AM263X OTP Keywriter User Guide

SITARA MCU SW

Exported on 03/07/2023

Table of Contents

1 Introduction

Note

- For detailed understanding of Secure Device (HS-FS/HS-SE) life cycle, Keywriter SW scope, this document has dependencies on Device Security HW Addendum[1], MCU+ SDK documentation for SBL[2] and tools are required.
- This package is applicable to HS-FS device variant of AM263x SR1.1 only.
- The OTP Keywriter for AM263x SR1A is available on MySecureSW.[3]

1.1 Scope

Key writer firmware is a fixed functionality SW which enables the transitioning of the Secure device life cycle from HS-FS (development variant with out security enforcement) to HS-SE (production variant with security enforced).

This device type transition for every SoC can happen only once either in development environment using CCS or in production environment relying on device ROM boot modes to load and execute key writer firmware. The default configuration of the SBL Keywriter is developed and validated on TI HW reference CCs and would require bootmode/Vpp customization for other HW.

OTP (One Time Programmable) Keywriter is a combination of Firmware, Software(SBL) and Tools(Certificate generation) which together when executed on device enables

- Provisioning up to 2 sets of Customer Keys (Both Public Keys for Root of Trust of Boot Images and Encryption Keys for decryption of Boot Images)
- Programming of OTP fields KEY_COUNT, KEY_REVISION which will enable device transition from HS-FS (Field Securable) to HS-SE (Security Enforced)
- Programming of OTP fields like SWREV for SBL/HSM/APP images which will enforce Anti Roll Back checks by Secure Boot on HS-SE device
- Programming of Extended OTP fields which are additional OTP fields available for customer specific usage
- Enable OTP protection for locking and disabling read/write access permissions via e-fuse controller

Once these fields are programmed, the device state will be transitioned to HS-SE, boot ROM will enforce secure boot with Image Authentication and decryption based on the keys provisioned and configured in the device.

Important

This action of burning the keys is irreversible across the fields, so caution should be exercised to provide the Key values in correct format and correct Key configurations.
Fields once programmed are irreversible and providing incorrect values or configuration can permanently damage the device.
Once the device is transitioned to HS-SE, device will not be able to boot Keywriter images.

1.2 Features Supported:

New Features in this release highlighted in bold.

- **Support for SR1.1 Revision of AM263x**

1 https://www.ti.com/licreg/docs/swlicexportcontrol.tsp?form_id=337487&prod_no=AM263X-RESTRICTED-SECURITY&ref_url=EP-proc-Sitara-MCU
2 https://software-dl.ti.com/mcu-plus-sdk/esd/AM263X/08_05_00_24/exports/docs/api_guide_am263x/EXAMPLES_DRIVERS_SBL.html
3 https://www.ti.com/licreg/docs/swlicexportcontrol.tsp?form_id=337487&prod_no=AM263X-RESTRICTED-SECURITY&ref_url=EP-proc-Sitara-MCU

- Signed Key writer firmware for HSM which accepts x.509 customer keys certificate with all OTP fields configured.
- **SBL Keywriter (which loads and Keywriter firmware to HSM) integrated with MCU+ SDK build and tools**
- **Protective checks in keywriter usage to prevent bricking of devices/ROM failures.**
- **Firewall updated to protect TOP_EFUSE_FARM**
- **Support for IO expander for controlling LDO for VPP supply in AM263x-cc revision-E2A.**
- **eFuse driver WritePulseWidth changed from 2us to 1us.**
- **Supports configuration of delay between eFuse row writes with default value for delay as 0.** (Please see appendix for more details)
- **Supports mcu_plus_sdk windows/Linux build environment**
- Supports programming keys at one-pass/Multi-pass customer key certificates
- Customer Key certificate generation scripts and utilities for configuring different fields
- Supports UART, QSPI and **JTAG (via DEV boot)** modes for key programming.
- Following are the keys programmable
    - MSV
    - SMPK , SMEK
    - BMPK , BMEK
    - EXT OTP
    - KEY COUNT
    - SWREV - HSM, SWREV - APP, SWREV-SBL, KEY REV

Important

SBL Keywriter images does not have any dependencies on Boot modes supported by ROM, SBL Keywriter will work on all boot modes supported by the device.

1.3 Summary of HS-FS Vs HS-SE device features

	HS-FS	HS-SE
R5F Boot Image (SBL)	Secure boot not enforced	Secure boot enforced with customer keys programmed by key writer
HSM Boot Image	Secure boot enforced (Only TI signed Key writer binary)	Secure boot enforced with customer keys programmed by key writer
R5 JTAG	Open by default	Closed by default
SoC Firewalls	Open by default	Disabled for HSM, and enabled for R5F
SoC DevBoot Mode / JTAG development	Enabled by default	Disabled by default

1.4 Programmable Keys:

Keywriter supports programming of following key types and fields described in the below table.

Key	Description	KeyWriter usage notes	Impact on HS-SE Device
SMPKH	Secondary Manufacturer Public Key Hash SMPKH Length: 512 Bits BCH Length: 64 Bits	Customer primary key set: Public Key Hash SMPK is 4096-bit customer RSA Public key used for verifying RSA signature included in Boot Images. SMPKH Value: 512 bit hash of SMPK generated via SHA512 hashing algo which will be split into two parts of 256-bits i.e SMPKH_P1/SMPKH_P2 BCH Value: 32-bit BCH will be computed using algorithm (288, 261, 7) for each part of SMPKH_P1 and SMPKH_P2, BCH algorithm that can support 1 bit error correction and detect 2 bit errors Key Protection: It is mandatory to Protect SMPKH Efuses as altering these efuses accidentally can lead to permanent failure of device boot. Checks enforced by KeyWriter: N/A	Secure boot active key to validate Root of Trust for boot Image x.509 certificate when key configuration field KEYREV=1
SMEK	Secondary Manufacturer Encryption Key SMEK Length: 256 Bits BCH Length: 32 Bits	Customer primary key set: Encryption Key SMEK is 256-bit customer encryption key for Boot Image Encryption SMEK Value: Original 256 Bit Symmetric Key used for AES-CBC Encryption of Boot Images BCH Value: 32-bit BCH will be computed using algorithm (288, 261, 7) for 256 bit SMEK supports 1 bit error correction and detect 2 bit errors Key Protection: It is mandatory to Protect SMEK Efuses as this is a Symmetric Key and altering these efuses accidentally can lead to permanent failure of device boot. Checks enforced by KeyWriter: N/A	Active key for secure boot to decrypt the boot Image if encrypted and enabled via x509 certificate when key configuration field KEYREV=1

Key	Description	KeyWriter usage notes	Impact on HS-SE Device
BMPKH	Backup Manufacturer Public Key Hash BMPKH Length: 512 Bits BCH Length: 64 Bits	Customer backup key set: Public Key Hash **Note: Device supports Backup key pair which should be provisioned by Key writer and can be activated once product is deployed in field via incremental programming of KEVREV field in field. Note that if backup key pair is not provisioned by keywriter on HS-FS device then it is not possible to program backup key pair later once device transitions to HS-SE.** BMPK is 4096-bit customer RSA Public key used for verifying RSA signature included in Boot Images. BMPKH Value: 512 bit hash of BMPK generated via SHA512 hashing algo which will be split into two parts of 256-bits i.e BMPKH_P1/BMPKH_P2 BCH Value: 32-bit BCH will be computed using algorithm (288, 261, 7) for each part of BMPKH_P1 and BMPKH_P2, BCH algorithm that can support 1 bit error correction and detect 2 bit errors Key Protection: It is mandatory to Protect BMPKH Efuses as altering these efuses accidentally can lead to permanent failure of device boot. Checks enforced by KeyWriter: N/A	Secure boot active key to validate Root of Trust for boot Image x.509 certificate when key configuration field KEYREV=2
BMEK	Backup Manufacturer Encryption Key BMEK Length: 256 Bits BCH Length: 32 Bits	Customer backup key set: Encryption Key BMEK is 256-bit customer encryption key for Boot Image Encryption BMEK Value: Original 256 Bit Symmetric Key used for AES-CBC Encryption of Boot Images BCH Value: 32-bit BCH will be computed using algorithm (288, 261, 7) for 256 bit BMEK supports 1 bit error correction and detect 2 bit errors Key Protection: It is mandatory to Protect BMEK Efuses as this is a Symmetric Key and altering these efuses accidentally can lead to permanent failure of device boot. Checks enforced by KeyWriter: N/A	Active key for secure boot to decrypt the boot Image if encrypted and enabled via x.509 certificate when key configuration field KEYREV=2

Key	Description	KeyWriter usage notes	Impact on HS-SE Device
KEYCNT	Key count configuration field Length: 16 bits	Active Key Sets provisioned in the device, and supports following values 1 if only Primary Key Set is Plan of Record for device (SMPKH /SMEK) 2 if both Primary Key Set and Backup Key Set is Plan of Record for device (SMPKH /SMEK) and (BMPKH / BMEK) **Note: This field will control the active Key sets in the device, if user wants to enable backup Key set it is mandatory to provision backup key pair and set KEYCNT as 2 by Keywriter. If this field is programmed as 1 then device can only support primary Key set for Secure boot and this field can not be updated later.** Key Protection: It is mandatory to Write Protect KEYCNT efuses as altering these efuses accidentally can lead to permanent failure of device boot. Checks enforced by KeyWriter: For Programming KEYCNT value of 1, KeyWriter will enforce SMPK Keys are write protected and for KEYCNT value of 2, KeyWriter will enforce SMPK Keys and BMPK Keys are write protected	Active Key Sets provisioned in the device for Key Manager to decode and setup the Keys.

Key	Description	KeyWriter usage notes	Impact on HS-SE Device
KEYREV	Key revision configuration field Length: 16 bits	Current active key revision in the device for Root of Trust and supports following values 1 for Primary Key Set (SMPK/SMEK) used for Secure Boot **(Recommended config for KeyWriter)** 2 for Backup Key Set (BMPK/BMEK) used for Secure Boot Key Protection: It is **NOT** recommended to Write Protect KEYREV efuses as this field can be incremented during the life time of device to change Root Of Trust. Checks enforced by KeyWriter: For programming KEYREV value, KeyWriter will enforce a check to verify KEYCNT is programmed in efuses and KEYREV value <= KEYCNT programmed in efuses **Note: Once this fields is programmed device will transition from HS-FS to HS-SE and device will stop booting of Key writer firmware. In Multi Pass mode usage of Key Writer this field should be programmed as the last field in the flow of different iterations**	Current active key revision for Secure Boot
MSV	Model specific value Length: 20 bits BCH: 12 bits	20 bit Model Specific Value, Customers can program 20 bit values to differentiate product variants using same SoC either in production flow Or in boot flow of the device. BCH Value: 12-bit BCH will be computed using algorithm (32, 20, 12) for 20 bit MSV supports 1 bit error correction and detect 2 bit errors	No Impact for Boot ROM, SW needs to comprehend the usage of this field
SWREV-SBL	SBL software revision Length: 64 bits	Supports 32 values (1 to 32 states with double redundancy) **Recommended to set initial value of 0x1** Key Protection: It is **NOT** recommended to Write Protect SWREV efuses as this field can be incremented during the life time of device to enforce anti Roll back for R5 SBL Images.	Enables Anti Roll back feature for SBL Image x.509 Certificate via SWRV extension

Key	Description	KeyWriter usage notes	Impact on HS-SE Device
SWREV-HSM	TIFS-MCU software revision Length: 64 bits	Supports 32 values (1 to 32 states with double redundancy) **Recommended to set initial value of 0x1** Key Protection: It is **NOT** recommended to Write Protect SWREV efuses as this field can be incremented during the life time of device to enforce anti Roll back for TIFS MCU Images.	Enables Anti Roll back feature for TIFS-MCU Image x.509 Certificate via SWRV extension
SWREV-APP	Application Image software revision Length: 128 bits	Supports 64 values (1 to 64 states with double redundancy) **Recommended to set initial value of 0x1** Key Protection: It is **NOT** recommended to Write Protect SWREV efuses as this field can be incremented during the life time of device to enforce anti Roll back for App Images.	No Impact for Boot ROM, TIFS-MCU needs to comprehend the usage of this fields
EXTENDED OTP	Extended OTP array Length: 1664 bits	1664 bit extended otp array for customer usage	No Impact for Boot ROM

1.5 Components

OTP Keywriter contents include:

1. MCU+SDK keywriter app source code

- This is the keywriter application and It boots directly on the primary boot core of the device (R5F) without any other bootloader. It is responsible for sending the OTP configuration to the secure keywriter component running on the security subsystem to blow the keys into the device eFuses.
- **x.509 certificate** configuration template, and **generation script** as a part of source code
  - This is a reference script. It can either be used as-is with the customer's keys or integrated within the customer's HSM environment.

2. Encrypted TIFS keywriter binary

   This is the secure OTP keywriter firmware, and runs on the secure subsystem of the device (M4F,). It is responsible for verifying the OTP configuration from x.509 certificate and blowing the data into the device efuses.

> **Note**
>
> This binary differs from the standard TIFS-MCU binary used in production.

3. TI FEK (public) key - TI Field Encryption Key

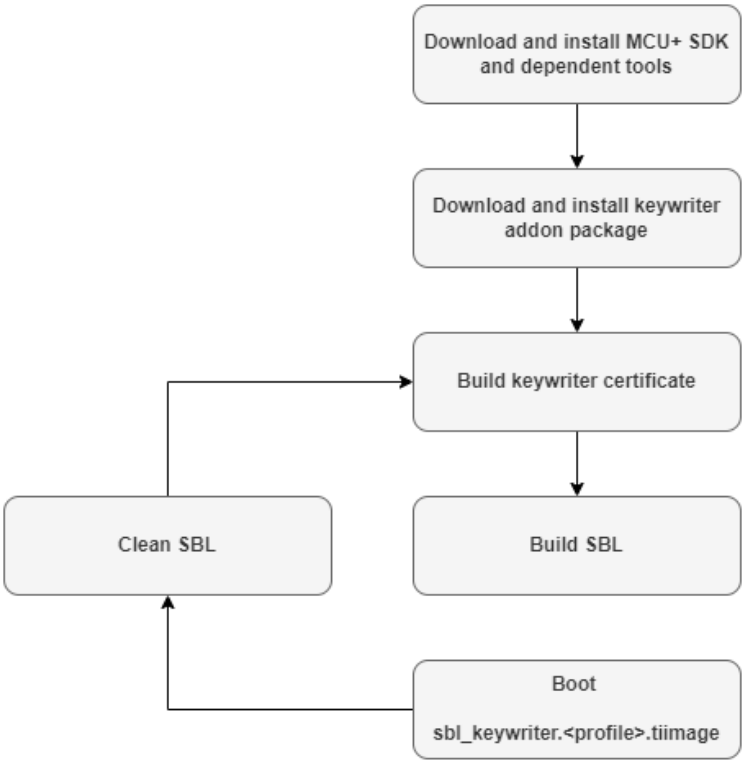
This will be required by the certificate generation script, to ensure confidentiality of the customer's key material. TI FEK Public key will be used to encrypt content, and TI FEK private key (not shared, but part of encrypted TIFS keywriter binary) will be used to decrypt it (at runtime, inside the secure subsystem).

4. Tools

1. load_sbl.js - Script used to load keywriter executables to R5 Core-0 on the SOC.
2. socid_parser (mcup_uart_boot_socid.py)(see page 34) - A python based parser to convert the hexadecimal numbers reported by ROM to human readable text.

1.6 Keywriter app build flow and directory structure

Following pictures show overview of keywriter build flow.


Download and install MCU+ SDK and dependent tools
Download and install keywriter addon package
Build keywriter certificate
Build SBL
Boot
sbl_keywriter.<profile>.tiimage
Clean SBL

```
.
├── tifs
│   └── sbl_keywriter
│       ├── am263x
│       │   └── r5fss0-0_nortos
│       │       ├── board.c
│       │       ├── example.syscfg
│       │       ├── keywriter_utils.c
│       │       ├── keywriter_utils.h
│       │       ├── main.c
│       │       └── ti-arm-clang
│       │           ├── example.projectspec
│       │           ├── linker.cmd
│       │           ├── makefile
│       │           ├── makefile_ccs_bootimage_gen
│       │           ├── makefile_projectspec
│       │           └── syscfg_c.rov.xs
│       ├── docs
│       │   └── am263x
│       │       ├── AM263x_CC_E2_Modifications_rev1.pdf
│       │       └── AM263X_OTP_Keywriter_User_Guide.pdf
│       ├── keywr_bin
│       │   └── am263x
│       │       └── SR_11
│       │           ├── Cust_KeysCert.h
│       │           ├── hsmRt_HSSEboot.bin
│       │           ├── hsmRtImage_testHSSE.h
│       │           └── hsmRtImg_keywr.h
│       ├── manifest
│       │   └── am263x
│       │       ├── AM263X_1.1_SDK_app_firmware_for_keywriter_08.06.00_manifest.html
│       │       └── TILAW-#347661-v12A-Production_Clickwrap_Template-_EP_Analog.DOCX
│       ├── scripts
│       │   └── cert_gen
│       │       └── am263x
│       │           ├── construct_ext_otp_data.sh
│       │           ├── gen_keywr_cert_helpers.sh
│       │           ├── gen_keywr_cert.sh
│       │           ├── keys_devel
│       │           │   ├── aes256.key
│       │           │   ├── bmek.key
│       │           │   ├── bmpk.pem
│       │           │   ├── smek.key
│       │           │   └── smpk.pem
│       │           ├── templates
│       │           │   ├── keycert_bmpk_template.txt
│       │           │   └── keycert_template.txt
│       │           └── tifek
│       │               └── SR_11
│       │                   └── ti_fek_public.pem
│       └── tools
│           ├── ccs_load
│           │   └── load_sbl.js
│           ├── hsfs_flashwriter
│           └── socid_parser
│               ├── mcup_uart_boot_socid.py
│               ├── parser.md
│               └── socid.txt
├── uninstall
└── uninstall.dat
```

1.7 Board Dependencies

- VPP pin of the SoC must be powered (1.7v) when the efuses are being programmed. Refer to device datasheet for details on VPP requirements.

- For AM263x-CC board revision PROC110E2A, the SBL enables VPP by default to 1.7V before programming the e-fuses.
- For AM263x-CC board revision PROC110E2, please refer to document "AM263x_CC_E2_Modifications_rev1.pdf" as part of the keywriter package.
- For custom hardware, user needs to comprehend how to provide 1.7V to VPP pin for the SoC. in the flow.

2 Setting up the build environment

2.1 Linux Host

2.1.1 Install MCU+ SDK for AM263X

> **Note:**
>
> This step is optional for those who already have MCU+ SDK 8.5 environment installed. Users already using SDK Linux must install MCU+SDK in order to use keywriter.

Keywriter example app works as an addon on MCU+ SDK for AM263X. It requires driver libraries from MCU+ SDK.

1. Download and install latest MCU+ SDK release[4]for AM263X from https://www.ti.com/tool/MCU-PLUS-SDK-AM263X#downloads.
   - Select "MCU PLUS SDK Linux Installer" for Linux hosts.
2. Download and install following dependent packages from the same link.
   - CCS 12.1.0[5]
   - SYSCONFIG 1.14.0[6]
3. Ideally above tools can be installed anywhere in the system, but we recommend it to install it at default location : ~/ti/ . We will be calling this location as <MCU_PLUS_SDK_INSTALL_DIR>.

2.1.2 Install keywriter addon package

1. Download keywriter addon package (.run file) from mySecureSW[7].
2. Install addon package at location `<MCU_PLUS_SDK_INSTALL_DIR>/source/security` . It will create a folder named "tifs" upon successful installation.

2.1.3 Build Keywriter Certificates

> This section provides a sample certificate generation process taking MSV as an example. For detailed documentation on how to generate various field certificates, see section 3.

Make sure openssl and python3[8] are installed on your system before following below steps.

1. Go to directory `<MCU_PLUS_SDK_INSTALL_DIR>/source/security/tifs/sbl_keywriter/scripts/cert_gen/am263x/`
2. Run : `./gen_keywr_cert.sh --msv 0x1E22D -t tifek/SR_11/ti_fek_public.pem`

4 https://dr-download.ti.com/software-development/software-development-kit-sdk/MD-r5FY9rRaGv/08.05.00.24/mcu_plus_sdk_am263x_08_05_00_24-linux-x64-installer.run
5 https://www.ti.com/tool/CCSTUDIO
6 https://dr-download.ti.com/software-development/ide-configuration-compiler-or-debugger/MD-nsUM6f7Vvb/1.14.0.2667/sysconfig-1.14.0_2667-setup.exe
7 https://www.ti.com/licreg/docs/swlicexportcontrol.tsp?form_id=337487&prod_no=AM263X-RESTRICTED-SECURITY&ref_url=EP-proc-Sitara-MCU
8 https://software-dl.ti.com/mcu-plus-sdk/esd/AM263X/08_05_00_24/exports/docs/api_guide_am263x/TOOLS.html#autotoc_md885

a. This will generate certificate with MSV certificate data at `<MCU_PLUS_SDK_INSTALL_DIR>/source/security/tifs/sbl_keywriter/scripts/cert_gen/x509cert/final_certificate.bin and also converts the certificate to C header file at <MCU_PLUS_SDK_INSTALL_DIR>/source/security/tifs/sbl_keywriter/keywr_bin/am263x/SR_11/Cust_KeysCert.h`

2.1.4 Build the example:

1. Go to directory: `<MCU_PLUS_SDK_INSTALL_DIR>/source/security/tifs/sbl_keywriter/am263x/r5fss0-0_nortos/ti-arm-clang`
2. Clean the example: `make -sj clean PROFILE=debug` `PROFILE can be either debug or release.`
3. Run: `make -sj PROFILE=debug`
4. This will build the example and generate sbl_keywriter.debug.tiimage in the same location.
5. Use sbl_keywriter.debug.tiimage to boot application using supported boot modes. (Refer section 4)

2.2 Windows Host

2.2.1 Install MCU+ SDK for AM263X

> **Note:**
>
> This step is optional for those who already have MCU+ SDK 8.5 environment installed. Users already using SDK Linux must install MCU+SDK in order to use keywriter.

Keywriter example app works as an addon on MCU+ SDK for AM263X. It requires driver libraries from MCU+ SDK.

1. Download and install latest MCU+ SDK release[9] for AM263X from https://www.ti.com/tool/MCU-PLUS-SDK-AM263X#downloads.
   - Select "MCU PLUS SDK Windows Installer" for Windows hosts.
2. Download and install following dependent packages
   - CCS 12.1.0[10]
   - SYSCONFIG 1.14.0[11]
3. Ideally above tools can be installed anywhere in the system, but we recommend it to install it at default location : c/ti/. We we be calling this location as <MCU_PLUS_SDK_INSTALL_DIR>.

9 https://dr-download.ti.com/software-development/software-development-kit-sdk/MD-r5FY9rRaGv/08.05.00.24/mcu_plus_sdk_am263x_08_05_00_24-windows-x64-installer.exe
10 https://www.ti.com/tool/CCSTUDIO
11 https://dr-download.ti.com/software-development/ide-configuration-compiler-or-debugger/MD-nsUM6f7Vvb/1.14.0.2667/sysconfig-1.14.0_2667-setup.exe

2.2.2 Install keywriter addon package

1. Download keywriter addon package (.exe/.run file) from mySecureSW[12].
2. Install addon package and copy "tifs" folder at location `<MCU_PLUS_SDK_INSTALL_DIR>/source/security` .

2.2.3 Prerequisites for windows

- Download and install **git bash** terminal for windows from the link - https://git-scm.com/download/win
- Set up **gmake** by referring MCU+SDK documentation section[13] : Developer Guides → Using SDK with Makefiles → Enabling "make" in Windows.

2.2.4 Build Keywriter Certificates

> ⚠ This section provides a sample certificate generation process taking MSV as an example. For detailed documentation on how to generate various field certificates, see section 3.

Make sure openssl and python3 installed on your system before following below steps

Run below commands in **git bash** terminal running on your windows PC.

1. Go to directory `<MCU_PLUS_SDK_INSTALL_DIR>/source/security/tifs/sbl_keywriter/scripts/cert_gen/am263x/`
2. Run : `./gen_keywr_cert.sh --msv 0x1E22D -t tifek/SR_11/ti_fek_public.pem`
   a. This will generate certificate with MSV certificate data at `<MCU_PLUS_SDK_INSTALL_DIR>/source/security/tifs/sbl_keywriter/scripts/x509cert/final_certificate.bin and also converts the certificate to C header file at <MCU_PLUS_SDK_INSTALL_DIR>/source/security/tifs/sbl_keywriter/keywr_bin/am263x/SR_11/Cust_KeysCert.h`

2.2.5 Build the example:

1. Go to directory: `<MCU_PLUS_SDK_INSTALL_DIR>/source/security/tifs/sbl_keywriter/am263x/r5fss0-0_nortos/ti-arm-clang`
2. Clean the example: `gmake -sj clean PROFILE=debug`
3. Run: `gmake -sj PROFILE=debug`
   `PROFILE can be either debug or release.`

12 https://www.ti.com/licreg/docs/swlicexportcontrol.tsp?form_id=337487&prod_no=AM263X-RESTRICTED-SECURITY&ref_url=EP-proc-Sitara-MCU

13 https://software-dl.ti.com/mcu-plus-sdk/esd/AM64X/08_02_00_08/exports/docs/api_guide_am64x/MAKEFILE_BUILD_PAGE.html#autotoc_md175

4. This will build the example and generate sbl_keywriter.debug.tiimage in the same location.
5. Use sbl_keywriter.debug.tiimage to boot application using supported boot modes. (Refer section 4)

3 Keywriter Certificate Generation

invoke gen_keywr_cert.sh with -h option to see help.
./gen_keywr_cert.sh -h

This is the master script to generate a x.509 certificate in binary format which should be included in SBL keywriter with different configuration of Key Values, Key Configuration and other OTP fields.

The **scripts** folder has the necessary tools for generating x.509 certificate for eFuse programming. It can be copied to a secure server (customer HSM) where the customer keys are stored, and used to generate the x.509 certificate.

- `./gen_keywr_cert.sh -g` will generate random keys for testing, in `keys/` folder
- `gen_keywr_cert.sh` will also generate SHA-512 hash of SMPK, SMEK, BMPK, and BMEK and store them in `verify_hash.csv`. HSM (M4F) UART logs will also print out SHA-512 hash of the keys. `verify_hash.csv` can be used for quick comparison with HSM (M4F) logs.

3.1 Generating x.509 certificate from customer HSM

The Customer keys are supposed to be private, and not to be distributed out in open. For the scope of this document, **Customer HSM** would mean *the secure server/ system* where the customer intends to generate the x.509 certificate. Once the certificate is generated, since the customer key information is encrypted, it is safe to share the certificate in open without revealing the actual keys.

The customer provided key material consists of both Public Key hashes (xMPKH) and secret/secure symmetric keys (xMEK). In order to maintain security, the symmetric keys are encrypted on the secure server (referred to here as Customer HSM), so that the resulting x.509 certificate can be exported and embedded in the Keywriter binary without exposing the secret keys.

Note

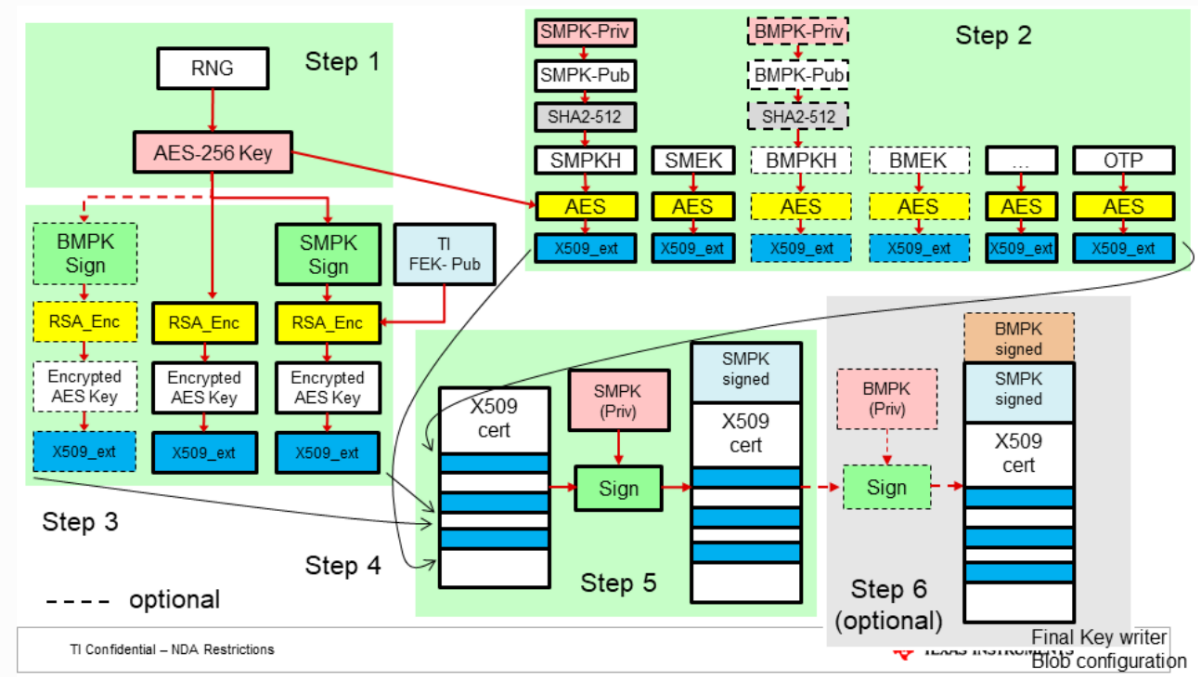
OpenSSL (1.1.1 11 Sep 2018) or later is required for building the OTP Keywriter. Check if OpenSSL is installed by typing “openssl version” at the command prompt. If OpenSSL is not installed, download and install OpenSSL for your OS.

- For Windows : The easiest way is to download and install Strawberry Perl[14] The Strawberry Perl installer automatically installs and sets up OpenSSL. Alternately, users can also use any of these Third Party OpenSSL Distributions[15] for Windows. Refer individual links for instructions on how to setup OpensSSL using a particular distribution.
- For Linux : Execute the command `sudo apt-get install openssl` at the linux command prompt.

Note

OTP keywriter firmware supports a maximum certificate length of 8192 bytes. This length is for individual certificates, and not the `x509cert/final_certificate.bin`. Check size of `primary_cert.bin` and `seconday_cert.bin` (optional depending on BMPK). If certificate length exceeds the supported limit, the keys can be programmed using incremental approach.

14 http://strawberryperl.com/download/5.28.0.1/strawberry-perl-5.28.0.1-64bit.msi
15 https://wiki.openssl.org/index.php/Binaries



1. OEM generates a random 256-bit number to be used as an AES encryption key for protecting the OTP data.
2. The AES-256 key from step 1 is used to encrypt all X509 extension fields, which require encryption protection.
3. Following X509 extensions are created, using TI FEK (public):
   - Encrypting the AES-256 key with TI FEK
   - Signing the AES-256 key with the SMPK, and encrypting that with the TI FEK
   - (optionally, refer step 6) signing the AES-256 key with the BMPK, and encrypting that with the TI FEK
4. All of the extensions from steps 1-3 are combined into a X.509 configuration which is used to generate and sign a certificate with the SMPK.
5. This X509 config is signed using SMPK (priv).
6. (Optional) If the OEM chooses to write BMPK/BMEK fields, X509 config from step 5 needs to be signed using BMPK (priv).

3.2 Reference commands and programming sequence.

Key Writer Firmware can be used in 2 different ways

1) Single Shot Mode

2) Multi Pass Mode

3.2.1 Single Shot Mode: Program via Single Certificate

- In this mode user is expected to generate a single x.509 Certificate with all the required fields
- Key writer firmware will process this certificate and upon successful parsing and verification it will extract all the fields in certificate, program different Keys, key configurations, SWREV, protection and will transition the device from HS-FS to HS-SE

- This is the recommended mode for majority of the use cases
- Below is the example command to program multiple fields at once (in one boot). Certificate given below will program MSV, smpk, smek, bmpk, bmek, key count and key revision at once. After reboot or power on reset device will become HSSE and enforce secure booting.
- Note that while programming everything at once, certificate size should be less than 8192 bytes. If certificate does not fit in size, program incrementally.

The default customer certificate packaged in installer is created with command mentioned below.

```
./gen_keywr_cert.sh -t tifek/SR_11/ti_fek_public.pem --msv 0x1E22D --msv-protect -b
keys_devel/bmpk.pem --bmek keys_devel/bmek.key -b-protect --bmek-protect -s
keys_devel/smpk.pem --smek keys_devel/smek.key -s-protect --smek-protect --sr-sbl 1
 --sr-hsmRT 1 --sr-app 1 --keycnt 2 --keycnt-protect --keyrev 1
```

Note

Attention: There are no separate controls for Read and Write protection. There is only single protect bit to control Read and Write.

3.2.2 Multi Pass Mode: Program via Multiple Certificates

- In this mode user is expected to program the Key Values, Key configuration and other OTP fields via multiple certificates by booting Key writer on device in multiple iterations
- This is an advanced mode and user needs to ensure that care is taken to follow the sequence of the certificates to the Key writer firmware.
- This mode can be used when user wants to program different fields with different certificates, validate the fields and finally transition the device type to HS-SE.
- One crucial requirement in this mode is that the Key Configuration field KEY_REVISION should be the last certificate that Keywriter firmware should process.
- Once KEY_REVISION certificate is processed by Keywriter, device will transition to HS-SE and will stop booting up of Keywriter firmware in next iteration.
- If all the required fields of Keys are not programmed by key writer before programming KEY_REVISION it can permanently damage the device.
- Below are the example commands to program fields in multiple iterations and keywriter can process at one certificate per boot .

Note

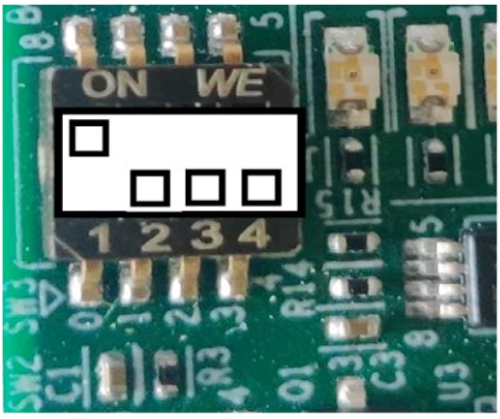
- Each step in below table is a new boot. i.e in each step we program certain fields, build keywriter app freshly with new certificate, reboot the device.
- Until the KEYREV value is set to either 1 or 2, the device is considered as HS-FS and key values can continue to be programmed incrementally. This allows key programming to be done in multiple passes.
- Once the KEYREV is set to 1 or 2, the device becomes HS-SE, and OTP keywriter application will no longer boot as the user root key takes over as root of trust.
- **So programming KEYREV should be done in the last step.**

Command	Description
./gen_keywr_cert.sh --msv 0x1E22D -t tifek/SR_11/ti_fek_public.pem --msv-protect	• Generates certificate with MSV value =0x1E22D for programming • --msv-protect is optional and should be passed only if row needs to be write protected
./gen_keywr_cert.sh -t tifek/SR_11/ti_fek_public.pem --sr-app 1 --sr-sbl 1	• Generates certificate to program software revision sbl = 1 and software revision app = 1
./construct_ext_otp_data.sh -extotp 0x80000001 -indx 0 -size 32 ./gen_keywr_cert.sh -t tifek/SR_11/ti_fek_public.pem --ext-otp ext_otp_data.bin --ext-otp-indx 0 --ext-otp-size 32 --ext-otp-protect 0x0000000000000001	• Generates certificate to program extended OTP with index 0 and size 32 • --ext-otp-protect is optional and should be passed only if row needs to be write protected
./gen_keywr_cert.sh -t tifek/SR_11/ti_fek_public.pem -b am263x/keys_devel/bmpk.pem --bmek /am263x/keys_devel/bmek.key -b-protect --bmek-protect	• Generates certificate to program bmpk and bmek • keys are taken from am263x/keys_devel/(dummy keys). provide path of keys to program your own keys. • It is recommended to Write Protect BMPKH and BMEK Efuses as altering these efuses accidentally can lead to permanent failure of device boot.
./gen_keywr_cert.sh -t tifek/SR_11/ti_fek_public.pem -s am263x/keys_devel/smpk.pem --smek am263x/keys_devel/smek.key -s-protect --smek-protect	• Generates certificate to program smpk and smek • keys are taken from am263x/keys_devel/(dummy keys). provide path of keys to program your own keys. • It is recommended to Write Protect SMPKH and SMEK Efuses as altering these efuses accidentally can lead to permanent failure of device boot.
./gen_keywr_cert.sh -t tifek/SR_11/ti_fek_public.pem --keycnt 2 --keycnt-protect	• Generates certificate to program key count = 2
./gen_keywr_cert.sh -t tifek/SR_11/ti_fek_public.pem --keyrev 1	• Generates certificate to program key revision = 1 • It is **NOT** recommended to Write Protect KEYREV efuses as this field can be incremented during the life time of device to change Root Of Trust. • After programming this device becomes HS-SE and enforces secure booting.

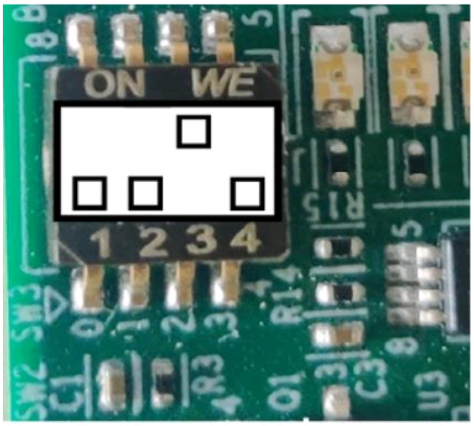
4 Booting and running the keywriter

Keywriter application has been validated with boot modes - UART, QSPI on AM263X control card.

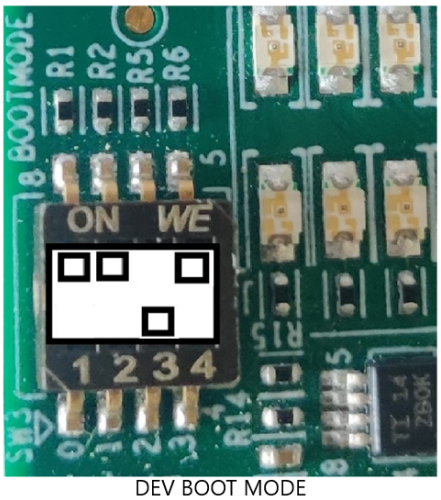
Recommended to go though and understand SOC boot procedures SDK documentation for more details. This section provides the steps for AM263X CC for UART and QSPI boot modes.



BOOTMODE 1:4 (SW3)
UART BOOT MODE



BOOTMODE 1:4 (SW3)
QSPI BOOT MODE



DEV BOOT MODE

After building keywriter application, we get **sbl_keywriter.<PROFILE>.tiimage** binary which can be booted as below.

4.1 UART boot

4.1.1 For Linux users:

Use python script **uart_bootloader.py** provided in `<MCU_PLUS_SDK_INSTALL_DIR>/tools/boot/` directory. Make sure you have necessary dependencies installed to run that python script.

1. Setup up CC for UART boot mode by configuring boot mode switches.
2. Go to directory : `<MCU_PLUS_SDK_INSTALL_DIR>/tools/boot/`

3. Run : `python3 uart_bootloader.py -p "/dev/ttyUSB0" -b <path_to_sbl_keywriter.<PROFILE>.tiimage>`

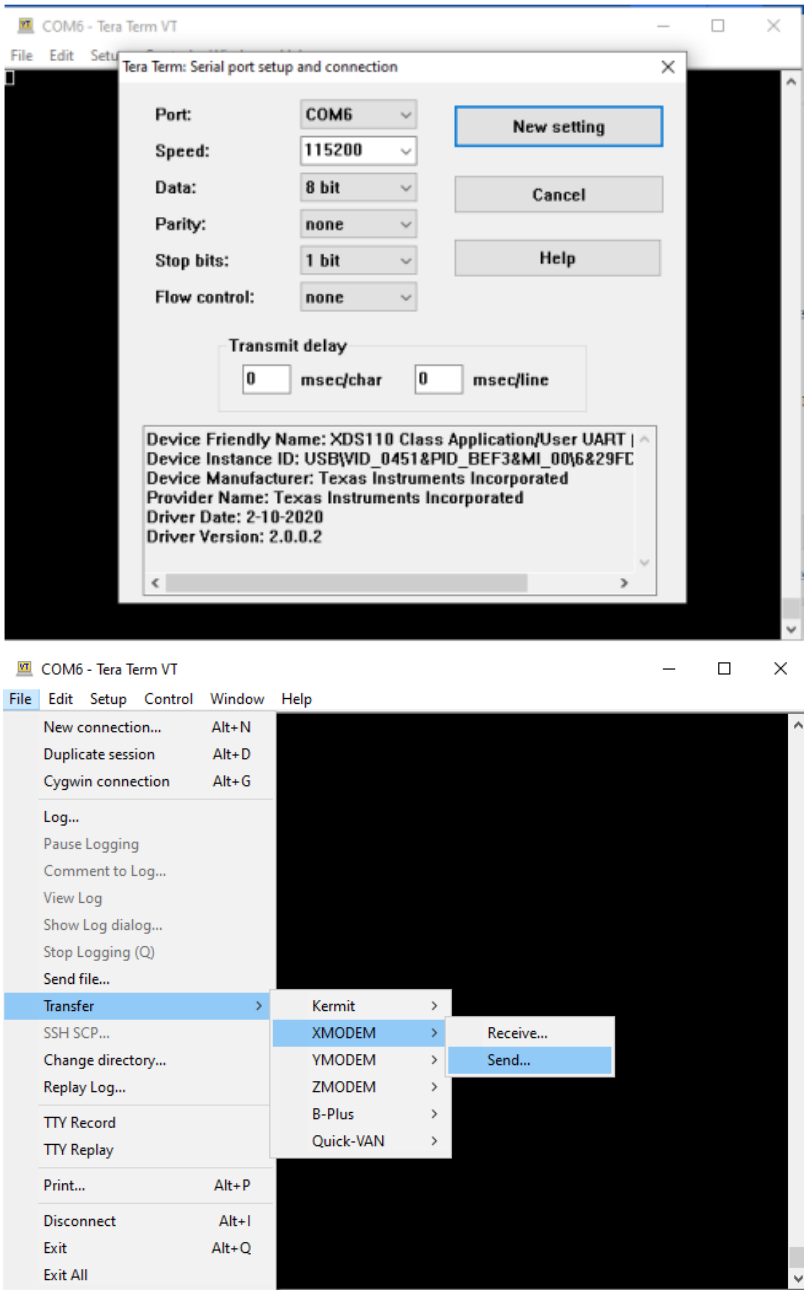
Update -p <port_number> and -b <binary_path> as necessary

4.1.2 For windows users:

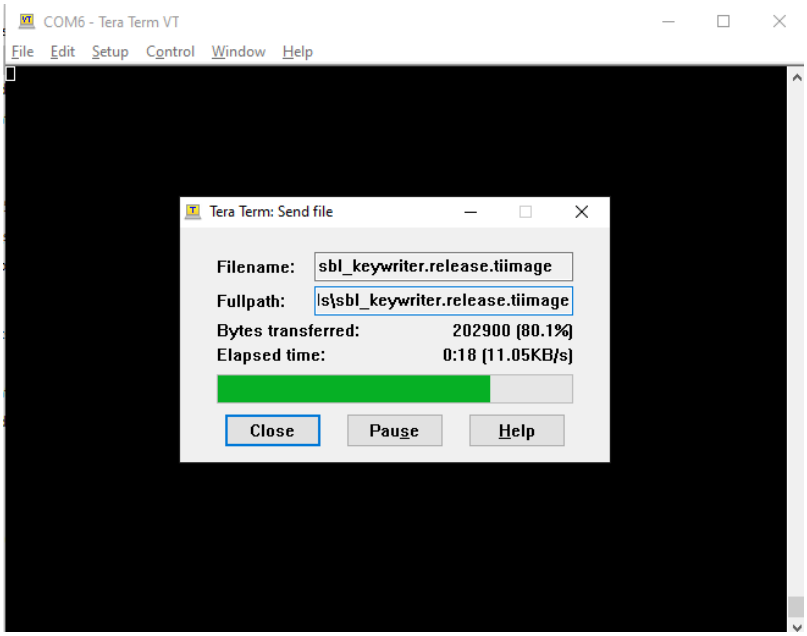
> uart_bootloader.py python script explained above can also be used on windows as long as you have all dependencies installed.

An alternative and quick way on windows is to use Tera Term as shown below.

1. Download and install Tera Term from : https://ttssh2.osdn.jp/index.html.en
2. Set up serial port by selecting Setup → Serial Port . Select appropriate COM port and settings shown in the picture below.
3. Send `sbl_keywriter.<PROFILE>.tiimage` over XMODEM by selecting File → Transfer → XMODEM → Send as shown in the picture below.


COM6 - Tera Term VT
File Edit
Tera Term: Serial port setup and connection
Port: COM6
Speed: 115200
Data: 8 bit
Parity: none
Stop bits: 1 bit
Flow control: none
New setting
Cancel
Help
Transmit delay
0 msec/char
0 msec/line
Device Friendly Name: XDS110 Class Application/User UART
Device Instance ID: USB\VID_0451&PID_BEF3&MI_00\6&29FD
Device Manufacturer: Texas Instruments Incorporated
Provider Name: Texas Instruments Incorporated
Driver Date: 2-10-2020
Driver Version: 2.0.0.2

COM6 - Tera Term VT
File Edit Setup Control Window Help
New connection... Alt+N
Duplicate session Alt+D
Cygwin connection Alt+G
Log...
Pause Logging
Comment to Log...
View Log
Show Log dialog...
Stop Logging (Q)
Send file...
Transfer
SSH SCP...
Change directory...
Replay Log...
TTY Record
TTY Replay
Print... Alt+P
Disconnect Alt+I
Exit Alt+Q
Exit All
Kermit
XMODEM
YMODEM
ZMODEM
B-Plus
Quick-VAN
Receive...
Send...



4.2 QSPI boot

To flash and boot from QSPI flash memory on CC, use provided flashwriter binary from SDK as per the below steps:

1. Set up CC for UART boot mode by configuring boot mode switches.
2. Go to directory: `<MCU_PLUS_SDK_INSTALL_DIR>`
3. Run the following command
   `make -s -C examples/drivers/boot/sbl_uart_uniflash/am263x-cc/r5fss0-0_nortos/ti-arm-clang all`
4. Go to directory: `<MCU_PLUS_SDK_INSTALL_DIR>/home/tools/boot/sbl_prebuilt/am263x-cc/` and edit file `default_sbl_null.cfg` and replace --file path to `<path_to_sbl_keywriter.<PROFILE>.tiimage>`

```
#-------------------------------------------------------------------------#
#                                                                         #
#     DEFAULT CONFIGURATION FILE TO BE USED WITH THE FLASHWRITER SCRIPT    #
#                                                                         #
#-------------------------------------------------------------------------#
#
# By default this config file,
# - points to pre-built flash writer, and SOC init bootloader or SBL NULL bootloader for this EVM
# - The SBL NULL bootloader does below
#   - power-ON and clock setup for the R5F CPUs.
#   - reset and run all the CPUs in a "WFI" loop.
#   - It does NOT boot any applicaiton binary
# - SBL NULL is useful to init the SOC when the EVM is powered ON,
#   so that one can do connect CCS and load application from CCS without needing
#   to run any DMSC load scripts
# - This make the CCS development flow lot simpler as long as this one time flashing step is done.
#

# First point to sbl_uart_uniflash binary, which function's as a server to flash one or more files
--flash-writer=sbl_prebuilt/am263x-cc/sbl_uart_uniflash.release.tiimage

# When sending bootloader make sure to flash at offset 0x0. ROM expects bootloader at offset 0x0
--file=/home/fargo/ti/mcu_plus_sdk/source/security/tifs/sbl_keywriter/am263x/r5fss0-0_nortos/ti-arm-clang/sbl_keywriter.release.tiimage --operation=flash --flash-offset=0x0


```

5. Go to directory : `<MCU_PLUS_SDK_INSTALL_DIR>/tools/boot/ and` Invoke uart_uniflash.py script to flash keywriter binary on flash memory.

- `python3 uart_uniflash.py -p /dev/ttyUSB0 --cfg=` sbl_prebuilt/am263x-cc/default_sbl_null.cfg

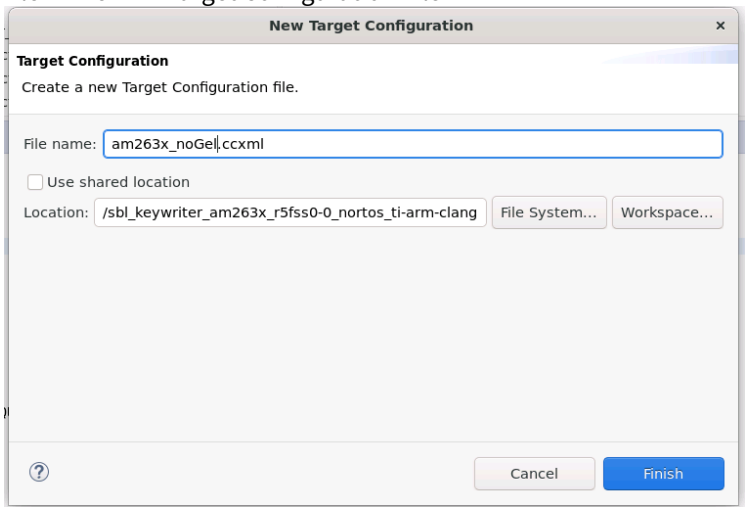
6. Set up CC in QSPI boot mode by configuring boot mode switches .
7. Reset the SOC and boot from QSPI memory.

4.3 DEV boot

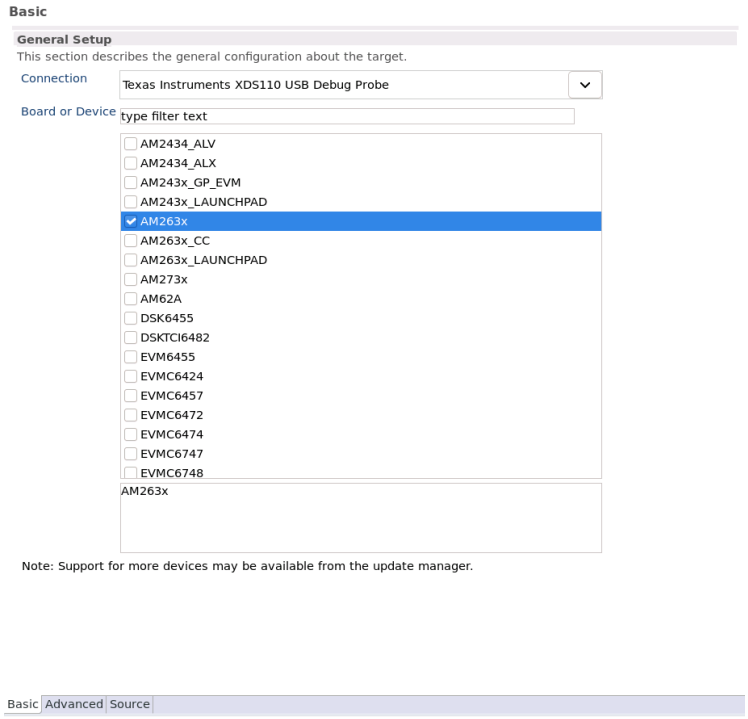
> ⚠ The sbl keywriter build with flag SBL_TEST_CCS_LOAD=1 is not compatible with UART boot and QSPI boot flow and is only compatible with DEV boot flow.

1. Set up CC for DEV boot mode by configuring boot mode switches.
2. Go to directory: `<MCU_PLUS_SDK_INSTALL_DIR>/source/security/tifs/sbl_keywriter/am263x-cc/r5fss0-0_nortos/ti-arm-clang`
3. Clean the example: `gmake clean PROFILE=debug`
4. Run: `gmake PROFILE=debug SBL_TEST_CCS_LOAD=1`

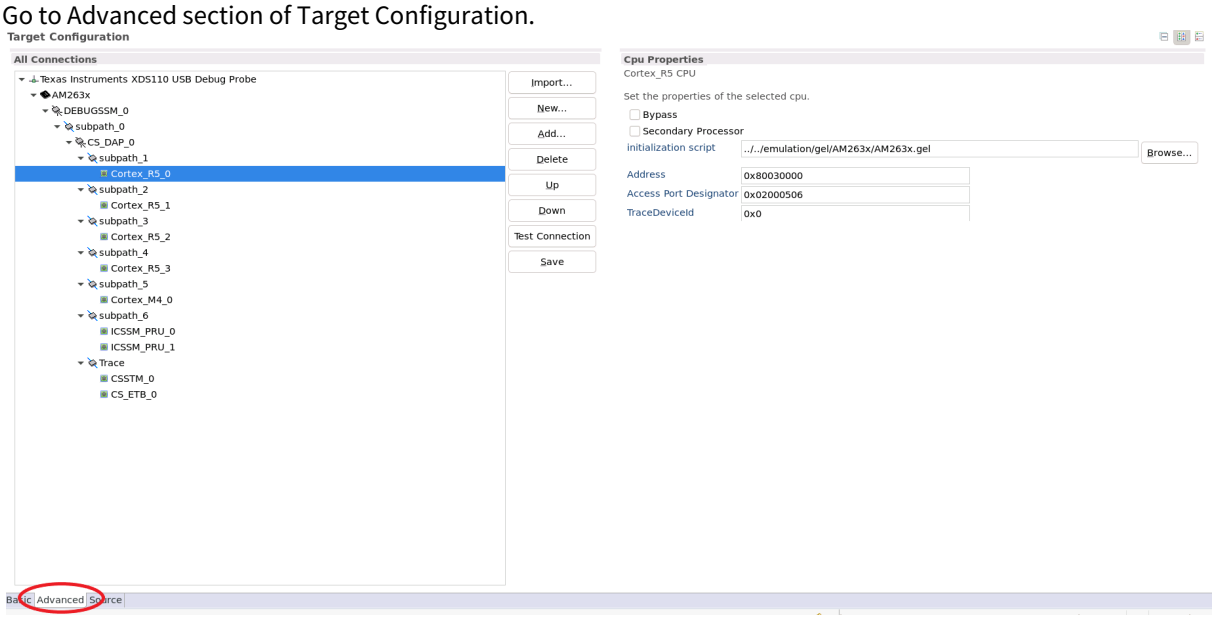
   PROFILE can be either debug or release.
5. This will build the example and generate sbl_keywriter.debug.out in the same location.
6. Open CCS
7. File → New → Target Configuration File



8. Check AM263x in General Setup.

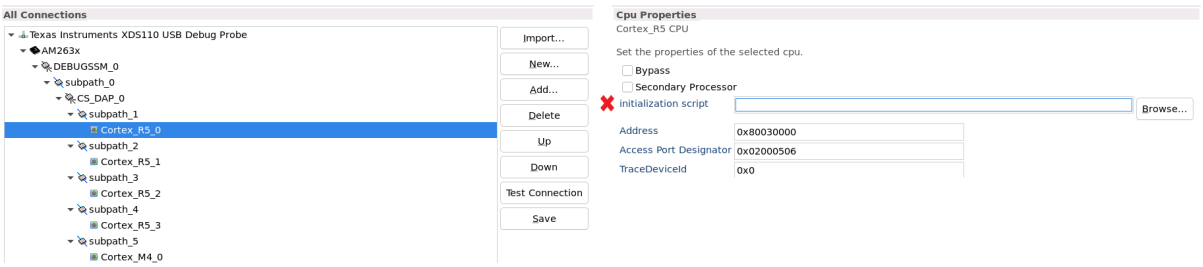


9. Go to Advanced section of Target Configuration.

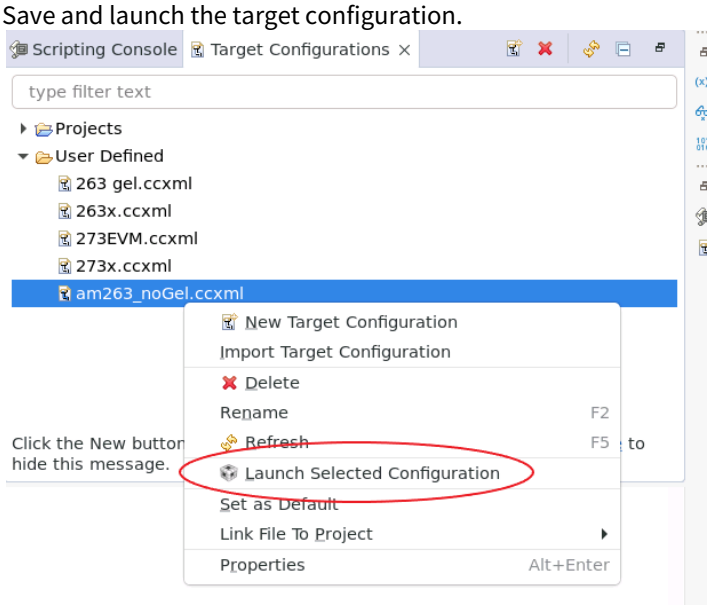


10. Delete initialization script from Cortex_R5_0.

> **Note**
>
> - Make sure no initialization scripts are loaded for any core.

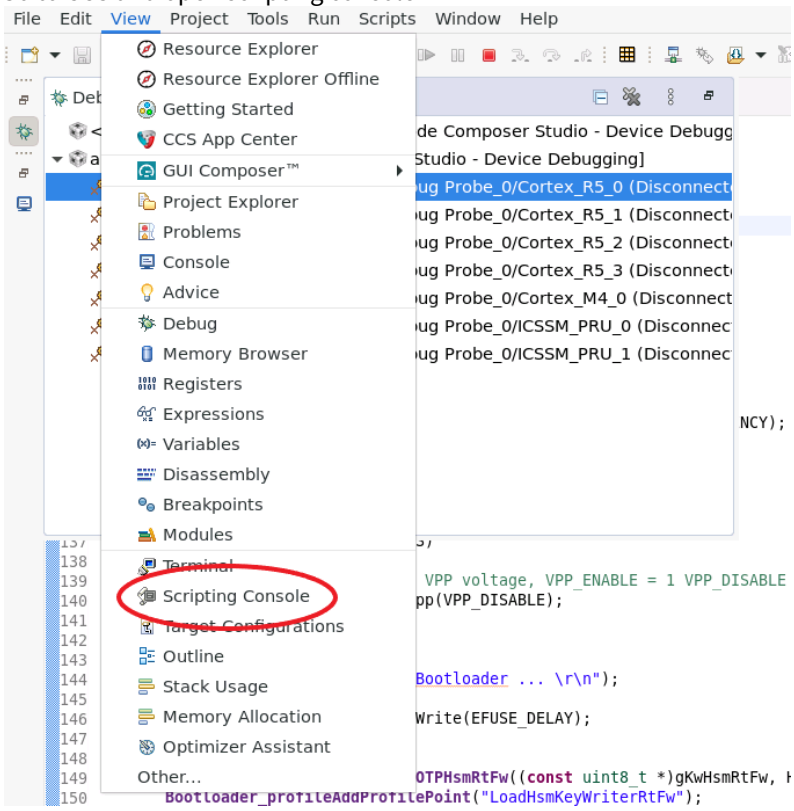


11. Save and launch the target configuration.



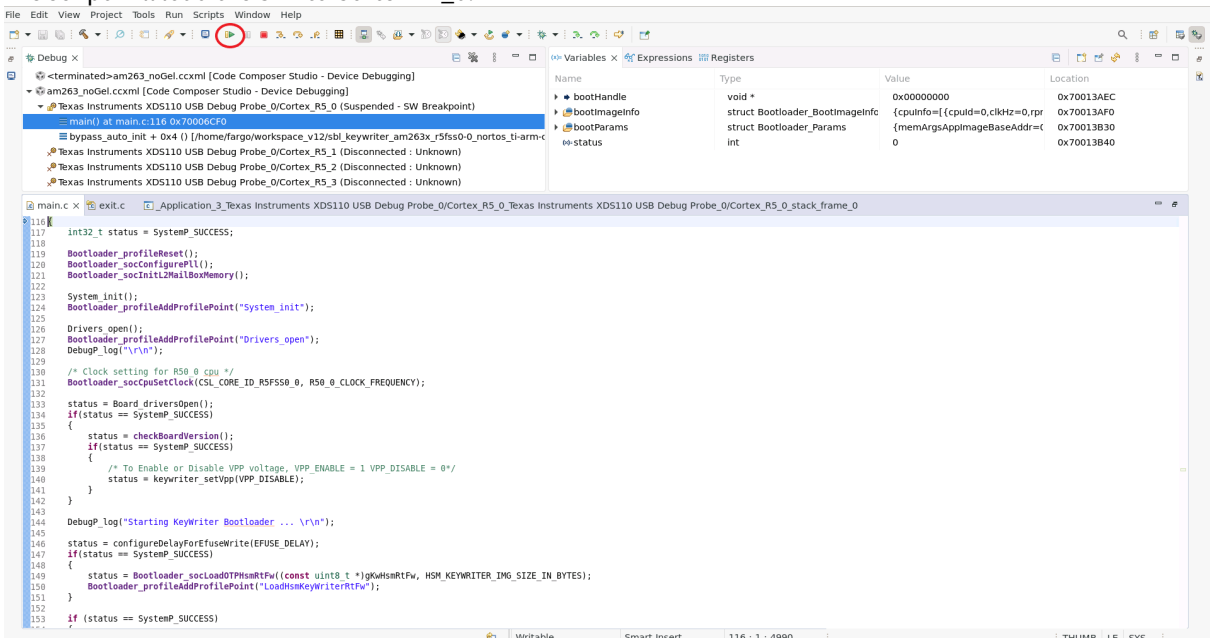
12. Go to directory:
    `<MCU_PLUS_SDK_INSTALL_DIR>/source/security/tifs/sbl_keywriter/tools/ccs_load/`
13. Add path of <MCU_PLUS_SDK_INSTALL_DIR> at line 19 of load_sbl.js

14. Go to CCS and open scripting console



15. Load load_sbl.js via Scripting console.

16. The script will load the SBL to Cortex RF_0.



17.

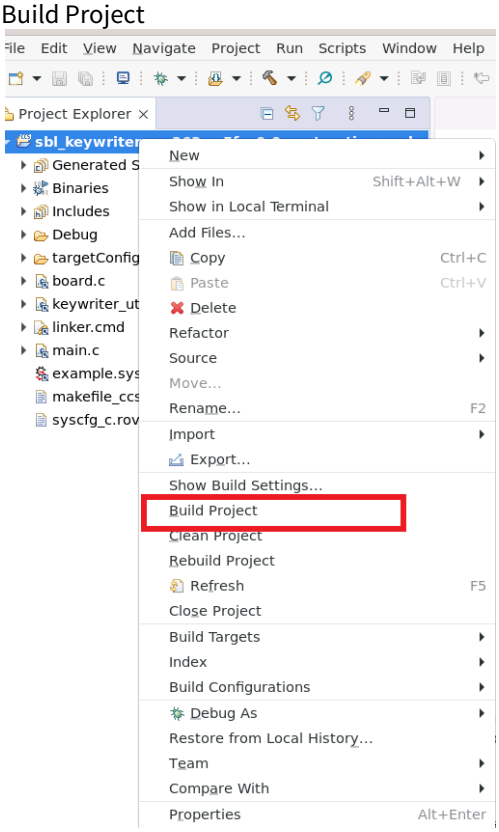
> **Note**
>
> - After execution, logs will be available on **UART0**

Once loaded, one can run or step through the code.

4.3.1 DEV boot via CCS Project

1. Open CCS
2. File → Import → CCS Projects
3. `Select search-directory: <MCU_PLUS_SDK_INSTALL_DIR>/source/security/tifs/sbl_keywriter/am263x/r5fss0-0_nortos/ti-arm-clang`

4. Build Project



5. Once the build is completed, the .out file will be generated in ccs workspace. The location of sbl_keywriter_am263x_r5fss0-0_nortos_ti-arm-clang.out will be replaced in line 45 in load_sbl.js.

```
function makeElfFileName(cpu, os, compiler, exampleName, profile)
{
    // return sdkPath + "/" + examplePath + "/" + board + "/" + cpu + "_" + os + "/" + compiler + "/" + exampleName + "." + profile + ".out"
    return "workspace_v12/sbl_keywriter_am263x_r5fss0-0_nortos_ti-arm-clang/Debug/sbl_keywriter_am263x_r5fss0-0_nortos_ti-arm-clang.out"
}
```

6. Load load_sbl.js via Scripting console.

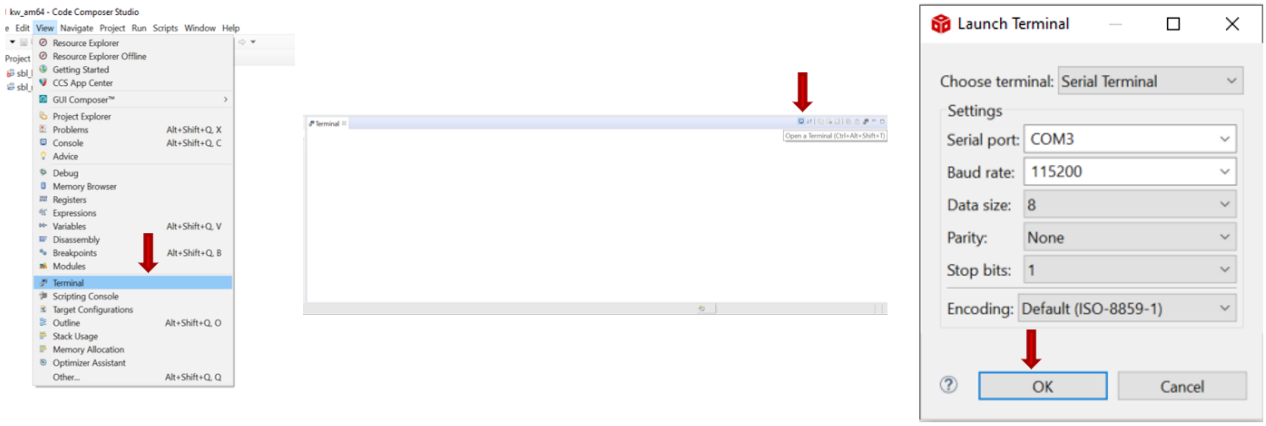
5 Checking device state (HS-FS / HS-SE)

5.1 SOC id parser

Boot ROM reports SoC ID on UART console of the device when UART boot mode is selected. It reports on both GP and HS devices and it provides insights into device configuration which would be helpful for checking if device is converted from

HSFS to HSSE after key programming. **mcup_uart_boot_socid.py** is a python based parser to convert the hexadecimal numbers reported by ROM to human readable text, below are the steps involved to use this parser.

- Set up the SoC in UART boot mode by setting appropriate boot mode switches.
- Open CCS, select view → terminal. It will show terminal window.
- Open launch terminal window and select serial terminal for R5 UART port (COM port on windows and /dev/ttyUSBxx on linux). Leave setting to default and press OK.



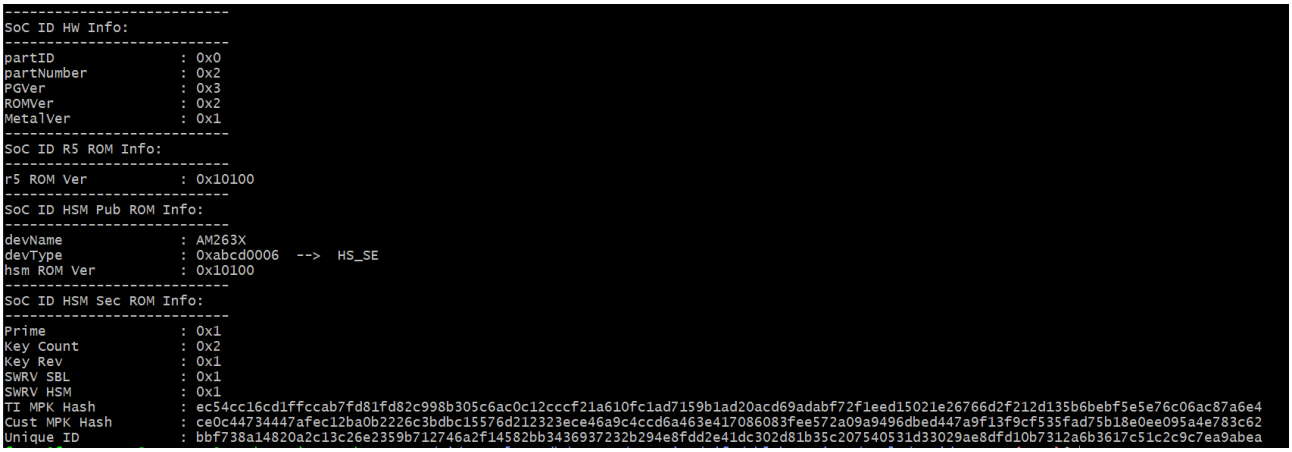
- Reset the SoC by pressing reset button. Terminal will show SOC ID followed by characters CCC (ping characters) on terminal.
- Copy entire string untill CCC (ping characters), and paste in `<MCU_PLUS_SDK_INSTALL_DIR>/source/security/tifs/sbl_keywriter/tools/ socid_parser/socid.txt` file
- `Run below command in the directory <MCU_PLUS_SDK_INSTALL_DIR>/source/security/tifs/sbl_keywriter/tools/ socid_parser`

```
$ python mcup_uart_boot_socid.py socid.txt
```

This will parse SoC ID and display the information as shown below.

Below picture is parsed SOC ID of HSFS device.

While the picture below is parsed SOC ID of HS-FS to HS-SE converted device after programming keys via keywriter application. Values of Cust MPK Hash, Key Count and Key Revision reported on right picture confirms device conversion from HS-FS to HS-SE is successful.



Check HS boot support sections of MCU+ SDK documentation (Developer Guides → Enabling Secure Boot) and boot example/applications on new HSSE device to confirm secure boot on HSSE devices.

5.2 HSSE Functional boot test

> ⚠ If KEYREV is 1 then MPK mentioned below must be SMPK, else if KEYREV is 2 then it should be BMPK.

1. Go to directory: `<MCU_PLUS_SDK_INSTALL_DIR>/source/security/tifs/sbl_keywriter/keywr_bin/am263x/SR_11`
2. Sign hsmRt_HSSEboot.bin with customer's MPK and convert into a header(.h) file with same name and format as hsmRtImg_keywr.h.
3. Go to directory: `<MCU_PLUS_SDK_INSTALL_DIR>/source/security/tifs/sbl_keywriter/am263x/r5fss0-0_nortos/`
4. Uncomment the lines 66-67 and comment line 69-70

```
const uint8_t
gKwHsmRtFw[(HSSE_FUNCTIONAL_TEST_IMG_SIZE_IN_BYTES)]__attribute__((section(
".rodata.hsmrtKr")))
    =HSSE_FUNCTIONAL_TEST_IMG;

/* const uint8_t
gKwHsmRtFw[(HSM_KEYWRITER_IMG_SIZE_IN_BYTES)]__attribute__((section(".rodata.hs
mrtKr")))
    =HSM_KEYWRITER_IMG; */
```

5. Go to directory: `<MCU_PLUS_SDK_INSTALL_DIR>/devconfig/devconfig.mak`
6. In devconfig.mak, replace the path of CUST_MPK for am263x with customer's MPK based on KEYREV.
7. Go to directory: `<MCU_PLUS_SDK_INSTALL_DIR>/source/security/tifs/sbl_keywriter/am263x/r5fss0-0_nortos/ti-arm-clang`
8. Clean the example: `gmake clean PROFILE=debug DEVICE_TYPE=HS PROFILE=debug`
9. `Run:  gmake -sj all DEVICE=am263x DEVICE_TYPE=HS PROFILE=debug`

   PROFILE can be either debug or release.
10. This will build the example and generate sbl_keywriter.debug.hs.tiimage in the same location.
11. Use sbl_keywriter.debug.hs.tiimage to boot application using supported boot modes. (Refer section 4)

6 Debugging

- Following error codes are updated to UART console as part debug information to the user.
- OTP Keywriter error code bit indices

Enum	Value	Description
KEYWR_ERR_DECRYPT_AES256_KEY	0	Error in Decrypting AES256 key randomly generated by customer
KEYWR_ERR_DECRYPT_BMEK	1	Error in Decrypting BMEK extension field
KEYWR_ERR_DECRYPT_BMPKH	2	Error in Decrypting BMPKH extension field
KEYWR_ERR_DECRYPT_SMEK	3	Error in Decrypting SMEK extension field
KEYWR_ERR_DECRYPT_SMPKH	4	Error in Decrypting SMPKH extension field
KEYWR_ERR_INTERAL_OP	5	Internal Operation Error
KEYWR_ERR_INVALID_EXT_COUNT	6	Invalid extension count in x509 certificate. Either SMPKH, SMEK pair or BMPKH, BMEK, SMPKH, SMEK should be used. Any other combination will trigger error
KEYWR_ERR_PARSE_CERT	7	Error in parsing certificate
KEYWR_ERR_PARSE_FEK	8	Error in parsing TI FEK (appended to TIFS binary, before encryption)
KEYWR_ERR_PARSE_SMPK_CERT	9	Error in parsing SMPK signed certificate (certificate that contains customer key data)
KEYWR_ERR_PROGR_BMEK	10	Error in programming BMEK into SoC efuses
KEYWR_ERR_PROGR_BMPKH_PART_1	11	Error in programming BMPKH part 1 into SoC eFuses
KEYWR_ERR_PROGR_BMPKH_PART_2	12	Error in programming BMPKH part 2 into SoC eFuses
KEYWR_ERR_PROGR_KEYCOUNT	13	Error in programming KEY COUNT into SoC eFuses

KEYWR_ERR_PROGR_KEYREV	14	Error in programming KEY REV into SoC eFuses
KEYWR_ERR_PROGR_SMEK	15	Error in programming SMEK into SoC eFuses
KEYWR_ERR_PROGR_SMPKH_PART_1	16	Error in programming SMPKH part 1 into SoC eFuses
KEYWR_ERR_PROGR_SMPKH_PART_2	17	Error in programming SMPKH part 2 into SoC eFuses
KEYWR_ERR_VALIDATION_CERT	18	Error validating certificate
KEYWR_ERR_VALIDATION_SMPK_CERT	19	Error validating SMPK signed certificate
KEYWR_ERR_VALIDATION_BMPK_KEY	20	Error validating BMPK key
KEYWR_ERR_VALIDATION_SMPK_KEY	21	Error validating SMPK key
KEYWR_ERR_WRITE_PROT_KEYCOUNT	22	Error write protecting key count row
KEYWR_ERR_WRITE_PROT_KEYREV	23	Error write protecting key revision row
KEYWR_ERR_IMG_INTEG_SMPK_CERT	24	SMPK signed certificate image integrity failed
KEYWR_ERR_PROGR_MSV	25	Error in programming MSV into SoC eFuses
KEYWR_ERR_PROGR_SWREV	26	Error in programming SWREV into SoC eFuses
KEYWR_ERR_PROGR_FW_CFG_REV	27	Error in programming FW CFG REV into SoC eFuses
KEYWR_ERR_DECRYPT_EXT_OTP	28	Error in Decrypting EXT OTP extension field
KEYWR_ERR_PROGR_EXT_OTP	29	Error in programming EXT OTP extension field

7 Appendix

7.1 Creating x.509 certificates for Incremental Programming

Pass 1

- If user wants to protect row 0 and row 1, we need to set --ext-otp-protect 0x0000000000000003
- If user wants to protect row 0 and row 9, we need to set --ext-otp-protect 0x00000000000003FF
- Each bit in --ext-otp-protect represents a row of extended otp region.

OTP keywriter can program extended otp array, when supplied with an index, and size of bits to program (both index and size have to be muliples of 8).

Constructing extended otp array

```
# Step 1: Generate array ext_otp_data.bin
./construct_ext_otp_data.sh -extotp 0x80000001 -indx 0 -size 32

# Step 2: Edit ext_otp_data.txt for any more changes
vim ext_otp_data.txt

# Step 3: Convert txt to bin file
xxd -r -p ext_otp_data.txt > ext_otp_data.bin

# Setp 4: Generate x.509 certificate
./gen_keywr_cert.sh -t tifek/SR_11/ti_fek_public.pem --ext-otp ext_otp_data.bin
--ext-otp-indx 0 --ext-otp-size 32
```

Note

Steps 2, 3 can be skipped if no changes are needed in extended OTP array
After creating this certificate, building the example keywriter app, and programming ext_otp[0:7], we could also program another offset of ext_otp

Pass 2

```
./construct_ext_otp_data.sh -extotp 0x80000001 -indx 0 -size 32
./gen_keywr_cert.sh -t tifek/SR_11/ti_fek_public.pem --ext-otp ext_otp_data.bin
--ext-otp-indx 0 --ext-otp-size 32 --ext-otp-protect 0x0000000000000001
```

Note

The index and size arguments should be same in both the commands
This will create another certificate. Keywriter example app needs to be built after every change in certificate. This could be repeated many times, but without any overlap in the ext_otp index, size parameters.

Pass 3

Since the KEYREV value is not modified until now, we can still use OTP keywriter to program other keys.

Following code snippet would set MSV, BMPKH, SMPKH, KEYCNT and KEYREV.

> **Note**
>
> It is recommended to write protect SMPKH, SMEK, BMPKH, BMEK efuse rows by providing the specific arguments to `gen_keywr_cert.sh` script.

```
./gen_keywr_cert.sh -s keys_devel/smpk.pem -s-protect --smek keys_devel/smek.key
--smek-protect -t tifek/SR_11/ti_fek_public.pem -a keys_devel/aes256.key --msv
0x1E22D --msv-protect --keycnt 1 --keyrev 1
```

> Important
>
> - KEYCNT should match the number of keys programmed into efuses.
> - KEYREV should always be <= KEYCNT

7.2 OTP Keywriter Logs

UART Log for SMPK/SMEK programming

```
Starting KeyWriter Bootloader ...

# Key programming sequence initiated
INFO: Bootloader_socLoadOTPHsmRtFw:68: Key Writer HSMRT Size in Bytes : 94196
INFO: Bootloader_socLoadOTPHsmRtFw:69: Key Writer hsm runtime firmware load complete
INFO: Bootloader_socLoadOTPHsmRtFw:70: Below Device DETECTED
Device Type : HSFS
SR version 1.1
Current connected device have LDO support ...
[I2C] IO expander found at device address 0x20
[I2C] Setup I/O expander Port 1, bit 3 as output
[I2C]  I/O expander Port 1, bit 3 output set to 1
... VPP Enabled ...

# IPC message for CustCertificate received

#
# Decrypting extensions..
#

 * MPK Options: 0x0

 * MEK Options: 0x0

 * MPK Opt P1 : 0x0
```

```
 * MPK Opt P2 : 0x0

 * MEK Opt    : 0x0

* SMPKH Part 1 BCH code: 0x20f65682

* SMPKH Part 2 BCH code: 0x000f704c

* SMPK Hash (part-1,2):

0xce0c44734447afec12ba0b2226c3bdbc15576d212323ece46a9c4ccd6a463e4100

0x7086083fee572a09a9496dbed447a9f13f9cf535fad75b18e0ee095a4e783c6200

* SMEK Hash:
0x716ded828b2731a19c961aa866ca07fcb99603d52ea1b3fa9d5b23c19e99c60a2ca76a82e34fa0a12cf
c4d7cc5352bb0ec74e313ddd42edfda52832055b662bf

* BMPKH Part 1 BCH code: 0x4006cd64

* BMPKH Part 2 BCH code: 0x806234dd

* BMPK Hash (part-1,2):

0xc090ae52abdb03fee89446518eb34f23c971fe29c0094e52d0a4f303fc8d7c7500

0x6ae38cc1830a7aa8bd5abc2901d76bf7f7780544543f136eb6b67069af7d1bf100

* BMEK Hash:
0x83b1b25a53f4863a27d4e2ddc496771d31a3b131bc74b959eecc3475900ff19ac82c062da18e8cb0b64
6219b379d987392b3df7919c666fd3fa14952552e699e

* EXT OTP Hash:
0xa6b88f45e3a4430c7b2ad907ef32b473a81111e8cb26961e2c0791b090c69f8caadaa388954635872a0
70586bc2db0f640b90853498d0b0567ab73233df75d35



[u32] MSV     : 0xF1E22D


[u32] MSV_BCH : 0x11


[u32] key_cnt : 0x303


[u32] key_REV : 0x101

* SWREV_SBL: 0x0000000100000001
```

```
* SWREV_HSM: 0x0000000100000001


* SWREV_APP: 0x00000000000000000000000100000000000000000000001

* KEYWR VERSION: 0x20000

#
# Programming Keys..
#


* MSV and MSV_BCH:

[u32]  MSV:0x0

[u32]  MSV+BCH:0x0
Programmed 1/1 rows successfully
Programmed 1/1 rows successfully


* SWREV SBL:
bl: 1, r: 1 d: 0x0
bl: 1, r: 2 d: 0x0
Programmed 2/2 rows successfully
bl: 1, r: 1 d: 0x1
bl: 1, r: 2 d: 0x1


* SWREV HSMRT:
bl: 1, r: 3 d: 0x0
bl: 1, r: 4 d: 0x0
Programmed 2/2 rows successfully
bl: 1, r: 3 d: 0x1
bl: 1, r: 4 d: 0x1


* SWREV APP:
bl: 1, r: 5 d: 0x0
bl: 1, r: 6 d: 0x0
bl: 1, r: 7 d: 0x0
bl: 1, r: 8 d: 0x0
bl: 1, r: 9 d: 0x0
bl: 1, r: 10 d: 0x0
Programmed 6/6 rows successfully
bl: 1, r: 5 d: 0x0
bl: 1, r: 6 d: 0x0
bl: 1, r: 7 d: 0x1
bl: 1, r: 8 d: 0x0
bl: 1, r: 9 d: 0x0
bl: 1, r: 10 d: 0x1

* EXT OTP:
```

```
Programmed 1/1 rows successfully


* BMPKH, BMEK:
Programmed 8/8 rows successfully
Programmed 1/1 rows successfully


Programmed 8/8 rows successfully
Programmed 1/1 rows successfully
Programmed 8/8 rows successfully
Programmed 1/1 rows successfully


* SMPKH, SMEK:

Programmed 8/8 rows successfully
Programmed 1/1 rows successfully


Programmed 8/8 rows successfully
Programmed 1/1 rows successfully
Programmed 8/8 rows successfully
Programmed 1/1 rows successfully


* KEYCNT:
[u32] keycnt: 0x0
Programmed 1/1 rows successfully


* KEYREV:
[u32] keyrev: 0x0
Programmed 1/1 rows successfully
[u32] keyrev: 0x101

# Cust_Keys program execution completed
  INFO: Bootloader_LoadOTPCustCert:96: Key Writer Customer key Certificate Size in
Bytes : 7158
INFO: Bootloader_LoadOTPCustCert:97: Key Writer Customer key Certificate load
complete ...
[I2C] IO expander found at device address 0x20
[I2C] Setup I/O expander Port 1, bit 3 as output
[I2C]  I/O expander Port 1, bit 3 output set to 0
... VPP Disabled ...
[BOOTLOADER_PROFILE] Boot Media       : undefined
[BOOTLOADER_PROFILE] Boot Image Size  : 0 KB
[BOOTLOADER_PROFILE] Cores present    :
[BOOTLOADER PROFILE] System_init                      :         94us
[BOOTLOADER PROFILE] Drivers_open                     :         70us
[BOOTLOADER PROFILE] LoadHsmKeyWriterRtFw             :    1035815us
[BOOTLOADER PROFILE] LoadHsmCustomerKeyCertificate    :     675331us
[BOOTLOADER_PROFILE] SBL Total Time Taken             :    1758015us
```

```
KeyWriter Bootloader Execution Complete...
```

7.3 X509 Configuration Template

The following x.509 configuration template is used, by `gen_keywr_cert.sh` file, to create the x509 certificate including customer keys and configuration fields. Each key has an extension field (OID 1.3.6.1.4.1.294.1.64 - 1.3.6.1.4.1.294.1.81), mentioning information such as SHA-512 value, size, IV, RS used in AES encryption.

Details about individual extensions can be obtained form Security x509 Certificate Documentation[16]

```
[ req ]
distinguished_name = req_distinguished_name
x509_extensions = v3_ca
prompt = no
dirstring_type = nobmp

# This information will be filled by the end user.
# The current data is only a place holder.
# Keywriter does not make decisions based
# on the contents of this distinguished name block.
[ req_distinguished_name ]
C = oR
ST = rx
L = gQE843yQV0sag
O = dqhGYAQ2Y4gFfCq0t1yABCYxex9eAxt71f
OU = a87RB35W
CN = x0FSqGTPWbGpuiV
emailAddress = kFp5uGcgWXxcfxi@vsHs9C9qQWGrBs.com

[ v3_ca ]
basicConstraints = CA:true
1.3.6.1.4.1.294.1.64 = ASN1:SEQUENCE:enc_aes_key
1.3.6.1.4.1.294.1.65 = ASN1:SEQUENCE:enc_smpk_signed_aes_key
1.3.6.1.4.1.294.1.66 = ASN1:SEQUENCE:enc_bmpk_signed_aes_key
1.3.6.1.4.1.294.1.67 = ASN1:SEQUENCE:aesenc_smpkh
1.3.6.1.4.1.294.1.68 = ASN1:SEQUENCE:aesenc_smek
1.3.6.1.4.1.294.1.69 = ASN1:SEQUENCE:plain_mpk_options
1.3.6.1.4.1.294.1.70 = ASN1:SEQUENCE:aesenc_bmpkh
1.3.6.1.4.1.294.1.71 = ASN1:SEQUENCE:aesenc_bmek
1.3.6.1.4.1.294.1.72 = ASN1:SEQUENCE:plain_mek_options
1.3.6.1.4.1.294.1.73 = ASN1:SEQUENCE:aesenc_user_otp
1.3.6.1.4.1.294.1.74 = ASN1:SEQUENCE:plain_key_rev
1.3.6.1.4.1.294.1.76 = ASN1:SEQUENCE:plain_msv
1.3.6.1.4.1.294.1.77 = ASN1:SEQUENCE:plain_key_cnt
1.3.6.1.4.1.294.1.78 = ASN1:SEQUENCE:plain_swrev_hsmRT
1.3.6.1.4.1.294.1.79 = ASN1:SEQUENCE:plain_swrev_sbl
1.3.6.1.4.1.294.1.80 = ASN1:SEQUENCE:plain_swrev_sec_app
1.3.6.1.4.1.294.1.81 = ASN1:SEQUENCE:plain_keywr_min_version
```

16 http://downloads.ti.com/tisci/esd/latest/2_tisci_msgs/security/sec_cert_format.html#extensions

```
[ enc_aes_key ]
# Replace PUT-ENC-AES-KEY with acutal Encrypted AES Key
val = FORMAT:HEX,OCT:PUT_ENC_AES_KEY
size = INTEGER:PUT_SIZE_ENC_AES

[ enc_bmpk_signed_aes_key ]
# Replace PUT-ENC-BMPK-SIGNED-AES-KEY with acutal Encrypted BMPK signed AES Key
val = FORMAT:HEX,OCT:PUT_ENC_BMPK_SIGNED_AES_KEY
size = INTEGER:PUT_SIZE_ENC_BMPK_SIGNED_AES

[ enc_smpk_signed_aes_key ]
# Replace PUT-ENC-SMPK-SIGNED-AES-KEY with acutal Encrypted SMPK signed AES Key
val = FORMAT:HEX,OCT:PUT_ENC_SMPK_SIGNED_AES_KEY
size = INTEGER:PUT_SIZE_ENC_SMPK_SIGNED_AES

[ aesenc_smpkh ]
# Replace PUT-AESENC-SPMKH with acutal Encrypted AES Key
val = FORMAT:HEX,OCT:PUT_AESENC_SMPKH
iv = FORMAT:HEX,OCT:PUT_IV_AESENC_SMPKH
rs = FORMAT:HEX,OCT:PUT_RS_AESENC_SMPKH
size = INTEGER:PUT_SIZE_AESENC_SMPKH
action_flags = INTEGER:PUT_ACTFLAG_AESENC_SMPKH

[ aesenc_smek ]
# Replace PUT-AESENC-SMEK with acutal Encrypted AES Key
val = FORMAT:HEX,OCT:PUT_AESENC_SMEK
iv = FORMAT:HEX,OCT:PUT_IV_AESENC_SMEK
rs = FORMAT:HEX,OCT:PUT_RS_AESENC_SMEK
size = INTEGER:PUT_SIZE_AESENC_SMEK
action_flags = INTEGER:PUT_ACTFLAG_AESENC_SMEK

[ aesenc_bmpkh ]
# Replace PUT-AESENC-BMPKH with acutal Encrypted BMPKH
val = FORMAT:HEX,OCT:PUT_AESENC_BMPKH
iv = FORMAT:HEX,OCT:PUT_IV_AESENC_BMPKH
rs = FORMAT:HEX,OCT:PUT_RS_AESENC_BMPKH
size = INTEGER:PUT_SIZE_AESENC_BMPKH
action_flags = INTEGER:PUT_ACTFLAG_AESENC_BMPKH

[ aesenc_bmek ]
# Replace PUT-AESENC-BMEK with acutal Encrypted BMEK
val = FORMAT:HEX,OCT:PUT_AESENC_BMEK
iv = FORMAT:HEX,OCT:PUT_IV_AESENC_BMEK
rs = FORMAT:HEX,OCT:PUT_RS_AESENC_BMEK
size = INTEGER:PUT_SIZE_AESENC_BMEK
action_flags = INTEGER:PUT_ACTFLAG_AESENC_BMEK

[ plain_msv ]
# Replace PUT-PLAIN-MSV with actual MSV value
val = FORMAT:HEX,OCT:PUT_PLAIN_MSV
action_flags = INTEGER:PUT_ACTFLAG_PLAIN_MSV
```

```
[ plain_mpk_options ]
# Replace PUT-PLAIN-MPK-OPT with actual MPK OPT value
val = FORMAT:HEX,OCT:PUT_PLAIN_MPK_OPT
action_flags = INTEGER:PUT_ACTFLAG_PLAIN_MPK_OPT

[ plain_mek_options ]
# Replace PUT-PLAIN-MEK-OPT with actual MEK OPT value
val = FORMAT:HEX,OCT:PUT_PLAIN_MEK_OPT
action_flags = INTEGER:PUT_ACTFLAG_PLAIN_MEK_OPT

[ plain_key_rev ]
# Replace PUT-PLAIN-KEY-REV with actual Key Rev value
val = FORMAT:HEX,OCT:PUT_PLAIN_KEY_REV
action_flags = INTEGER:PUT_ACTFLAG_PLAIN_KEY_REV

[ plain_key_cnt ]
# Replace PUT-PLAIN-KEY-CNT with actual Key Count value
val = FORMAT:HEX,OCT:PUT_PLAIN_KEY_CNT
action_flags = INTEGER:PUT_ACTFLAG_PLAIN_KEY_CNT

[ plain_swrev_hsmRT ]
# Replace PUT-PLAIN-SWREV-HSMRT with actual SWREV HSMRT value
val = FORMAT:HEX,OCT:PUT_PLAIN_SWREV_HSMRT
action_flags = INTEGER:PUT_ACTFLAG_PLAIN_SWREV_HSMRT

[ plain_swrev_sbl ]
# Replace PUT-PLAIN-SWREV-SBL with actual SWREV SBL value
val = FORMAT:HEX,OCT:PUT_PLAIN_SWREV_SBL
action_flags = INTEGER:PUT_ACTFLAG_PLAIN_SWREV_SBL

[ plain_swrev_sec_app ]
# Replace PUT-PLAIN-SWREV-SEC-APP with actual SWREV SEC APP value
val = FORMAT:HEX,OCT:PUT_PLAIN_SWREV_SEC_APP
action_flags = INTEGER:PUT_ACTFLAG_PLAIN_SWREV_SEC_APP

[plain_keywr_min_version ]
# Replace PUT-PLAIN-KEYWR-MIN-VER with actual KEYWR-MIN-VER value
val = FORMAT:HEX,OCT:PUT_PLAIN_KEYWR_MIN_VER

[ aesenc_user_otp ]
# Replace PUT-AESENC-USER-OTP with actual Encrypted OTP
val = FORMAT:HEX,OCT:PUT_AESENC_USER_OTP
iv = FORMAT:HEX,OCT:PUT_IV_AESENC_USER_OTP
rs = FORMAT:HEX,OCT:PUT_RS_AESENC_USER_OTP
wprp = FORMAT:HEX,OCT:PUT_WPRP_AESENC_USER_OTP
index = INTEGER:PUT_INDX_AESENC_USER_OTP
size = INTEGER:PUT_SIZE_AESENC_USER_OTP
action_flags = INTEGER:PUT_ACTFLAG_AESENC_USER_OTP
```

7.4 Delay configuration in SBL

There is a configurable delay available in SBL code that helps to set the delay between each row write. This option is provided to help with slower VPP replenishment circuits and is expected to be within range of 0 to 1.125 seconds. The default value is 0.

> **Note**
>
> Keywriter validation is done with delay value of 0. For any issues with non-zero values for the delay, please contact TI representative.

The delay is specified in sbl_keywriter/am263x/r5fss0-0_nortos/keywriter_utils.h

```
/* Efuse Delay in milliseconds */
#define EFUSE_DELAY                                    (0U)
```