AN13175

AES Encryption/Decryption Using RT5xx

Rev. 0 — 03/2021 Application Note

Contents

1 Introduction

The RT5xx HASH-Crypt IP contains the HASH engine and AES engine. HASH and AES share the same HASH-Crypt register set and cannot operate concurrently. The AES engine, compliant with the AES standard, process 128 bits at a time, and supports 128-bit, 192-bit, and 256-bit keys. The key can either be supplied directly loading from software or be a secret key from OTP or from the PUF (Physically Unclonable Function) IP. The AES engine supports different modes and is defined in both US (NIST FIPS PUB 197) and international standards (ISO/IEC 18033-3).

The AES engine is used extensively by the RT5xx boot ROM. If the image is encrypted for recovery boot, or for setup of the OTFAD allowed XIP images to be encrypted in flash, decrypted on the fly during execution.

This application note focuses on use of the AES engine for runtime encryption and decryption. Encryption can be used for data saved to a non-volatile memory decrypted as needed by the application or for encryption/decryption of data created/modified/stored in RAM during runtime. Two devices could communicate with encrypted messages and/or encrypted data, but the key has to be agreed upon between devices.

The RT5xx SDK includes an HASH-Crypt API that simplifies configuration of the modes supported by the AES engine, key management, and encryption/decryption of data.

This application note demonstrates how to configure the AES engine and write code using the SDK API to encrypt/decrypt data using keys either manually supplied or generated by the PUF IP. Configuration and use of PUF IP is described.

2 Basic features

- AES key, IV, or counter registers cannot be read once loaded.
- AES engine peak performance of 0.5 bytes/clock cycle.
- AES engine supports 128-bit, 192-bit, or 256-bit key in:
  - — Electronic Code Book (ECB) mode.
  - — Cipher Block Chaining (CBC) mode.
  - — Counter (CTR) mode.
- The AES engine supports 128-bit key in ICB (Indexed Code Book) mode, that offers protection against side-channel attacks.
- AES offers programmability to select little-endian or big-endian mode of operation.
- It may use the processor, DMA, or AHB Master for data movement. AHB Master may only be used to load data, DMA may be used to read-out results. DMA-based result reading is a “trigger”, so the application must set the size correctly.

3 Block diagram

Figure 1 illustrates the data flow and required inputs to the HASH-CRYPT engine allowing it to perform AES encrypt and decrypt operations. In all modes, a key is required. Depending on mode, the IV (CBC mode) or COUNTER (CTR mode) inputs may be required. Key source (USER, OTP, PUF) is determined by memory mapped register (SYSCON and Hash-AES registers) settings. Per the AES standard, the AES engine processes 128-bits at a time. Input data can be streamed manually, in software, using the HASH-CRYPT built-in AHB bus master, or using DMA. Output data can be read out manually (upon interrupt or via polling) or using DMA.

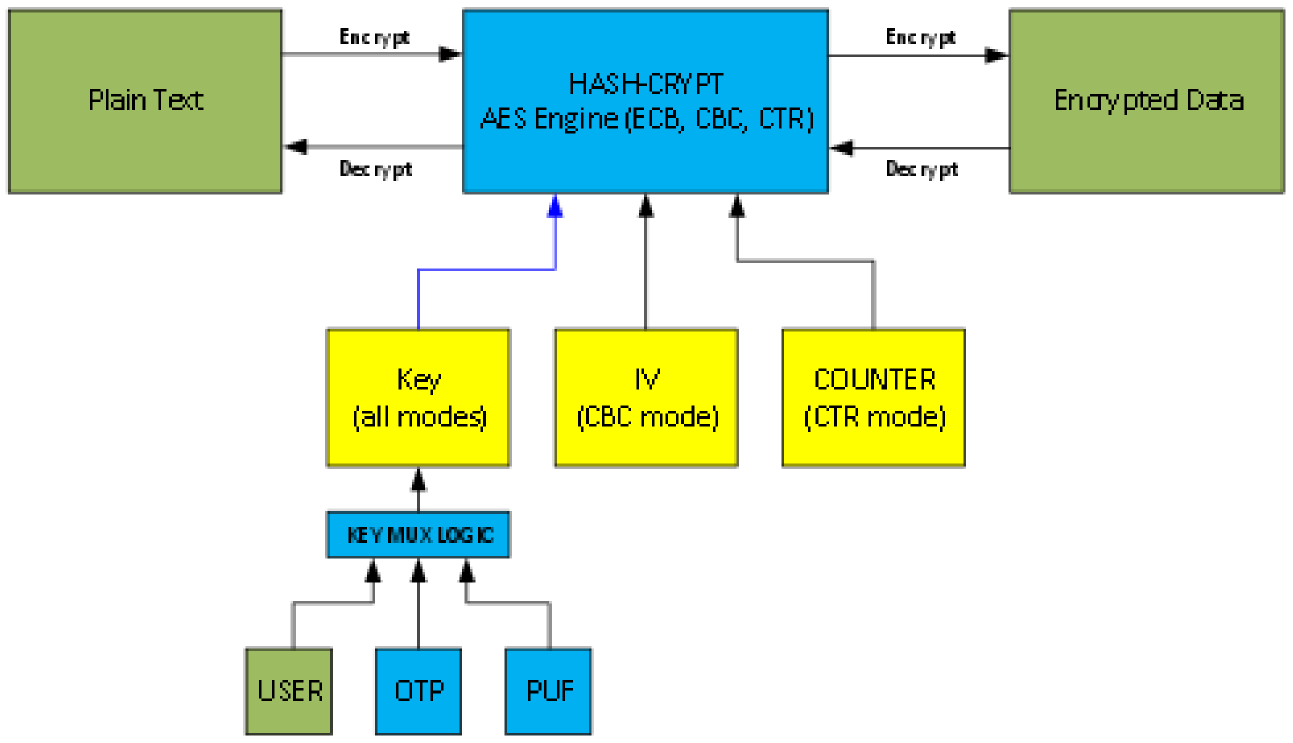


Figure 1. HASH-CRYPT AES engine Data Flow

4 Tools

4.1 EVK board

Assuming the development platform is NXP i.MX RT500 Evaluation Kit (part # MIMXRT595-EVK). The board contains an NXP MIMXRT595SFVKB device which contains a 200 MHz Arm® Cortex®-M33 CPU and a 200 MHz Cadence® Tensilica® Fusion F1. See MIMXRT595-EVK: i.MX RT500 Evaluation Kit.

The board contains onboard debug and virtual COM port through a single Micro USB interface, onboard flash memory and SD card socket for power-on boot, stereo audio input/output, and multiple expansion ports for prototyping.

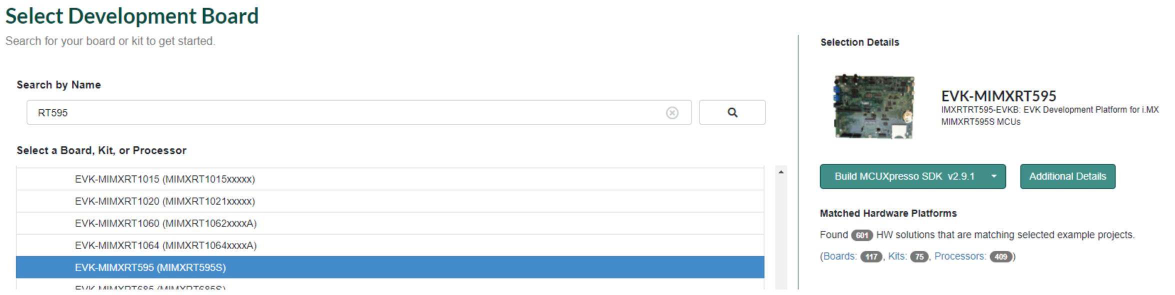
4.2 SDK

The NXP RT595 SDK includes peripheral libraries, configuration tools, documentation, and application examples using the SDK drivers. The SDK is downloadable from the NXP website for free and supports three IDEs including NXP MCUXpresso, Keil uVision, and IAR Embedded Workspace.

This application note contains three examples, each demonstrating one the three IDEs (MCUXpresso, Keil, IAR). These examples use the SDK API extensively including for the AES engine. Each example has a workspace for each of three IDEs (MCUXpresso, Keil, IAR). The examples are similar to those shipped with SDK install. The examples are provided in a zip file to be unzipped into the SDK examples, once installed.

4.3 SDK Download

1. If not registered, register on NXP website (https://www.nxp.com).
2. Go to the SDK Builder site https://mcuxpresso.nxp.com/en/welcome.
   a. Click Select Development Board.
   b. In Search by Name text box " enter "RT595".
   c. In Name your SDK text box, use the name assigned to you or choose a custom name for the SDK install zip, therefore folder name when unzipped.
   d. In search results, under Boards, click "EVK-MIMXRT595 (MIMXRT595S)".
   e. On the right of the page, under Actions, click "Build MCUXpresso SDK".



Select Development Board

Search for your board or kit to get started.

Search by Name

RT595

Select a Board, Kit, or Processor

EVK-MIMXRT1015 (MIMXRT1015xxxxx)
EVK-MIMXRT1020 (MIMXRT1021xxxxx)
EVK-MIMXRT1060 (MIMXRT1062xxxxA)
EVK-MIMXRT1064 (MIMXRT1064xxxxA)
EVK-MIMXRT595 (MIMXRT595S)

Selection Details

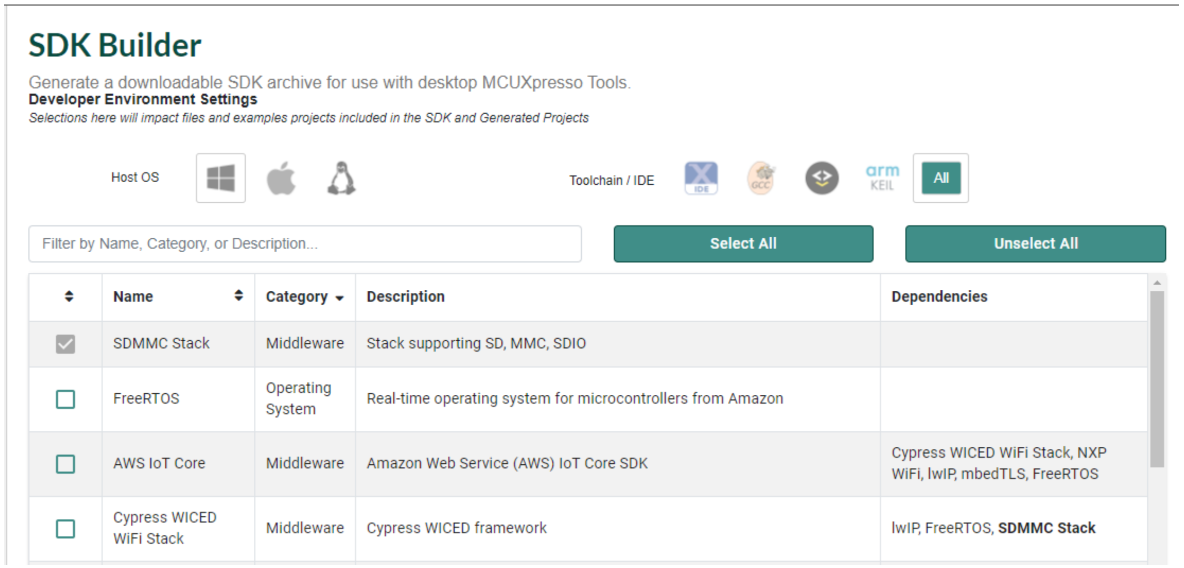
EVK-MIMXRT595
IMXRTRT595-EVKB: EVK Development Platform for i.MX MIMXRT595S MCUs

Matched Hardware Platforms

Found 601 HW solutions that are matching selected example projects.

(Boards: 117, Kits: 75, Processors: 409)

   f. The following page allows configuration of the install including selection of SDK Version, Toolchain/IDE, Host OS, and Components.



SDK Builder

Generate a downloadable SDK archive for use with desktop MCUXpresso Tools.

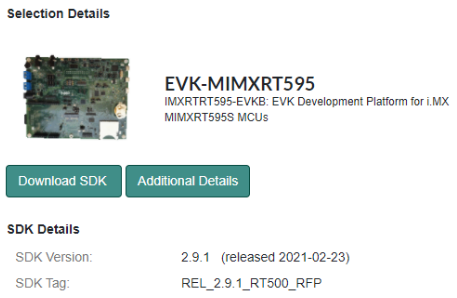
Developer Environment Settings

Selections here will impact files and examples projects included in the SDK and Generated Projects

	Name	Category	Description	Dependencies
✓	SDMMC Stack	Middleware	Stack supporting SD, MMC, SDIO	
☐	FreeRTOS	Operating System	Real-time operating system for microcontrollers from Amazon	
☐	AWS IoT Core	Middleware	Amazon Web Service (AWS) IoT Core SDK	Cypress WICED WiFi Stack, NXP WiFi, lwIP, mbedTLS, FreeRTOS
☐	Cypress WICED WiFi Stack	Middleware	Cypress WICED framework	lwIP, FreeRTOS, **SDMMC Stack**

   g. Under the Components, check the following unchecked items: mbedtls.
   h. Other useful unchecked components include mcu-boot, FAT-FS, and USB stack.

i. A second chance to edit the zip file name inside text box Archive Name.



Selection Details

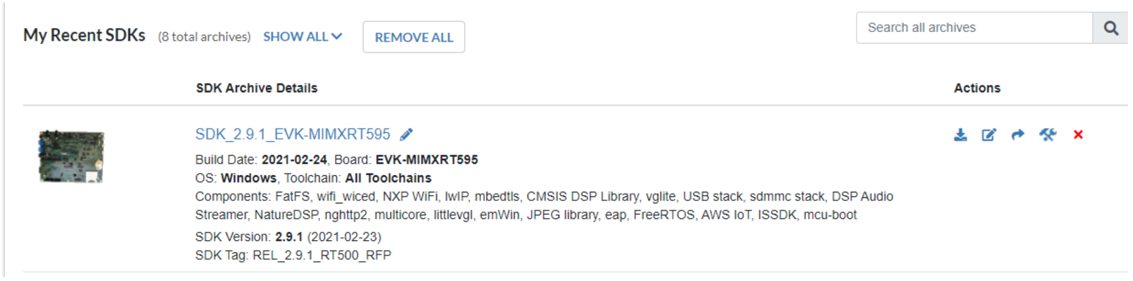
EVK-MIMXRT595
IMXRTRT595-EVKB: EVK Development Platform for i.MX MIMXRT595S MCUs

SDK Details

SDK Version:	2.9.1 (released 2021-02-23)
SDK Tag:	REL_2.9.1_RT500_RFP

j. Finally, click Download SDK.

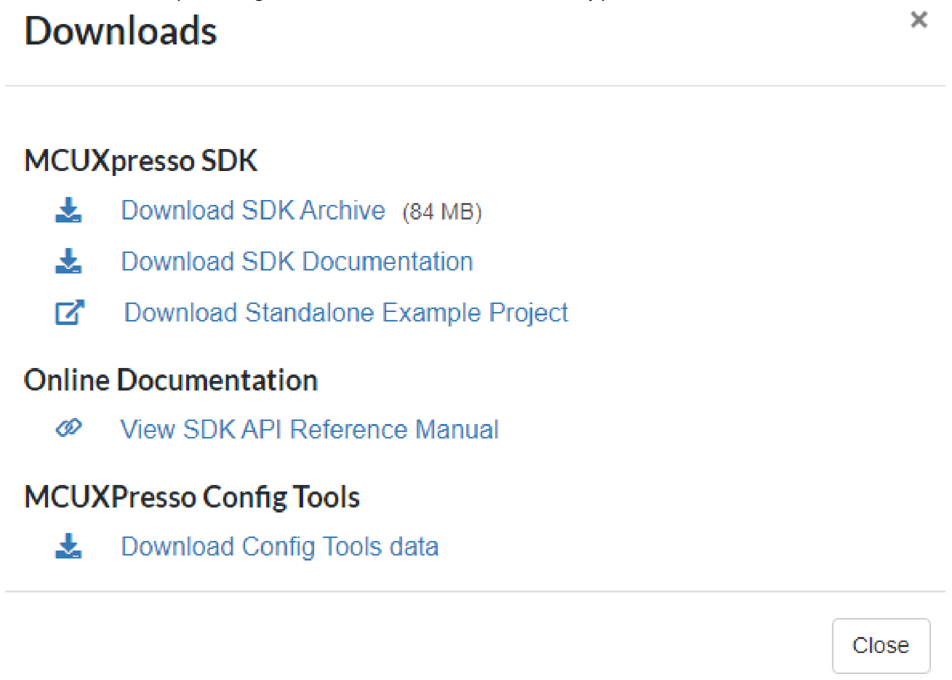
k. The site if building the archive. Once completed, an email is sent with the download path for the archive, but otherwise, the following webpage is presented.



SDK_2.9.1_EVK-MIMXRT595
Build Date: **2021-02-24**, Board: **EVK-MIMXRT595**
OS: **Windows**, Toolchain: **All Toolchains**
Components: FatFS, wifi_wiced, NXP WiFi, lwIP, mbedtls, CMSIS DSP Library, vglite, USB stack, sdmmc stack, DSP Audio Streamer, NatureDSP, nghttp2, multicore, littlevgl, emWin, JPEG library, eap, FreeRTOS, AWS IoT, ISSDK, mcu-boot
SDK Version: **2.9.1** (2021-02-23)
SDK Tag: REL_2.9.1_RT500_RFP

l. Click the download icon (red arrow) and then click Download SDK Archive. Best to save the archive to a local drive unless confident a network drive does not cause issue.

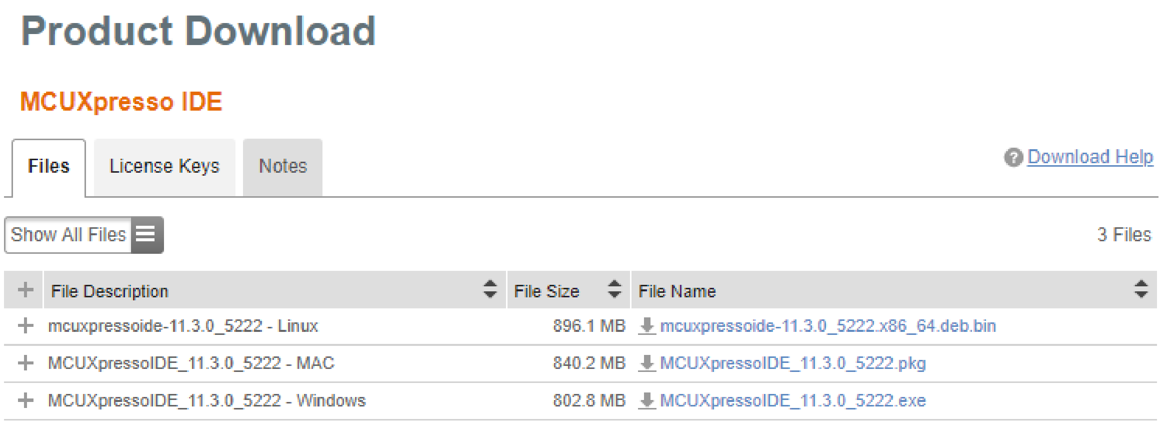
m. Find the zip file in the file manager application, copy into the projects folder, and unzip the file into folder (creating folder of the same name as zip).



Downloads
MCUXpresso SDK
Download SDK Archive (84 MB)
Download SDK Documentation
Download Standalone Example Project
Online Documentation
View SDK API Reference Manual
MCUXPresso Config Tools
Download Config Tools data
Close

4.4 Install MCUXpresso IDE (not using Keil/IAR)

It is required that MCUXpresso IDE 11.3.0 or newer is installed with RT595 SDK v2.9.1 and with the example code of this application note.



	File Description	File Size	File Name
+	mcuxpressoide-11.3.0_5222 - Linux	896.1 MB	mcuxpressoide-11.3.0_5222.x86_64.deb.bin
+	MCUXpressoIDE_11.3.0_5222 - MAC	840.2 MB	MCUXpressoIDE_11.3.0_5222.pkg
+	MCUXpressoIDE_11.3.0_5222 - Windows	802.8 MB	MCUXpressoIDE_11.3.0_5222.exe

Import the SDK_2.9.1_EVK-MIMXRT595 folder (not zip) into the MCUXpresso workspace.

4.5 Add AES example code to SDK

1. Select the SDK folder (that is, SDK_2.9.1_EVK-MIMXRT595) in the file manager application.
2. Go to subdirectory SDK_2.9.1_EVK-MIMXRT595\boards\evkmimxrt595.
3. Download the code supplied with this application note contained in zip file RT5xx_aes_appnote_examples.zip. Copy to and unzip in folder SDK_2.9.1_EVK-MIMXRT595\boards\evkmimxrt595 (extract here since all folders are contained in zip).

5 C Programming with AES engine

5.1 Low-Level initialization

The AES engine (Hash-AES) like most other RT5xx IP blocks has a reset and clock control in the SYSCON registers. Also key selection (OTP versus PUF) control in SYSCON when a secret (hidden) key is selected. Most configuration is handled via the API calls, but this section will outline configuration of the IP without API calls.

1. HASHCRYPT structure referenced in the following header file.

```
#include"fsl_de vice_registers.h"
```

2. AES engine clock is connected and the IP is reset.

```
/* reset the HASHCrypt Engine - done in API function HASHCRYPT_Init() */
CLOCK_EnableClock(kCLOCK_HashCrypt);
RESET_PeripheralReset(kHASHCRYPT_RST_SHIFT_RSTn);
```

3. Initialize the AES engine for AES mode (ECB, CBC, CTR), key source (user versus secret), key size (128, 192, 256), direction (encrypt/decrypt).

```
/* configure AES engine */
/* parameter: configWord */
HASHCRYPT->CRYPTCFG = configWord;
```

```
/* select PUF secret key, OTP = 0x2 */
SYSCTL0->AESKEY_SRCSEL = 0x0;

/* start new AES encrypt/decrypt – start state machine */
HASHCRYPT->CTRL = 0x10;
HASHCRYPT->CTRL = 0x14;
```

4. Load key (user or secret)

   After AES engine is started, a key is loaded, either manually if the user (software) key is selected in CRYPTCFG or automatically from PUF index 0 or OTP key source if the secret key is selected.

```
/* key check – if user key loaded, as above, continue */
/*    – if secret key loaded from PUF or OTP, wait */
/* parameters: keySource, key[] */ if (keySource = kSecret)    {

volatile uint32_t wait = 50u;
/* wait until STATUS register is non-zero */ while ((wait > 0U) && (HASHCRYPT->STATUS == 0U))
{
wait--;
}
/* if NEEDKEY bit is not set, HW key is available */ if (0U != (HASHCRYPT->STATUS & 0x10)) {
return FAIL;
}
}
/* else load user key */ else {
if (0U != (HASHCRYPT->STATUS & 0x10)) {
/* load the key */ HASHCRYPT->INDATA = key[0];
for (int i = 1; i < (keySizeInBytes>>2); i++)    { HASHCRYPT->ALIAS[i-1] = key[i];
}
}
else {
return FAIL;
}
}
```

5. Load IV (CBC mode) or CTR (CTR mode)

```
/* if CBC or CTR mode need to load 128-bit IV or CTR respectively */
/* parameter: iv_ctr[] */
if ((HASHCRYPT->CRYPTCFG & 0x30) != 0)    { if (0U != (HASHCRYPT->STATUS & 0x20)) {
/* load the IV or CTR */
HASHCRYPT->INDATA = iv_ctr[0];
HASHCRYPT->ALIAS[0] = iv_ctr[1];
HASHCRYPT->ALIAS[1] = iv_ctr[2];
HASHCRYPT->ALIAS[2] = iv_ctr[3];
}
else {
return FAIL;
}
}
```

6. Encrypt or decrypt – max 256 bytes per pass through outer while loop

   The code below can be optimized to use DMA, but it illustrates the data flow through the AES engine.

```
/* parameters input[], output[], size */
uint32_t temp[256 / sizeof(uint32_t)];
int cnt = 0;
while (size != 0U)
```

```
{
size_t actSz    = size >= 256u ? 256u : size;
size_t actSzOrig = actSz;
void)memcpy(temp, (const uint32_t *)(uintptr_t)(input + 256 * cnt), actSz);
size -= actSz;
HASHCRYPT->MEMADDR = HASHCRYPT_MEMADDR_BASE(temp);
HASHCRYPT->MEMCTRL = HASHCRYPT_MEMCTRL_MASTER(1) | HASHCRYPT_MEMCTRL_COUNT(actSz>>4);
uint32_t outidx = 0; while (actSz != 0U)
{
while (0U == (HASHCRYPT->STATUS & HASHCRYPT_STATUS_DIGEST_MASK))
{
}
for (int i = 0; i < 4; i++)
{
(temp + outidx)[i] = swap_bytes(HASHCRYPT->DIGEST0[i]);
}
outidx += HASHCRYPT_AES_BLOCK_SIZE / 4U; actSz -= HASHCRYPT_AES_BLOCK_SIZE;
}
(void)memcpy(output + 256 * cnt, (const uint8_t *)(uintptr_t)temp, actSzOrig); cnt++;
}
```

5.2 SDK API

The previous section was provided to illustrate the functionality and data flow through the AES engine. Typically you use the SDK HASHCRYPT API to use the AES engine. The API consists of a handful of function calls which contain the code of the previous section. Since the HASHCRYPT IP contains both the AES and SHA Engines, they also share the same API from header file fsl_hashcrypt.h.

Application code must include the following header files:

```
#include "fsl_device_registers.h"
#include “fsl_hashcrypt.h”
```

HASHCRYPT parameter structure – defined in application code, passed to API functions:

```
/*! @brief Specify HASHCRYPT's key resource. */
typedef struct
{
uint32_t keyWord[8];
hashcrypt_aes_keysize_t keySize;
hashcrypt_key_t keyType;
} attribute ((aligned)) hashcrypt_handle_t;
```

Initialization API functions – connect and disconnect clocks, reset IP:

```
void HASHCRYPT_Init(HASHCRYPT_Type *base);
void HASHCRYPT_Deinit(HASHCRYPT_Type *base);
```

Set user (software key):

```
HASHCRYPT_AES_SetKey
status_t HASHCRYPT_AES_SetKey(HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *key,
size_t keySize);
```

ECB mode functions:

```
status_t HASHCRYPT_AES_EncryptEcb(HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t
*plaintext, uint8_t *ciphertext, size_t size);
```

```
status_t HASHCRYPT_AES_DecryptEcb(HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t
*ciphertext, uint8_t *plaintext, size_t size);
```

CTR mode function (similar to ECB but also pass pointer to 128-bit counter) – last two parameters set to NULL:

```
status_t HASHCRYPT_AES_CryptCtr(HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t
*input, uint8_t *output, size_t size, uint8_t counter[HASHCRYPT_AES_BLOCK_SIZE], uint8_t
counterlast[HASHCRYPT_AES_BLOCK_SIZE], size_t *szLeft);
```

6 Examples

Each of the following sections overviews the AES demo applications, each demonstrating a different key type: user (software) key, OTP key, and PUF key. Each is demonstrated with a different IDE as well (Keil, IAR, MCUXpresso).

6.1 Example Software Data Flow Diagram

Figure 2 is the software flow diagram using the PUF example, since it is slightly more complex with the PUF initialization. The OTP and Software Key examples follow the same flow without PUF initialization.

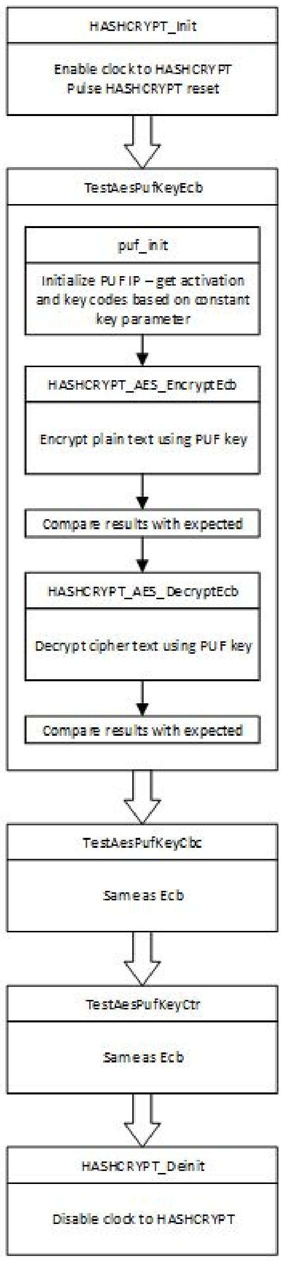

HASHCRYPT_Init
Enable clock to HASHCRYPT
Pulse HASHCRYPT reset
TestAesPufKeyEcb
puf_init
Initialize PUF IP – get activation and key codes based on constant key parameter
HASHCRYPT_AES_EncryptEcb
Encrypt plain text using PUF key
Compare results with expected
HASHCRYPT_AES_DecryptEcb
Decrypt cipher text using PUF key
Compare results with expected
TestAesPufKeyCbc
Same as Ecb
TestAesPufKeyCtr
Same as Ecb
HASHCRYPT_Deinit
Disable clock to HASHCRYPT

Figure 2. Software flow diagram using the PUF example

6.2 Example using Software Key – with Keil uVision

The simplest way to use the AES engine is to load a software key, either downloaded from a host, saved to external flash memory, or like in this case, defined in code for demonstration purposes.

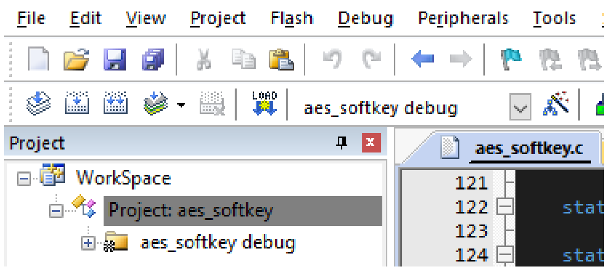
This example, like the others in this application note, can be compiled and debugged using MCUXpresso, Keil uVision or IAR Embedded Workbench. The Keil uVision IDE will be used for this example.

1. Install the Keil uVision IDE – using version 5.33
2. Download and unzip the SDK as described in the above section “SDK Download”.

3. Download and unzip the example code for this application note as described in the above section “Add AES Example Code to SDK”
4. Navigate through the SDK to the following folder:

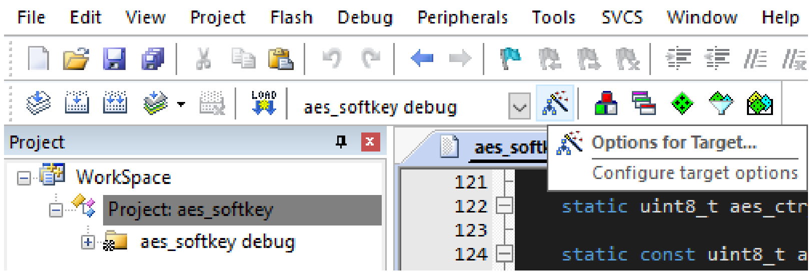
```
SDK_2.9.1_EVK-MIMXRT595\boards\evkmimxrt595\rt5xx_aes_appnote_examples\software_key\mdk
```

5. Double-click on workspace file: aes_softkey.uvmpw
6. Select project target: aes_softkey debug

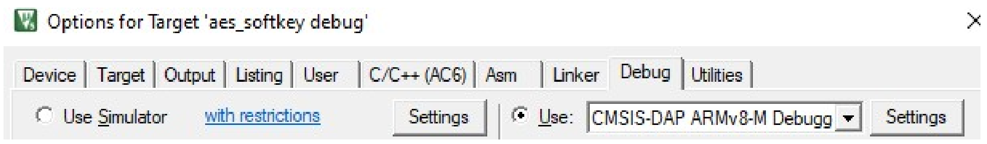


   a. This is debug mode, image executed out of RAM
   b. Other options allow release build and/or image executed out of external flash
      i. aes_softkey_flash_debug
      ii. aes_softkey_release
      iii. aes_softkey_flash_release

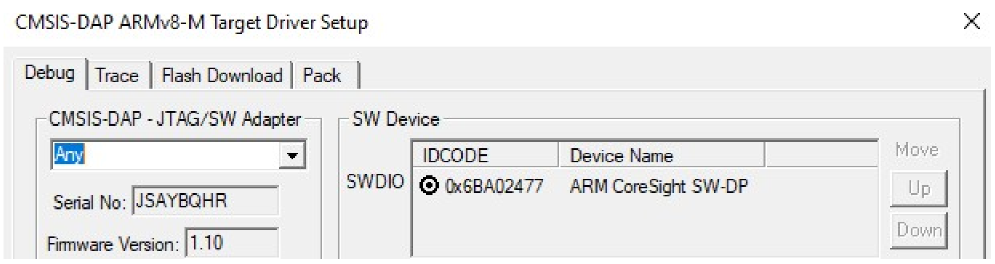
7. Rebuild the project: Project->Rebuild
8. Connect EVK board
   a. Board must be RT595 EVK .
   b. Need LPC Link2 Driver – if not already installed, install the following driver from the NXP site. https://www.nxp.com/downloads/en/software/lpc_driver_setup.exe
   c. Connect free USB port to (microUSB) to connector J40. The port serves as a virtual COM for console I/O and as a debugger.
   d. Verify the board is available as a debugger:
      i. Click “Options for Target” button or Project->Options from menu.



      ii. Click the “Debug” tab, verify “CMSIS-DAP ARMv8-M Debugger” is selected, and click “Settings”.



      iii. The resulting dialog box should show the board is connected as shown below.


CMSIS-DAP ARMv8-M Target Driver Setup

CMSIS-DAP - JTAG/SW Adapter: Any

Serial No: JSAYBQHR

Firmware Version: 1.10

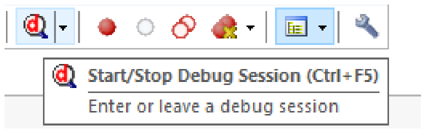
SW Device

	IDCODE	Device Name
SWDIO	0x6BA02477	ARM CoreSight SW-DP

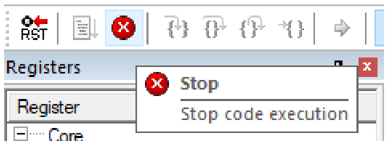
   iv. If not showing, follow the instructions, refer to the RT595 EVK Getting Started section on the NXP site. https://www.nxp.com/document/guide/getting-started-with-the-mimxrt595-evk:GS-MIMXRT595-EVK

   v. Else, click OK to the two open dialog boxes to close.

9. Next install an RS-232 Terminal Emulation program such as TeraTerm (Windows).
10. Configure the terminal emulation for the EVK.
    a. Select the COM port assigned by the OS such as “COM40: LPC-LinkII UCom port”.
    b. For terminal settings under New-line for RX and TX select “CR+LF”.
    c. Set baud rate and port settings to 115200-8-N-1, no flow control required.
11. **Code is built, EVK set up, connected, ready to debug.**
    a. Click the Start/Stop Debug Session button or select Debug->Start/Stop Debug Session from menu.


Start/Stop Debug Session (Ctrl+F5)
Enter or leave a debug session

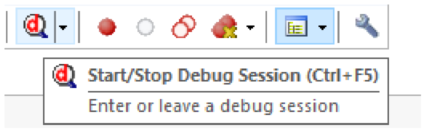
    b. Code will download and debugger will point to function “main” in project source file aes_softkey.c.
    c. The code has similar initialization to the other demos in the SDK, specifically ported from the “hello_world” demo. Clocks, pins, and the RS-232 port are all configured before the AES specific code is executed.
    d. To single-step over functions in Keil, type F10. To step into a function, type F11. To run, type F5, to stop click the Stop button. To the left of the Stop button is the Run (continue) button, greyed out below.



    e. Breakpoints are supported, but are not discussed in this application note.
    f. To verify the code works, simply type F5 to run to click the Run button.
    g. The console window in the terminal program should display the following:

```
AES ECB, CBC, CTR testing using key loaded via software
AES ECB Test - 128-bit key loaded via software - pass
AES CBC Test - 128-bit key loaded via software - pass
AES CTR Test - 128-bit key loaded via software - pass
```

    h. If not, refer to the RT595 EVK Getting Started section on the NXP site referred to in section 8.d.iv.
    i. To leave the debugger, click the “Start/Stop Debug Session” button or select menu Debug->Start/Stop Debug Session.


Start/Stop Debug Session (Ctrl+F5)
Enter or leave a debug session

12. **Run the code again, this time to step through code.**
    a. Follow steps a-e in step 11 above.
    b. Code is at "main" in project source file aes_softkey.c.
    c. The AES specific code is:

```
/* Initialize Hashcrypt */ HASHCRYPT_Init(HASHCRYPT);

/* test description */
PRINTF("\r\nAES ECB, CBC, CTR testing using key loaded via software\r\n\r\n");

/* Call HASH APIs */ TestAesSoftKeyEcb(); TestAesSoftKeyCbc(); TestAesSoftKeyCtr();

HASHCRYPT_Deinit(HASHCRYPT);
```

    d. The functions with "HASHCRYPT" in the name are SDK API calls and are described above in "SDK API" section.
    e. The functions with "TestAesSoftKey" in the name each test a different AES mode (ECB, CBC, CTR) supported by the AES engine and are found in project source file aes_softkey.c.
    f. Function "TestAesSoftKeyEcb" tests ECB mode using a software key.
        i. Press F10 until the cursor (yellow Triangle) is pointed to "TestAesSoftKeyEcb".
        ii. Press F11 to step into the function.
        iii. The function has three constant arrays defined, each 128 bits or 16 bytes.
            i. keyAes128 – softwarekey
            ii. plainAes128 – plaintext – unencrypted data
            iii. cipherAes128 – cipher – encrypted data
        iv. Sinceinitialization of the AES engine was done in the main code, the first operation is to load the "hashcrypt_handle_t" handle structure with key type and call the SDK API function "HASHCRYPT_AES_SetKey"to load the key into the handle structure.
        v. Call to SDK API function "HASHCRYPT_AES_EncryptEcb" passes the handle, plaintext, returns encrypted results into a cipherarray.
        vi. Thereturned cipher data is compared to the expected, defined in the constant array, and if the compare fails,the program aborts with a message that ECB failed.
        vii. If results match expected, a call to SDK API function "HASHCRYPT_AES_DecryptEcb" passes the handle,cipher, returns plaintext results into a plaintext array.
        viii.Thereturned plaintext data is compared to the expected, defined in the constant array, and if the compare fails,the program aborts with a message that ECB failed.
        ix. Ifresults match expected, a message is displayed that ECB passed.
        x. Functioncalls to similar CBC and CTR tests are made from "main". All in this example are 128-bit softwarekeys, 128-bit data arrays, and 128-bit IV or CTR arrays for CBC and CTR modes respectively.
        xi. Ifresults match expected, a messages are displayed that CBC and CTR passed.

6.3 Example using OTP Key – with IAR Embedded Workbench

The AES engine can use a secret (hidden) key either from OTP memory or from the PUF IP. A SYSCON register selects whether the secret key is sourced by OTP or by PUF. The AES engine CRYPTCFG register selects whether a software key or secret key is used.

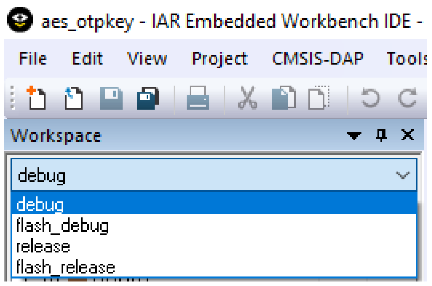
The OTP key is exposed unless OTP settings prevent reading that specific OTP register bank (read lock loaded by boot ROM). Also, OTP has a bank of shadow registers (RAM) to allow testing of the OTP feature before permanently programming OTP cells from 0 to 1. The steps below explain how to program the shadow registers. Programming the actual OTP bits (fuses) is beyond the scope of this application note.

This example, like the others in this application note, can be compiled and debugged using MCUXpresso, Keil uVision or IAR Embedded Workbench. The IAR Embedded Workbench IDE is used for this example.

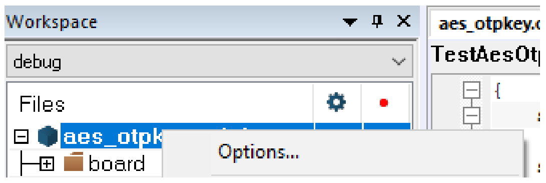
1. Install IAR Embedded Workbench IDE – using version 8.50.9
2. Download and unzip the SDK as described in the above section “SDK Download”.
3. Download and unzip the example code for this application note as described in the above section “Add AES Example Code to SDK”
4. Navigate through the SDK to the following folder:

   ```
   SDK_2.9.1_EVK-MIMXRT595\boards\evkmimxrt595\rt5xx_aes_appnote_examples\otp_key\iar
   ```

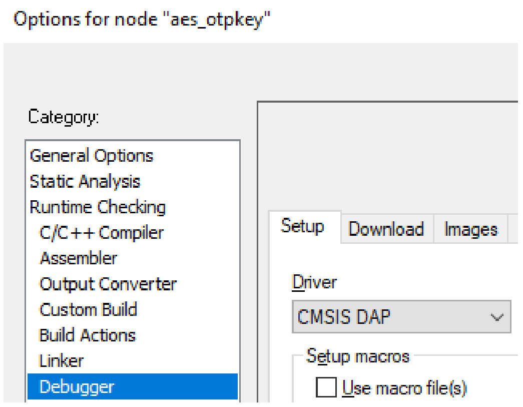
5. Double-click workspace file: aes_otpkey.eww
6. One the workspace is loaded, select the Debug target clicking the pulldown menu.


aes_otpkey - IAR Embedded Workbench IDE -
File Edit View Project CMSIS-DAP Tools
Workspace
debug
debug
flash_debug
release
flash_release

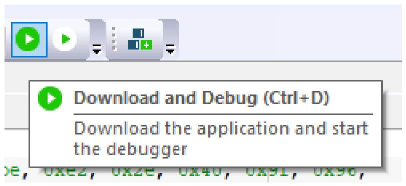
7. To rebuild the project, click Project->Rebuild All.
8. Configure the EVK and RS-232 terminal application. From section 6.2, “Example using Software Key”, verify steps 9 and 10.
9. IAR does not have a way to verify a hardware interface as Keil does, but verify that the CMSIS-DAP debugger is selected.
   a. Right-click project options in the Workspace pane or select menu Project->Options



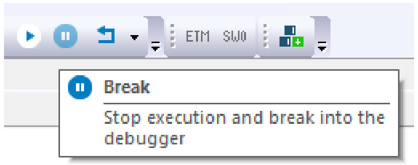
   b. Select Debugger and verify that CMSIS-DAP is selected. Click OK.



10. Code is built, EVK set up, connected, ready to debug.
    a. Click the Download Debug Session button or select Project->Download and Debug from menu.


Download and Debug (Ctrl+D)
Download the application and start the debugger

    b. Code downloads and debugger points to function “main” in project source file aes_otpkey.c.
    c. The code has similar initialization to the other demos in the SDK, ported from the “hello_world” demo. Clocks, pins, and the RS-232 port are all configured before the AES-specific code is executed.
    d. To single-step over functions in IAR, type F10. To step into a function, type F11. To run, type F5, to stop click the Break button. To the left of the Break button is the Go (Run) button.

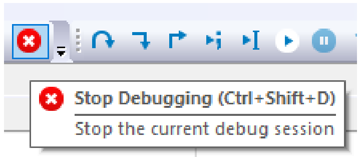


    e. Breakpoints are supported, but are not discussed in this application note.
    f. To verify the code works, simply type F5 to run to click the Break button.
    g. The console window in the terminal program should display the following:

```
AES ECB, CBC, CTR testing using OTP key (via OTP shadow registers)
If OTP shadow register 107 (KEY_SCRAMBLE_SEED) is 0,
OTP shadow registers 119-112 (OTP_MASTER_KEY) function as a test key for the AES engine

AES ECB Test - 192-bit OTP key - pass AES CBC Test - 192-bit OTP key - pass
AES CTR Test - 256-bit OTP key - pass
```

    h. If not, refer to the RT595 EVK Getting Started section on the NXP site: https://www.nxp.com/document/guide/getting-started-with-the-mimxrt595-evk:GS-MIMXRT595-EVK
    i. To leave the debugger, click the Red “X” button or select menu Debug->Stop Debugging.



11. Run the code again, this time to step through code.
    a. Follow steps a-c in step 10 above.
    b. Code is at “main” in project source file aes_otpkey.c.
    c. The AES-specific code is:

    ```
    /* test description */
    PRINTF("\r\nAES ECB, CBC, CTR testing using OTP key (via OTP shadow registers)\r\n");
    PRINTF("If OTP shadow register 107 (KEY_SCRAMBLE_SEED) is 0,\r\n");
    PRINTF("OTP shadow registers 119-112 (OTP_MASTER_KEY) function\r\n");
    PRINTF("as a test key for the AES engine\r\n\r\n");

    /* Initialize Hashcrypt */ HASHCRYPT_Init(HASHCRYPT);

    /* Call HASH APIs */ TestAesOtpKeyEcb(); TestAesOtpKeyCbc(); TestAesOtpKeyCtr();

    HASHCRYPT_Deinit(HASHCRYPT);
    ```

    d. The functions with “HASHCRYPT” in the name are SDK API calls and are described above in “SDK API” section.
    e. The functions with “TestAesOtpKey” in the name each test a different AES mode (ECB, CBC, CTR) supported by the AES engine and are found in project source file aes_otpkey.c.
    f. Function “TestAesOtpKeyCbc” tests CBC mode using an OTP key.
        i. Press F10 until the cursor (green arrow) is pointed to “TestAesOtpKeyCbc”.
        ii. Press F11 to step into the function.
        iii. The function has three constant arrays defined, each 128 bits, or 16 bytes.

            i. plainAes128 – plaintext – unencrypted data

            ii. cipherAes128 – cipher – encrypted data

            iii. ive – IV - initial vector
        iv. Sinceinitialization of the AES engine was done in the main code, the first operation is the load the “hashcrypt_handle_t”handle structure with key type and key size.
        v. Incomparison to the software key example, where the key is loaded into the AES engine directly, the key isloaded into the OTP shadow registers, and is presented to the AES engine as a secret key. Also, the key size is loaded into the handle. The RT5xx User’s Manual (IMXRT500RM) section Chapter 38 On-Chip OTP Controller explains how the OTP key is being configured via OTP shadow registers in this example.

        ```
        /* secret key default is OTP, 192-bit key */
        SYSCTL0->AESKEY_SRCSEL = 0x2;
        m_handle.keyType = kHASHCRYPT_SecretKey;

        /* 192-bit key loaded to OTP shadow register */
        OCOTP0->OTP_SHADOW[107] = 0;
        OCOTP0->OTP_SHADOW[112]= 0x03020100;
        OCOTP0->OTP_SHADOW[113]= 0x07060504;
        OCOTP0->OTP_SHADOW[114]= 0x0B0A0908;
        OCOTP0->OTP_SHADOW[115]= 0x0F0E0D0C;
        OCOTP0->OTP_SHADOW[116]= 0x13121110;
        OCOTP0->OTP_SHADOW[117]= 0x17161514;
        m_handle.keySize = kHASHCRYPT_Aes192;
        ```

        vi. Call to SDK API function “HASHCRYPT_AES_EncryptCbc” passes the handle, plaintext, IV, returns encrypted results into a cipher array.

vii. The returned cipher data is compared to the expected, defined in the constant array, and if the compare fails, the program aborts with a message that CBC failed.

viii. If results match expected, a call to SDK API function “HASHCRYPT_AES_DecryptCbc” passes the handle, cipher, IV, returns plaintext results into a plaintext array.

ix. The returned plaintext data is compared to the expected, defined in the constant array, and if the compare fails, the program aborts with a message that CBC failed.

x. If results match expected, a message is displayed that CBC passed.

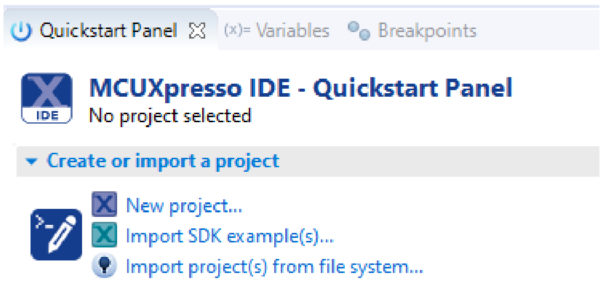
6.4 Example using PUF Key – with MCUXpresso

The AES engine can use a secret (hidden) key from OTP memory or from the PUF IP. A SYSCON register selects whether the secret key is sourced by OTP or by PUF. The AES engine CRYPTCFG register selects whether a software key or secret key is used. This example sources the PUF key to the AES engine.

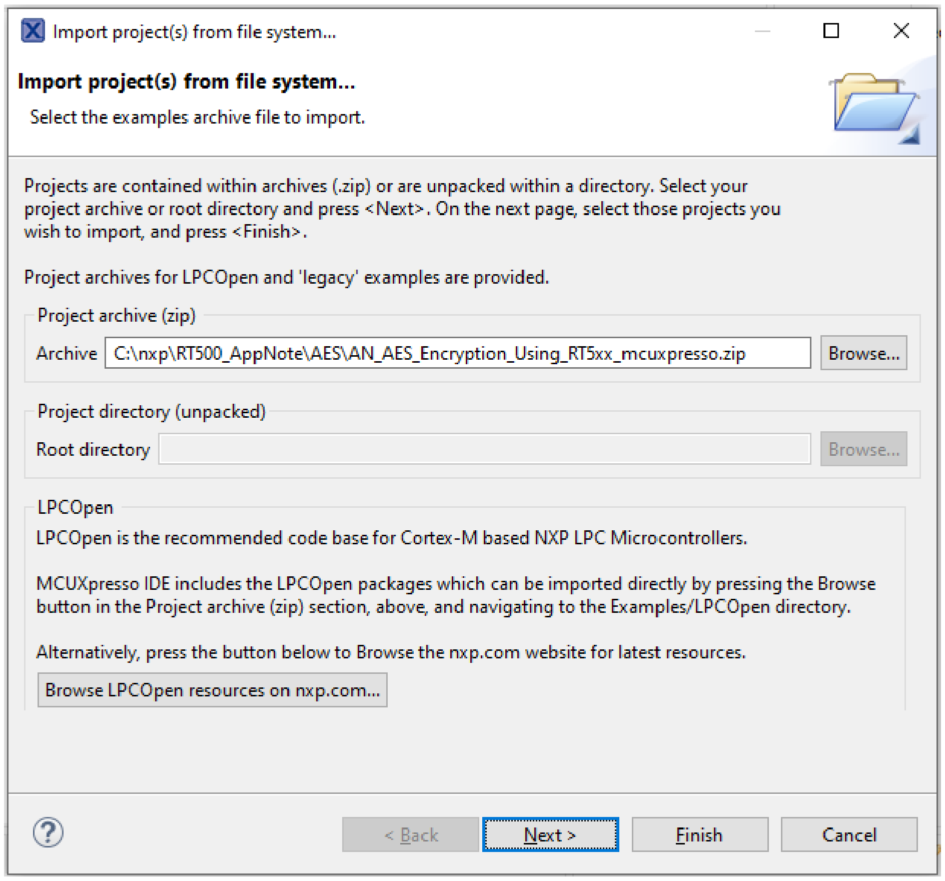
The PUF can maintain 16 keys at a time, but only PUF key, index 0, is unreadable, sourced directly to the AES engine IP. The AES PUF key example provides example code to show how to configure the PUF IP and render the key. That code is not described in detail in this lab, but is in project source file aes pufkey.c in function “puf_init”.

This example, like the others in this application note, can be compiled and debugged using MCUXpresso, Keil uVision or IAR Embedded Workbench. The MCUXpresso is used for this example.

1. Install MCUXpresso IDE – see above section “Install MCUXpresso IDE – if not using Keil or IAR”
2. Download the MCUXpresso archive supplied with this application note contained in zip file AN_AES_Encryption_Using_RT5xx_mcuxpresso.zip.
3. From Project Explorer, select Import project(s) from file system …



4. Chose AN_AES_Encryption_Using_RT5xx_mcuxpresso.zip as Project archive (zip). Click Finish to load all three demos.



Import project(s) from file system...

Import project(s) from file system...

Select the examples archive file to import.

Projects are contained within archives (.zip) or are unpacked within a directory. Select your project archive or root directory and press <Next>. On the next page, select those projects you wish to import, and press <Finish>.

Project archives for LPCOpen and 'legacy' examples are provided.

Project archive (zip)

Archive: C:\nxp\RT500_AppNote\AES\AN_AES_Encryption_Using_RT5xx_mcuxpresso.zip

Project directory (unpacked)

Root directory

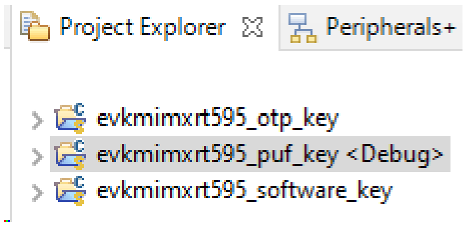
LPCOpen

LPCOpen is the recommended code base for Cortex-M based NXP LPC Microcontrollers.

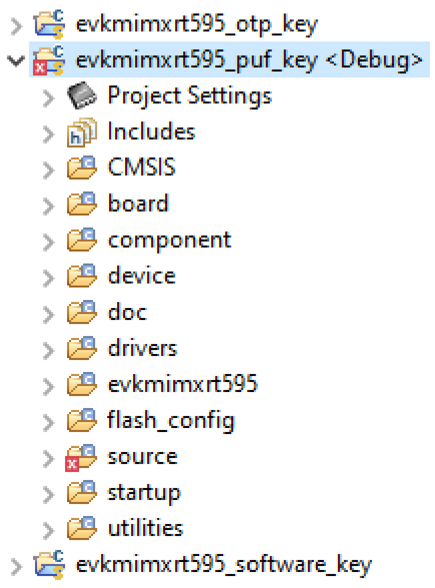
MCUXpresso IDE includes the LPCOpen packages which can be imported directly by pressing the Browse button in the Project archive (zip) section, above, and navigating to the Examples/LPCOpen directory.

Alternatively, press the button below to Browse the nxp.com website for latest resources.

5. The Project Explorer contains other code that was imported to the IDE from the SDK and from the zip.

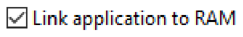


6. Expand the evkmimxrt595_puf_key example:



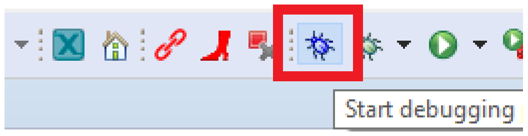
- evkmimxrt595_otp_key
- evkmimxrt595_puf_key <Debug>
  - Project Settings
  - Includes
  - CMSIS
  - board
  - component
  - device
  - doc
  - drivers
  - evkmimxrt595
  - flash_config
  - source
  - startup
  - utilities
- evkmimxrt595_software_key

7. Build code to run from flash (XIP) or from RAM:
   a. Default is to link to execute in place (XIP) from flash memory. Code may instead reside in RAM when booting from flash memory or code is loaded from a host over a serial interface. To link the image to load to RAM:
      i. Right click project "evkmimxrt595_puf_key", select Properties.
      ii. Select C/C++ Build->Settings
      iii. Choose the Tools Settings tab
      iv. Click MCU Linker->Managed Linker Script
      v. Click the checkbox for Link application to RAM

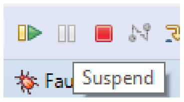
      

      ☑ Link application to RAM

      vi. If unchecked, code is linked to reside in and execute from external flash memory (XIP). The image is programmed into flash as part of the debug step.
      vii. Click Apply and Close
8. Console I/O is configured to a terminal emulation program. Configure the EVK and RS-232 terminal application. From section 6.2, "Example using Software Key", verify steps 9 and 10.
9. Right click project "evkmimxrt595_puf_key", select Build Project
10. Code is built, EVK set up, connected, ready to debug.
    a. Click the blue debug icon shown below.

    b. Code downloads and debugger points to function "main" in project source file aes_pufkey.c.
    c. The code has similar initialization to the other demos in the SDK, specifically ported from the "hello_world" demo. Clocks, pins, and the RS-232 port are all configured before the AES-specific code is executed.
    d. To single-step over functions in MCUXpresso, type F6. To step into a function, type F5. To run, type F8, to stop click the Suspend button (pause button below). To the left of the Suspend button is the Resume (Run) button.



e. Breakpoints are supported, but are not discussed in this application note.

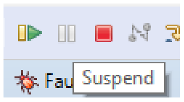
f. To verify the code works, simply type F8 to run to click the Resume button.

g. The console window in the terminal program should display the following:

```
AES ECB, CBC, CTR testing using PUF key

AES ECB Test - 128-bit PUF key - pass
AES CBC Test - 192-bit PUF key - pass
AES CTR Test - 256-bit PUF key - pass
```

h. To leave the debugger, click the Red square button or select menu Run->Terminate.



11. Run the code again, this time to step through code.

    a. Follow steps a-b in step 10 above.

    b. Code is at “main” in project source file aes_pufkey.c.

    c. The AES-specific code is:

```
/* test description */
PRINTF("\r\nAES ECB, CBC, CTR testing using PUF key\r\n\r\n");

/* Initialize Hashcrypt */
HASHCRYPT_Init(HASHCRYPT);


/* Call HASH APIs */
TestAesPufKeyEcb();
TestAesPufKeyCbc();
TestAesPufKeyCtr();

HASHCRYPT_Deinit(HASHCRYPT);
```

    d. The functions with “HASHCRYPT” in the name are SDK API calls and are described above in “SDK API” section.

    e. The functions with “TestAesPufKey” in the name each test a different AES mode (ECB, CBC, CTR) supported by the AES engine and are found in project source file aes_pufkey.c.

    f. Function “TestAesPufKeyCtr” tests CBC mode using a software key.

        i. Press F6 until the cursor (green arrow) is pointed to “TestAesPufKeyCtr”.

        ii. Press F5 to step into the function.

        iii. The function has three constant arrays defined, each 128 bits or 16 bytes.

            i. aes_ctr_test01_plaintext – plaintext – unencrypted data

            ii. aes_ctr_test01_ciphertext – cipher – encrypted data

            iii. aes_ctr_test01_counter_1 initial counter

            iv. aes_ctr_test01_counter_2 initial counter

v. aes_ctr_test01_key – key – key loaded into to PUF

iv. Since initialization of the AES engine was done in the main code, the first operation loads the "hashcrypt_handle_t" handle structure with key type and key size.

v. In comparison to the software key example, where the key is loaded into the AES engine directly, the key is sourced by the PUF IP and presented as the secret key. The PUF IP requires discharging its built-in SRAM, then allowing the SRAM charge, then to initializing it to a known state via acquisition of an activation code (AC) and a key code (KC) for each key. The key can be presented to the IP to generate the key code if key is known or can be generated intrinsically, never to be seen, but still returning a key code. For simplicity, for this example, the key is known.

vi. Key configuration includes programming the SYSCON register to select the PUF key (versus OTP key) and configuring the handle with key type and key size. The "puf_init" function called to initialize and configure the PUF with the known key.

```
SYSCTL0->AESKEY_SRCSEL = 0x0;
m_handle.keyType = kHASHCRYPT_SecretKey; m_handle.keySize =
kHASHCRYPT_Aes256;
/* PUF init */
status = puf_init((uint8_t *) aes_ctr_test01_key, 32);
TEST_ASSERT(0 == status, "PUF");
```

vii. Call to SDK API function "HASHCRYPT_AES_CryptCtr" passes the handle, plaintext, counter, returns encrypted results into a cipher array.

viii. The returned cipher data is compared to the expected, defined in the constant array, and if the compare fails, the program aborts with a message that CTR failed.

ix. If results match expected, a call to SDK API function "HASHCRYPT_AES_DecryptCtr" passes the handle, cipher, counter, returns plaintext results into a plaintext array.

x. The returned plaintext data is compared to the expected, defined in the constant array, and if the compare fails, the program aborts with a message that CTR failed.

xi. If results match expected, a message is displayed that CTR passed.

7 Features and capabilities not demonstrated

It is useful to list some of the features and capabilities that were not tested in the lab or in the application note.

- Plaintext and cipher streams longer than a single 128-bit (16 byte) block.
- The HASHCRYPT SDK API passes input and output array pointers to encrypt/decrypt functions. The functions support use of the AHB bus master input data streaming via use of registers MEMCTRL and MEMADDR. This is effectively HASHCRYPT's built-in DMA, but not tied to the RT5xx DMA engine. Results are read out manually and saved to the output array. Other methods of data input and output streaming not supported by the API:
  - — Manual loading of input data (plaintext or cipher) to the AES engine is not implemented by the API although is less efficient than AHB bus master.
  - — DMA of input and output data is not implemented by the API.
- The API functions poll for completion. Interrupt capability is not implemented. Interrupt capability along with AHB bus master or DMA on input and DMA on output could allow for background AES processing while other tasks are executed in the foreground.
- Indexed Codebook mode (ICB) where a random mask is introduced to prevent Side Channel Attacks (SCA) where current fluctuations can determine actual data.
- Loading of OTP key from programmed OTP bits. Using an EVK and part being soldered, risky since permanent, but that is the proper way to implement an OTP key.

- Loading a PUF key using an intrinsic key (where the key is created randomly inside the IP, never to be seen). To create a test with an intrinsic key, would have to encrypt and decrypt and verify that the plaintext was the output of the decrypt. There is no way to know if the cipher text is correct since the key is not available.

8 Conclusion

This application note provides simple examples of using the AES engine testing ECB, CBC, and CTR modes with software, OTP and PUF generated keys, testing 128-bit, 192-bit, and 256-bit key sizes. Each example includes projects for Keil uVision, IAR Embedded Workbench, and NXP MCUXpresso.

9 Revision history

Table 1. Revision history

Revision number	Date	Substantive changes
0	03/2021	Initial release

Date of release: 03/2021

Document identifier: AN13175