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 evaluation kit. 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.

Target Devices

- RA8M1
- RA8D1
- RA8T1

Required Resources

Software and development tools

- e^2 studio IDE v2024-10
- Renesas Flexible Software Package (FSP) v5.6.0

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

- Renesas Flash Programmer (RFP) v3.15 or later
  https://www.renesas.com/us/en/products/software-tools/tools/programmer/renesas-flash-programmer-programming-gui.html
- Renesas Security Key Management Tool v1.07 or later
  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)
- Workstation running Windows® 10 and the Tera Term console or similar application
- One USB device cable (type-A male to micro-B male)

Prerequisites and Intended Audience

This application project assumes that the user has experience using the Renesas e^2 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. 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

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

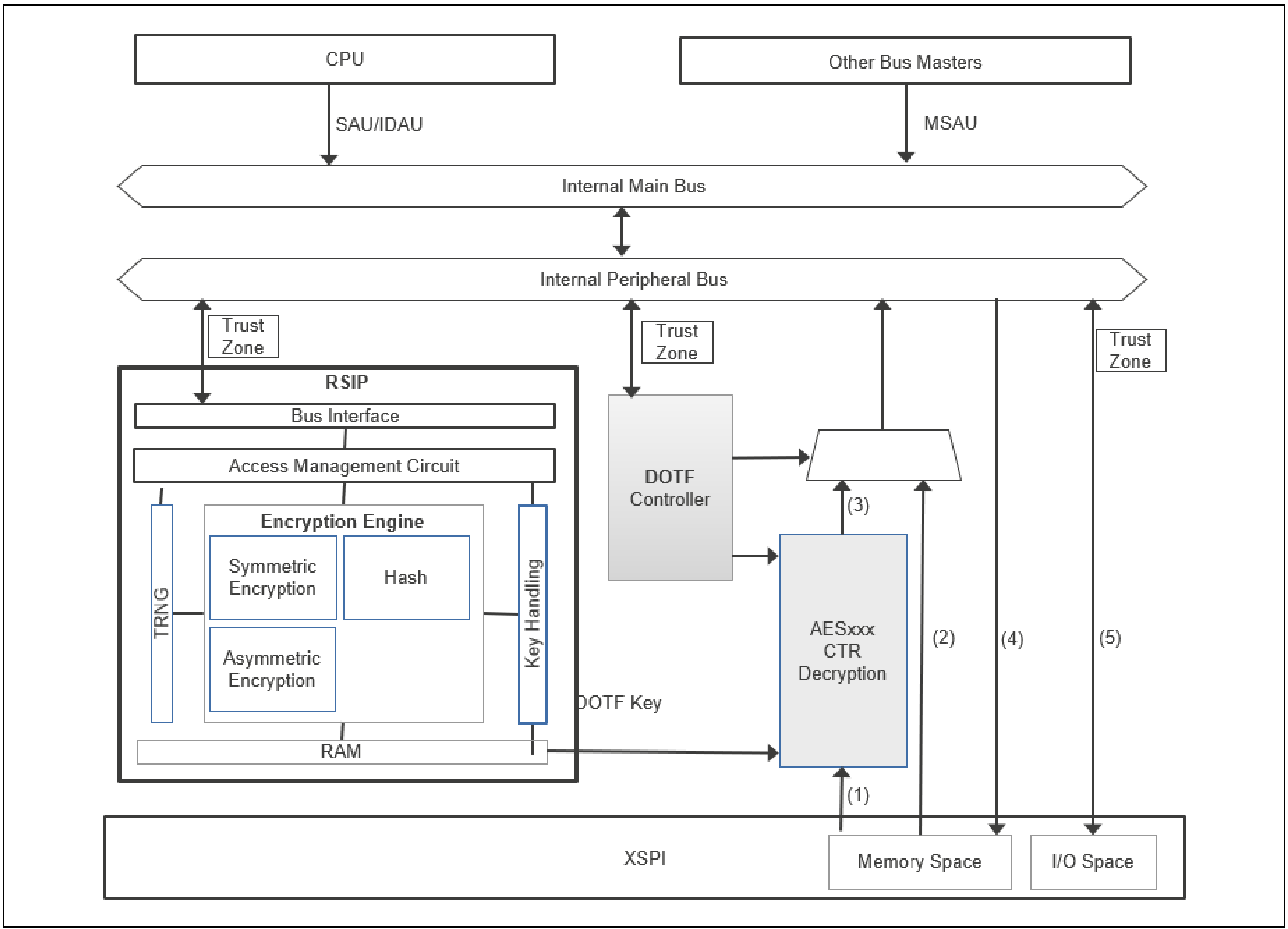


Figure 1. RA8 Decrypt on the Fly

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 the EK-RA8M1, an OctaFlash is connected to the RA8M1 on channel 1, which starts 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.

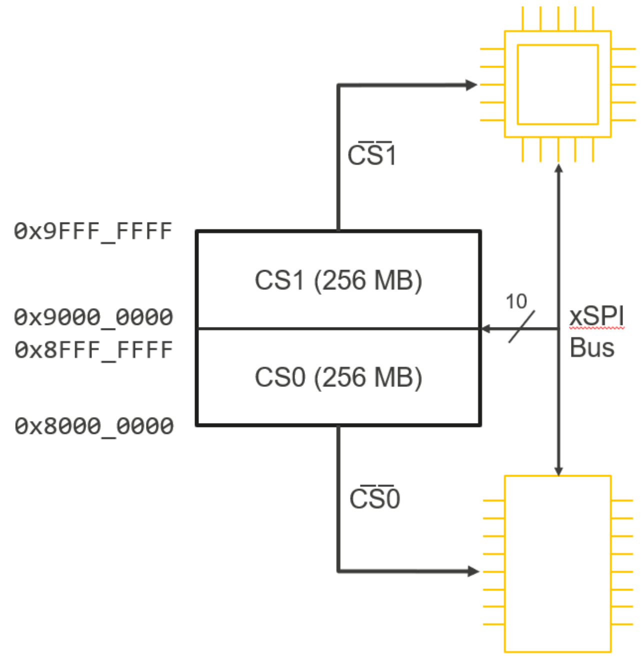


Figure 2. OSPI Memory Space

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.

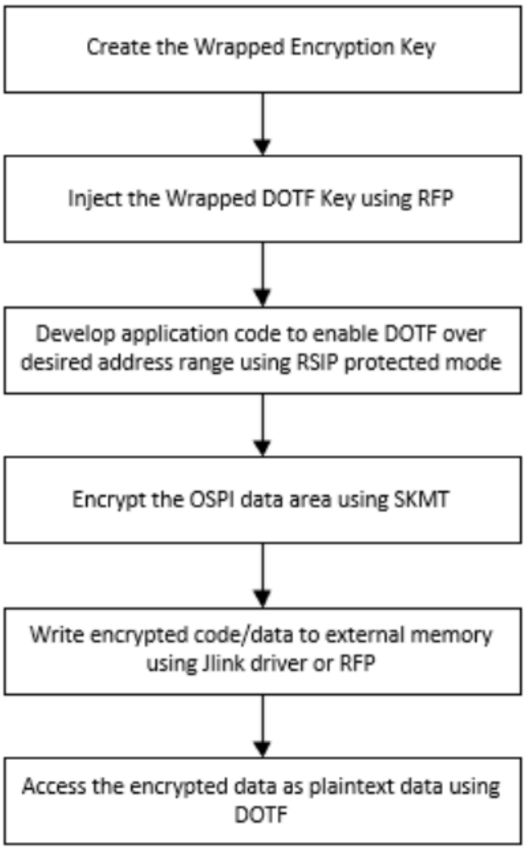


Figure 3. Example DOTF Operational Flow using RSIP Protected Mode

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.

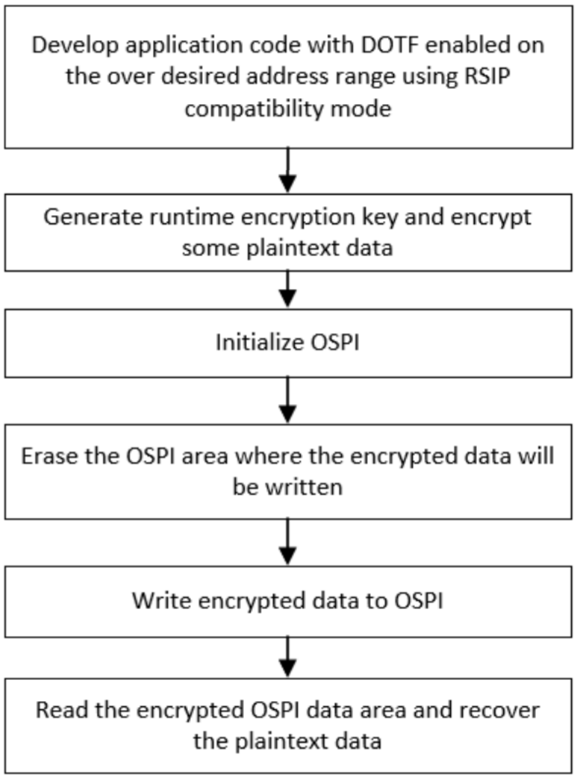


Figure 4. Runtime DOTF Key Generation and Data Encryption using RSIP Compatibility Mode

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 Initialization Vector (IV) are provided 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 Specifical 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.

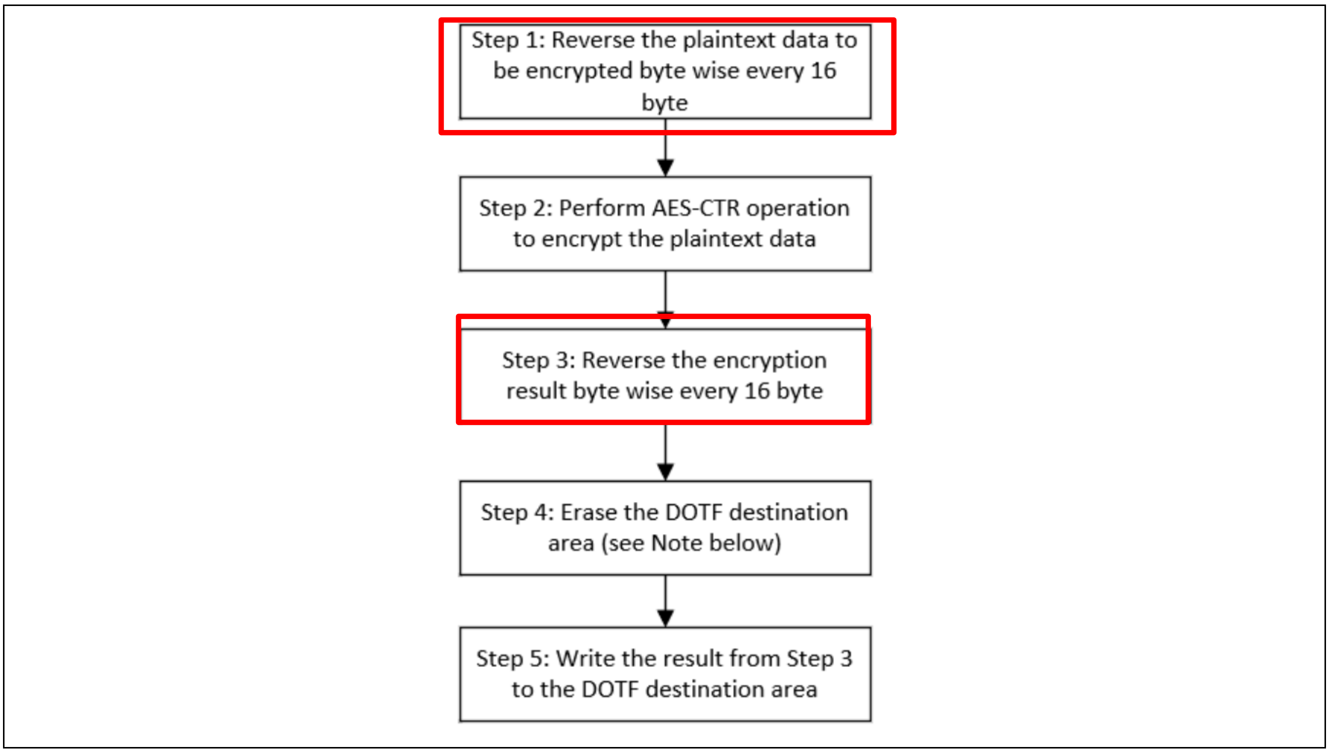


Figure 5. Runtime Encryption Operational Flow

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 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.

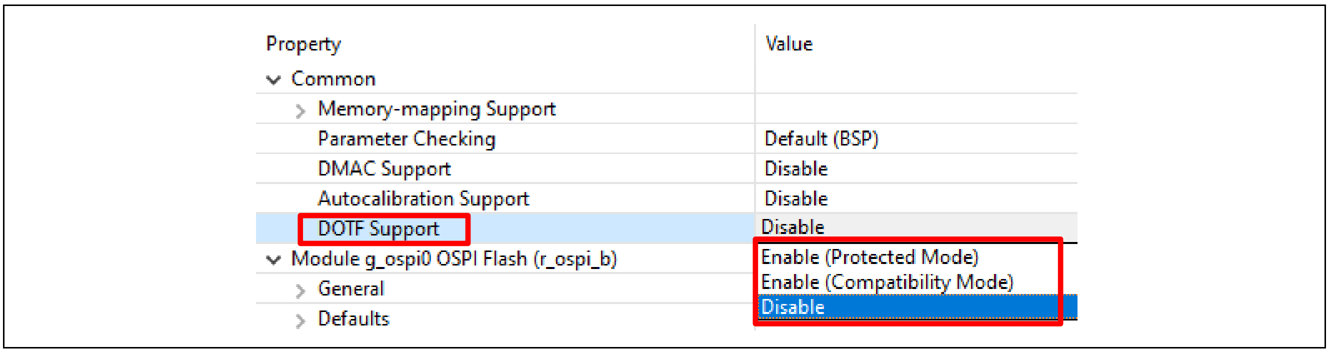


Figure 6. Select the DOTF Operation Mode

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	g_ospi_dotf	DOTF Configuration name.	Name must be a valid C symbol
AES Key	g_ospi_dotf_key	Name of Key variable.	Name must be a valid C symbol
AES IV	g_ospi_dotf_iv	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, 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. The GCC compiler is used for the Compatibility Mode example project. Data Cache is also enabled to achieve better system performance.

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 is from 0x90000000 to 0x90000FFF.

Figure 7 is the OSPI memory layout for both the DOTF RSIP Compatibility Mode and the DOTF RSIP Protected Mode example projects.

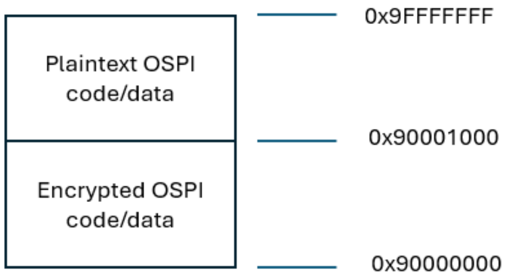


Figure 7. OSPI Memory Map of the Example Projects

Figure 8 is the key DOTF configurations for the Compatibility Mode example project.

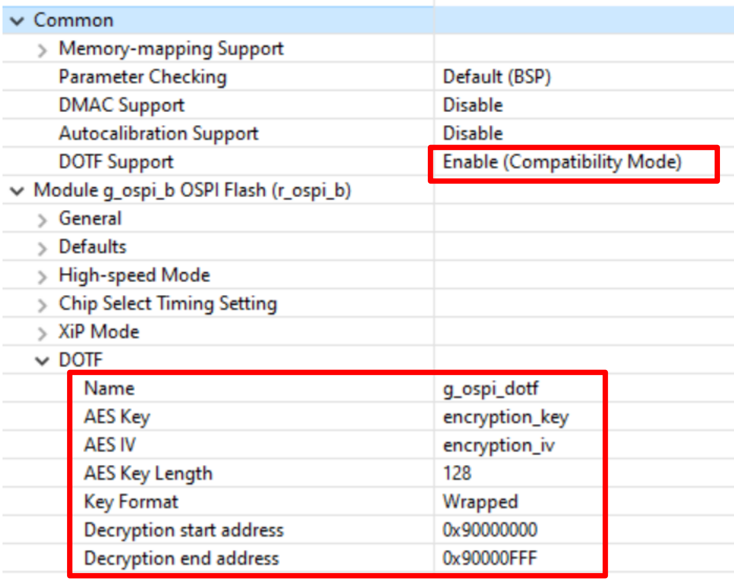


Figure 8. Configure the DOTF with Compatibility Mode

Figure 9 is an overview of the software components used in the example application project. 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.

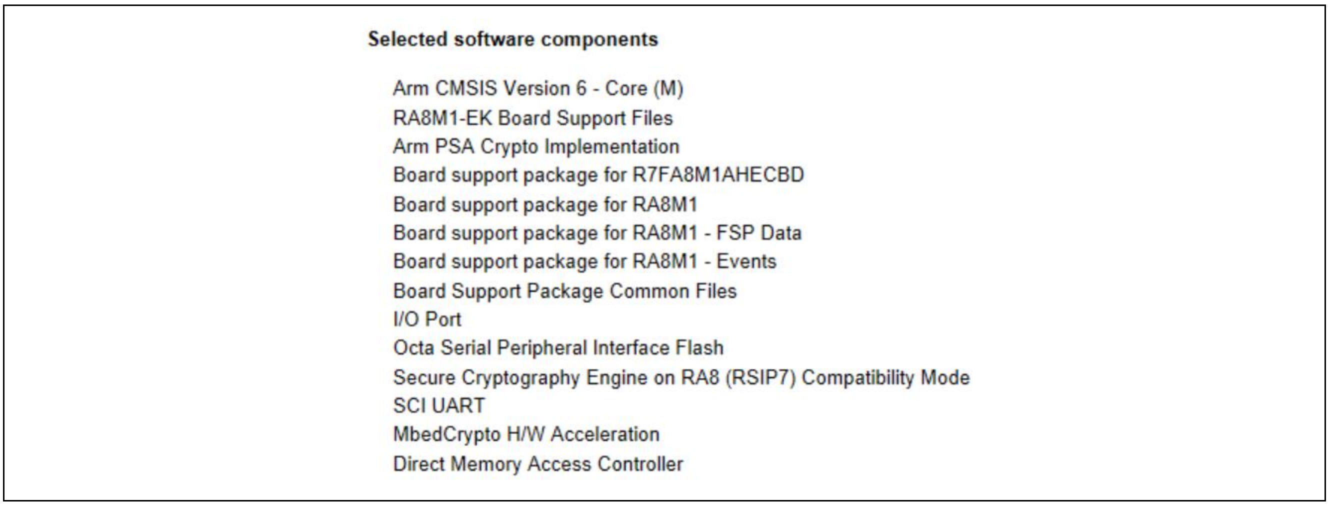


Figure 9. Software Components Used for Runtime Encryption and DOTF Support

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.

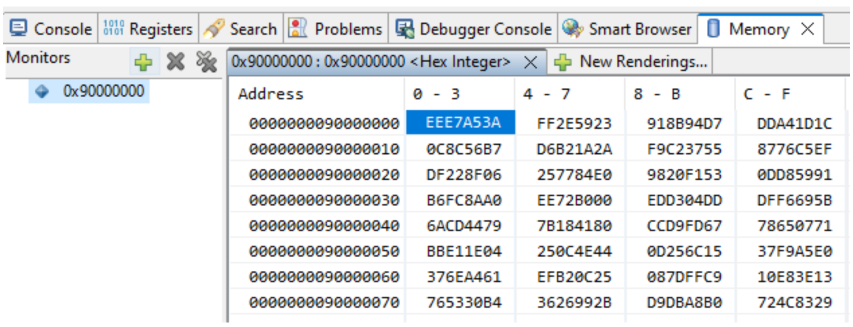
- In this example implementation, the MbedTLS Crypto module is configured with AES128 CTR encryption enabled. The Asymmetric algorithms like ECC and RSA are disabled to reduce flash and SRAM usage.
- Set up the AES-CTR algorithm IV. As explained in section 1.4.3, the last 28 bit of the IV is the counter which is initialized with the first 28 bit of the destination address of the DOTF operation. In this example project, the destination address is 0x90000000. So, the first 28-bit 0x9000000 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[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x09, 0x00, 0x00, 0x00};

  In this example project, any nonce (which is the first 100 bit of the IV) would work so long as the initial counter (which is the last 28 bit of the IV) is 0x9000000.
- 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\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.



Address	0 - 3	4 - 7	8 - B	C - F
0000000090000000	EEE7A53A	FF2E5923	918B94D7	DDA41D1C
0000000090000010	0C8C56B7	D6B21A2A	F9C23755	8776C5EF
0000000090000020	DF228F06	257784E0	9820F153	0DD85991
0000000090000030	B6FC8AA0	EE72B000	EDD304DD	DFF6695B
0000000090000040	6ACD4479	7B184180	CCD9FD67	78650771
0000000090000050	BBE11E04	250C4E44	0D256C15	37F9A5E0
0000000090000060	376EA461	EFB20C25	087DFFC9	10E83E13
0000000090000070	765330B4	3626992B	D9DBA8B0	724C8329

Figure 10. 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_to_encrypt (\src\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.

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 default, the FSP linker script defines one region OSPI_DEVICE_1 with two subregions for the OSPI flash connected on the EK-RA8M1. This example project updated the default linker script and memory region configuration so the plaintext data can be allocated to the correct location at compile time.

Table 2 Linker Script Customization

	Default FSP configuration	Example project configuration
Linker script	\linkers\fsp.ld	\linkers\fsp_app.ld

The following is the default OSPI_DEVICE_1 region for application to allocate OSPI data at compile time. This configuration does not allow compile time OSPI data allocation at specific locations.

```
/* OSPI_DEVICE_1 section to be downloaded via debugger */
.OSPI_DEVICE_1 :
{
    __ospi_device_1_start__ = .;
    KEEP(*(.ospi_device_1*))
    KEEP(*(.code_in_ospi_device_1*))
    __ospi_device_1_end__ = .;
} > OSPI_DEVICE_1
```

Figure 11. Default Memory Section "OSPI_DEVICE_1"

The following is the updated OSPI_DEVICE_1 region.

```
/* OSPI_DEVICE_1 section to be downloaded via debugger */
.OSPI_DEVICE_1 :
{
    __ospi_device_1_start__ = .;
    KEEP(*(.ospi_device_1*))
    . = ORIGIN(OSPI_DEVICE_1) + 0x1000;
     __ospi_device_1_plaintext_start__ = .;

    KEEP(*(.code_in_ospi_device_1*))
    __ospi_device_1_end__ = .;
} > OSPI_DEVICE_1
```

Figure 12. Updated Memory Section "OSPI_DEVICE_1"

The subsection `.ospi_device_1` is used as the DOTF destination region. The subsection `.code_in_ospi_device_1` is used as the plaintext OSPI data section. Since the OSPI is configured to use 0x90000000 to 0x90000FFF as the DOTF destination region, an offset of 0x1000 is defined for the plaintext OSPI region to start from 0x90001000. Additionally, a global variable `__ospi_device_1_plaintext_start__` pointing to 0x90001000 is defined, which is accessible for the application code. The application code uses this variable to access the plaintext data array. The application code needs to guarantee not writing 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.

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. When the Generate Project Content is clicked, the default linker scripts are extracted to the workspace in the folders described in Table 6. This is okay as it is not used in the compilation process.

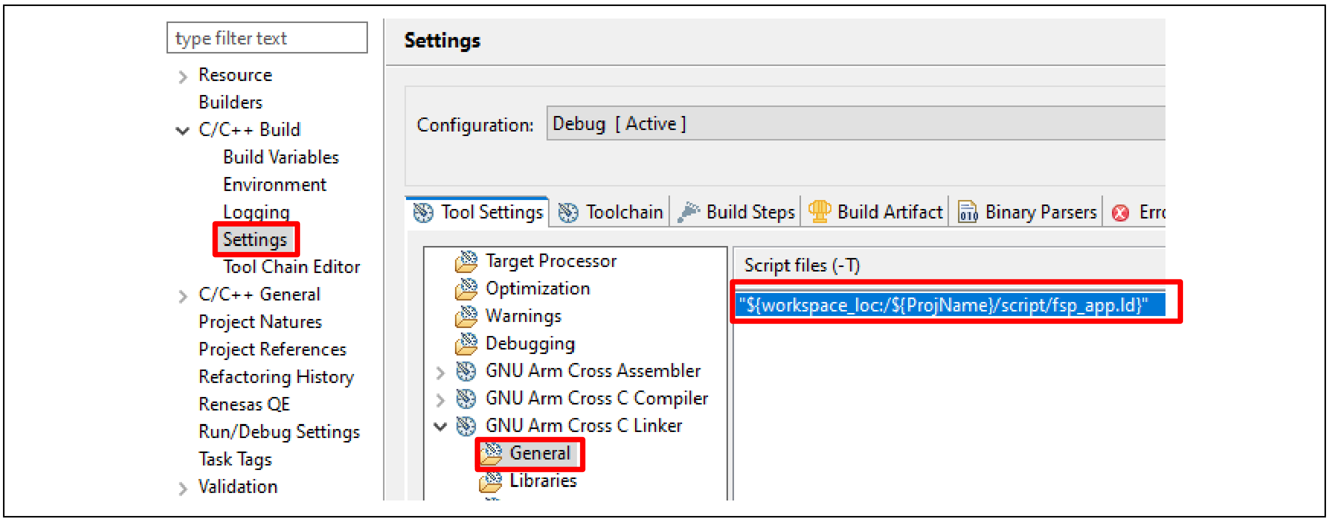


Figure 13. Configure to Use the Custom Linker Script

Figure 14 is the plaintext data allocated to the subsection `.code_in_ospi_device_1`.

```
/* 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(".code_in_ospi_device_1") = {
    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 14. 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.

2.4 Running the Example Application

2.4.1 Set up the Hardware and Import the Application

Using the EK-RA8M1 default jumper setting, connect J10 on EK-RA8M1 to the Development PC using the USB type-A male to micro-B male cable to provide the power, Debug, and Virtual COM port connections to the board. Next follow section 2.4.1.1 to Initialize the MCU, then power cycle the EK-RA8M1 and then erase the OSPI follow section 2.4.1.2 and power cycle the EK-RA8M1. 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):

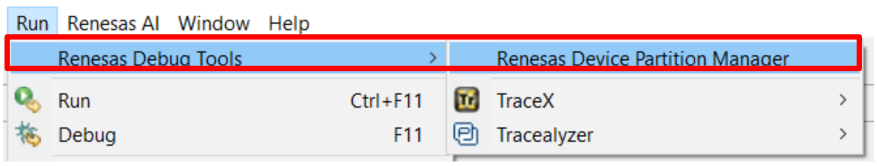


Figure 15. Open the Renesas Device Partition Manager

Next, check **Initialize device back to factory default**, 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 a custom PCB board, the Connection Type should be selected based on the Boot Mode interfaces available.

.

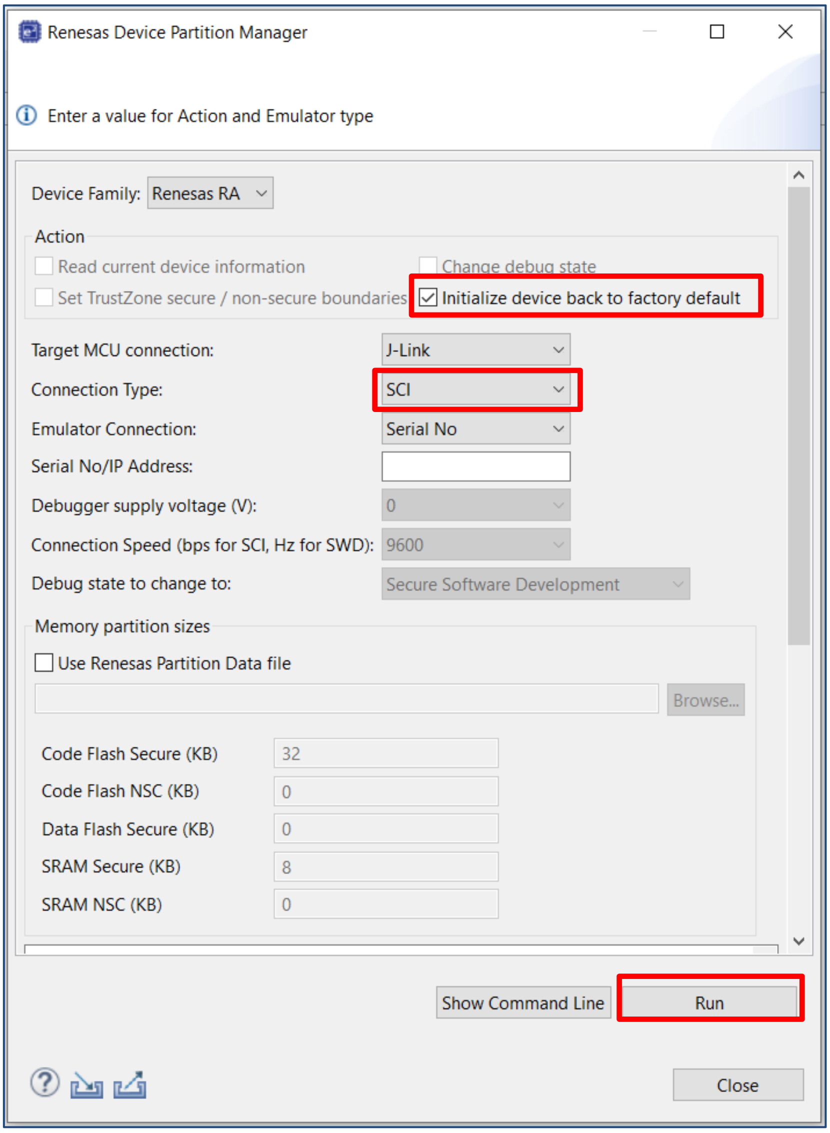


Figure 16. Initialize RA8M1 using Renesas Device Partition Manager

Ensure the following output is achieved and power cycle the EK-RA8M1.

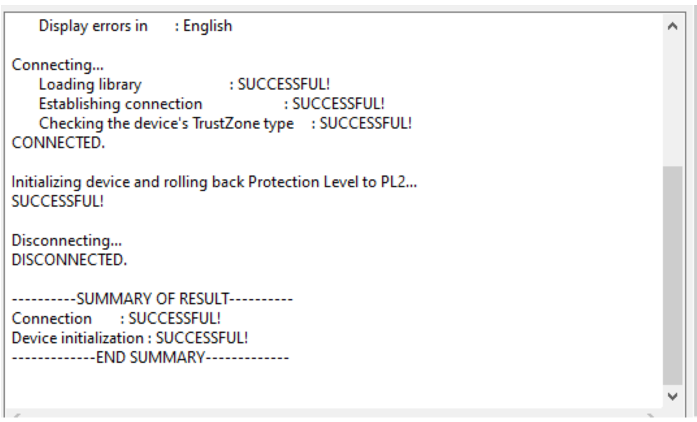


Figure 17. MCU Initialization Successful

2.4.1.2 Erase the OSPI

Additionally, perform these actions to erase the OSPI and then power cycle the EK-RA8M1.

- Unzip the `jlink_scripts.zip`. double click the `\jlink_scripts\erase_ospi_8kB.bat` to erase the first 8kB of the OSPI area. Ensure the following output is achieved.

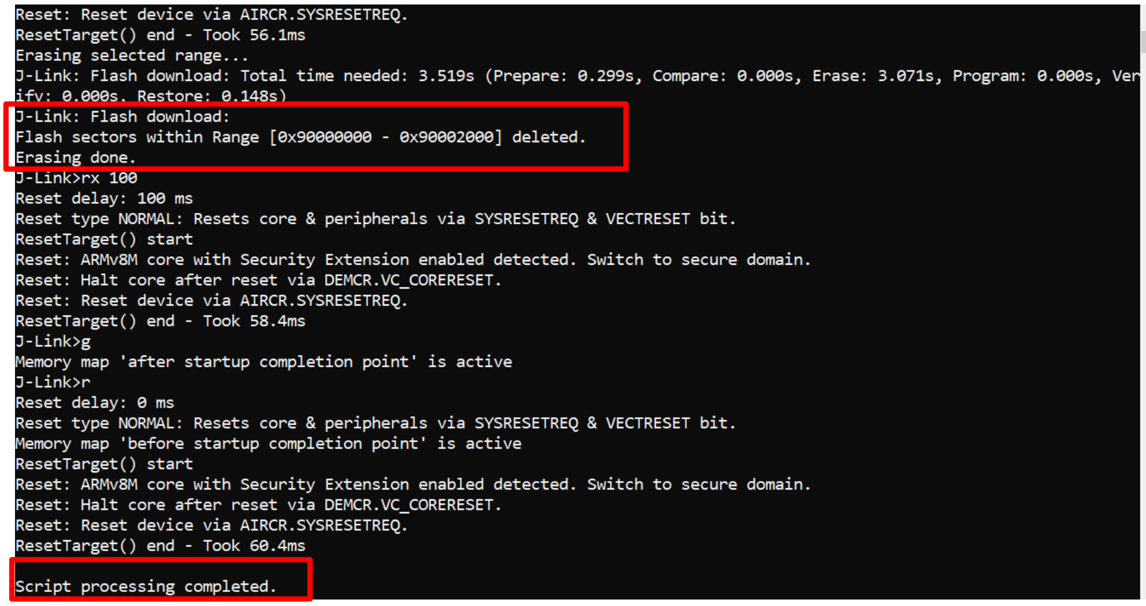


Figure 18. Erase 8kB OSPI Using the Script

- Must Power cycle the EK-RA8M1 before the next step.

For other OSPI applications that use more than 8kB 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 Protected Mode project `dotf_rsip_compatibility_mode_ek_ra8m1.zip`. After the project is imported, double-click configuration.xml to open the RA configurator. Click **Generate Project Content** and build the application project. There are warnings generated from the third-party libraries.

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).

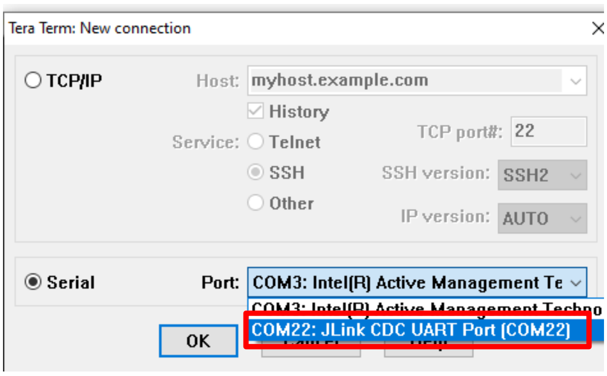


Figure 19. Select the JLink Console Connection

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

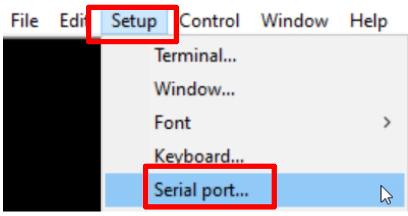


Figure 20. Open the “Serial port” interface

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

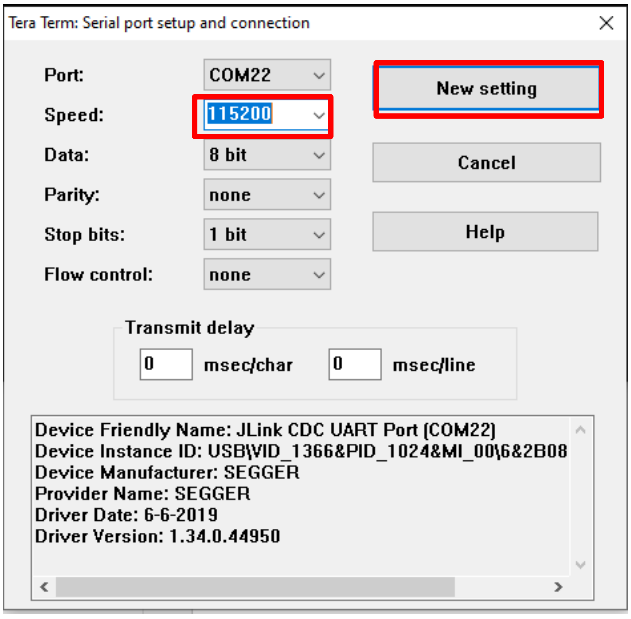


Figure 21. Configure the Tera Term

Resume the Debug session. Similar output as shown in Figure 22 should be observed in the Tera Term terminal. Time used to access the DOTF area is comparable with the time used to access the plaintext OSPI area. The blue LED should be blinking after the evaluation is done.

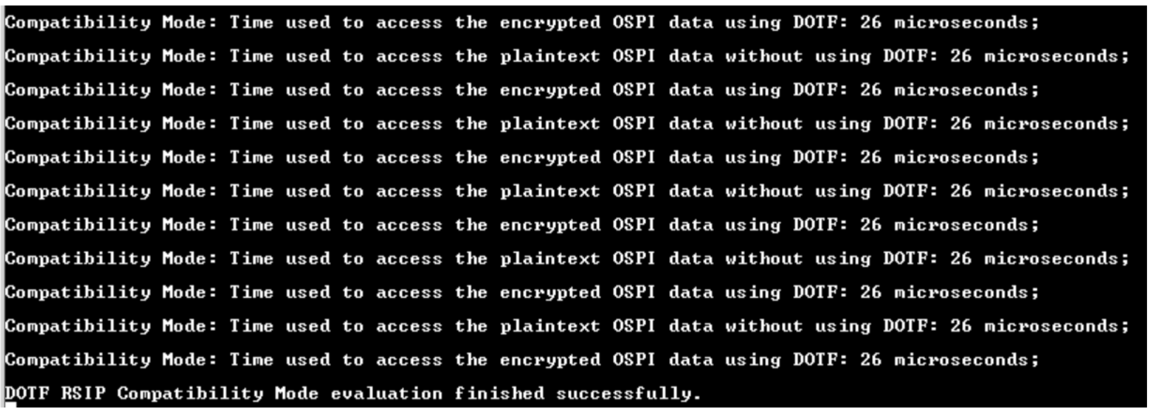


Figure 22. Access Plaintext and DOTF Decrypted Data in Compatibility Mode

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 is included in the application project: the DOTF_AES_128_RA8M1.rkey 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 **RA Family, RSIP-E51A Security Functions and Protected Mode**

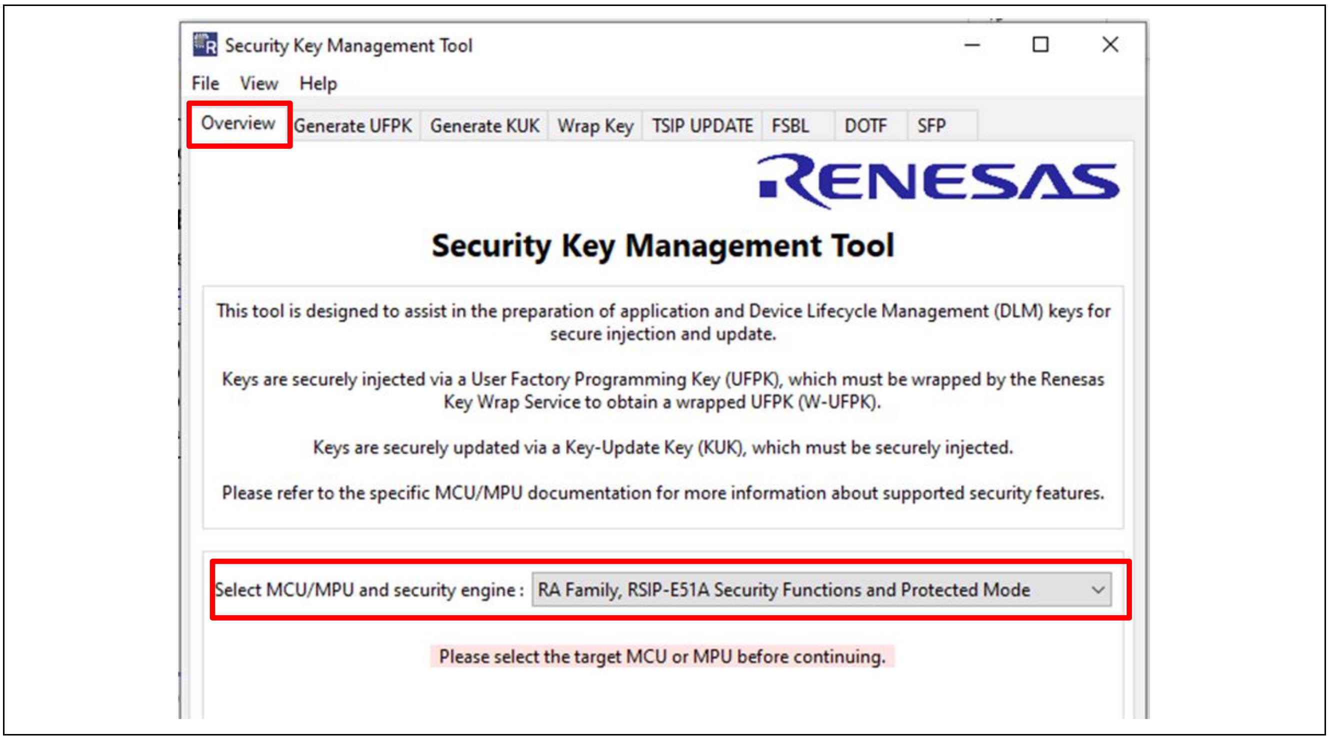


Figure 23. Select RSIP Protected Mode

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)

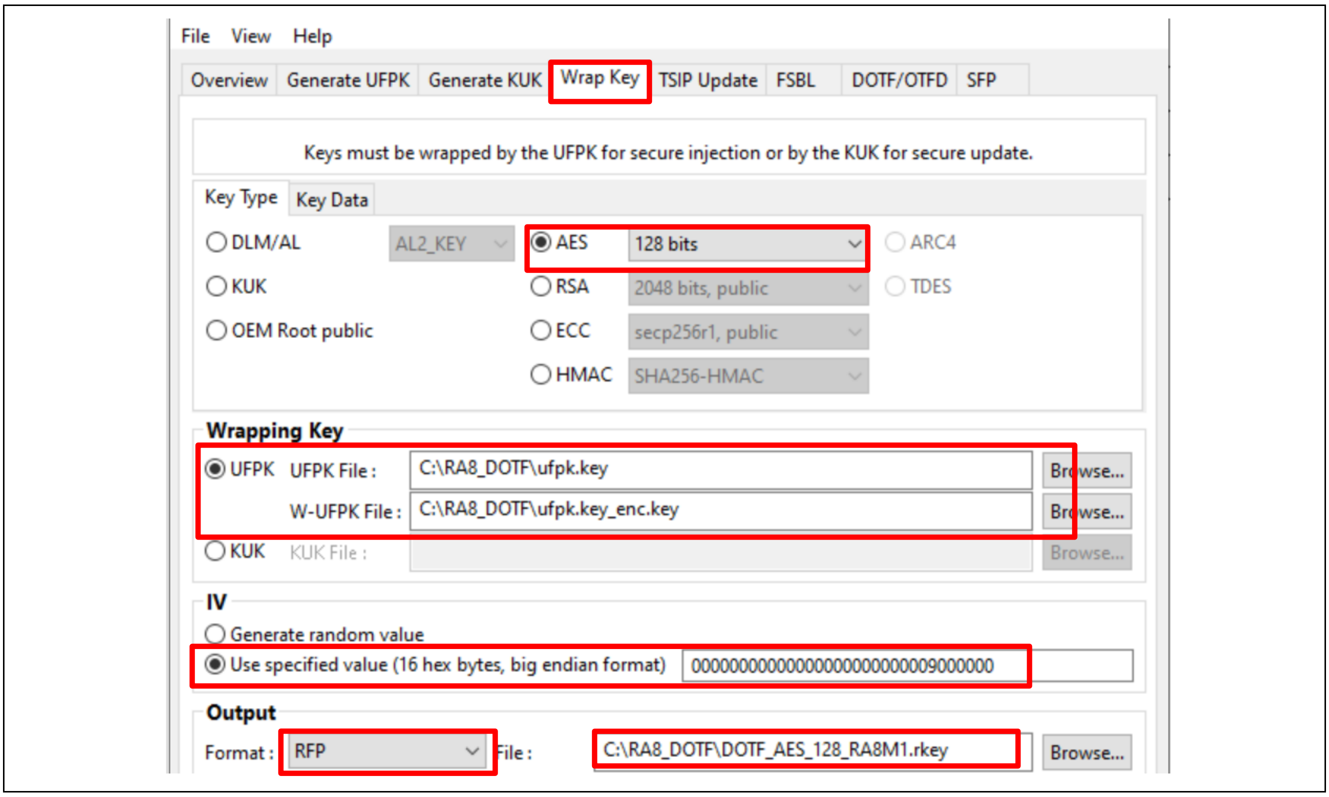


Figure 24. Select AES-128 and Provide WUFPK

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

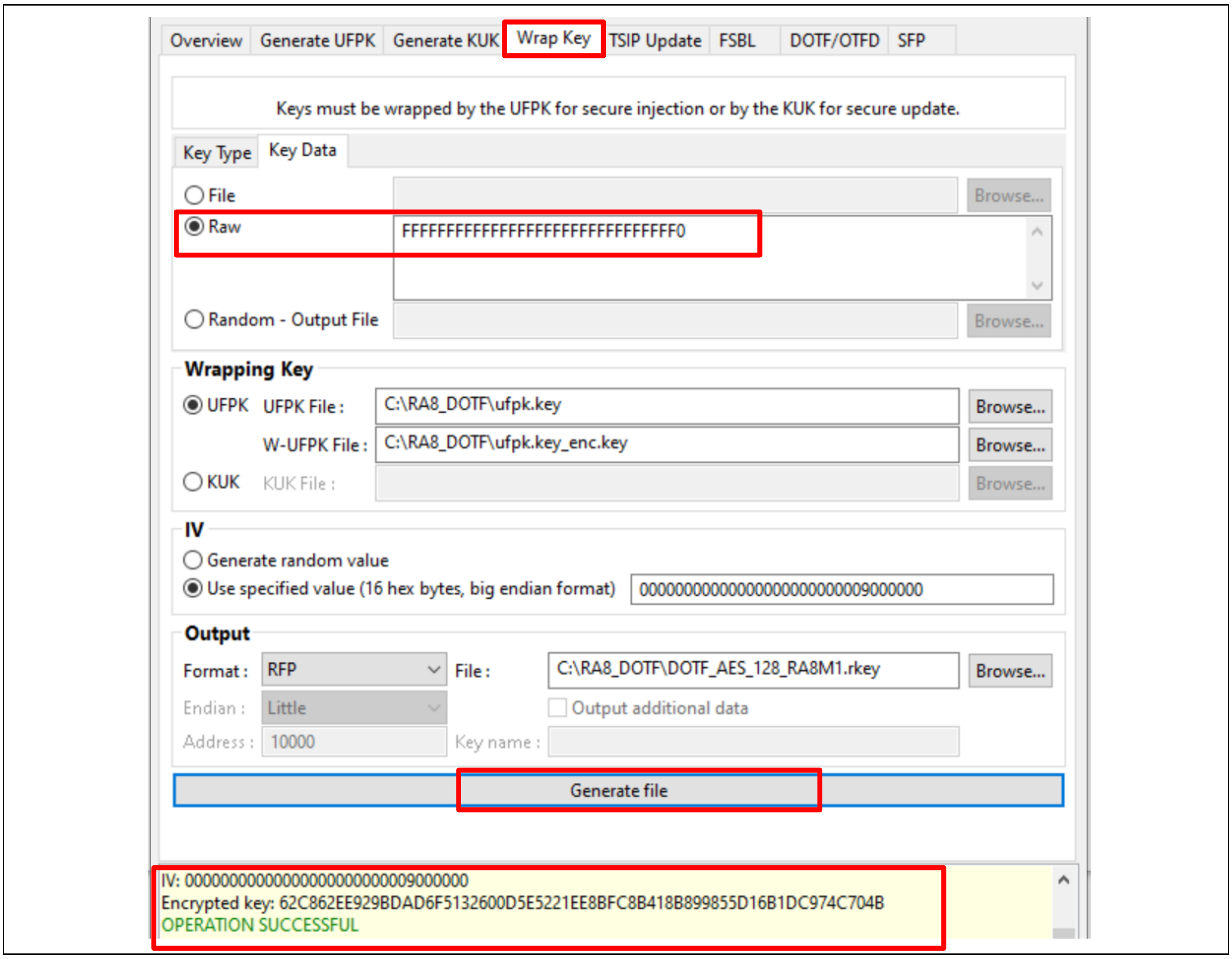


Figure 25. Generate the Wrapped DOTF Key

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 as shown in Figure 7.

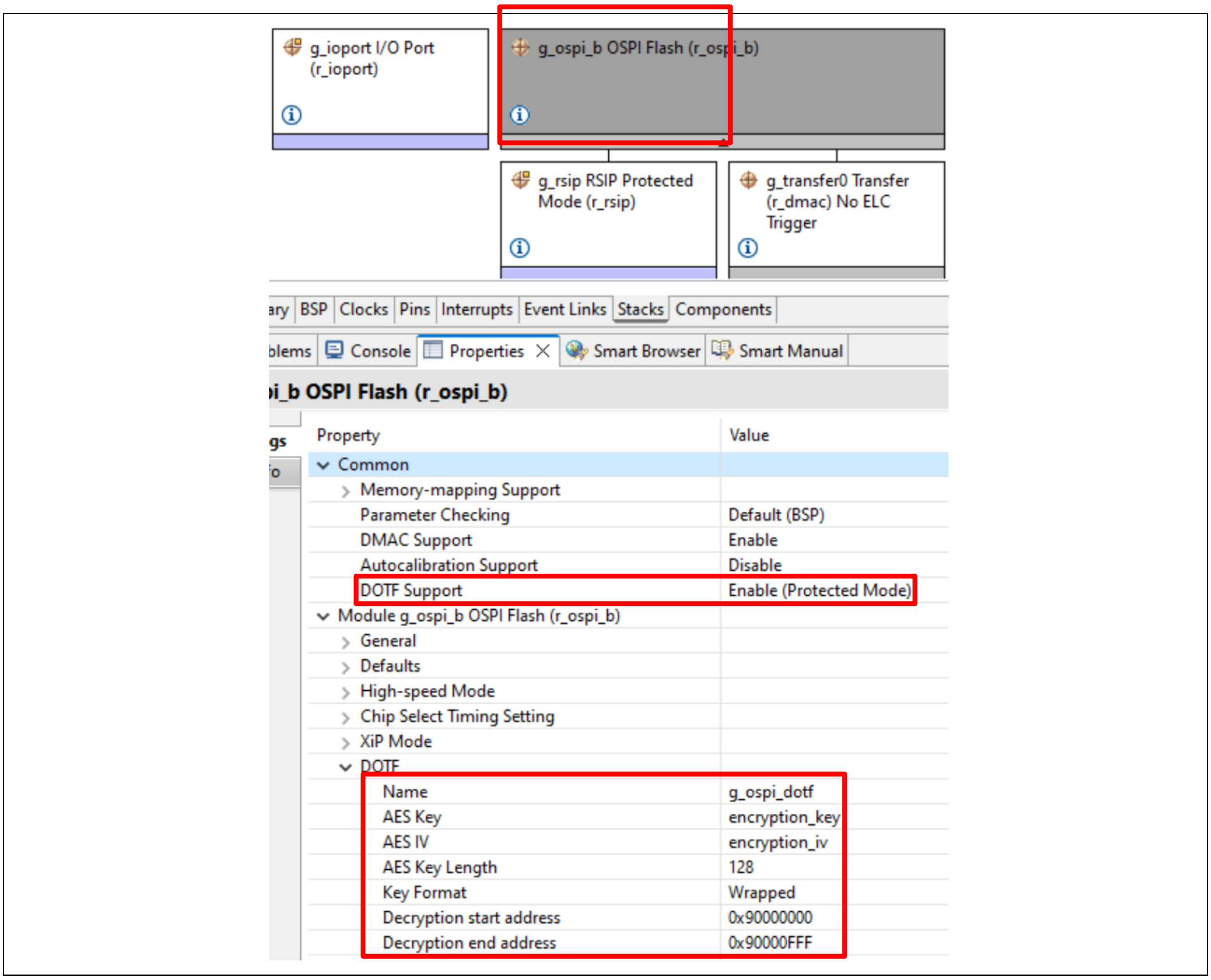


Figure 26. Configure the DOTF with Protected Mode

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.

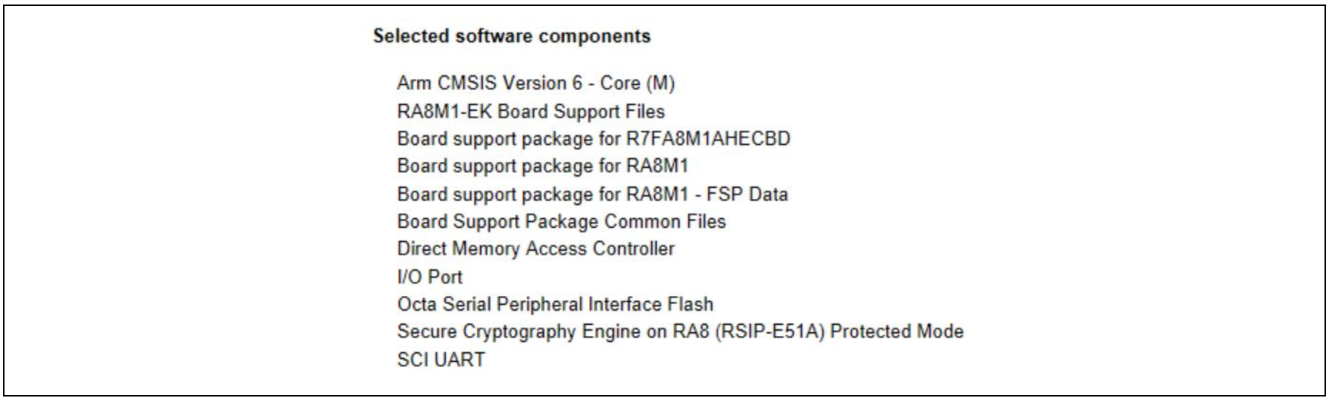


Figure 27. Software Components used for DOTF example with RSIP Protected Mode

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. 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. */
.data_flash (NOLOAD):
{
    . = ORIGIN(DATA_FLASH);
    __tz_DATA_FLASH_S = .;
    __Data_Flash_Start = .;
    KEEP(*(.data_flash*))
    __Data_Flash_End = .;

    __tz_DATA_FLASH_N = DEFINED(DATA_FLASH_NS_START) ? ABSOLUTE(DATA_FLASH_NS_START) : __RESERVE_NS_RAM ? ABSOLUTE(DATA_FLASH_START + DATA_FLASH_LENGTH) : ALIGN(1024);
} > DATA_FLASH
```

Figure 28. Set NOLOAD for the dash_flash section

The updated linker script fsp_app.lld is configured to be used in the example project.

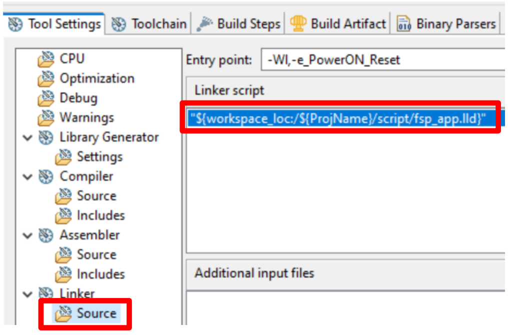


Figure 29. Set NOLOAD for the dash_flash section

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.

When allocating code to the OSPI regions, only one continuous region can be used. The DOTF RSIP Protected Mode example project uses the default "`.OSPI_DEVICE_1`" region linker script configuration as shown in Figure 11. Either of the two subsections "`.ospi_device_1`" or "`.code_in_ospi_device_1`" can be used to allocate the code to the OSPI area as long as all the functions are allocated to one of these two subregions.

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.

```
uint32_t fibonacci(uint32_t num) __attribute__((noinline)) __attribute__((aligned(4096))) BSP_PLACE_IN_SECTION(".ospi_device_1");
```

Figure 30. Allocate a Function to DOTF Destination Area (to be Encrypted by SKMT)

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.

```
uint32_t fibonacci2(uint32_t num) __attribute__((noinline)) __attribute__((aligned(4096))) BSP_PLACE_IN_SECTION(".ospi_device_1");
```

Figure 31. Allocate a Function to OSPI Plaintext Area

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

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`. After the project is imported, double-click configuration.xml to open the RA configurator. Click **Generate Project Content** and build the application project. 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 32.

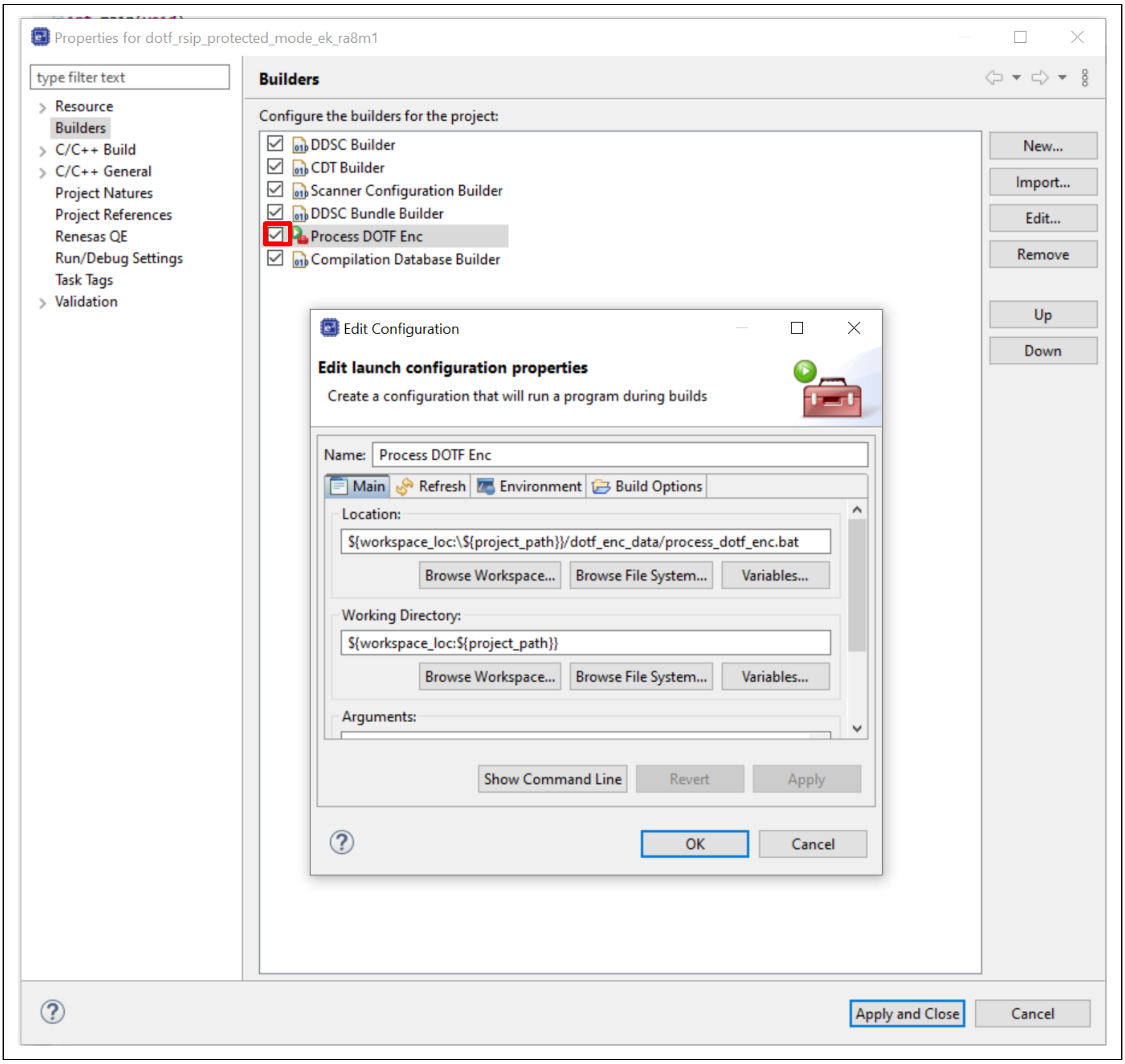


Figure 32. Add a Custom Builder

Figure 33 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_rsip_protected_mode_ek_ra8m1.srec -crop 0x00000000 0x7FFFFFFF -o ra_app_image_wo_ospi_area.srec
srec_cat.exe ..\Debug\dotf_rsip_protected_mode_ek_ra8m1.srec -crop 0x80000000 0x9FFFFFFF -o ra_app_image_ospi_area.srec
dotf_cli_v107\skmt.exe /encdotf /keytype "AES-128" /enckey "FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0" /nonce "00000000000000000000000000000009" /startaddr
"90000000" /endaddr "90000FFF" /prg "./ra_app_image_ospi_area.srec" /incplain /output "./ra_app_image_ospi_area_encrypted_and_plaintext.srec"
```

Figure 33. 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.

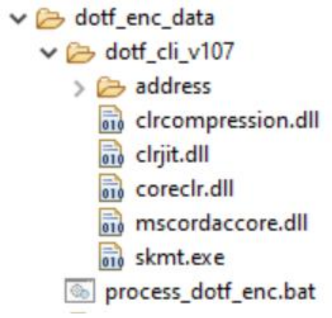


Figure 34. Include the skmt.exe in the e2studio Project

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\github\RA8_DOTF\protected_mode\dotf_rsip_protected_mode_ek_ra8m1>cd dotf_enc_data

C:\RA8_DOTF\github\RA8_DOTF\protected_mode\dotf_rsip_protected_mode_ek_ra8m1\dotf_enc_data>srec_cat.exe ..\Debug\dotf_rsip_protected_mode_ek_ra8m1.srec -
crop 0x00000000 0x7FFFFFFF -o ra_app_image_wo_ospi_area.srec

C:\RA8_DOTF\github\RA8_DOTF\protected_mode\dotf_rsip_protected_mode_ek_ra8m1\dotf_enc_data>srec_cat.exe ..\Debug\dotf_rsip_protected_mode_ek_ra8m1.srec -
crop 0x80000000 0x9FFFFFFF -o ra_app_image_ospi_area.srec

C:\RA8_DOTF\github\RA8_DOTF\protected_mode\dotf_rsip_protected_mode_ek_ra8m1\dotf_enc_data>dotf_cli_v107\skmt.exe /encdotf /keytype "AES-128" /enckey
"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0" /nonce "00000000000000000000000000000009" /startaddr "90000000" /endaddr "90000FFF" /prg "./ra_app_image_ospi_area.srec"
/incplain /output "./ra_app_image_ospi_area_encrypted_and_plaintext.srec"
Output File: C:\RA8_DOTF\github\RA8_DOTF\protected_mode\dotf_rsip_protected_mode_ek_ra8m1\dotf_enc_data\ra_app_image_ospi_area_encrypted_and_plaintext.srec
Key: FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0
Counter: 00000000000000000000000009000000
```

Figure 35. Custom Builder Execution Result

3.6.2 Encrypt the DOTF Destination Area using SKMT GUI

Figure 36 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_ek_ra8m1.mot includes the code flash content, the encrypted DOTF destination area content as well as the plaintext OSPI area content.

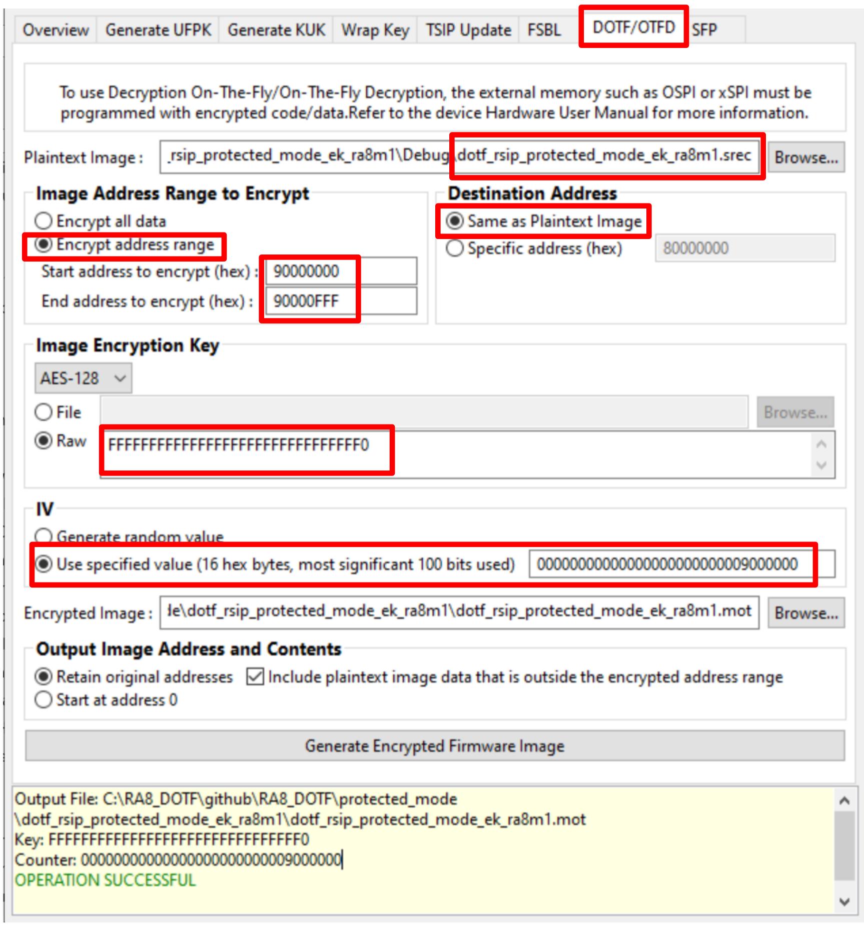


Figure 36. Use SKMT to Encrypt the Data for the DOTF Destination Area

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.

Connect the USB Debug J10 on the EK-RA8M1 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**.

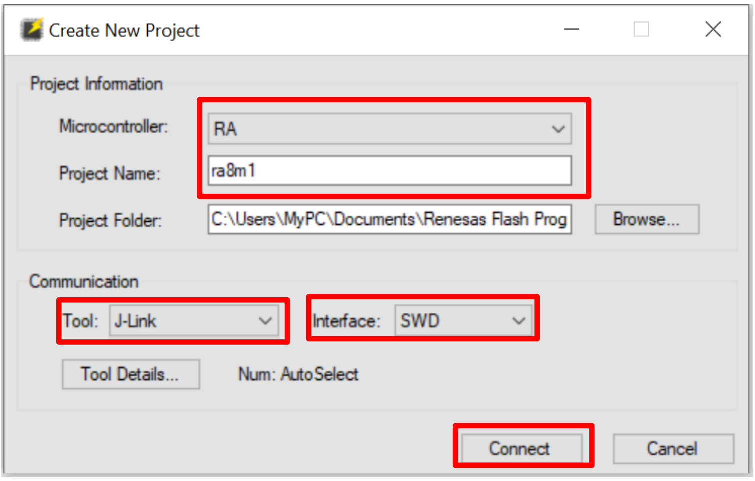


Figure 37. Establish an RFP Project to Communicate with the MCU Boot Interface

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

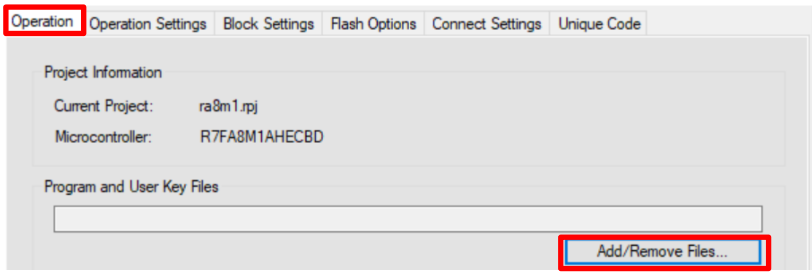


Figure 38. Select the Wrapped DOTF Key

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).

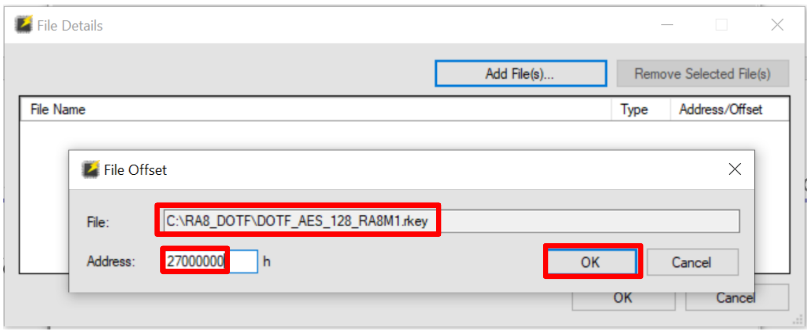


Figure 39. Choose the Data Flash Area to Inject the DOTF key

Configure the following **Operation Settings**.

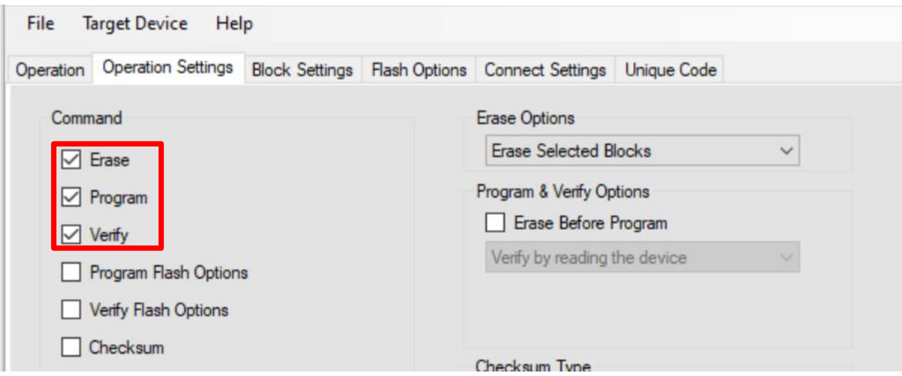


Figure 40. Operation Setting for Injecting the Wrapped DOTF Key

On the **Operation** page, click **Start** to Inject the Wrapped DOTF Key

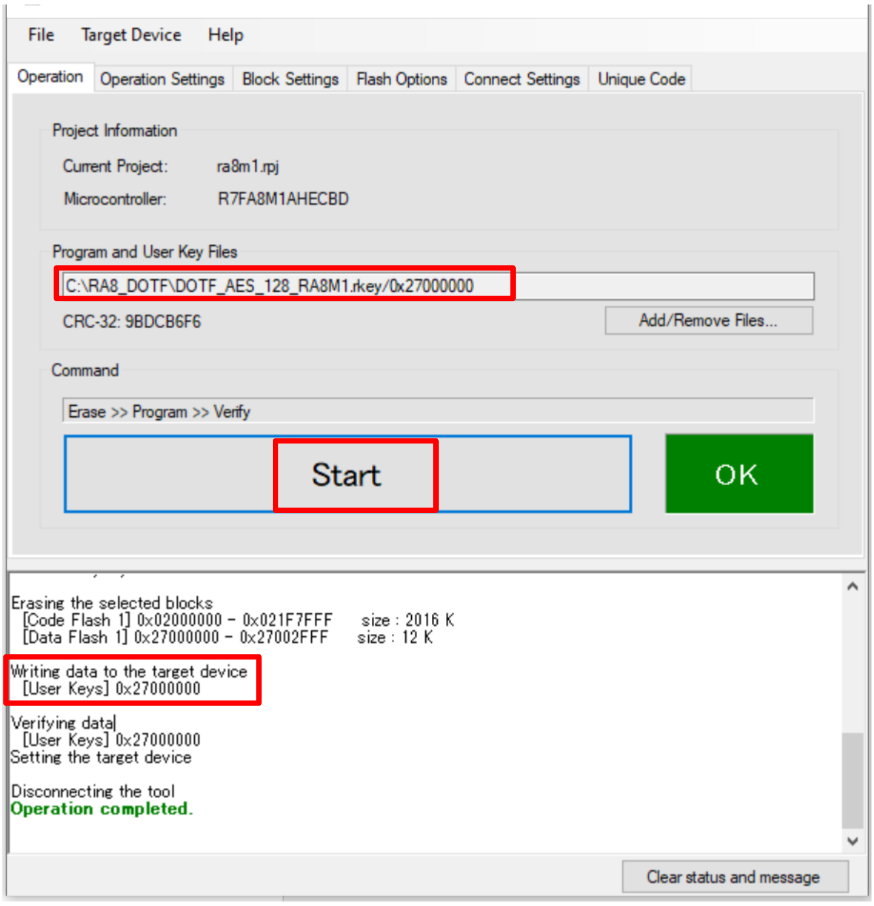


Figure 41. Inject the Wrapped AES128 Key as DOTF Key

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.

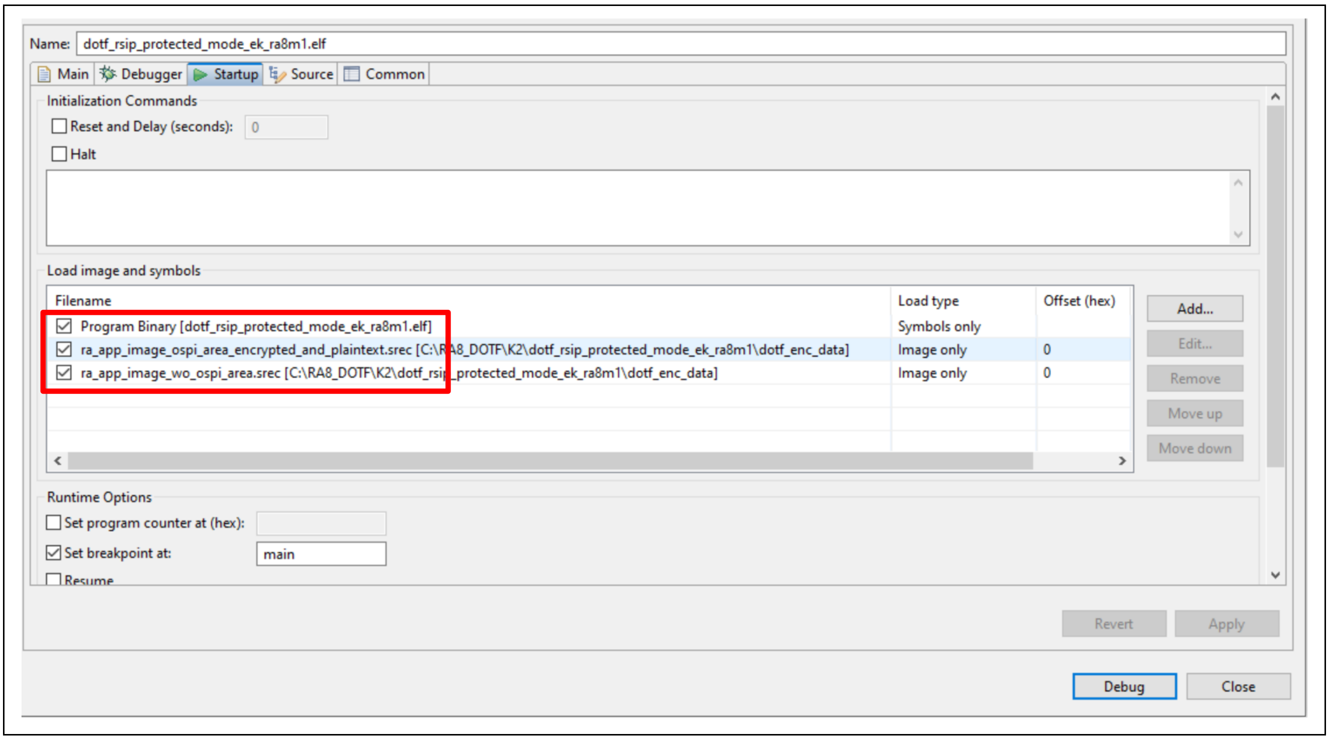


Figure 42. Configure the Debug Configuration

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 43 should be observed. The blue LED should be blinking after the evaluation is done.

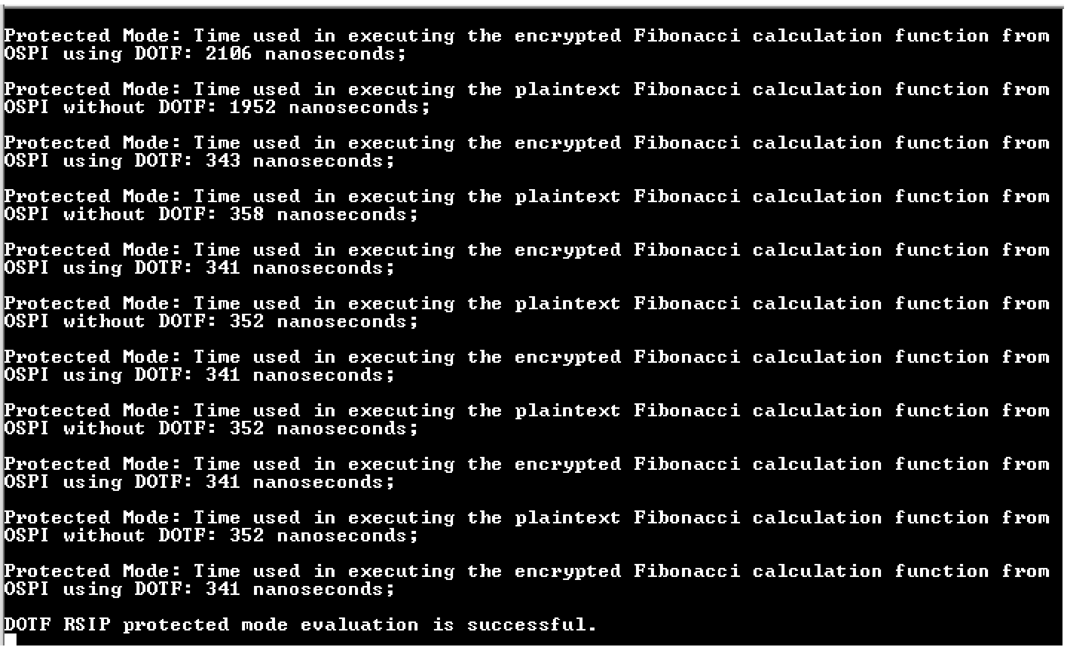


Figure 43. Code Execution Result DOTF RSIP Protected Mode

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 44. 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 80% 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: 3006 nanoseconds;

Protected Mode: Time used in executing the encrypted Fibonacci calculation function from
OSPI using DOTF: 2389 nanoseconds;

Protected Mode: Time used in executing the plaintext Fibonacci calculation function from
OSPI without DOTF: 3006 nanoseconds;

Protected Mode: Time used in executing the encrypted Fibonacci calculation function from
OSPI using DOTF: 2385 nanoseconds;

Protected Mode: Time used in executing the plaintext Fibonacci calculation function from
OSPI without DOTF: 3006 nanoseconds;

Protected Mode: Time used in executing the encrypted Fibonacci calculation function from
OSPI using DOTF: 2393 nanoseconds;

Protected Mode: Time used in executing the plaintext Fibonacci calculation function from
OSPI without DOTF: 3006 nanoseconds;

Protected Mode: Time used in executing the encrypted Fibonacci calculation function from
OSPI using DOTF: 2385 nanoseconds;

Protected Mode: Time used in executing the plaintext Fibonacci calculation function from
OSPI without DOTF: 3006 nanoseconds;

Protected Mode: Time used in executing the encrypted Fibonacci calculation function from
OSPI using DOTF: 2385 nanoseconds;

Protected Mode: Time used in executing the plaintext Fibonacci calculation function from
OSPI without DOTF: 3006 nanoseconds;

Protected Mode: Time used in executing the encrypted Fibonacci calculation function from
OSPI using DOTF: 2393 nanoseconds;

DOTF RSIP protected mode evaluation is successful.
```

Figure 45. Testing Result for DOTF Overhead Evaluation

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 46.

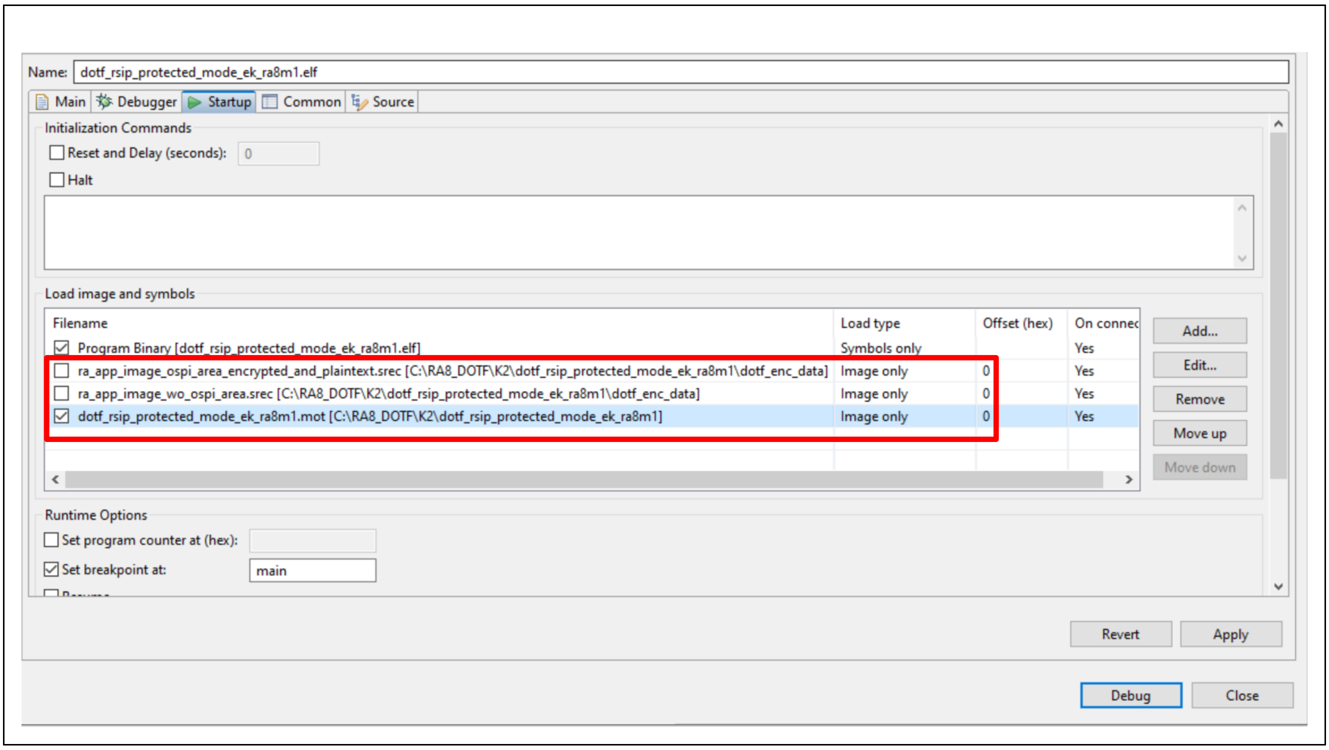


Figure 46. Update the RSIP Protected Mode DOTF Example Project Debug Configuration

Next Start the **Debug** session and use Tera Term to observe the execution result. A result similar to Figure 43 should be observed.

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 36 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.

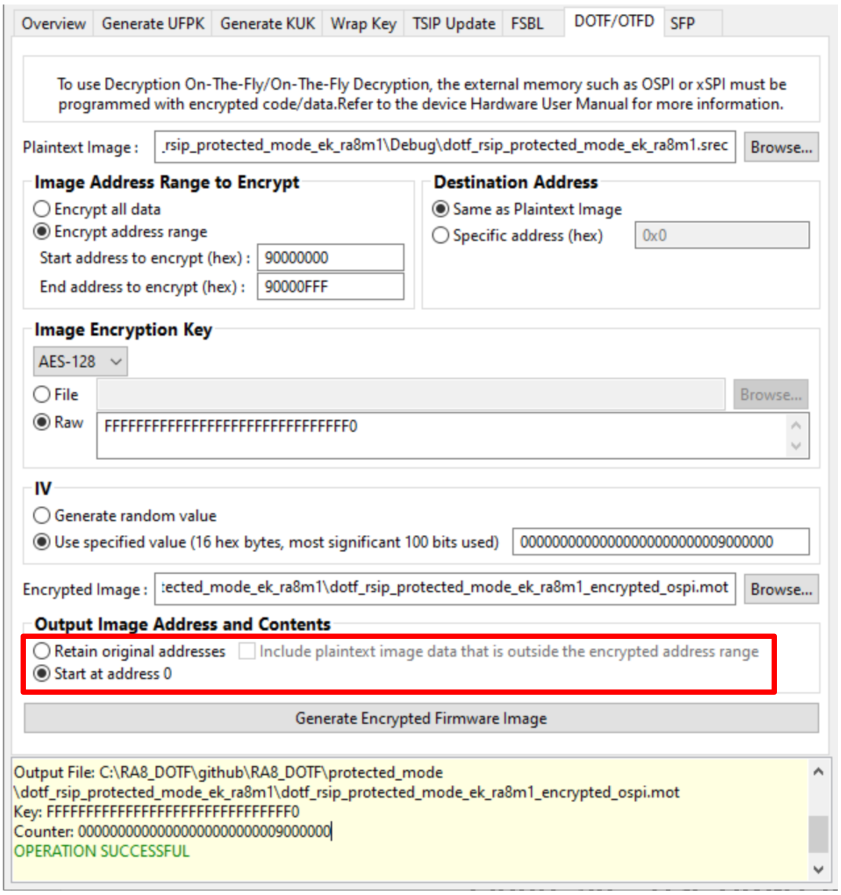


Figure 47. Example: Generate Encrypted OSPI Data for Third-party Tool

5. Appendix

5.1 Update the Linker Script for the Compatibility Mode Example Project

The included Compatibility Mode example project uses GCC compiler. As shown in Figure 12, the default FSP linker script is updated to help OSPI data allocation.

To achieve the same goal when LLVM is used, the same linker script updates as the GCC compiler should be performed.

To achieve the same goal when AC6 is used, the following linker script updates should be performed.

```
LOAD_REGION_OSPI_DEVICE_1 OSPI_DEVICE_1_START OSPI_DEVICE_1_PRV_LENGTH
{
  __tz_OSPI_DEVICE_1_S OSPI_DEVICE_1_S_START EMPTY 0
  {
  }
  OSPI_DEVICE_1 +0 FIXED
  {
    *(.ospi_device_1*)
    . = OSPI_DEVICE_1_START + 0x1000;                    /* Offset by 0x1000 starting from OSPI_DEVICE_1_START*/
    ospi_device_1_plaintext_start = .;
    *(.code_in_ospi_device_1*)
  }
  __tz_OSPI_DEVICE_1_N OSPI_DEVICE_1_NS_START EMPTY 0
  {
  }
}
```

Figure 48. Update 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 28, 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.

```
fsp_app.ld
        KEEP(*(.ns_buffer*))
    } > RAM

    /* Data flash. */
    .data_flash (NOLOAD):
    {
        . = ORIGIN(DATA_FLASH);
        __tz_DATA_FLASH_S = .;
        __Data_Flash_Start = .;
        KEEP(*(.data_flash*))
        __Data_Flash_End = .;

        __tz_DATA_FLASH_N = DEFINED(DATA_FLASH_NS_START) ? ABSOLUTE(DATA_FLASH_NS_START) : __RESERVE_NS_RAM
    } > DATA_FLASH
```

Figure 49. 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. NOLOAD key word is used similar to the LLVM linker script update.

```
LOAD_REGION_DATA_FLASH DATA_FLASH_START DATA_FLASH_LENGTH NOLOAD
{
  __tz_DATA_FLASH_S DATA_FLASH_S_START EMPTY 0
  {
  }
  DATA_FLASH +0
  {
    *(.data_flash*)
  }
  __tz_DATA_FLASH_N DATA_FLASH_NS_START EMPTY 0
  {
  }
}
```

Figure 50. Update the AC6 Linker Script for the Project Mode Project

6. References

1. Flexible Software Package (FSP) User's Manual
2. Renesas RA8M1 Group User's Manual: Hardware
3. Renesas RA Family RA8 MCU Series Device Lifecycle Management (R11AN0785)
4. Renesas RA Family MCU Injecting and Updating Secure User Keys (R11AN0496)
5. Renesas RA Family MCU Injection Plaintext User Keys (R11AN0473)
6. Renesas RA Family MCU Renesas RA Family Security Engine Operational Modes (R11AN0498)

7. 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
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

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

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/.

Trademarks

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