injection or by the KUK for secure update.

Key Type Key Data

☐ File

☒ **Raw** FFFFFFFFFFFFFFFFFFFFFFFFFF0

☐ Random - Output File

Wrapping Key

☒ **UFPK** UFPK File: C:\RA8_DOTF\ufpk.key

W-UFPK File: C:\RA8_DOTF\ufpk.key_enc.key

☐ **KUK** KUK File:

IV

☐ Generate random value

☒ Use specified value (16 hex bytes, big endian format) 00000000000000000000000009000000

Output

Format: RFP File: C:\RA8_DOTF\DOTF_AES_128_RA8M1.rkey

Endian: Little ☐ Output additional data

Address: 10000 Key name:

IV: 00000000000000000000000009000000
 Encrypted key: 62C862EE929BDAD6F5132600D5E5221EE8BFC8B418B899855D16B1DC974C704B
 OPERATION SUCCESSFUL

Figure 41. Generate the Wrapped DOTF Key

Note: To set the IV value correctly, refer to section 1.4.3. For users convenience during the wrapped DOTF key generation process, the IV value is provided in the table below.

Table 2. The IV value based on the destination address

Device	The IV value
EK-RA8M1	00000000000000000000000009000000
EK-RA8P1 (CPU0)	00000000000000000000000009000000
EK-RA8P1 (CPU1)	00000000000000000000000009000400

3.3 Configure the Application Project with RSIP in Protected Mode

When using DOTF with the RSIP Protected Mode, the **Wrapped Key Format** must be used. The OSPI memory map of the DOTF RSIP Protected Mode example project is same as the DOTF RSIP Compatibility Mode example project.

Figure 42 is the key DOTF configurations for the Protected Mode example project on RA8M1.

g_ospi_b OSPI (r_ospi_b)		
Settings	Property	Value
API Info	▼ Common	
	> Memory-mapping Support	
	Parameter Checking	Default (BSP)
	DMAC Support	Enable
	Autocalibration Support	Disable
	DOTF Support	Enable (Protected Mode)
	Row Addressing Support	Disable
	▼ Module g_ospi_b OSPI (r_ospi_b)	
	> General	
	> Command Sets	
	> Timing Settings	
	> XiP Mode	
	▼ DOTF	
	Name	g_ospi_dotf
	AES Key	encryption_key
	AES IV	encryption_iv
	AES Key Length	128
	Key Format	Wrapped
	Decryption start address	0x90000000
	Decryption end address	0x90000FFF
	> Row Addressing	

Figure 42. Configure the DOTF with Protected Mode on RA8M1

Figure 43 and Figure 44 show the key DOTF configurations for the Protected Mode example project on RA8P1 (CPU0 and CPU1).

g_ospi_b OSPI (r_ospi_b)		
Settings	Property	Value
API Info	▼ Common	
	> Memory-mapping Support	
	Parameter Checking	Default (BSP)
	DMAC Support	Enable
	Autocalibration Support	Disable
	DOTF Support	Enable (Protected Mode)
	Row Addressing Support	Disable
	▼ Module g_ospi_b OSPI (r_ospi_b)	
	> General	
	> Command Sets	
	> Timing Settings	
	> XiP Mode	
	▼ DOTF	
	Name	g_ospi_dotf
	AES Key	encryption_key
	AES IV	encryption_iv
	AES Key Length	128
	Key Format	Wrapped
	Decryption start address	0x90000000
	Decryption end address	0x90001FFF
	> Row Addressing	

Figure 43. Configure the DOTF with Protected Mode on RA8P1 (CPU0)

g_ospi_b OSPI (r_ospi_b)		
Settings	Property	Value
API Info	▼ Common	
	> Memory-mapping Support	
	Parameter Checking	Default (BSP)
	DMAC Support	Enable
	Autocalibration Support	Disable
	DOTF Support	Enable (Protected Mode)
	Row Addressing Support	Disable
	▼ Module g_ospi_b OSPI (r_ospi_b)	
	> General	
	> Command Sets	
	> Timing Settings	
	> XiP Mode	
	▼ DOTF	
	Name	g_ospi_dotf
	AES Key	encryption_key
	AES IV	encryption_iv
	AES Key Length	128
	Key Format	Wrapped
	Decryption start address	0x90004000
	Decryption end address	0x90005FFF
	> Row Addressing	

Figure 44. Configure the DOTF with Protected Mode on RA8P1 (CPU1)

As in the Compatibility Mode example project, the `r_sci_b_uart` module is used for the J-link virtual console to communicate with a PC end terminal (eg. Tera Term). The virtual console can be used to print the system status information, for example error message or time usage information.

Selected software components
Board Support Package Common Files
I/O Port
Arm CMSIS Version 6 - Core (M)
Board support package for R7FA8M1AHECBD
Board support package for RA8M1
Board support package for RA8M1 - FSP Data
Board support package for RA8M1 - Events
Board support package for RA8M1 - Linker
RA8M1-EK Board Support Files
Direct Memory Access Controller
Octa Serial Peripheral Interface Flash
Renesas Secure IP (RSIP-E51A) Protected Mode
SCI UART

Figure 45. Software Components used for DOTF example with RSIP Protected Mode on RA8M1

Selected software components

Board Support Package Common Files
 I/O Port
 Arm CMSIS Version 6 - Core (M)
 Board support package for R7KA8P1KFLCAC
 Board support package for RA8P1
 Board support package for RA8P1 - FSP Data
 Board support package for RA8P1 - Events
 Board support package for RA8P1 - Linker
 RA8P1-EK Board Support Files
 Direct Memory Access Controller
 Octa Serial Peripheral Interface Flash
 Renesas Secure IP (RSIP-E50D) Protected Mode
 SCI UART

Figure 46. Software Components used for DOTF example with RSIP Protected Mode on RA8P1

3.4 Update the Linker Script

In this example application project, the wrapped DOTF key will be injected to the beginning of the Data Flash region (refer to section 3.7.1.2), which starts at 0x27000000 on RA8M1.

By default, e2studio erases the Code and Data flash prior to downloading the application image. To avoid erasing the injected DOTF key, the linker script is updated to disable the data_flash erase by defining the .data_flash region as "(NOLOAD)".

```

__data_flash_readonly$$ (NOLOAD)
{
    __data_flash_readonly$$Base = .;
    /* section.data_flash.readonly */
    *(.data_flash)
    /* section.data_flash.code */
    *(.data_flash_code)
    __data_flash_readonly$$Limit = .;
}> DATA_FLASH
  
```

Figure 47. Set NOLOAD for the dash_flash section on RA8M1

For RA8P1, the wrapped DOTF key will be injected into MRAM are instead. Please refer to section 3.7.1.2.

In addition, users need to remove the NOLOAD attribute for __plaintext_cpu0_s and __plaintext_cpu1_s sections in both CPU0 and CPU1 projects.

```

/* User region PLAINTEXT_CPU0_S */
__plaintext_cpu0_s$$ :
{
    KEEP(*(.plaintext_cpu0_s))
}> PLAINTEXT_CPU0_S

/* User region PLAINTEXT_CPU1_S */
__plaintext_cpu1_s$$ :
{
    KEEP(*(.plaintext_cpu1_s))
}> PLAINTEXT_CPU1_S
  
```

Figure 48. Remove NOLOAD for the Plaintext section on RA8P1

The updated linker script `fsp_app.ld` is configured to be used in the example project for both RA8M1 and RA8P1.

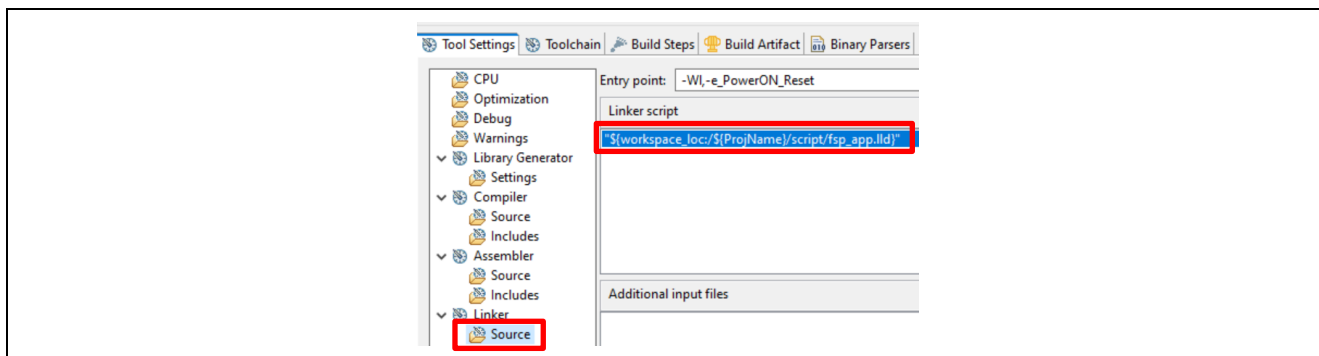


Figure 49. Configure to Use the Custom Linker Script

3.5 Allocating Code to the DOTF Destination Area

The DOTF demonstration project using the RSIP Protected Mode allows evaluation of DOTF for both data storage and code execution, but only one at a time. The protected mode project enables DOTF evaluation of executing encrypted code in the OSPI area.

For RA8M1, the memory partitioning is the same as in Compatibility Mode. Refer to section 2.3.1 for more details.

For RA8P1, please refer to section 2.3.2 for its memory partition details.

The application code needs to guarantee not allocating encrypted data or code outside the DOTF destination area, and the application code should not allocate plaintext data or code to the DOTF destination area. These rules must be followed for the DOTF application to operate correctly.

In the DOTF RSIP Protected Mode example project, one function `fibonacci` is allocated to the encrypted OSPI area.

```
static uint32_t fibonacci(uint32_t num) __attribute__((noinline)) __attribute__((aligned(4096))) BSP_PLACE_IN_SECTION(".ospi0_cs1_code");
```

Figure 50. Allocate a Function to DOTF Destination Area (to be Encrypted by SKMT) on RA8M1

When the project is compiled, the plaintext data of this function is allocated to 0x90000000 in the `.srec` file. The SKMT is then used to encrypt this area and generate the encrypted version of this function. The encrypted function will then be programmed to the encrypted OSPI area through the e2studio Debugging process.

The same functionality is implemented in another function named `fibonacci2`. It is allocated to the beginning of the plaintext OSPI area.

```
static uint32_t fibonacci2(uint32_t num) __attribute__((noinline)) __attribute__((aligned(4096))) BSP_PLACE_IN_SECTION(".plaintext_data_s");
```

Figure 51. Allocate a Function to OSPI Plaintext Area on RA8M1

When the project is compiled, this plaintext function is allocated to 0x90001000.

For RA8P1, the function should also be allocated to the DOTF Destination, and Plaintext Area is configured similarly to RA8M1.

3.6 Import and Build the RSIP Protected Mode Example Project

Follow the “Importing an Existing Project into e² studio” section in the FSP User’s Manual to import the Protected Mode project:

- `dotf_rsip_protected_mode_ek_ra8m1.zip` for RA8M1, or
- `dotf_rsip_protected_mode_ek_ra8p1.zip` for RA8P1.

After importing the project, users need to follow these steps:

1. Open the RA Configurator

- Double-click on `configuration.xml` located in the project:
 - `dotf_rsip_protected_mode_ra8m1_CM85` for EK-RA8M1.
 - `dotf_rsip_protected_mode_ra8p1_CPU0`,
`dotf_rsip_protected_mode_ra8p1_CPU1` for EK-RA8P1.
- Click **Generate Project Content** to apply the configuration settings.

Note: For the EK-RA8P1, users must **Generate Project Content** and **Build** the CPU0 project first before opening the `configuration.xml` of the CPU1 project in RA Configurator. Otherwise, a Smart Bundle Error will occur.

2. Build the solution project

- Right-click on the solution project:
 - `dotf_rsip_protected_mode_ra8m1` for EK-RA8M1.
 - `dotf_rsip_protected_mode_ra8p1` for EK-RA8P1.
- Select **Build Project**. This command will build all projects within the solution project.

Note: There are some warnings from the third-party libraries.

3.6.1 Encrypt the DOTF Destination Area Using the SKMT CLI

The example project `dotf_rsip_protected_mode_ek_ra8m1` integrated a custom builder that is included in the project as shown in Figure 52.

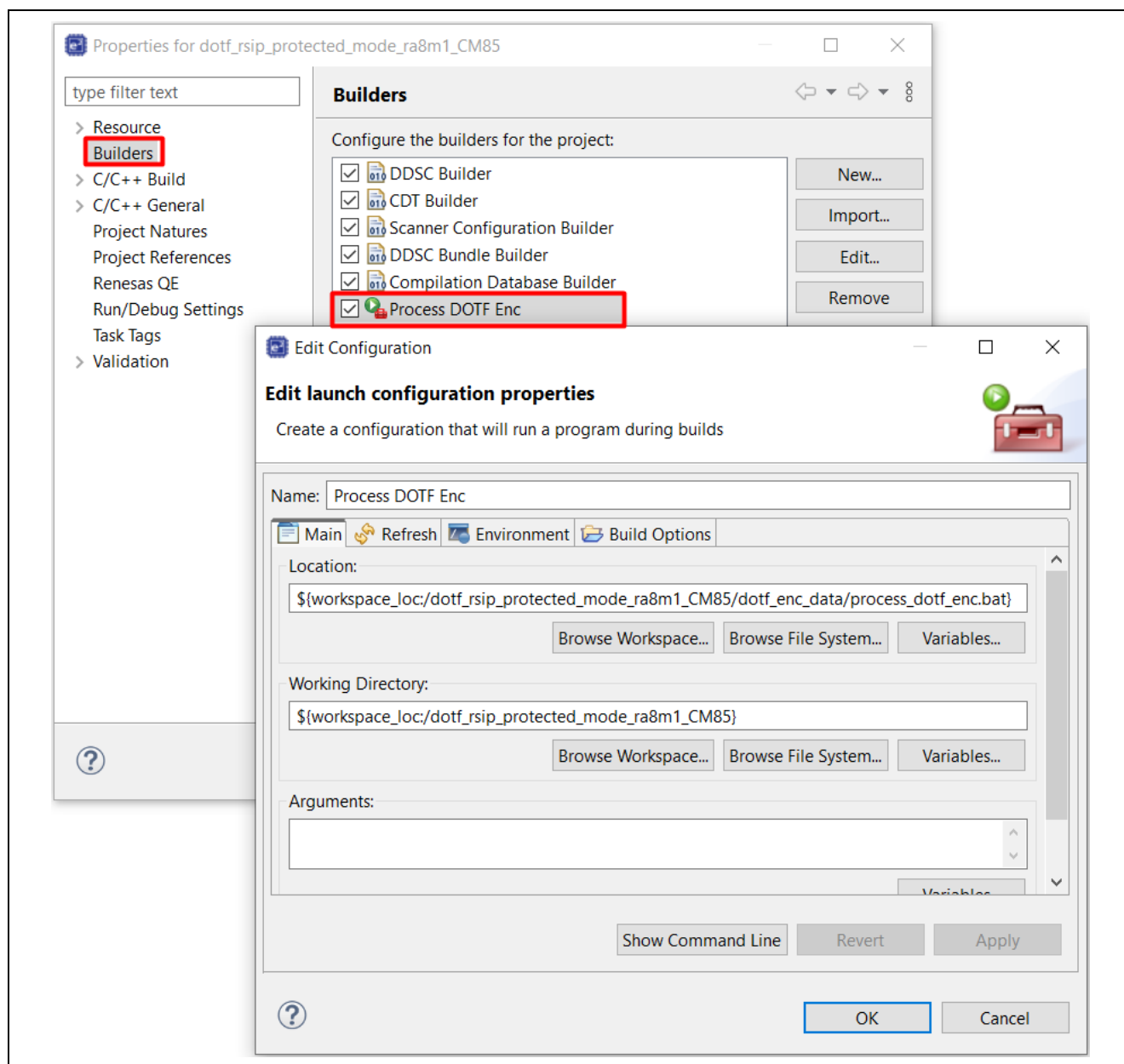


Figure 52. Add a Custom Builder

Figure 53 is the content of the `process_dotf_enc.bat` file. The `srec_cat.exe` is used to generate the OSPI area image before encryption: `ra_app_image_ospi_area.srec` and the code flash image: `ra_app_image_wo_ospi_area.srec`. The `skmt.exe` is then used to encrypt the OSPI data which is intended for the DOTF destination. This process generates an OSPI image (`ra_app_image_ospi_area_encrypted_and_plaintext.srec`) which includes the encrypted DOTF destination area data as well as the plaintext OSPI area. This `.srec` file can be programmed to the OSPI DOTF area with a Debug session or an OEM third party tool.

```
cd dotf_enc_data
srec_cat.exe ..\Debug\dotf_rspi_protected_mode_ra8m1_CM85.srec -crop 0x00000000 0x7FFFFFFF -o ra_app_image_wo_ospi_area.srec
srec_cat.exe ..\Debug\dotf_rspi_protected_mode_ra8m1_CM85.srec -crop 0x80000000 0x9FFFFFFF -o ra_app_image_ospi_area.srec
dotf_cli_v110\skmt.exe /encdotf /keytype "AES-128" /enckey "FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF" /nonce "00000000000000000000000000000000" /startaddr "90000000"
/endaddr "90000FFF" /prg "../ra_app_image_ospi_area.srec" /incplain /output "../ra_app_image_ospi_area_encrypted_and_plaintext.srec"
```

Figure 53. Functionality of the Custom Builder (process_dotf_enc.bat)

The \dotf_enc_data folder includes the srec_cat.exe, the skmt.exe and its supporting utilities.

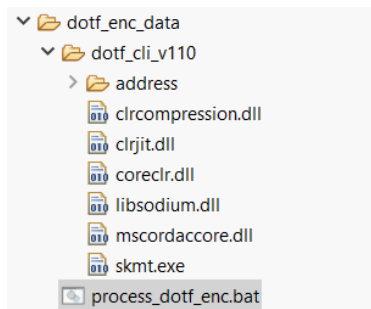


Figure 54. Include the skmt.exe in the e2studio Project

With the custom builder enabled, compile the application. The custom builder will generate ra_app_image_ospi_area_encrypted_and_plaintext.srec and ra_app_image_wo_ospi_area.srec.

```
C:\RA8_DOTF\dotf_rsip_protected_mode_ra8m1_CM85>cd dotf_enc_data

C:\RA8_DOTF\dotf_rsip_protected_mode_ra8m1_CM85\dotf_enc_data>srec_cat.exe ..\Debug\dotf_rsip_protected_mode_ra8m1_CM85.srec -crop 0x00000000 0x7FFFFFFF -o
ra_app_image_wo_ospi_area.srec

C:\RA8_DOTF\dotf_rsip_protected_mode_ra8m1_CM85\dotf_enc_data>srec_cat.exe ..\Debug\dotf_rsip_protected_mode_ra8m1_CM85.srec -crop 0x80000000 0x9FFFFFFF -o
ra_app_image_ospi_area.srec

C:\RA8_DOTF\dotf_rsip_protected_mode_ra8m1_CM85\dotf_enc_data>dotf_cli_v110\skmt.exe /encdotf /keytype "AES-128" /enckey "FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF" /nonce
"00000000000000000000000000000000" /startaddr "90000000" /endaddr "9000FFFF" /prg ".\ra_app_image_ospi_area.srec" /incplain /output
"/ra_app_image_ospi_area_encrypted_and_plaintext.srec"
Output File: C:\RA8_DOTF\dotf_rsip_protected_mode_ra8m1_CM85\dotf_enc_data\ra_app_image_ospi_area_encrypted_and_plaintext.srec
Key: FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0
Counter: 00000000000000000000000000000000
```

Figure 55. Custom Builder Execution Result

All actions described in this section should be performed similarly for RA8P1.

3.6.2 Encrypt the DOTF Destination Area using SKMT GUI

Figure 56 is an example of using the SKMT GUI to encrypt a section of data between 0x90000000 and 0x90000FFF. The Image Encryption Key uses a Raw 128-bit key and a specified 128 bit IV is used. The Image Encryption Key and IV must match the Wrapped DOTF Key and the IV used in the key wrapping. The image encryption key and the same IV must be accessible from the application project for the DOTF to function. The DOTF accesses the wrapped key via its location in the application project. When RSIP protected mode is used, the location of the wrapped key is as global variables for the OSPI driver to access. The generated dotf_rsip_protected_mode_ra8m1.mot includes the code flash content, the encrypted DOTF destination area content as well as the plaintext OSPI area content.

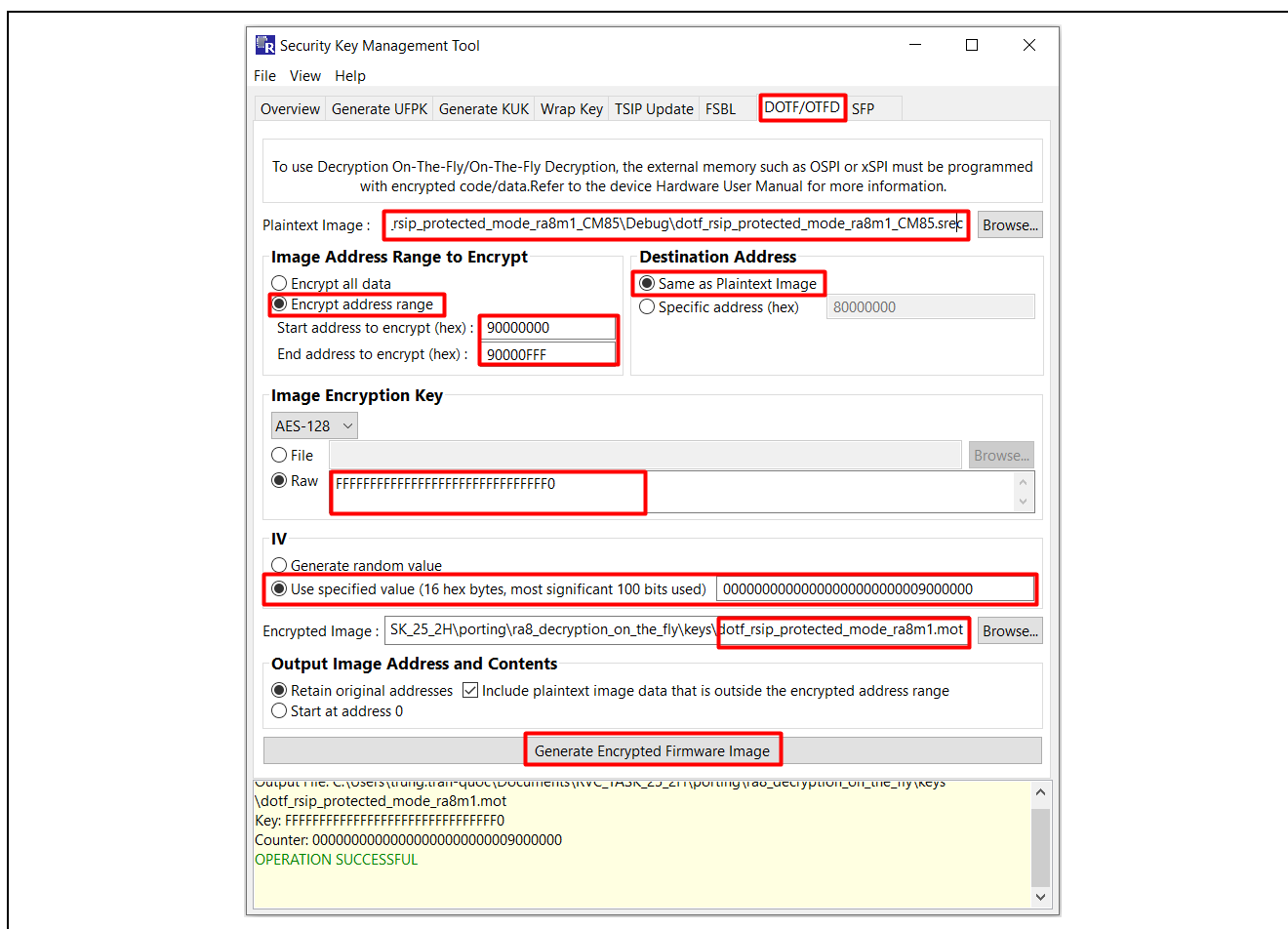


Figure 56. Use SKMT to Encrypt the Data for the DOTF Destination Area on RA8M1

For RA8P1 (CPU0), when using SKMT GUI to encrypt the Data for the DOTF Destination Area, all configuration parameters are the same as for RA8M1. However, for RA8P1 (CPU1), some parameters need to be adjusted accordingly.

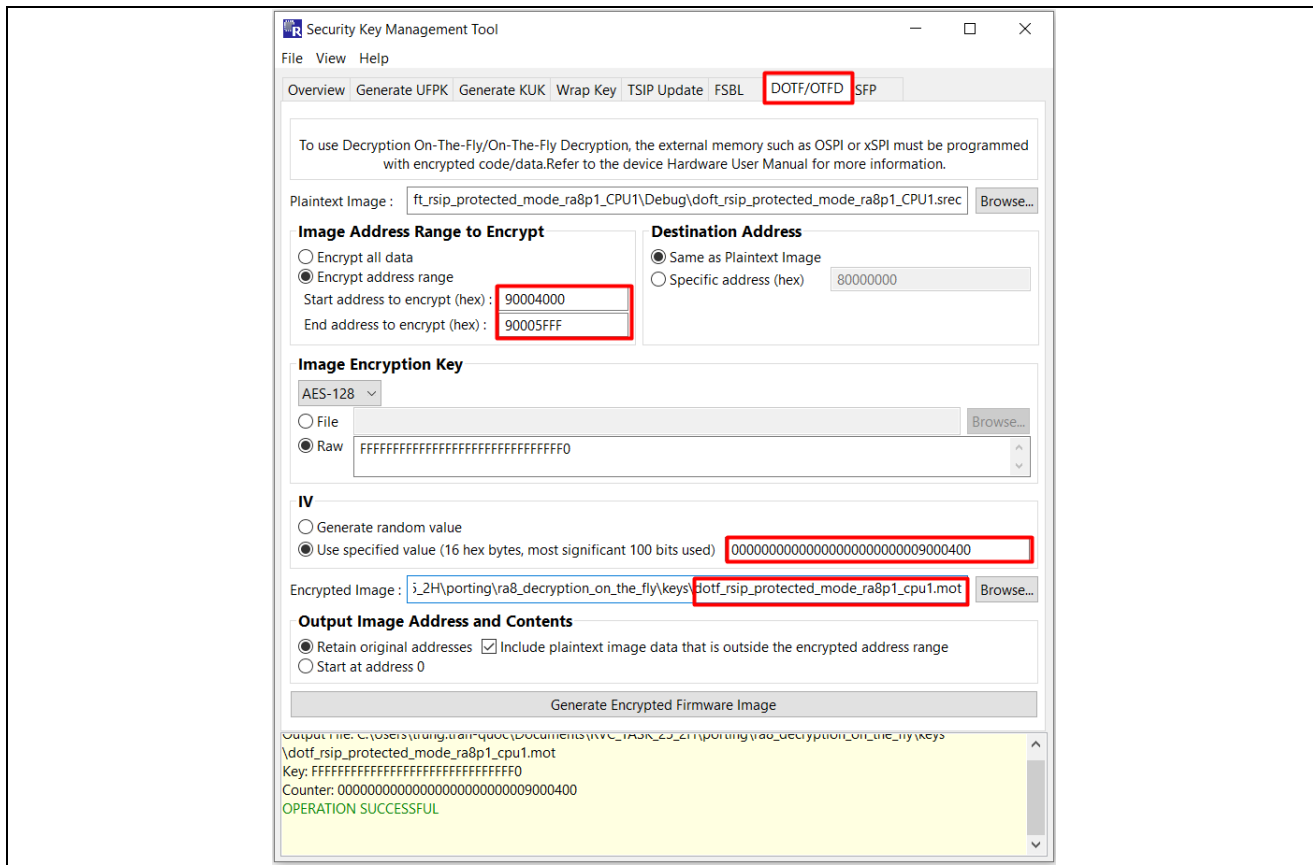


Figure 57. Use SKMT to Encrypt the Data for the DOTF Destination Area on RA8P1 (CPU1)

3.7 Running the Example Application

3.7.1.1 Set up the Hardware

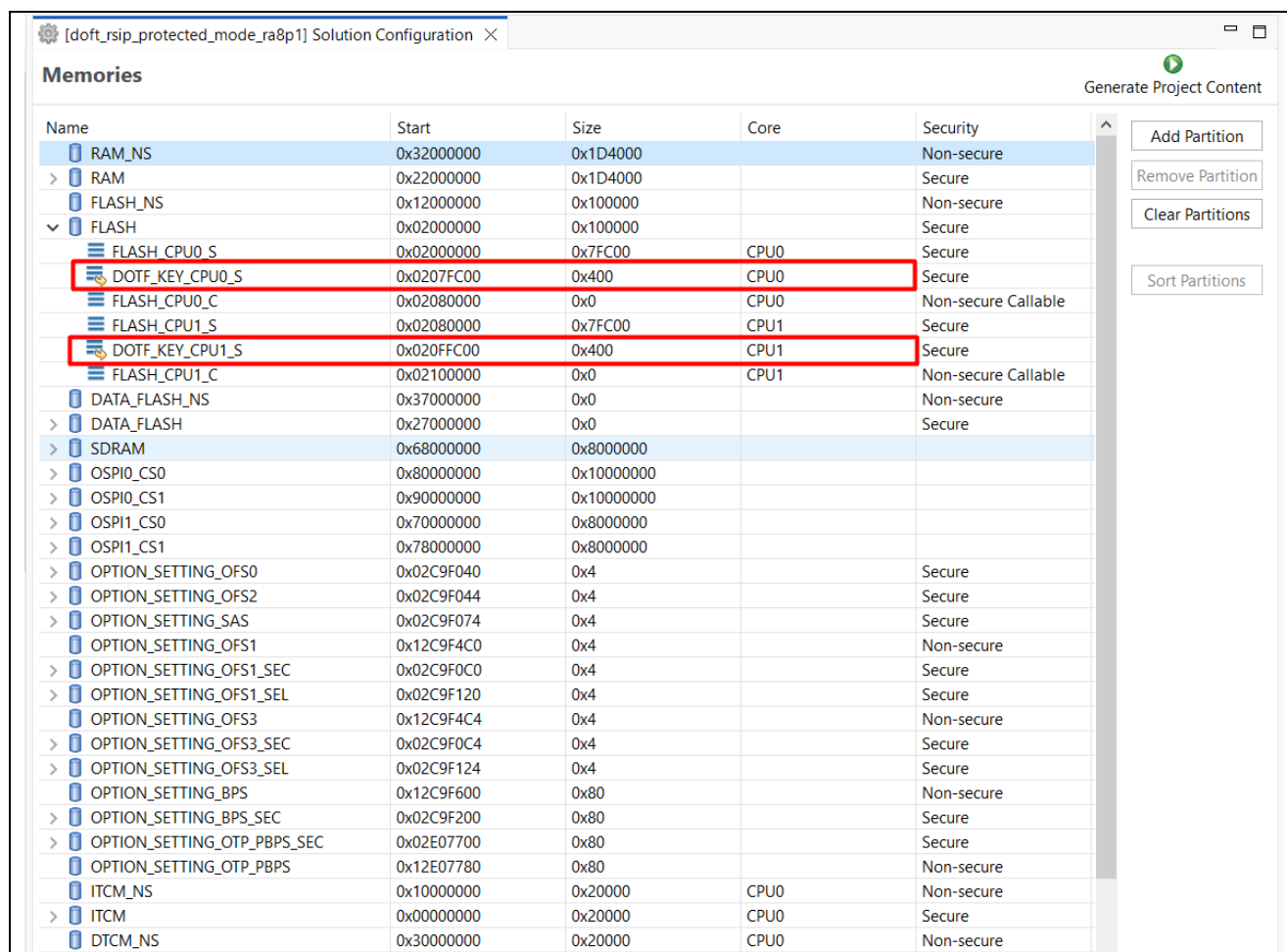
First follow sections 2.4.1, 2.4.1.1 and 2.4.1.2 to set up the hardware, initialize the MCU, and erase the OSPI flash.

3.7.1.2 Injecting the Wrapped DOTF Key

To run the Protected Mode example project, the Wrapped DOTF Key must be first injected into the MCU.

For RA8M1, the wrapped DOTF key will be injected to the beginning of the Data Flash region, which starts at 0x27000000.

For RA8P1, Flash Memory is replaced by MRAM. Therefore, we divided the memory partition accordingly to inject the wrapped DOTF key on both CPU0 and CPU1, as shown in Figure 58.



Name	Start	Size	Core	Security
RAM_NS	0x32000000	0x1D4000		Non-secure
RAM	0x22000000	0x1D4000		Secure
FLASH_NS	0x12000000	0x100000		Non-secure
FLASH	0x02000000	0x100000		Secure
FLASH_CPU0_S	0x02000000	0x7FC00	CPU0	Secure
DOTF_KEY_CPU0_S	0x0207FC00	0x400	CPU0	Secure
FLASH_CPU0_C	0x02080000	0x0	CPU0	Non-secure Callable
FLASH_CPU1_S	0x02080000	0x7FC00	CPU1	Secure
DOTF_KEY_CPU1_S	0x020FFC00	0x400	CPU1	Secure
FLASH_CPU1_C	0x02100000	0x0	CPU1	Non-secure Callable
DATA_FLASH_NS	0x37000000	0x0		Non-secure
DATA_FLASH	0x27000000	0x0		Secure
SDRAM	0x68000000	0x8000000		
OSPI0_CS0	0x80000000	0x10000000		
OSPI0_CS1	0x90000000	0x10000000		
OSPI1_CS0	0x70000000	0x8000000		
OSPI1_CS1	0x78000000	0x8000000		
OPTION_SETTING_OFS0	0x02C9F040	0x4		Secure
OPTION_SETTING_OFS2	0x02C9F044	0x4		Secure
OPTION_SETTING_SAS	0x02C9F074	0x4		Secure
OPTION_SETTING_OFS1	0x12C9F4C0	0x4		Non-secure
OPTION_SETTING_OFS1_SEC	0x02C9F0C0	0x4		Secure
OPTION_SETTING_OFS1_SEL	0x02C9F120	0x4		Secure
OPTION_SETTING_OFS3	0x12C9F4C4	0x4		Non-secure
OPTION_SETTING_OFS3_SEC	0x02C9F0C4	0x4		Secure
OPTION_SETTING_OFS3_SEL	0x02C9F124	0x4		Secure
OPTION_SETTING_BPS	0x12C9F600	0x80		Non-secure
OPTION_SETTING_BPS_SEC	0x02C9F200	0x80		Secure
OPTION_SETTING_OTP_PBPS_SEC	0x02E07700	0x80		Secure
OPTION_SETTING_OTP_PBPS	0x12E07780	0x80		Non-secure
ITCM_NS	0x10000000	0x20000	CPU0	Non-secure
ITCM	0x00000000	0x20000	CPU0	Secure
DTCM_NS	0x30000000	0x20000	CPU0	Non-secure

Figure 58. Memory Partition: MRAM Area on RA8P1

Note: Although the region is named “FLASH”, its start address points to the beginning of the MRAM. For more details, please refer to chapter 60 in the **User’s Manual: Hardware on RA8P1**.

Connect the USB Debug J10 on the EK-RA8M1 or EK-RA8P1 to the development PC. Launch **RFP** and click **File > New Project**. Assign the name of the project, select the Tool and Interface for Communication, then click **Connect**.

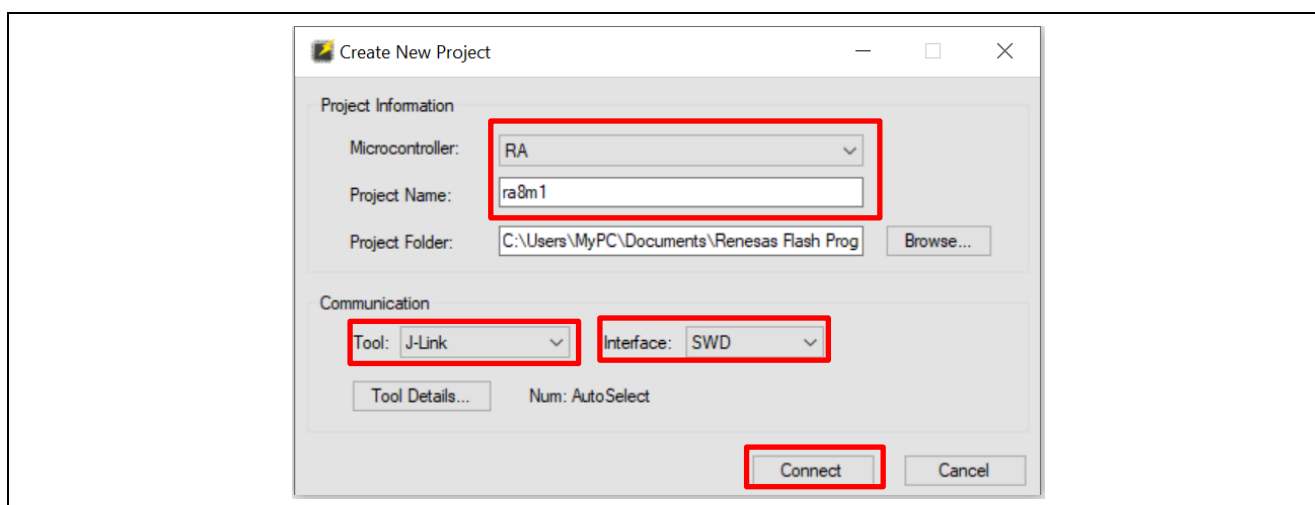


Figure 59. Establish an RFP Project to Communicate with the MCU Boot Interface

Once the connection is established, navigate to the **Operation** tab. Select the **Add/Remove Files** button.

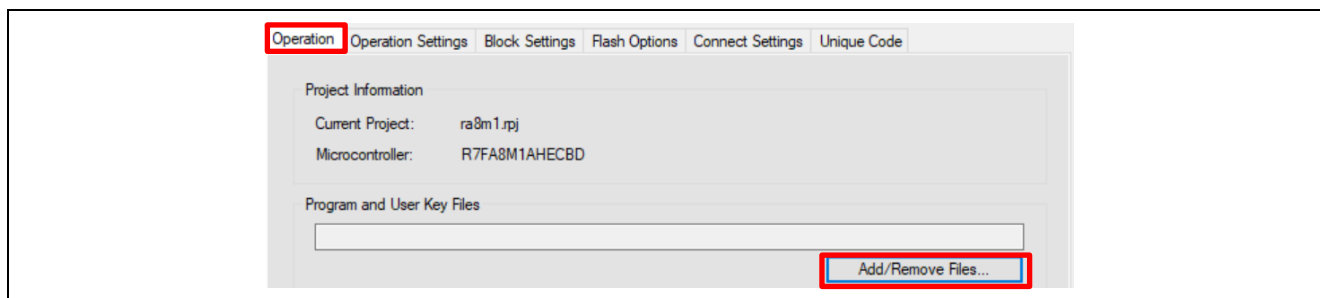


Figure 60. Select the Wrapped DOTF Key

Browse the .rkey file generated in section 3.2 or select the included DOTF_AES_128_RA8M1.rkey file and set the Address to 27000000 (which is the start to the Data Flash first sector), as shown in Figure 61.

For RA8P1:

- Inject DOTF_AES_128_RA8P1_CPU0.rkey at address 0207FC00.
- Inject DOTF_AES_128_RA8P1_CPU1.rkey at address 020FFC00.

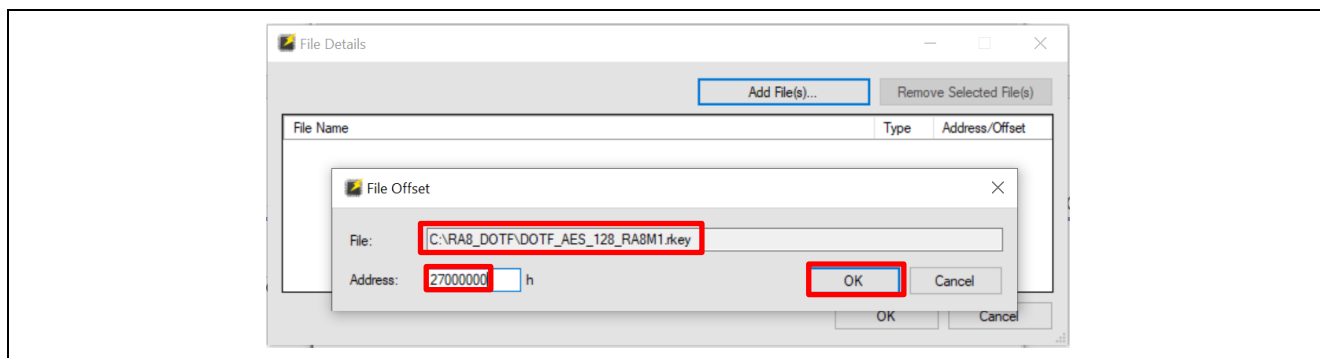


Figure 61. Choose the Data Flash Area to Inject the DOTF key

Configure the following **Operation Settings**.

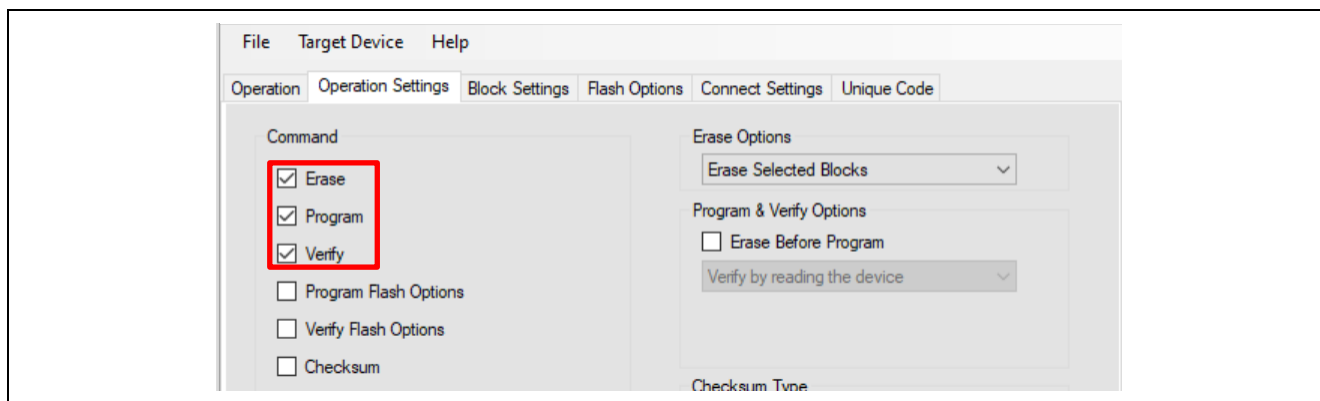


Figure 62. Operation Setting for Injecting the Wrapped DOTF Key

On the **Operation** page, click **Start** to Inject the Wrapped DOTF Key, as shown in Figure 63.

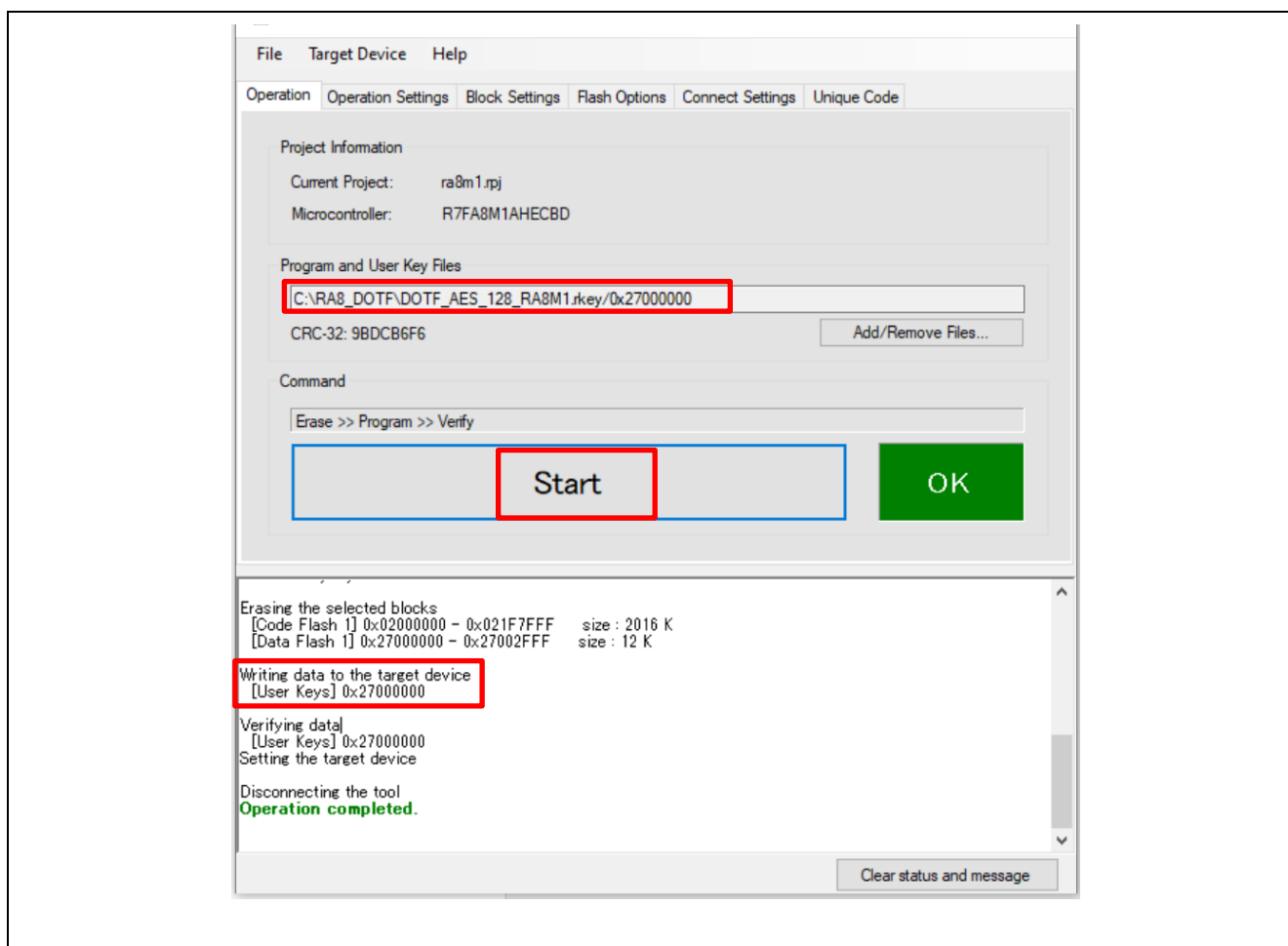


Figure 63. Inject the Wrapped AES128 Key as DOTF Key

All actions described in this section should be performed similarly for RA8P1.

3.7.2 Launch the Debug Session using the SKMT CLI Generated Images

By default, the RSIP Protected Mode example project Debug configuration uses the encryption result generated in section 3.6.1.

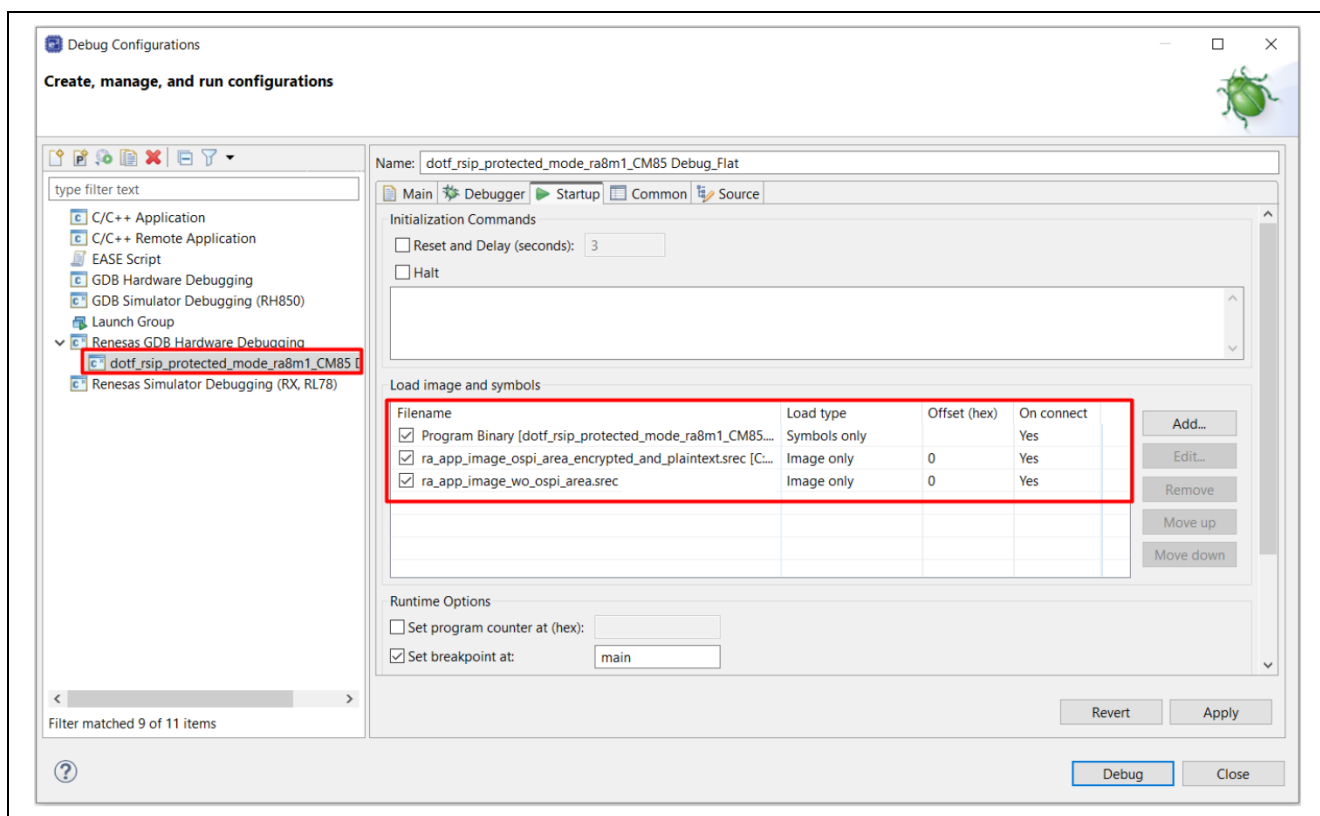


Figure 64. Configure the Debug Configuration on RA8M1

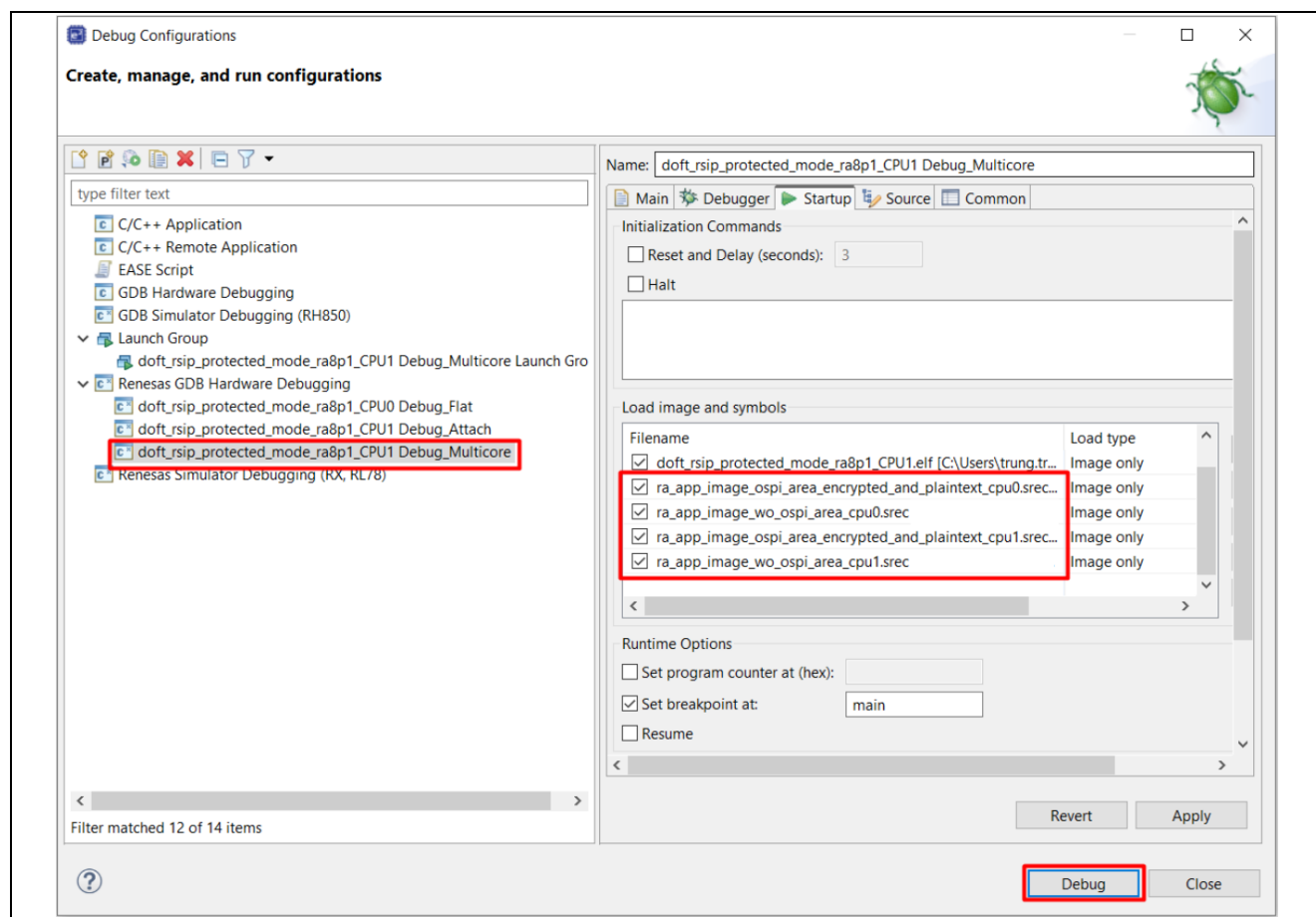


Figure 65. Configure the Debug Configuration on RA8P1

Start the Debug session and use the Tera Term to observe the execution result following the instructions in section 2.4.2. A result similar to Figure 66 should be observed. The blue LED should be blinking after the evaluation is done.

```
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 2462 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 1625 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 543 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 1535 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 462 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 456 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 452 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 456 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 452 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 456 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 452 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 456 nanoseconds;
DOTF RSIP protected mode evaluation is successful.
```

Figure 66. Code Execution Result DOTF RSIP Protected Mode on RA8M1

To validate the DOTF Protected Mode Operation on RA8P1, users should perform the same steps as in Compatibility Mode. Please refer to section 2.4.2.

```
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 12802 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 6576 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 6490 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 223 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 226 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 223 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 226 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 223 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 226 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 223 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 226 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 223 nanoseconds;
DOTF RSIP protected mode evaluation is successful on CPU0.
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 280000 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 861000 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 279000 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 861000 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 279000 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 861000 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 279000 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 861000 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 279000 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 861000 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 279000 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 861000 nanoseconds;
DOTF RSIP protected mode evaluation is successful on CPU1.
```

Figure 67. Code Execution Result DOTF RSIP Protected Mode on RA8P1

The Cortex-M85 Instruction Cache (I Cache) is always enabled by FSP BSP which boosts the MCU performance for code execution. The I Cache contributed to achieving similar code execution result when DOTF is used compared with plaintext OSPI code execution. Note that the first couple of executions take longer time as the Instructions need to be fetched and stored in the Instruction Cache. In addition, the Branch Prediction also needs time to stabilize to be more accurate.

If we invalidate the I Cache prior to the OSPI operations, we can see the overhead of DOTF on the OSPI code execution. Add the `SCB_InvalidateICache` function call to the example project (`hal_entry()` function in `hal_entry.c`):


```

SCB_InvalidateICache();
/* execution using plaintext OSPI code */
ResetCycleCounter();
execute_plaintext_fun_ospi();
SCB_InvalidateICache();
/* execution using encrypted OSPI code */
ResetCycleCounter();
execute_encrypted_func_ospi();

```

Figure 68. Invalidate the I-Cache to Evaluate the DOTF Overhead

Recompile and run the example project. Similar results as shown in can be observed in the terminal output. We can see when DOTF is enabled, this example project shows the OSPI performs at about 90% of the plaintext OSPI code execution speed. Keep in mind this result will vary based on the specific application that is evaluated.

```

Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 2606 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 2077 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 2585 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 2072 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 2572 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 2089 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 2572 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 2081 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 2572 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 2089 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 2581 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 2089 nanoseconds;
DOTF RSIP protected mode evaluation is successful.

```

Figure 69. Testing Result for DOTF Overhead Evaluation on RA8M1

```

Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 12877 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 11777 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 12873 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 11777 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 12873 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 11777 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 12873 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 11769 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 12869 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 11769 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 12873 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 11773 nanoseconds;
DOTF RSIP protected mode evaluation is successful on CPU0.
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 280000 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 861000 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 279000 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 861000 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 279000 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 861000 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 279000 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 861000 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 279000 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 861000 nanoseconds;
Protected Mode: Time used in executing the plaintext Fibonacci calculation function from OSPI without DOTF: 279000 nanoseconds;
Protected Mode: Time used in executing the encrypted Fibonacci calculation function from OSPI using DOTF: 861000 nanoseconds;
DOTF RSIP protected mode evaluation is successful on CPU1.

```

Figure 70. Testing Result for DOTF Overhead Evaluation on RA8P1

3.7.3 Launch the Debug Session using the SKMT GUI Encryption Result

To evaluate the encryption result generated from section 3.6.2, update the Debug configuration as shown in Figure 71.

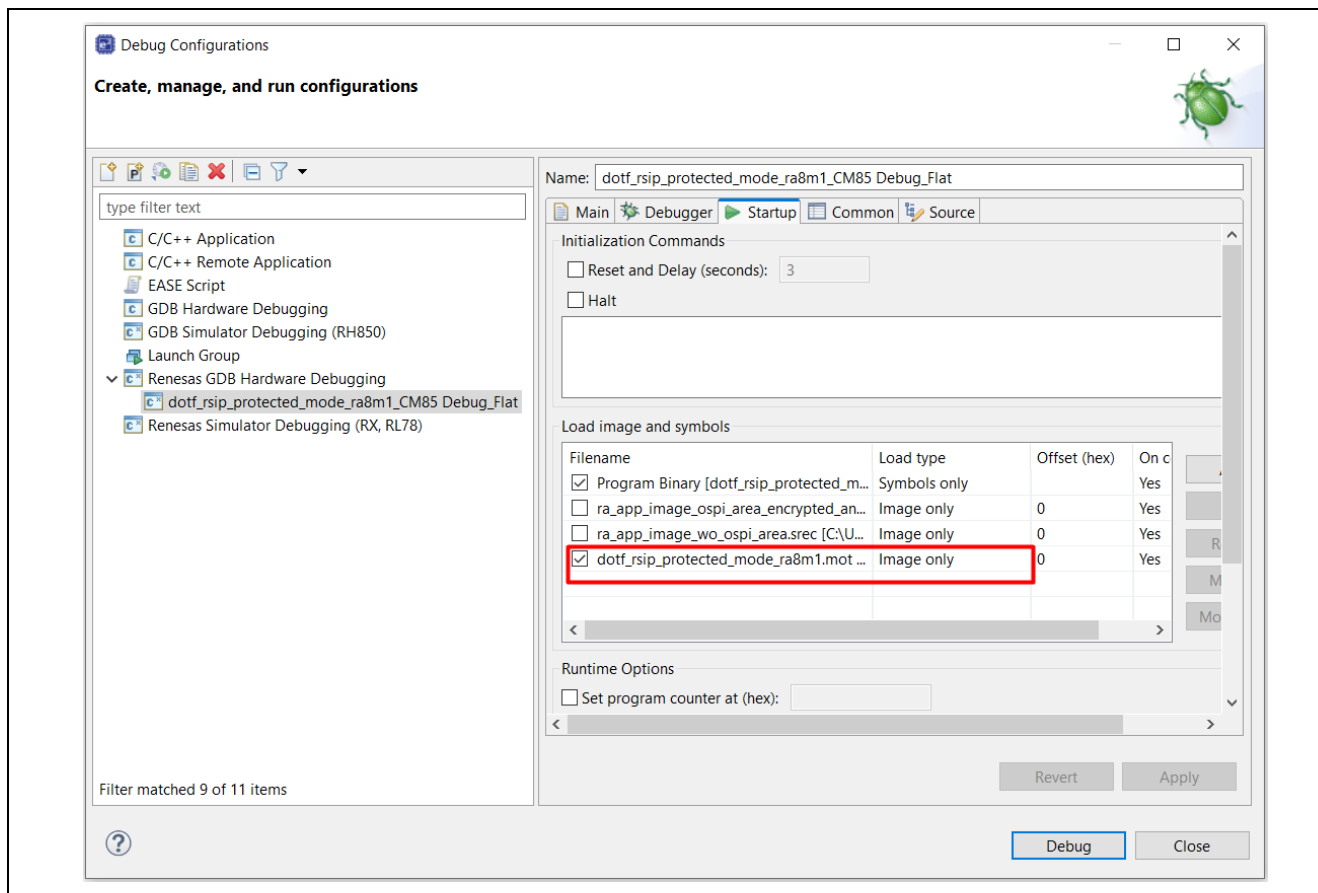


Figure 71. Update the RSIP Protected Mode DOTF Example Project Debug Configuration on RA8M1

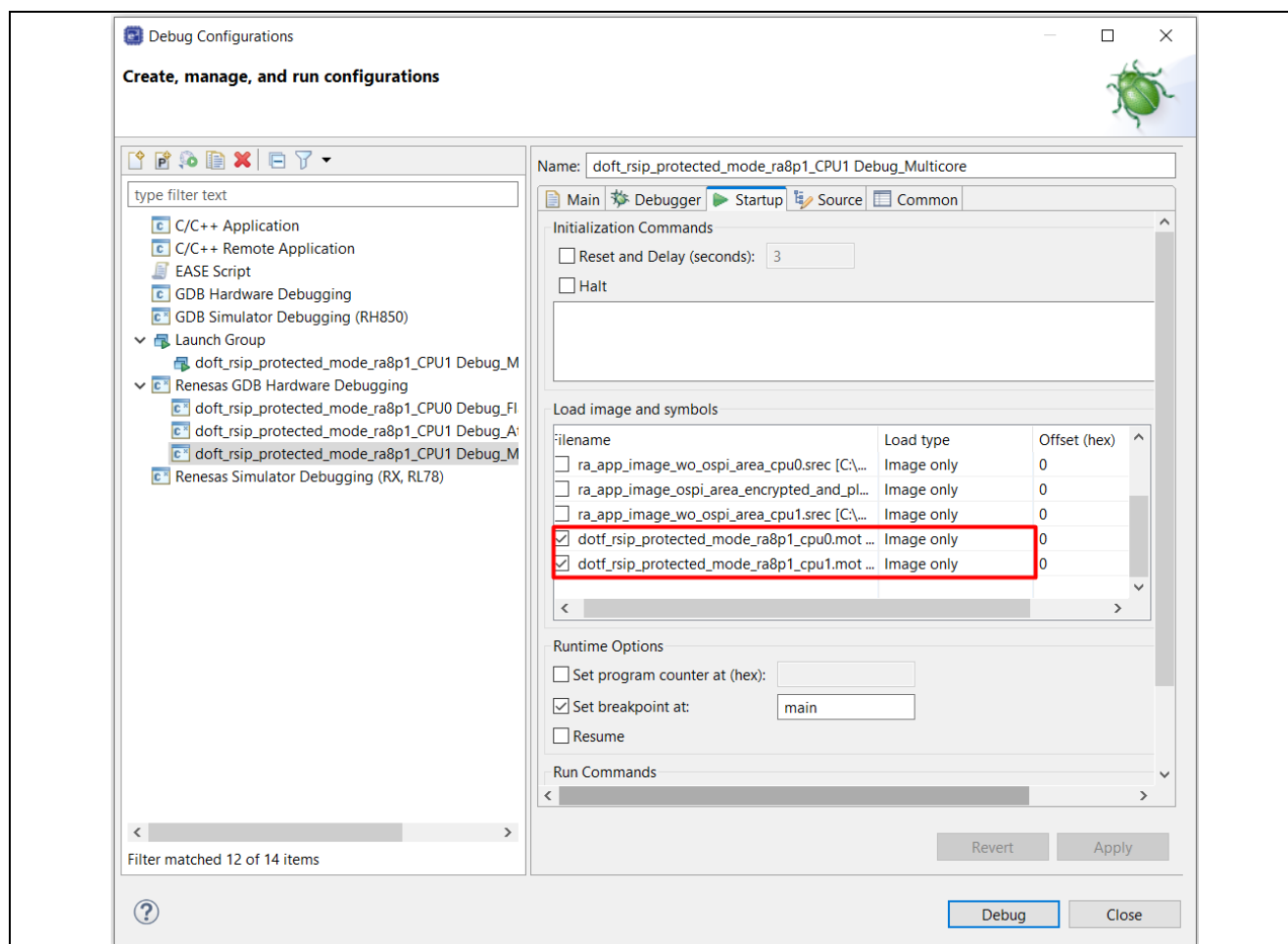


Figure 72. Update the RSIP Protected Mode DOTF Example Project Debug Configuration on RA8P1

Next, start the **Debug** session and use Tera Term to observe the execution result. A result similar to Figure 66 and Figure 67 should be observed for RA8M1 and RA8P1.

4. Guidelines for DOTF Production Support

The demonstrations in this application project assume the MCU addressing space is known. In a production environment, a third-party tool will program the OSPI chip independently of the MCU without prior knowledge of the MCU addressing space.

In this case, the encrypted OSPI data should be output to a separate file. Addressing in that file must use the addresses of the OSPI flash chip address space, NOT the MCU address space.

Using the use case of Figure 56 as an example, the following update should be performed when producing the encrypted data for the OSPI area that will be programmed by a third-party tool. Note that the MCU address space is required when specifying the address range to encrypt, since this address is incorporated into the encryption algorithm.

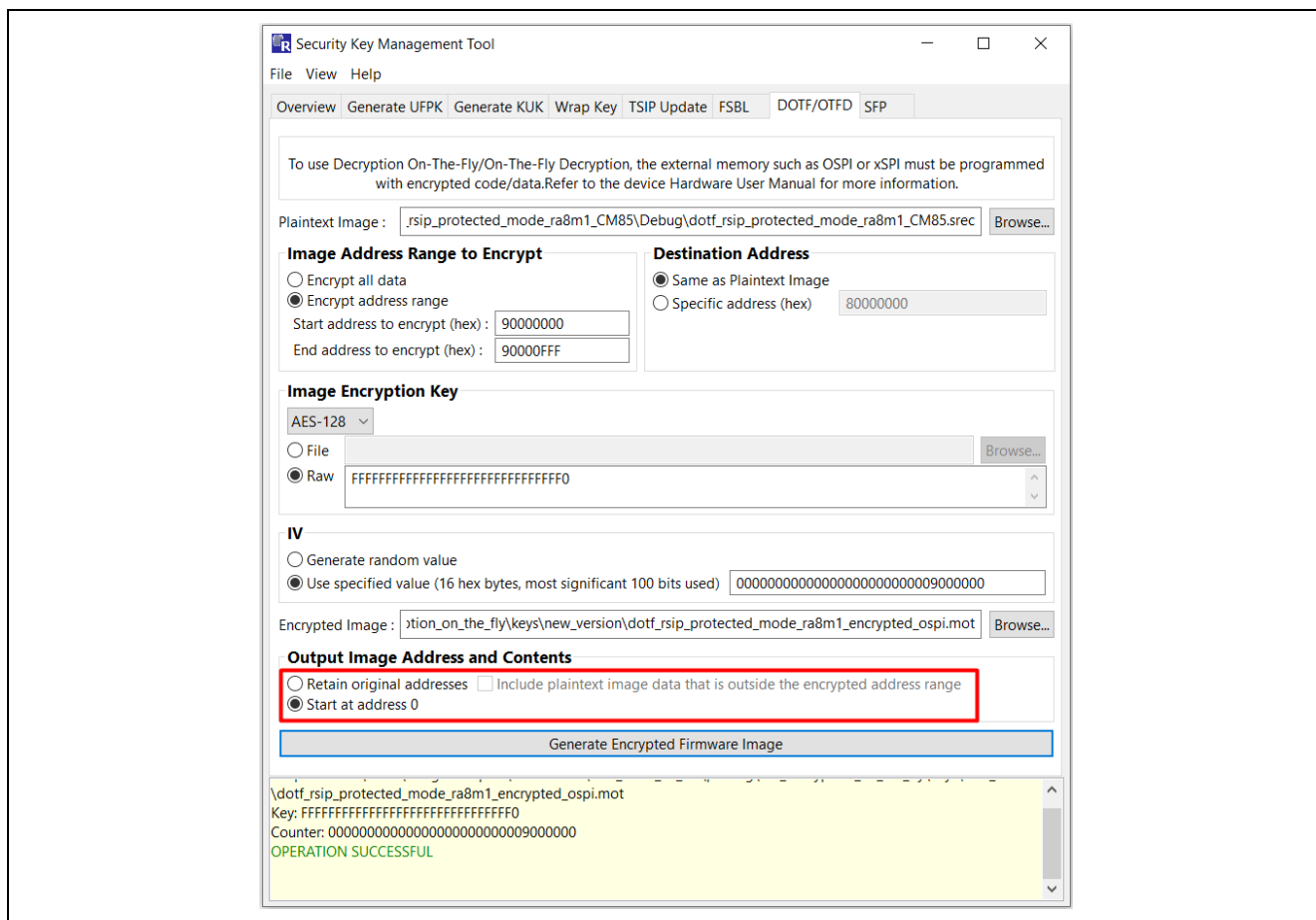


Figure 73. Example: Generate Encrypted OSPI Data for Third-party Tool

5. Appendix

5.1 Linker Script used in the Compatibility Mode Example Project with Arm Compiler (AC6)

The included Compatibility Mode example project uses the GCC compiler for RA8M1 and the LLVM compiler for RA8P1. To achieve the same result when using AC6, refer to sections 2.3.1 and 2.3.2.

```
; User region PLAINTEXT_DATA_S
LOAD_REGION_PLAINTEXT_DATA_S PLAINTEXT_DATA_S_START NOCOMPRESS PLAINTEXT_DATA_S_LENGTH
{
    __plaintext_data_s +0 FIXED UNINIT
    {
        *(.plaintext_data_s)
    }
}
```

Figure 74. The AC6 Linker Script for the Compatibility Mode Project

5.2 Update the Linker Script for the Protected Mode Example Project

The included Protected Mode example project uses LLVM compiler. As shown in Figure 47, the default FSP linker script is updated to keep the injected wrapped DOTF key during the Debug session launch process.

To achieve the same goal when GCC is used, the following linker script updates should be performed. Key word (NOLOAD) is added similar to the LLVM linker script update.

```
__data_flash_readonly$$ (NOLOAD) :
{
    __data_flash_readonly$$Base = .;
    /* section.data_flash.readonly */
    *(.data_flash)
    /* section.data_flash.code */
    *(.data_flash_code)
    __data_flash_readonly$$Limit = .;
}> DATA_FLASH
```

Figure 75. Update the GCC Linker Script for the Project Mode Project

To achieve the same goal when AC6 is used, the following linker script updates should be performed. UNINIT key word is used in linker script update.

```
LOAD_REGION_DATA_FLASH_JUMP +0 NOCOMPRESS
{
    __data_flash_readonly +0 FIXED UNINIT
    {
        ; section.data_flash.readonly
        *(.data_flash)
        ; section.data_flash.code
        *(.data_flash_code)
    }

    __data_flash_noinit +0 FIXED UNINIT
    {
        ; section.data_flash.noinit
        *(.bss.data_flash_noinit)
    }

    __ddsc_DATA_FLASH_END AlignExpr(+0, 1024) EMPTY 0
    {}
    .data_flash.endof AlignExpr(+0, 1024) EMPTY 0
    {
    }

    __DATA_FLASH_end +0 EMPTY 0
    {}
    ScatterAssert( (LoadBase(__DATA_FLASH_end) - LoadBase(__DATA_FLASH_start)) <= DATA_FLASH_LENGTH )
}
```

Figure 76. Update the AC6 Linker Script for the Project Mode Project

6.2 J-Link Script Failure

In cases where the J-Link Script described in section 2.4.1.2 does not work in the user's environment. Users can use J-Flash Lite v8.58 to erase the OSPI Flash. Note that erasing the entire OSPI device memory may take several minutes.

Make sure to initialize the device first using the **Renesas Device Partition Manager (RDPM)**, as mentioned in section 2.4.1.1.

To use the J-Flash Lite, connect the USB Debug Port J10 to the PC and launch J-Flash Lite. Select the **Device**, **Target Interface**, communication speed, and the OSPI area to be erased.

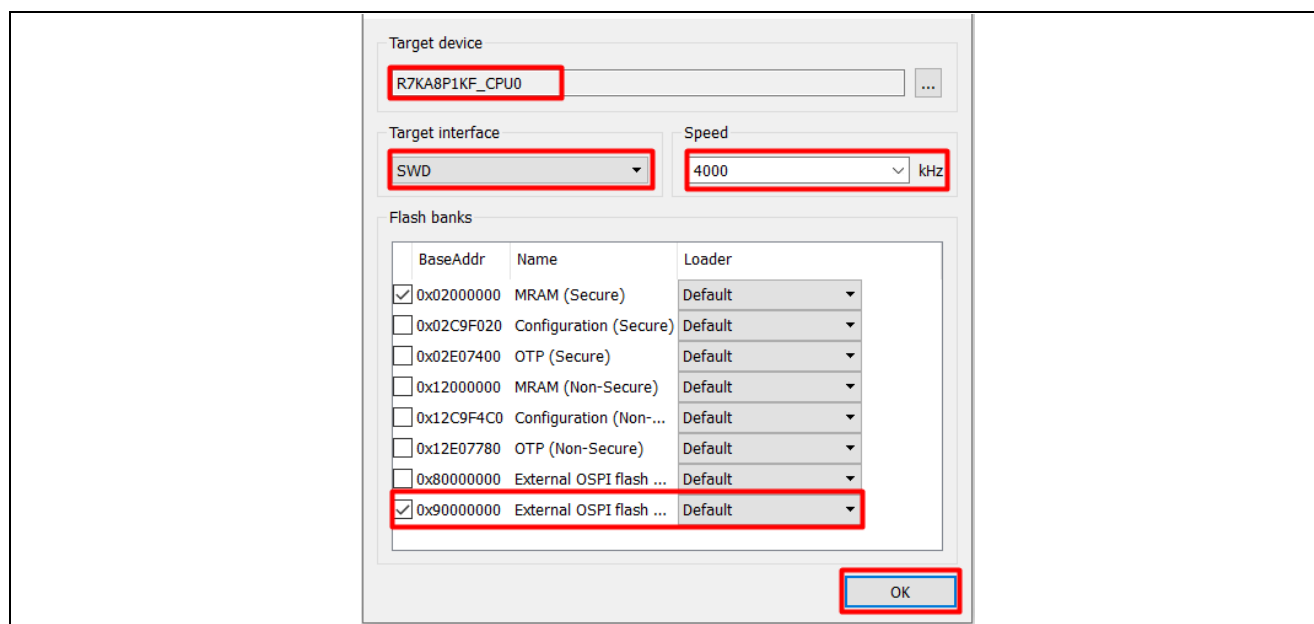


Figure 79. Launch the J-Flash Lite

Click **OK**. In the next screen, select **Erase Chip**.

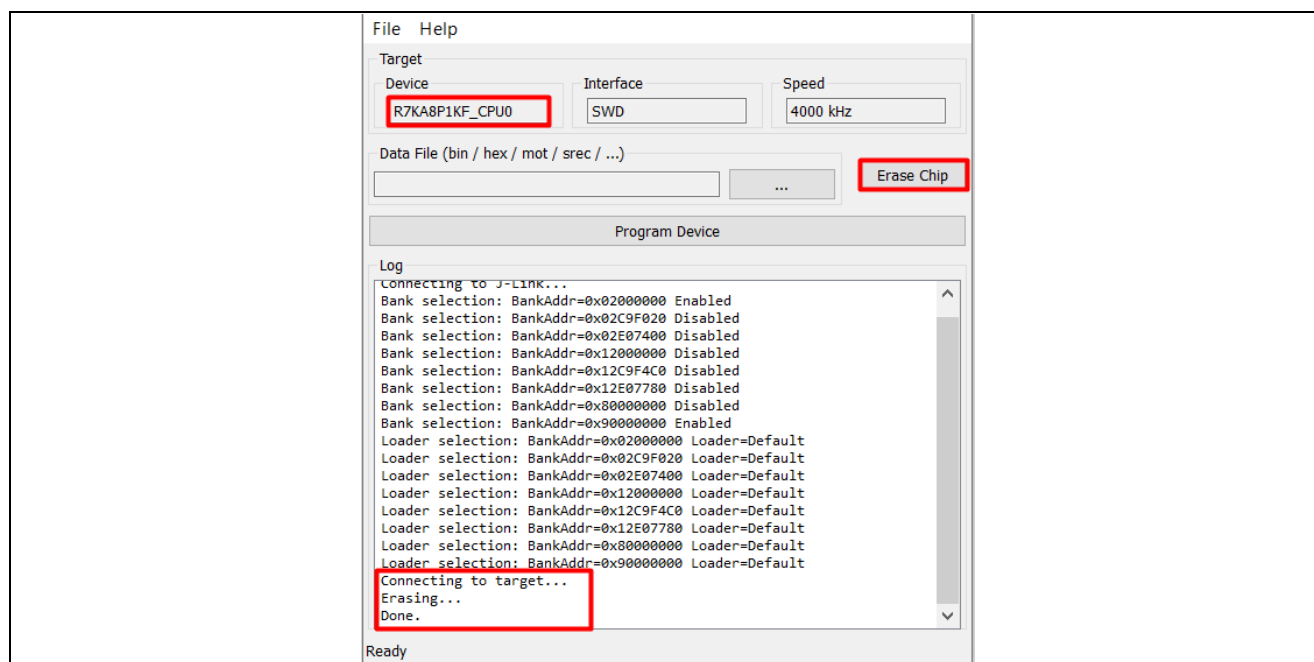


Figure 80. Erase the OSPI Flash using J-Flash Lite

Repeat the same procedure for CPU1 by selecting the device **R7KA8P1KF_CPU1**, as shown in Figure 79.

7. References

1. [Flexible Software Package \(FSP\) User's Manual](#)
2. [Renesas RA8M1 Group User's Manual: Hardware](#)
3. [Renesas RA8P1 Group User's Manual: Hardware](#)
4. Renesas RA Family RA8 MCU Series Device Lifecycle Management (R11AN0785)
5. Renesas RA Family MCU Injecting and Updating Secure User Keys (R11AN0496)
6. Renesas RA Family MCU Injection Plaintext User Keys (R11AN0473)
7. Renesas RA Family MCU Renesas RA Family Security Engine Operational Modes (R11AN0498)
8. Renesas RA Family MCU Developing with RA8 Dual Core MCU (R01AN7881)
9. Macronix International Co., Ltd — MX25LW51245GXDI00 Datasheet
10. Infineon Technologies — S28HL512TFPBHI010 Datasheet

8. Website and Support

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

EK-RA8M1 Resources	renesas.com/ra/ek-ra8m1
EK-RA8P1 Resources	renesas.com/ek-ra8p1
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	Nov.21.24	—	Initial release
1.10	Oct.01.25	—	Supported for RA8P1 Dual-Core with FSPv6.0.0

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/.