Table of Contents

Prerequisites: ... 2
Glossary ... 2
Objectives ... 2
Hardware ... 3
Lab high level description ... 3
AES on RT685 ... 3
SDK API ... 4
RT685 EVK ... 6
Examples ... 7
Example using Software Key ... 7
Example using OTP (One-Time Programmable) key ... 12
Example using PUF (Physically Unclonable Function) key ... 15
Features and capabilities not demonstrated ... 20
Conclusion ... 20

Prerequisites:

If this lab is done in sequence with the other labs, then there are no prerequisites.

All the software and setup will have been done in the “Getting Started” steps.

If you have not gone through these steps, please go back to the Getting Started lab guide, and completed the download and setup of the IDE and SDK.

1. Download the latest MCUXpresso IDE for your platform (https://mcuxpresso.nxp.com). It is required that MCUXpresso IDE 11.1.1 or newer is installed.

Glossary

A few terms used in this application note.

- AES – Advanced Encryption Standard. A specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST). AES modes use the same key for encryption and decryption. Key size can be 128-bit, 192-bit or 256-bit.
- HASH-CRYPT AES engine – AES IP implemented in the RT6xx. The RT6xx HASH-CRYPT engine supports three AES modes: ECB, CBC and CTR. All three modes are demonstrated in the following labs.
- OTP – One-Time Programmable memory. This memory is partially exposed to the end-user for settings, such as enabling security features and security parameters. The bits are defaulted to logic 0 and once programmed to logic 1, cannot be reverted.
- PUF – Physically Unclonable Function – IP to generate more random and secure keys versus saving/rendering keys from OTP. PUF initialization is typically performed at boot time when the boot image is secure, but as shown in this application note, PUF can be initialized at run-time in user code. The abstracted parameters can either be part of the image or separately saved in flash.

Objectives

In this lab, you will learn:

- RT6xx HASH-CRYPT AES engine architecture
- How to use the HASH-CRYPT AES engine with the SDK-API
- Use of the PUF and its advantages/features
- Use of the MCUXpresso SDK API Reference Manual
- Test the results with AES application note examples
    - How to load an AES key (software key) directly to the AES engine
    - How to load an AES key using OTP shadow registers (OTP key)
    - How to load an AES key using the PUF IP (PUF key)

Hardware

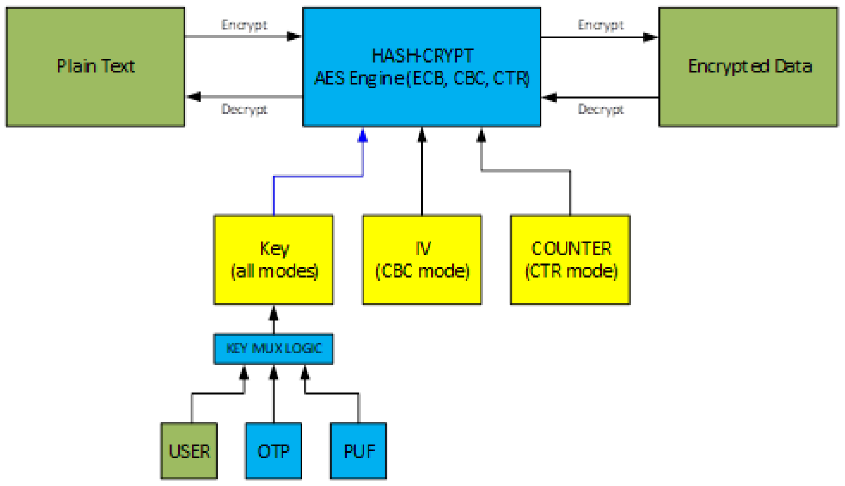
- NXP i.MX RT600 Evaluation Kit - MIMXRT685-EVK
- Micro-USB cable

Lab high level description

In this lab you will gain an understanding of the RT6xx HASH-CRYPT IP AES engine features and capabilities.

AES on RT685

The following is a brief description of the implementation of the AES on the RT6xx MCU series.



This diagram 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.

The key features of the AES engine are:

- 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-bits 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.

SDK API

The SDK HASHCRYPT API makes the AES engine easy-to-use. The API consists of a handful of function calls to allow use of the AES engine ECB, CBC and CTR modes. Since the HASHCRYPT IP contains both the AES and SHA Engines, they also share the same API source (fsl_hashcrypt.c) and header file (fsl_hashcrypt.h).

Application code must include the following header files:

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

The API documentation is available online:

https://mcuxpresso.nxp.com/api_doc/dev/1581/group_hashcrypt_driver.html
https://mcuxpresso.nxp.com/api_doc/dev/1581/group_hashcrypt_driver__aes.html

Below are the key excerpts from those sections of the SDK API manual.

HASHCRYPT mode and key enumerations:

```
enum   hashcrypt_aes_mode_t {
  kHASHCRYPT_AesEcb = 0U,
  kHASHCRYPT_AesCbc = 1U,
  kHASHCRYPT_AesCtr = 2U
}

enum   hashcrypt_aes_keysize_t {
  kHASHCRYPT_Aes128 = 0U,
  kHASHCRYPT_Aes192 = 1U,
  kHASHCRYPT_Aes256 = 2U,
  kHASHCRYPT_InvalidKey = 3U
}

enum   hashcrypt_key_t {
  kHASHCRYPT_UserKey = 0xc3c3U,
  kHASHCRYPT_SecretKey = 0x3c3cU
}
```

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);
```

CBC mode functions (similar to ECB but also pass pointer to 128-bit IV):

```
status_t HASHCRYPT_AES_EncryptCbc(HASHCRYPT_Type *base, hashcrypt_handle_t
*handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t
iv [16]) ;

status_t HASHCRYPT_AES_DecryptCbc(HASHCRYPT_Type *base, hashcrypt_handle_t
*handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t
iv [16]) ;
```

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

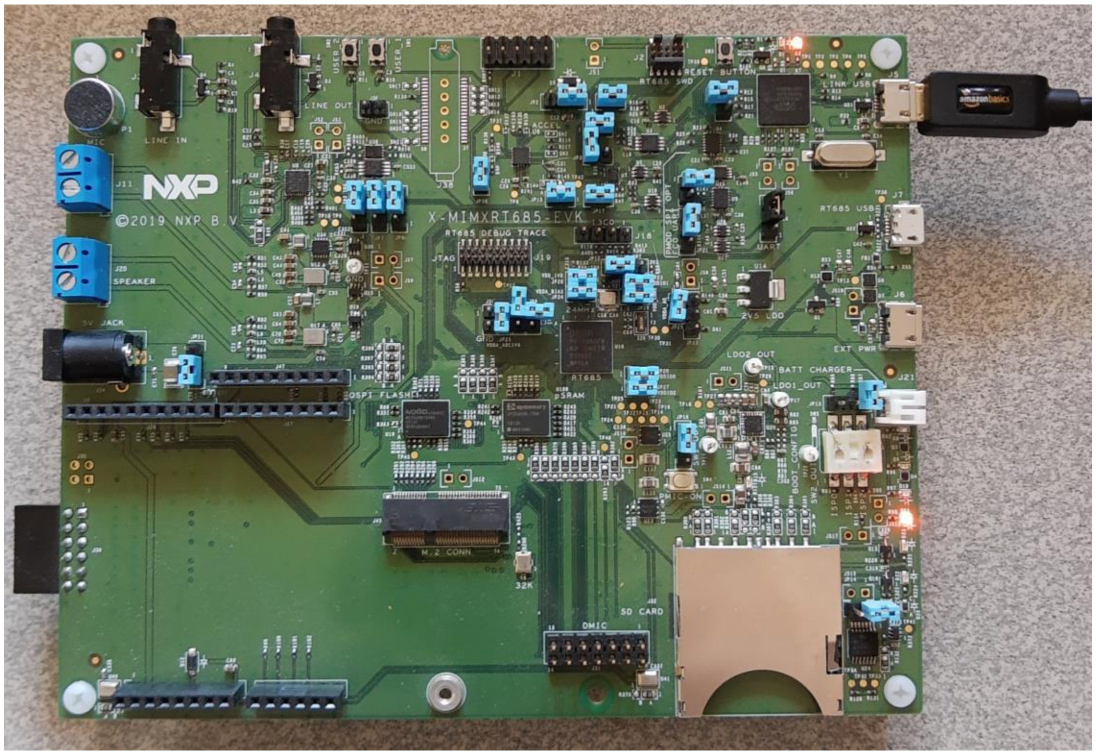
```
status_t HASHCRYPT_AES_CryptCtr(HASHCRYPT_Type *base, hashcrypt_handle_t *handle,
const uint8_t *input, uint8_t *output, size_t size, uint8_t
counter[HASHCRYPT_AES_BLOCK_SIZE], uint8_t counterlast[HASHCRYPT_AES_BLOCK_SIZE],
size_t *szLeft);
```

RT685 EVK

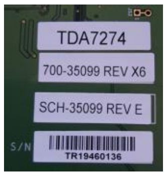
The development platform is the NXP i.MX RT600 Evaluation Kit (part # MIMXRT685-EVK). The board contains an NXP MIMXRT685SFVKB device a 300 MHz with an ARM® Cortex® M33 CPU and a 600 MHz Cadence® Tensilica® HiFi 4.

https://www.nxp.com/design/development-boards/i-mx-evaluation-and-development-boards/i-mx-rt600-evaluation-kit:MIMXRT685-EVK

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



A sticker on bottom of board must read REV E.



Examples

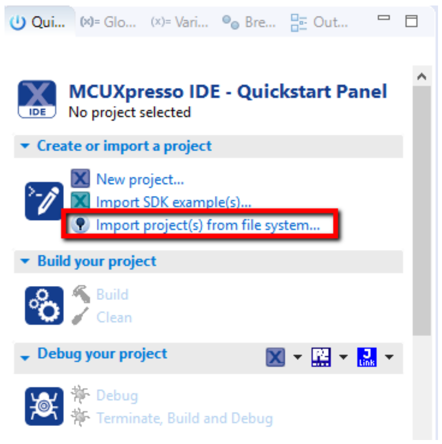
Each of the following sections overviews the AES demo applications, each demonstrating a different key type: user (software) key, OTP key and PUF key, using the MCUXpresso IDE.

Example using Software Key

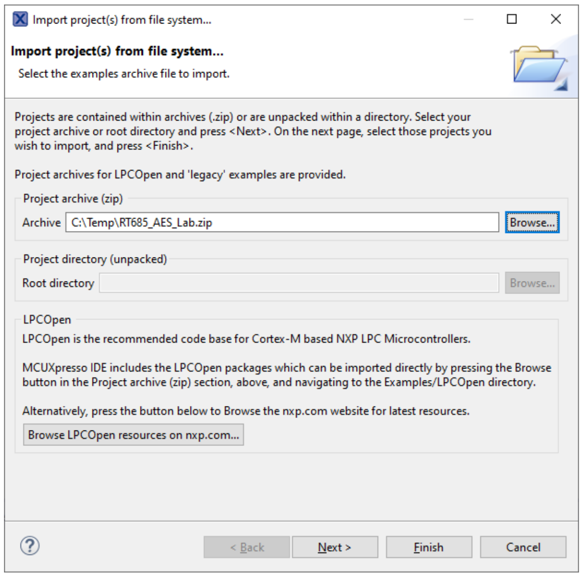
The simplest way to use the AES engine is to load a software key, either downloaded from a host, saved to external flash memory, or like in this case, defined in code for demonstration purposes.

This example, is compiled and debugged using MCUXpresso:

1) Install MCUXpresso IDE 11.1.1 or newer.
2) Download the RT685_AES_Lab.zip folder from the same folder as this document.
3) Open MCUXpresso IDE 11.1.1.
4) Close the Welcome window.
5) On the **Quickstart Panel**, select **Import project(s) from file system…**



6) At the Project archive (zip), click Browse and select RT685_AES_Lab.zip from the folder from where it is saved, then click Next.



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:\Temp\RT685_AES_Lab.zip

Project directory (unpacked)

Root directory

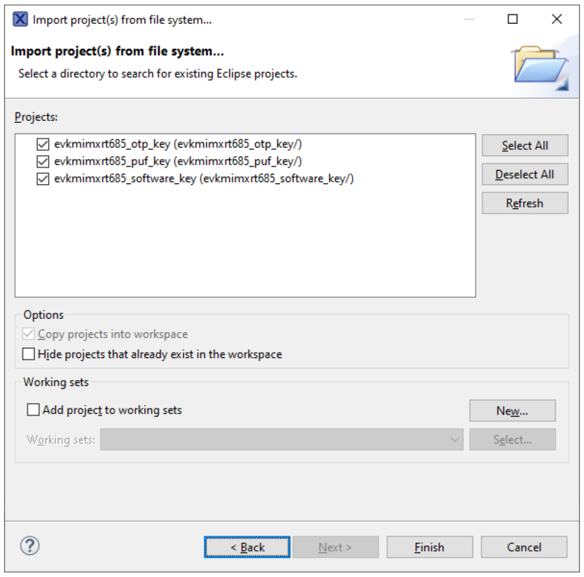
LPCOpen

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.

7) Make sure that all three projects are selected (checked), then click Finish.



Import project(s) from file system...

Select a directory to search for existing Eclipse projects.

Projects:

- ☑ evkmimxrt685_otp_key (evkmimxrt685_otp_key/)
- ☑ evkmimxrt685_puf_key (evkmimxrt685_puf_key/)
- ☑ evkmimxrt685_software_key (evkmimxrt685_software_key/)

Options

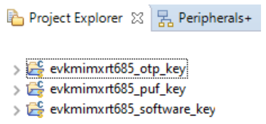
- ☑ Copy projects into workspace
- ☐ Hide projects that already exist in the workspace

Working sets

- ☐ Add project to working sets

Working sets:

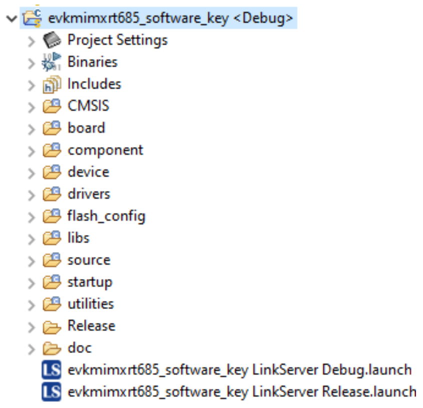
8) You should now see three new projects loaded into the Project Explorer as shown below.



- evkmimxrt685_otp_key
- evkmimxrt685_puf_key
- evkmimxrt685_software_key

PUBLIC

9) Expand the evkmimxrt685_software_key example:



```
evkmimxrt685_software_key <Debug>
  Project Settings
  Binaries
  Includes
  CMSIS
  board
  component
  device
  drivers
  flash_config
  libs
  source
  startup
  utilities
  Release
  doc
  evkmimxrt685_software_key LinkServer Debug.launch
  evkmimxrt685_software_key LinkServer Release.launch
```

10) Two project setting changes that may be useful, although not required:

   a. Default setting is to link to execute code “in place” (XIP) from flash memory. Code more typically resides in internal RAM when booting from flash memory or code is loaded from a host over a serial interface.

      To link the image as load to RAM:

      i. Right click on project, 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

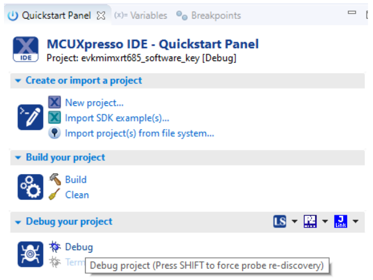
      vi. If unchecked, code will be linked to reside in and execute from external flash memory (XIP). The image is programmed into flash as part of the debug step.
      vii. Click Apply and Close.

   b. Default is Console I/O, which is directed to the MCUXpresso Console window.

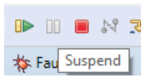
      To direct console I/O to a terminal emulation program:

      i. Right click on project “evkmimxrt685_sotfware_key”, select Properties.
      ii. Select C/C++ Build->Settings.
      iii. Choose the Tools Settings tab.
      iv. Click MCU C Compiler->Preprocessor.
      v. Under Defined symbols, change SDK_DEBUGCONSOLE=0 to SDK_DEBUGCONSOLE=1.
      vi. Click Apply and Close.

11) If using a terminal emulation program:
    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.
12) Right click on project "evkmimxrt685_software_key", select Build Project.
13) Code is built, EVK set up, connected, ready to debug.
    a. Select the "evkmimxrt685_software_key".
    b. Click Debug in the Quickstart Panel on the bottom-left of the MCUXpresso IDE.


Quickstart Panel
Variables
Breakpoints
MCUXpresso IDE - Quickstart Panel
Project: evkmimxrt685_software_key [Debug]
Create or import a project
New project...
Import SDK example(s)...
Import project(s) from file system...
Build your project
Build
Clean
Debug your project
Debug
Debug project (Press SHIFT to force probe re-discovery)

    c. Code will download and debugger will point to function "main" in project source file aes_softkey.c.
    d. 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.
    e. 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.

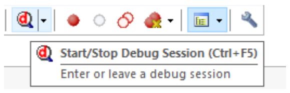


    f. Breakpoints are supported but are not discussed in this application note.
    g. To verify the code works, simply type F8 to run.
    h. The console window in the terminal program should display the following:

```
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
```

    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

14) Run the code again, this time to step through code.

   a. Follow steps a-b in step 13 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 "TestAesSoftkeyKeyECB" tests ECB mode using a software key.
      i. Press F6 until the cursor (green arrow) is pointed to "TestAesSoftKeyEcb".
      ii. Press F5 to step into the function.
      iii. The function has three constant arrays defined, each 128 bits or 16 bytes.
         1. keyAes128 – software key
         2. plainAes128 – plaintext – unencrypted data
         3. cipherAes128 – cipher – encrypted data
      iv. Since initialization 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 cipher array.
      vi. 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 ECB failed.

vii. If results match as expected a call to SDK API function "HASHCRYPT_AES_DecryptEcb" passes the handle, cipher, returns plaintext results into a plaintext array.

viii. 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 ECB failed.

ix. If results match as expected, a message is displayed that ECB passed.

x. Function calls to similar CBC and CTR tests are made from "main". All in this example are 128-bit software keys, 128-bit data arrays, and 128-bit IV or CTR arrays for CBC and CTR modes respectively.

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

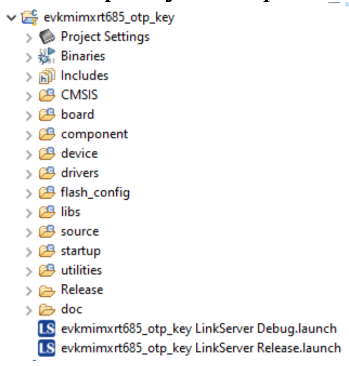
Example using OTP (One-Time Programmable) Key

The AES engine can use a secret (hidden) key either from RT6xx 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.

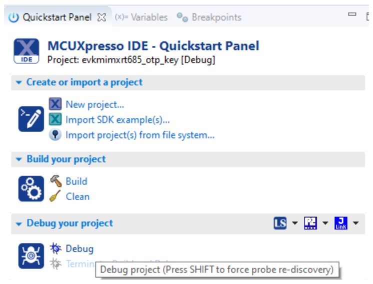
The OTP key is exposed unless OTP settings prevent reading that specific OTP register bank (read lock loaded by boot ROM). Also, OTP has a bank of shadow (RAM) registers to allow testing of the OTP feature before permanently programming OTP cells from 0 to 1. The steps below will explain how to program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

This example is compiled and debugged using MCUXpresso.

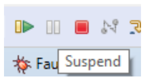
1) Follow steps 1-8 in the "Example using Software Key" above. If that example has been done, steps 1-8 can be skipped.
2) Expand the evkmimxrt685_otp_key example:



3) Project setting options – see above “Example using Software Key”, 10a and 10b.
4) If using a terminal emulation program:
    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.
5) Right click on project “evkmimxrt685_otp_key”, select Build Project.
6) Code is built, EVK set up, connected, ready to debug.
    a. Select the “evkmimxrt685_otp_key”.
    b. Click Debug in the Quickstart Panel on the bottom-left of the MCUXpresso IDE.


Quickstart Panel
Variables
Breakpoints
MCUXpresso IDE - Quickstart Panel
Project: evkmimxrt685_otp_key [Debug]
Create or import a project
New project...
Import SDK example(s)...
Import project(s) from file system...
Build your project
Build
Clean
Debug your project
Debug
Debug project (Press SHIFT to force probe re-discovery)

    c. Code will download and debugger will point to function “main” in project source file aes_otpkey.c.
    d. 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.
    e. 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.

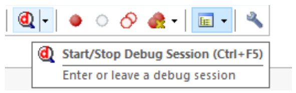


    f. Breakpoints are supported but are not discussed in this application note.
    g. To verify the code works, simply type F8 to run.
    h. The console window in the terminal program should display the following:

```
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
```

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

7) Run the code again, this time to step through code.
   a. Follow steps a-b in step 6 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 F6 until the cursor (yellow Triangle) is pointed to “TestAesOtpKeyCbc”.
      ii. Press F5 to step into the function.
      iii. The function has three constant arrays defined, each 128 bits or 16 bytes.
         1. plainAes128 – plaintext – unencrypted data
         2. cipherAes128 – cipher – encrypted data
         3. ive – IV - initial vector
      iv. Since initialization 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. In comparison to the software key example, where the key is loaded into the AES engine directly, the key is loaded 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 RT6xx User's Manual (UM11147) section 42.2.2 overviews OTP key storage. Table 1071 note 2c 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 */
OCOTP->OTP_SHADOW[107] = 0;
OCOTP->OTP_SHADOW[112] = 0x03020100;
OCOTP->OTP_SHADOW[113] = 0x07060504;
OCOTP->OTP_SHADOW[114] = 0x0B0A0908;
OCOTP->OTP_SHADOW[115] = 0x0F0E0D0C;
OCOTP->OTP_SHADOW[116] = 0x13121110;
OCOTP->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 as 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 as expected, a message is displayed that CBC passed.

Example using PUF (Physically Unclonable Function) Key

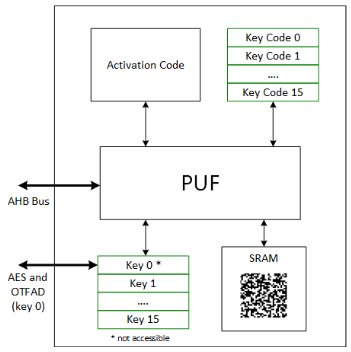
The AES engine can use a secret (hidden) key from OTP memory or from the PUF IP. A SYSCON register selects whether the secret key is sourced by OTP or by PUF. The AES engine CRYPTCFG register selects whether a software key or secret key is used. This example sources the PUF key to the AES engine.

The PUF can maintain 16 keys at a time, but only PUF key, index 0, is unreadable by software, 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".

PUF is IP from a 3rd party, Intrinsic ID, which can render unique keys (16 at a time) mainly based on the state of a PUF specific RAM (embedded, unreadable). Power up state of PUF RAM widely varies device to device, but is roughly consistent for any individual device, assuming a long enough time powered off for discharge. This allows for reliable use for rendering keys based on its state.

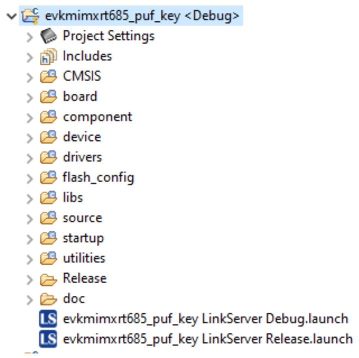
PUF is a memory mapped peripheral with an SDK API to allow configuration and use, but usually PUF is used by the bootloader for:

- boot image decryption
- configuration of the OTFAD feature that allows for encrypted external flash code to be decrypted/executed
- recovery boot
- generation of the DICE CDI code used for cloud system verification



This example, like the other examples, are compiled and debugged using MCUXpresso.

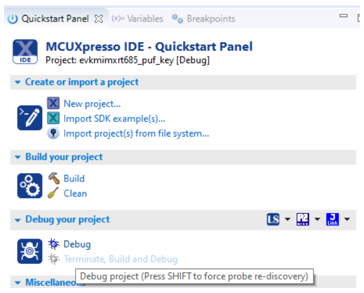
1) Follow steps 1-8 in the "Example using Software Key" above. If that example has been done, steps 1-8 can be skipped.
2) Expand the evkmimxrt685_puf_key example:



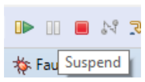
evkmimxrt685_puf_key <Debug>
- Project Settings
- Includes
- CMSIS
- board
- component
- device
- drivers
- flash_config
- libs
- source
- startup
- utilities
- Release
- doc
- evkmimxrt685_puf_key LinkServer Debug.launch
- evkmimxrt685_puf_key LinkServer Release.launch

3) Project setting options – see above "Example using Software Key", 10a and 10b.

4) If using a terminal emulation program:
    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.
5) Right click on project “evkmimxrt685_puf_key”, select Build Project
6) Code is built, EVK set up, connected, ready to debug.
    a. Select the “evkmimxrt685_puf_key”.
    b. Click Debug in the Quickstart Panel on the bottom-left of the MCUXpresso IDE.


Quickstart Panel
Variables
Breakpoints
MCUXpresso IDE - Quickstart Panel
Project: evkmimxrt685_puf_key [Debug]
Create or import a project
New project...
Import SDK example(s)...
Import project(s) from file system...
Build your project
Build
Clean
Debug your project
Debug
Terminate, Build and Debug
Miscellaneo
Debug project (Press SHIFT to force probe re-discovery)

    c. Code will download and debugger will point to function “main” in project source file aes_pufkey.c.
    d. 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.
    e. 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.

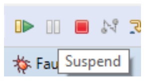


    f. Breakpoints are supported but are not discussed in this application note.
    g. To verify the code works, simply type F8 to run.
    h. The console window in the terminal program should display the following:

```
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
```

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



7) Run the code again, this time to step through code.
   a. Follow steps a-b in step 6 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.
         1. aes_ctr_test01_plaintext – plaintext – unencrypted data
         2. aes_ctr_test01_ciphertext – cipher – encrypted data
         3. aes_ctr_test01_counter_1 initial counter
         4. aes_ctr_test01_counter_2 initial counter
         5. 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.

```
/* secret key PUF, 256-bit 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.

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 RT6xx 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 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.

Conclusion

The lab provided 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. An application note titled entitled "AES Encryption/Decryption Using RT6xx" is like this lab, but requires code is first unzipped into an existing, unzipped SDK project versus from an archive as is done here. The application note code also includes Keil uVision and IAR Embedded Workbench projects along with NXP MCUXpresso projects.