How to use OTFDEC for encryption/decryption in trusted environment on STM32H7Bxxx and STM32H73xx microcontrollers

Introduction

The OTFDEC (on-the-fly decryption) engine allows on-the-fly decryption of the Arm® AXI or AHB traffic based on the read request address information. In addition, OTFDEC can encrypt code to be stored in an external Octo-SPI memory using a proprietary protection layer over the advanced encryption standard (AES) in counter mode.

This application note explains how to use the OTFDEC engine in a trusted environment using the STM32H7Bxxx and STM32H73xx microcontrollers:

- To decrypt on-the-fly data or code located in external Octo-SPI memories used in memory mapped mode that has been encrypted using a standard AES-128 counter mode part of the AES hardware accelerator or the proprietary protection layer over the standard AES-128 in counter mode part of the OTFDEC RSS services.
- To encrypt binary image using OTFDEC RSS services.

Sample code illustrates how to configure user application firmware to use the OTFDEC engine for encryption and decryption. Code examples are integrated in the STM32Cube release associated to the STM32 microcontrollers supporting OTFDEC (listed in Table 1. Applicable products).

This document describes from a user perspective the use of AES and OTFDEC standard and intends to be abstract of the hardware.

For further information on OTFDEC in STM32 devices, refer to the product reference manuals available on *www.st.com*.

Table 1. Applicable products

Type	Products
Microcontrollers	STM32H7B0AB, STM32H7B0IB, STM32H7B0LB, STM32H7B0RB, STM32H7B0VB, STM32H7B0ZB, STM32H7B3AI, STM32H7B3II, STM32H7B3LI, STM32H7B3NI, STM32H7B3QI, STM32H7B3RI, STM32H7B3VI, STM32H7B3ZI, STM32H730AB, STM32H730IB, STM32H730VB, STM32H730ZB, STM32H733VG, STM32H733ZG, STM32H735AG, STM32H735IG, STM32H735RG, STM32H735VG, STM32H735ZG.

1 General information

This document applies to STM32H7Bxxx and STM32H73xx Arm®-based microcontrollers.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

2 Encryption with standard AES-128 in counter mode using the AES hardware accelerator

The purpose of this section is to explain how to organize the data memory used for the encryption with the standard AES-128 in counter mode.

All STM32 products listed in Table 1. Applicable products embed the AES hardware accelerator in counter mode.

2.1 Endianness notation

The following notation is used in this document (see the table below):

- For each 128-bit data, a little-endian notation is used, with the bit on the left is noted as bit 127 and the bit on the right is noted as bit 0.
- For each 32-bit data, a little-endian notation is used, with the bit on the left is noted as bit31 and the bit on the right is noted as bit0.
- The 128-bit data consists of 4x 32-bit words, the left one is noted as the W3 and the right one as W0.

Table 2. Data structure illustration

Data	Bit number	Word number			
		W3	W2	W1	W0
IV[127:0]	b127: 0	0xF0F1F2F3	0xF4F5F6F7	0xF8F9FAFB	0xFCFDFEFF
KEY[127:0]	b127:0	0x2B7E1516	0x28AED2A6	0xABF71588	0x09CF4F3C
PLAIN_DIN[127:0]	b127:0	0x6BC1BEE2	0x2E409F96	0xE93D7E11	0x7393172A
CRYP_DOUT[127:0]	b127:0	0x874D6191	0xB620E326	0x1BEF6864	0x990DB6CE

In the rest of the document, it is important to understand that the endianness notation is referring to the loading order of the 4x32 bit words within the 128 bits word and the 32 bits word ordering does not change, as seen in the following table.

Table 3. 4x32 bit word endianness notation

Memory address	Little endian notation	Big endian notation
0x24000000	W0[31..0]	W3[31..0]
0x24000004	W1[31..0]	W2[31..0]
0x24000008	W2[31..0]	W1[31..0]
0x2400000C	W3[31..0]	W0[31..0]
0x24000010	W4[31..0]	W7[31..0]
0x24000014	W5[31..0]	W6[31..0]
0x24000018	W6[31..0]	W5[31..0]
0x2400001C	W7[31..0]	W4[31..0]

CPU memories and OTFDEC follow the little endian notation whereas AES hardware accelerator follows the big endian notation.

2.2 How to correctly configure keys and vectors for AES-128 in counter mode

To decrypt data stored in external Octo-SPI memory, the OTFDEC needs the following information:

- a 128-bit key
- a 64-bit nonce
- a 16-bit region firmware version
- a 2-bit OTFDEC regions identifier
- a 28-bit external Octo-SPI memory start address (a memory address is made of 32 bits, however the four last bits are ignored).

To run the AES-128 in counter mode, configure the OTFDEC with the correct IV (initialization vector) and key values. The memory mapping configuration parameters to apply is illustrated in the tables below.

Table 4. Memory mapping (SRAM example)

Memory address	Items
0x24000000	Key 0 [31:0]
0x24000004	Key 1 [31:0]
0x24000008	Key 2 [31:0]
0x2400000C	Key 3 [31:0]
0x24000010	Nonce 0 [31:0]
0x24000014	Nonce 1 [31:0]

Table 5. AES-IV[127..0] mapping

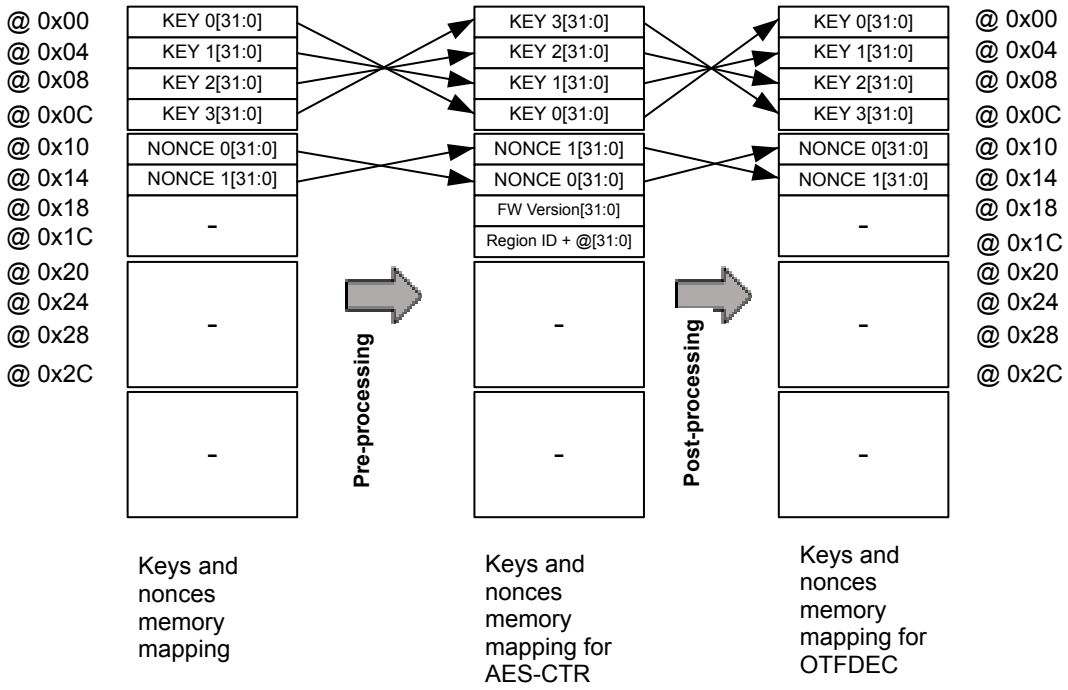
Words number						
W3	W2	W1		W0		
Bits number						
127:96	95:64	63:48	47:32	31:30	29:28	27:0
Description						
Nonce 1	Nonce 0	Not used	Region FW version	Not used	Region ID	External memory start @ (modulo 128-bit)

Table 6. AES-KEY[127..0] mapping

Words number			
W3	W2	W1	W0
Bits number			
127:96	95:64	63:32	31:0
Description			
Key 3	Key 2	Key 1	Key 0

The figure below illustrates the configuration of the AES-128 counter mode initial vector and keys.

Figure 1. AES-128 counter mode initial vector and keys register configuration



2.3 How to process plain data or code for encryption with AES-128 in counter mode

In all STM32 devices, the memory is organized in little-endian format. The AES hardware accelerator requires plain data in big-endian format. OTFDEC requires encrypted data in little-endian format. Therefore, there is a need to pre-process and post-process the data to be compatible with AES and OTFDEC.

Pre-processing converts the plain data stored in memory from little endian format to big endian format as shown in the table below.

Table 7. Plain data memory mapping (Flash example)

Flash address	Word number	Plain data	Pre processing conversion results	
			Word number	Plain data
0x08000000	W0	0x7393172A	W3	0x6BC1BEE2
0x08000004	W1	0xE93D7E11	W2	0x2E409F96
0x08000008	W2	0x2E409F96	W1	0xE93D7E11
0x0800000C	W3	0x6BC1BEE2	W0	0x7393172A
Until end of binary image			Until end of binary image	

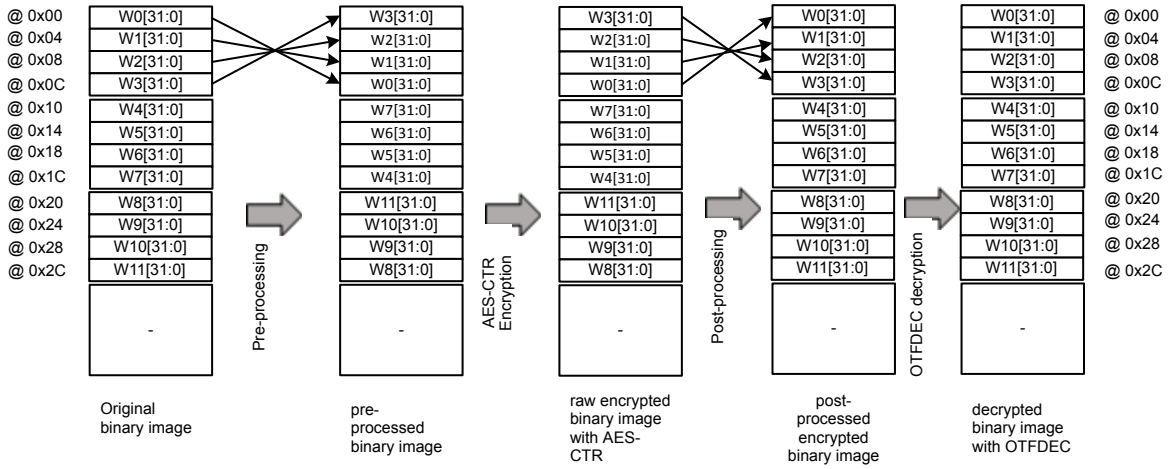
Post-processing converts the encrypt data stored in memory from big endian format to little endian format as shown in the table below.

Table 8. Encrypted data memory mapping (internal Flash example)

Flash address	Word number	Crypt data	Post processing result	
			Word number	Crypt data
0x08000000	W3	0x874D6191	W0	0x990DB6CE
0x0800 0004	W2	0xB620E326	W1	0x1BEF6864
0x08000008	W1	0x1BEF6864	W2	0xB620E326
0x0800000C	W0	0x990DB6CE	W3	0x874D6191
Until the end of the binary image			Until the end of the binary image	

The figure below illustrates the configuration steps of the AES-128 in counter mode for the encryption and OTFDEC for the decryption.

Figure 2. AES-128 in counter mode encryption and OTFDEC decryption process diagram



3 Encryption with proprietary protection layer over the standard AES-128 in counter mode

The purpose of this section is to describe how to encrypt code using the proprietary protection layer over the standard AES in counter mode part of the OTFDEC RSS services. OTFDEC only encrypts an executable binary to be stored in the external Octo-SPI memory. In this case, data must not be encrypted as it won't be decrypted.

In fact, this encryption method calls on the root secure services (RSS) that is executed after a system reset when the security feature is enabled and before executing any other software stored in the device. Using this additional proprietary encryption layer adds robustness to the standard AES in counter mode. To execute an OTFDEC RSS service systematically requires a device reboot. Depending on the size of the binary image to encrypt, this image is cut into manageable slices. Each slice is encrypted separately resulting in a device reboot. The number of reboots depends on the size of the code and the size of the internal SRAM. This defines the number of slices to be produced.

3.1 RSS services to use for OTFDEC

The RSS services used for the OTFDEC encryption are detailed in the tables below.

Table 9. resetAndEncrypt

Prototype	`Void resetAndEncrypt (RSS_OTFD_ENC_ Params_t* otfd_enc_data)`

Arguments	`typedef struct`
	`{`
	`  struct`
	`  {`
	`  uint32_t dest;  /* Octo-SPI memory @ to store the encrypted code */`
	`  uint8_t* buffer;  /* SRAM @ where code to encrypt is loaded and encrypted code`
	`  is located once RSS service called */`
	`  uint32_t size;  /* size of the SRAM code to encrypt */`
	`  }data;`
	`  struct`
	`  {`
	`  uint32_t number;  /* OTDFEC region number. Following number allowed`
	`  0: OTFDEC1 region 1`
	`  1: OTFDEC1 region 2`
	`  2: OTFDEC1 region 3`
	`  3: OTFDEC1 region 4`
	`  4: OTFDEC2 region 1`
	`  5: OTFDEC2 region 2`
	`  6: OTFDEC2 region 3`
	`  7: OTFDEC2 region 4 */`
	`  uint32_t mode;  /* encryption mode. Following mode allowed`
	`  0: Only instruction accesses are decrypted`
	`  1: Only data accesses are decrypted`
	`  2: All read accesses are decrypted (instruction or data)`
	`  3: Only instruction accesses are decrypted, and enhanced`
	`  encryption mode is activated */`
	`  uint32_t version;  /* region version, not used to be set to 0 */`
	`  void* nonce;  /* address of 64 bits nonce */`
	`  void* key;  /* address of 128 bits region key */`
	`  }region;  /* info about OTFD region */`
	`}RSS_OTFD_ENC_Params_t;`

Description	This service encrypts code loaded in the dedicated SRAM area. Depending of the size of the code to encrypt, several calls to this service can be requested. This service is located in the RSS secure area at the address 0x1FF09560 as defined in the RSS_OTFD.h. A system reset is triggered as soon as the service is called.

3.2 How to encrypt with OTFDEC and write code to external Octo-SPI memory

The principle of encrypting code with a proprietary protection layer above the standard AES in counter mode is as follows:

1. Make sure the security feature is enabled in order to activate the RSS services.
2. Copy the code to encrypt to the pre-defined SRAM area.
3. Call RSS service `resetAndEncrypt` with the argument `otfd_enc_data` as defined in Table 9 and described below:
   - Data information:
     - Dest: external Octo-SPI Flash address to store the encrypted code. Even if the RSS service resetAndEncrypt does not copy the encrypted code into the external Octo-SPI memory, it uses the destination address to encrypt the code.
     - Buffer: SRAM address where encrypted code is loaded.
     - Size: size of code in SRAM to encrypt.
   - OTFDEC region information:
     - Number: OTFDEC region number (0 to 7).
     - Mode: to be set to 3, only instruction accesses are decrypted, and enhanced encryption mode is activated (this is the purpose of this section).
     - Version: Not used, set to 0.
     - Nonce: address of the 64 bits nonce.
     - Keys: address of the 128 bits keys used for the above defined region.

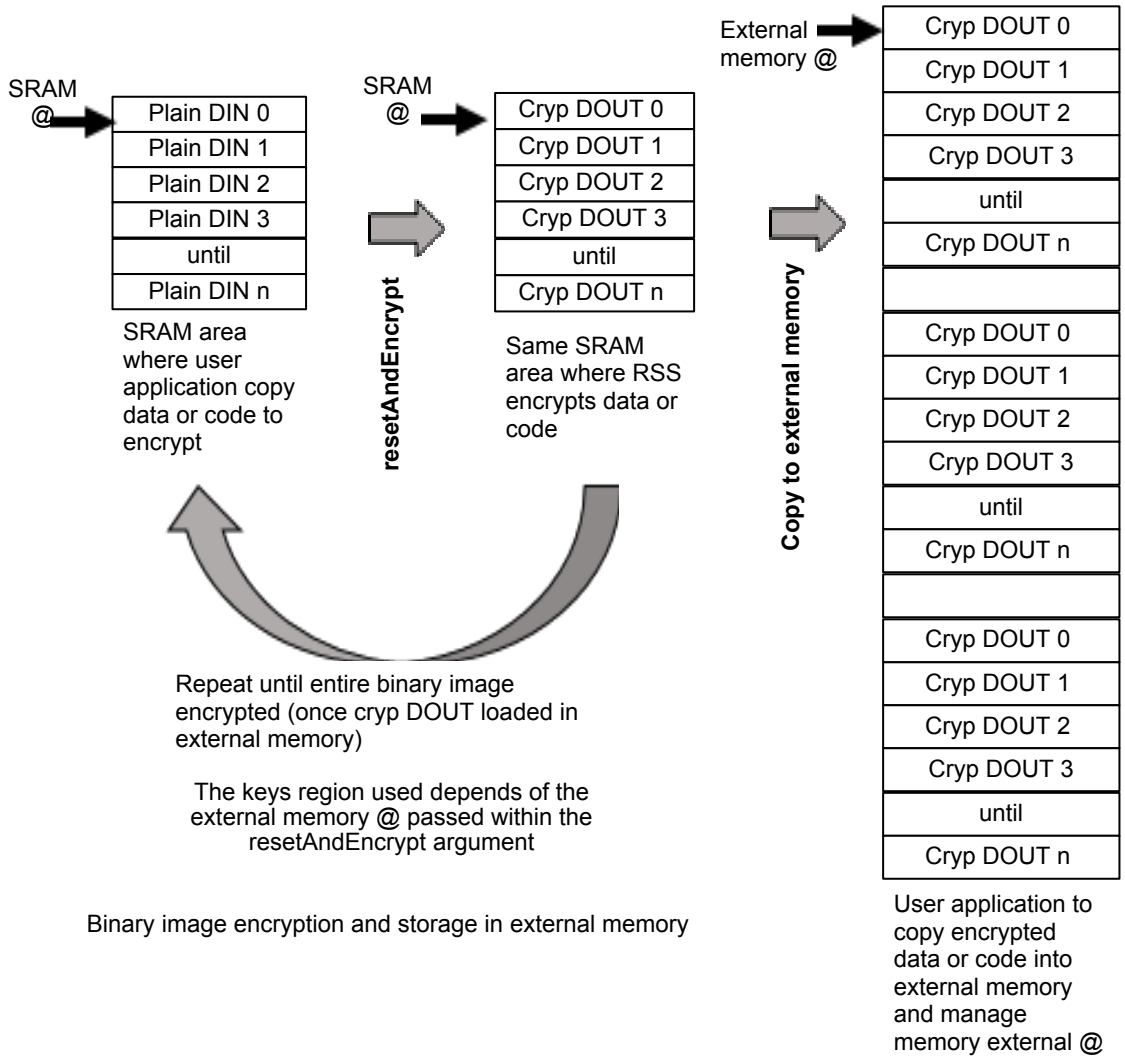
   The OTFDEC region information as part of the RSS service resetAndEncrypt must be the one loaded in the selected OTFDEC instances to decrypt the code in the Octo-SPI external flash memory at the destination adresse (Dest).

   The RSS service `resetAndEncrypt` triggers a system reset.
4. After reboot, the code in the pre-defined SRAM area has been encrypted by the OTFDEC RSS service. Write the encrypted code to the external Octo-SPI Flash memory at the destination address (Dest).
5. If there is still code to encrypt, repeat from step 2.
6. The security feature can be optionally disabled.

Note: The RSS encryption method can be used without a proprietary protection layer by selecting mode to 0, 1 or 2 in the OTFDEC region information. In this case, the standard AES is used.

The figure below shows the parameter memory mapping used to encrypt code with a proprietary protection layer above the standard AES in counter mode.

Figure 3. Binary image encryption and storage in external Octo-SPI memory



3.3 Constraints

All constraints to be taken into account when using the OTFDEC encryption through RSS services are listed below:

- Following each SRAM code encryption, the user application must write the encrypted code to the external Octo-SPI memory before calling a new encryption request via the `resetAndEncrypt` RSS service.
- After each OTFDEC RSS service call, a system reset is triggered, then the user application must ensure that global variables that need to be reused won't be corrupted by the scatter file (not be reset to a predefined value) or the RSS services that uses the following DTCM area: 0x2000800 to 0x20005000.
- In order to avoid too many system reset due to the encryption (OTFDEC `resetAndEncrypt` RSS service), it is important to define a SRAM area as big as possible to store the code to encrypt.

4 On-the-fly decryption

The embedded OTFDEC decrypts in real-time the encrypted content. This content is stored in the external Octo-SPI memories used in memory-mapped mode. This section explains how to configure the OTFDEC to decrypt code or data depending of the encryption method used:

- Standard AES-128 in counter mode
- Proprietary layer over the standard AES-128 in counter mode.

4.1 On-the-fly decryption of encrypted data or code with a standard AES-128 in counter mode

The decryption of encrypted data or code with a standard AES-128 in counter mode as described in Section 2 is outlined here:

1. Configure one of the four OTFDEC regions (same as the one used for the encryption using the standard AES-128 in counter mode) with the following parameters:
   – 128-bit key used for the AES-128 in counter mode encryption.
   – 64-bit nonce used for the AES-128 in counter mode encryption.
   – 16-bit region firmware version used for the AES-128 in counter mode encryption.
   – 32-bit external Octo-SPI memory start address used for the AES-128 in counter mode encryption.
   – 32-bit external Octo-SPI memory end address used for the AES-128 in counter mode encryption.
2. Program the selected OTFDEC region in one of the following modes (MODE bitfield in OTFDEC_RxCFGR register):
   – Only instruction accesses are decrypted (MODE = 00).
   – Only data accesses are decrypted (MODE = 01).
   – All read accesses are decrypted for both instruction and data (MODE = 10) .
3. Enable the memory map mode of the external Octo-SPI memory Flash controller.

Note: Data and code in the external Octo-SPI memory must be in little-endian format as described in Section 2 .

4.2 On-the-fly decryption of encrypted code with the proprietary layer above the standard AES-128 in counter mode

The principle of decrypting code encrypted with the proprietary layer over the standard AES-128 in counter mode as described in Section 2 , is as follows:

1. Configure one of the four OTFDEC regions (same as the one used for the encryption using the proprietary layer on top the standard AES-128 in counter mode) with the following parameters:
   – 128-bit key depending of the selected region to encrypt.
   – 64-bit nonce as defined for the encryption.
   – 16 bit region firmware version (set to 0).
   – 32-bit external Octo-SPI memory start address used for the AES-128 in counter mode encryption.
   – 32-bit external Octo-SPI memory end address used for the AES-128 in counter mode encryption.
2. Program the selected OTFDEC region with the following mode (MODE bitfield to 11 in OTFDEC_RxCFGR register, other modes do not activate the proprietary layer over the AES and therefore the decryption fails):
   – Only instruction accesses are decrypted, and enhanced encryption mode is activated.
3. Enable the memory map mode of the external Octo-SPI Flash controller.

Note: In this case, only code can be decrypted and executed. The external memory data read always returns zero, so it is crucial not to store data in a region programmed in this mode.

5 Firmware use-case examples

This section provides the two following examples:

- Encryption of a binary image using the standard AES-128 in counter mode, writing the encrypted binary image to external Octo-SPI memory and execution of the image.
- Encryption of a binary image using the proprietary layer over the standard AES-128 in counter mode, writing the encrypted binary image to external Octo-SPI memory and execution of the image.

5.1 Example using standard AES-128 in counter mode

5.1.1 Overview

In this example, a code calculating PI (π) is loaded in an array format from a hex file to the internal Flash memory . The standard AES-128 in counter mode encrypts the code using the AES hardware accelerator. The encrypted code is then programmed into the external Octo-SPI Flash memory. The user application executes the code straight from the external Octo-SPI memory with on-the-fly decryption supported by the OTFDEC. The result of the PI computation is displayed on a serial terminal via a UART interface.

5.1.2 Pre-requisite

PI computation code is compiled and linked using the external Octo-SPI memory mapping detailed in the table below.

Table 10. External mapping addressing

External memory address	Contents
0x90000000 to 0x90002000	Memory area containing the scatter loading and reset code of the PI computation algorithm. Used data is copied into the device SRAM area.
0x90002000 to 0x90006000	Memory area containing the PI computation algorithm in execute only mode.

This example relies on the OTFDEC and an accelerated AES-128 in counter mode algorithm with its associated crypto HAL driver.

5.1.3 Programming

The code examples use the OCTOSPI peripheral as external Flash memory to store the encrypted binary image. These examples are built in the following sequence:

1. Initialize the device.
2. Initialize the required peripherals (USART and OCTOSPI).
3. Configure the OCTOSPI in DTR mode.
4. Erase the Octo-SPI memory.

5. Encrypt the binary image as follows:
   - As stated in Section 2 , pre and post processing are required to encrypt the data with the AES-128 counter mode algorithm and to decrypted with the OTFDEC.
   - Assign the four following buffers:
     - `OSPI_Buffer`: initial buffer located in the Flash area, containing the binary image in little-endian format.
     - `CODE_Buffer_Swapped`: buffer located in SRAM, containing the binary image in big-endian format to be used as input for the AES-128 counter algorithm.
     - `OSPI_Buffer_Swapped_Enc`: buffer located in SRAM, containing the encrypted binary image in big-endian format as the output from the AES-128 counter algorithm.
     - `OSPI_Buffer_Enc`: SRAM buffer containing the encrypted binary image in little-endian format to be programmed in the Octo-SPI external Flash memory. This image is decrypted on-the-fly by the OTFDEC.
   - Initialize the AES-IV and AES-KEY to encrypt the binary code as stated in Section 2.2 .
   - Pre process `OSPI_Buffer` to be AES-128 in counter mode compatible (big-endian format). The result is stored in `Code_Buffer_Swapped`.
   - Call `HAL_CRYP_Encrypt` function to encrypt `CODE_Buffer_Swapped`. The result is stored in `OSPI_Buffer_Swapped_Enc`.
   - Post process `Ospi_Buffer_Swapped_Enc` to be OTFDEC compatible (little-endian format). The result is stored in `Ospi_Buffer_Enc`.
6. Flash `OSPI_Buffer_Enc` to the OCTOSPI.
7. Configure OTFDEC region as follows:
   - Initialize all the requested parameters to decrypt the binary code as stated in Section 4.1 .
   - Define OTFDEC region from 0x9000 0000 to 0x9000 6000, that contain both PI computation algorithm, initialization code and data.
   - Set the OTFDEC region mode to “All read accesses are decrypted (instruction or data)”, allowing read access of the Octo-SPI external memory.
8. Enable Octo-SPI memory map.
9. Jump to Octo-SPI external memory to execute PI computation algorithm.
10. The serial UART port terminal displays the messages shown in the figure below.

Figure 4. AES-128 in counter mode serial UART port terminal display

```
********** OCTOSPI1 OCTO MODE DTR **********************
CPU FREQUENCY: 60 MHz
OCTOSPI: Memory Mapped Mode - Memory Code Read OK
********** Execute encrypted code with AES-CTR from OCTOSPI1 memory **********
Starting PI Calculation with Cache ON: by CM7...
CM7 benchtime in ms  = 1483
x= 0.38631 y= 0.89070 low= 939239 j=1200001
Pi =  3.130797 ztot=   801773.75 itot= 1200000
```

5.2 Example using proprietary layer of protection above standard AES in counter mode

5.2.1 Overview

In this example, a code calculating PI (π) is loaded to the internal Flash memory from a hex file in an array format. It is encrypted using the OTFDEC RSS services. Once the encrypted code is programmed to the external Octo-SPI Flash memory, the user application jumps to the external Octo-SPI memory to execute the code with the on-the-fly decryption supported by the OTFDEC. The PI computation results are displayed on the serial terminal via a UART interface.

5.2.2 Pre-requisite

The memory mapping is defined when the PI computation code is compiled and linked, as illustrated in the table below.

Table 11. PI computation code memory mapping

External memory address	Contents
0x90000000 to 0x90002000	Contain the scatter loading and reset code of the PI computation algorithm. Used data is copied into device SRAM area.
0x90002000 to 0x90006000	Contain the PI computation algorithm in execution only mode

This example relies on the OTFDEC and the OTFDEC RSS services.

Before launching the example, make sure the security features are disabled.

It is recommended to have boot mode BOOT0 set to 0 and boot address set to the Flash memory area.

To encrypt the binary file, the device calls the proper OTFDEC RSS services that force the device to reboot. Therefore data must be stored in an SRAM area that cannot be overwritten during the reset.

When using the OTFDEC to encrypt, the binary image is always decrypted in execute only mode. Therefore, only code loaded from the address 0x90002000 to 0x90006000 is encrypted and defined as a OTFDEC region to be decrypted.

5.2.3 Programming

The code example uses the OCTOSPI peripheral as external Flash memory to store the encrypted binary image. These examples are built in the following sequence:

1. Initialize the device.
2. Initialize the required peripherals (USART and OCTOSPI).
3. Configure OCTOSPI in DTR mode.

4. If security is disabled:
   - Erase the Octo-SPI memory.
   - Copy the non-encrypted part of the code to the Octo-SPI Flash (reset and initialization from 0x9000 0000 to 0x9000 2000).
   - Enable the security.

   While the entire code is not encrypted:
   - If the encryption must be done:
     - Copy to the dedicated AXI SRAM1 area the data to encrypt.
     - Call OTFDEC RSS service `resetAndEncrypt` to encrypt the code located in the dedicated AXI SRAM1 area with the following argument `otfd_enc_data`:
       - Data information:
         - Dest: 0x9000 2000
         - Buffer: 0x2400 0000
         - Size: 13 KBytes.
       - OTFDEC region information:
         - Number: 0, OTFDEC1 region 1.
         - Mode: to be set to 3, only instruction accesses are decrypted, and enhanced encryption mode is activated (this is the purpose of this section).
         - Version: not used, set to 0.
         - Nonce: address of the 64 bits nonce, NonceIn[2] = {0x11223344, 0xAABBCCDD}.
         - Keys: address of the 128 bits keys used for the above defined region, OtfdecKey128 = {0x12345678, 0x9ABDCF0, 0x12345678, 0x9ABDCF0}.
     - A device reboot is generated and repeat the sequence from step 1.
   - Else, write encrypted code located in the dedicated AXI SRAM1 area to the Octo-SPI external memory.
5. All the code is now encrypted and programmed in the Octo-SPI memory. Disable the security mode.
6. Configure the OTFDEC region with the same parameters used for the encryption above:

   Initialize all the requested parameters to decrypt the binary code as stated in Section 4.2 .
   - Instantiate OTFDEC1 region1
   - Define OTFDEC region from 0x90020000 to 0x90006000 containing PI computation algorithm in Execute mode only.
   - Set Region mode to “Only instruction accesses are decrypted, and enhanced encryption mode is activated” (MODE bitfield to 11 in `OTFDEC_RxCFGR` register). In this mode, reading the encrypted external memory returns zero.
   - Version number set to 0.
   - 128 bits key as used for the encryption part: OtfdecKey128 = {0x12345678, 0x9ABDCF0, 0x12345678, 0x9ABDCF0}.
   - 64 bits nonce as used for the encryption part: NonceIn[2] = {0x11223344, 0xAABBCCDD}.
7. Enable the Octo-SPI memory map.
8. Jump to the Octo-SPI external memory to execute the PI computation algorithm.
9. The serial UART port terminal displays the messages shown in the figure below.

Figure 5. AES in counter mode serial UART port terminal display

```
********** OCTOSPI1 OCTO MODE DTR **********************
CPU FREQUENCY: 60 MHz
********** Execute encrypted code with proprietary enhanced AES-CTR from OCTOSPI1 memory **********
Starting PI Calculation with Cache ON: by CM7...
CM7 benchtime in ms = 1483
x= 0.38631 y= 0.89070 low= 939239 j=1200001
Pi =  3.130797 ztot=   801773.75 itot= 1200000
```

6 Conclusion

OTFDEC main purpose is to decrypt in real-time encrypted data or code stored in the Octo-SPI external memory. It can also be used as part of the OTFDEC RSS services, for code encryption based on a proprietary layer over the standard AES-128 counter mode.

In case of encryption, there are two possibilities:

- Use a standard AES-128 counter mode part of the AES hardware accelerator to encrypt data or code and use OTFDEC to decrypt on-the-fly; the data, or code programmed in an external Octo-SPI memory. Pre and post processing must be applied because standard AES-128 in counter mode uses the big-endian memory order whereas for encryption, OTFDEC uses the little-endian memory order.
- Use the OTFDEC RSS services to encrypt binary image only based on a proprietary layer over the standard AES-128 counter mode. The benefit of this OTFDEC encryption is to add robustness to the standard AES in counter mode, but may require several system resets to encrypt the entire binary image depending of the size of the binary image and the selected SRAM area used to encrypt the binary image

The OTFDEC allows real time decryption with negligible latency.

Revision history

Table 12. **Document revision history**

Date	Version	Changes
04-Mar-2019	1	Initial release.
07-Aug-2019	2	Removed: resetAndGenerateKeys table. Updated: • Table 9. resetAndEncrypt • Section 3.2 How to encrypt with OTFDEC and write code to external Octo-SPI memory • Figure 3. Binary image encryption and storage in external Octo-SPI memory • Section 3.2 How to encrypt with OTFDEC and write code to external Octo-SPI memory • Section 3.3 Constraints • Section 4.2 On-the-fly decryption of encrypted code with the proprietary layer above the standard AES-128 in counter mode Section 5.2.2 Pre-requisite Section 5.2.3 Programming
03-Feb-2020	3	Changed the document status from "ST Restricted" to "Public"
06-May-2020	4	Updated: • Section • Section 1 General information
10-Sep-2020	5	Updated the document title, Introduction, and Section 1 General information to support STM32H73xx and STM32H7Bxx microcontrollers.

Contents

List of figures

List of tables