NXP Semiconductors
Application Notes

Document Number: AN13495
Rev. 0, 03/2022

S32G PKCS Compile and Test Procedure

by: NXP Semiconductors

Contents

1. Introduction

The HSE support for PKCS#11 provides a user-space module that integrates with libp11 to enable communication with HSE when using command line tools, such as OpenSSL and pkcs11-tool. Moreover, the communication is also supported by directly calling libp11 functions.

This application note is developed with reference to the BSP31 Release, so steps may differ for other releases.

2. NXP Linux crypto architecture

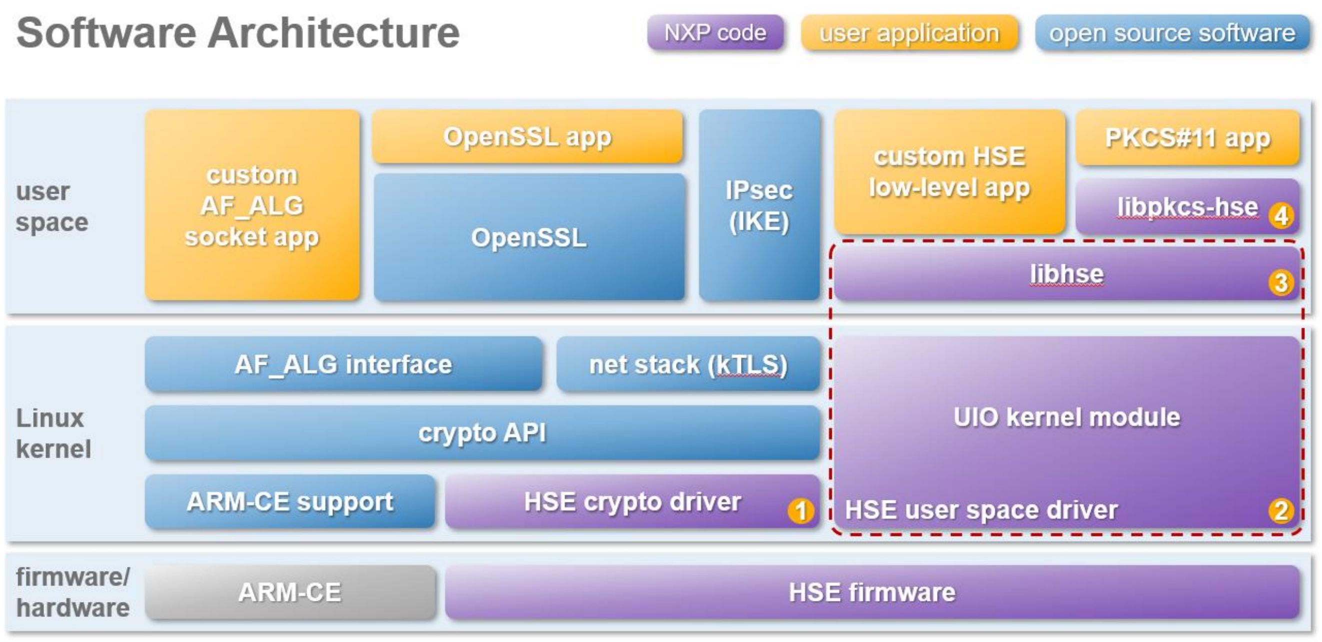


Figure 1. **Linux crypto architecture**

3. Prerequisites

Below Modules needs to be compiled for executing PKCS example in BSP31

1. BSP u-boot
2. BSP Linux kernel
3. pkcs11-hse
4. libp11 0.4.11 (2020-Oct-11)
5. OpenSSL 1.1.1 (2018-Sep-11)
6. OpenSC 0.21.0 (2020-Nov-24)
7. HSE_FW_S32G2_0_1_0_0
8. Install packages
    - sudo apt-get install pcscd libccid libpcsclite-dev libssl-dev libreadline-dev autoconf automake build-essential docbook-xsl xsltproc libtool pkg-config

All the modules are required to be compiled using aarch64 cross compiler and not Yocto based. PKCS11 is not automated as part of Yocto distribution in BSP 31 release.

Cross compiler can be downloaded from the following location.

4. Compilation of required modules for PKCS on Linux host

NOTE

Set the CROSS COMPILE. This is required for cross compiling all the modules going forward

```
#export CROSS_COMPILE= path/to/your/toolchain/dir/bin/aarch64-none-linux-gnu-#export ARCH=arm64
```

4.1. U-boot

The following sub sections describes the steps required to setup and build u-boot bootloader

4.1.1. Downloading the u-boot bootloader source code

There are two ways of obtaining the source for this component, each described below. Choose the one which is appropriate for your situation.

1. Cloning the GIT repository: In a Linux terminal window, type in the following commands

   ```
   #git clone https://source.codeaurora.org/external/autobsps32/u-boot
   #cd u-boot
   #git checkout -b <branch_name> bsp31.0-2020.04
   #ls
   ```

   The contents of the u-boot source code should appear.

2. If you already have a clone of the repository, you can run the following commands in the root directory of the existing repository:

   ```
   #git fetch origin bsp31.0-2020.04
   #git checkout -b <branch_name> bsp31.0-2020.04
   #ls
   ```

   The contents of the u-boot source code should appear.

   <branch_name> is a new branch to be created from the specified tag and can have any value (user choice)

4.1.2. Building the u-boot bootloader

1. In the same Linux terminal window as above, type the following commands

```
#make CROSS_COMPILE=/path/to/your/toolchain/dir/bin/ aarch64-none-linux-gnu-<board>_defconfig
```

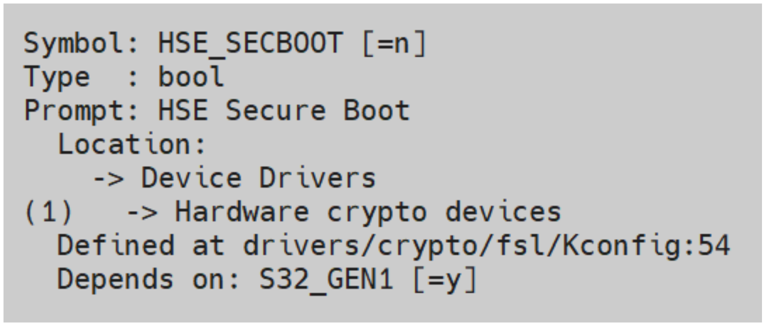
NOTE

If targeting the s32g274ardb, s32g274ardb2 board replace <board> with the board name in the first command above. Use s32g274ardb2_defconfig for RDB boards and s32g2xxaevb_defconfig for s32g274aevb, s32g254aevb and s32g233aevb boards.

2. Enable secure boot from menuconfig

   #make menuconfig

   Search for hse flag and make sure it is enabled. Set to Y as below

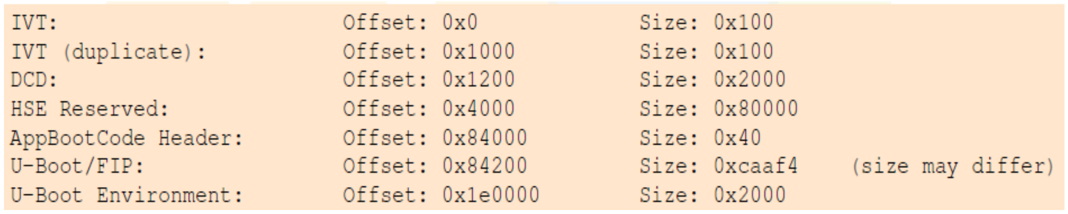


```
Symbol: HSE_SECBOOT [=n]
Type  : bool
Prompt: HSE Secure Boot
  Location:
    -> Device Drivers
(1)   -> Hardware crypto devices
  Defined at drivers/crypto/fsl/Kconfig:54
  Depends on: S32_GEN1 [=y]
```

3. Compile u-boot image

#make CROSS_COMPILE=/path/to/your/toolchain/dir/bin/aarch64-none-linux-gnu-

This command generates the u-boot image with IVT header and Program data (u-boot.s32) that can be written onto the SD. Also, the layout should have sufficient space to store HSE and required associated data on it. The following layout is used



```
IVT:                      Offset: 0x0        Size: 0x100
IVT (duplicate):          Offset: 0x1000     Size: 0x100
DCD:                      Offset: 0x1200     Size: 0x2000
HSE Reserved:             Offset: 0x4000     Size: 0x80000
AppBootCode Header:       Offset: 0x84000    Size: 0x40
U-Boot/FIP:               Offset: 0x84200    Size: 0xcaaf4    (size may differ)
U-Boot Environment:       Offset: 0x1e0000   Size: 0x2000
```

4.2. Linux kernel

The following sub sections shows the steps to set up and build Linux kernel

4.2.1. Downloading the Linux kernel source code

There are two ways of obtaining the source for this component, both of the ways are described below. Choose the one which is appropriate for your situation

1. Cloning the GIT repository: In a Linux terminal window, type in the following commands

   #git clone https://source.codeaurora.org/external/autobsps32/linux

   #cd linux

   #git checkout -b <branch_name> bsp31.0-5.4-rt

   #ls

   The contents of the linux kernel source code should appear here.

2. If you already have a clone of the repository, you can run the following commands in the root directory of the existing repository:

   ```
   # git fetch origin bsp31.0-5.4-rt
   # git checkout -b <branch_name> bsp31.0-5.4-rt
   # ls
   ```

   The contents of the Linux kernel source code appears here. <branch_name> is a new branch to be created from the specified tag and can have any value (user choice)

4.2.2. Building the Linux Kernel

1. In the same linux terminal window as above, type in the following commands.

   ```
   #make ARCH=arm64 CROSS_COMPILE=/path/to/your/toolchain/dir/bin/ aarch64-none-linux-gnu- <soc_name>_defconfig
   ```

   For s32g274aevb, s32g254aevb, s32g233aevb, s32g274ardb and s32g274ardb2, the defconfig is

   s32gen1_defconfig.

2. Linux must be compiled with CONFIG_UIO_NXP_HSE flag enabled. Also, MU can be set to user's choice through CONFIG_UIO_NXP_HSE_MU_ID. The option set can be enabled by running:

   ```
   #make ARCH=arm64 CROSS_COMPILE=/path/to/your/toolchain/dir/bin/ aarch64-none-linux-gnu- menuconfig
   # make ARCH=arm64 CROSS_COMPILE=/path/to/your/toolchain/dir/bin/ aarch64-none-linux-gnu-
   ```

4.3. Cross-compiling HSE PKCS11 module

1. Download HSE Firmware from Flexera, and unzip it on a known path, for example:

   ```
   =$HOME/ HSE_FW_S32G2_0_1_0_0
   #git clone https://source.codeaurora.org/external/autobsps32/pkcs11-hse
   #cd pkcs11-hse/
   #git checkout release/bsp31.0
   #export HSE_FWDIR=/path/to/HSE FW (i.e. HSE_FW_S32G274_0_0_9_0)
   #make
   ```

NOTE

If the compilation fails, open a new terminal, and run the below export commands

```
#export CROSS_COMPILE= path/to/your/toolchain/dir/bin/aarch64-none-linux-gnu-
```

```
#export ARCH=arm64
#make
```

5. Modules compiled for running PKCS example in Linux

5.1. Cross-compiling OpenSSL 1.1.1 for aarch64

5.1.1. Download and cross compile Open SSL

```
# wget https://www.openssl.org/source/old/1.1.1/openssl-1.1.1k.tar.gz
# tar -xvzf openssl-1.1.1k.tar.gz
#cd openssl-1.1.1k
#./Configure linux-aarch64 --prefix=$HOME/openssl-aarch64
```

-prefix indicates a directory separate from your host’s file system in which the crosscompiled files are placed. The path provided is an example.

5.1.2. Build Open SSL

```
#make
#sudo make install
```

You can find the compiled files under $HOME/openssl-aarch64

5.2. Cross-compiling LIBP11 for aarch64

5.2.1. Download and cross compile Libp11

```
# wget https://github.com/OpenSC/libp11/releases/download/libp11-0.4.11/libp11-0.4.11.zip
# unzip libp11-0.4.11.zip
# cd libp11-0.4.11
#export CFLAGS="-g -O2 -I $HOME/openssl-aarch64/include"
#./configure --host=aarch64-linux-gnu --prefix=$HOME/libp11-aarch64/ --with-enginesdir=$HOME/libp11-aarch64/ OPENSSL_LIBS="-lcrypto -L$HOME/openssl-aarch64/lib"
```

-prefix indicates a directory separate from your host’s file system in which the crosscompiled files are placed. The path provided is an example.

5.2.2. Build Libp11

```
#make
```

```
#sudo make install
```

5.3. Cross-compiling OpenSC's pkcs11-tool

5.3.1. Download and cross compile OpenSC

```
# wget https://github.com/OpenSC/OpenSC/releases/download/0.21.0/opensc-0.21.0.tar.gz
```

```
# tar -xvzf opensc-0.21.0.tar.gz
```

```
# ./bootstrap
```

```
#./configure --host=aarch64-linux --prefix="$HOME/opensc-aarch64-test" --enable-openssl
CC=aarch64-linux-gnu-gcc LDFLAGS="-g -Wl,-rpath,$HOME/openssl-aarch64/lib"
OPENSSL_LIBS="-lcrypto -L$HOME/openssl-aarch64/lib" OPENSSL_CFLAGS="-I$HOME/openssl-
aarch64/include"
```

-prefix indicates a directory separate from your host's file system in which the crosscompiled files are placed. The path provided is an example.

5.3.2. Build OpenSC

```
#make CFLAGS="-Wno-error=format-truncation"
```

```
#make
```

```
#sudo make install
```

5.4. Example from HSE PKCS11 Module

Compile the included example as part of HSE PKCS11 Module (Cross-compiling HSE PKCS11 module) to verify HSE functionality.

```
#cd examples
```

```
#make OPENSSL_DIR=$HOME/openssl-aarch64 LIBP11_DIR=$HOME/libp11-aarch64
```

Since the application is dynamically linked, both libcrypto and libp11 must be present in the target's file system, (usually) under /usr/lib/. In this case, you can use the files under openssl-aarch64/lib/ and libp11-aarch64/lib/, and copy the libp11.so* and libcrypto.so* files to the target's /usr/lib/ directory.

Afterwards, you can run the example on the target system

```
#./pkcs-keyop /path/to/libpkcs-hse.so
```

6. Prepare a secure u-boot image

Now that all the required images are compiled, make a secure u-boot image. For ease-of-use, the script tools/s32gen1_secboot.sh is provided as part of u-boot to automate the process of signing and writing the signed image to the SD Card

The following steps are required for users executing from windows PC using Cygwin.

Create a folder <example pkcs>

#mkdir pkcs

#cd pkcs

#mkdir tools

#cp u-boot/tools/s32gen1_secboot.sh to tools /*The folders can be created at any path*/

#cp u-boot/u-boot.s32 to pkcs folder created above

Download hse pink binary from Flexera and copy to pkcs folder. Make sure you are in pkcs folder and run the below command

# ./tools/s32gen1_secboot.sh -k keys -d /dev/sdb --hse 32g2xx_hse_fw_0.1.0_0.9.0_pb210331.rev2.pink

Usage: s32gen1_secboot.sh -k <key_path> -d <device> --hse <hse_pink_binary>

-k|--key Full path to key pair directory

NOTE: A new key pair will be created in the specified directory

-d|--dev Full path to device (i.e. /dev/sdb)

--hse Full path to HSE Firmware

-h|--help Display this help section

If running on a Linux machine, the s32gen1_secboot.sh script can be used directly from the u-boot repo, if the HSE pink binary is also on the Linux machine

After the secure boot image is written into the SD or eMMC, copy the files compiled above from the directories pointed as part of prefix option (--prefix=$HOME/) under examples Modules compiled for running PKCS example in Linux. These are the directories where the cross compiled files are created

1. openssl-aarch64/lib/libcrypto.so.1.1
2. opensc-aarch64-test/lib/libopensc.so.7.0.0
3. opensc-aarch64-test/bin/pkcs11-tool
4. libp11-aarch64/lib/libp11.so.3.4.3

Below files are created as part of pkcs11-hse folder

5. pkcs11-hse/libpkcs-hse.so
6. pkcs11-hse/examples/pkcs-keyop
7. kcs11-hse/libhse.so.1.0

7. Boot secure image from SD or eMMC

To use the HSE PKCS driver, the HSE Key Catalog must be formatted before use by using the following U-Boot command:

#hse_keystore_format

```
#reset
```

After completing the boot, to ensure that HSE is running type the following command:

```
# dmesg | grep hse
```

Expected output:

```
[    0.911425] uio_hse 40211000.mu1b: successfully registered device
```

8. PKCS test setup on target

NOTE

Below steps needs be executed when the host is a Windows machine. If the user is writing these files to the sdcard from a Linux machine, they can write them directly to the rootfs and follow from step 7

1. Boot S32G device
2. Make a directory under root

   ```
   #mkdir <directory name, For example, created pkcs>
   ```

3. Mount SD or eMMC

   ```
   #mount /dev/mmcblk0p1 pkcs/
   ```

4. cd to mount directory

   ```
   #cd pkcs
   ```

5. Copy the libraries from SD or eMMC into /usr/lib

   ```
   # cp libcrypto.so.1.1 /usr/lib
   # cp libopensc.so.7.0.0 /usr/lib
   # cp libp11.so.3.4.3 /usr/lib
   #cp libhse.so.1.0 /usr/lib
   # cp pkcs11-hse.so ~/
   # cp pkcs-keyop ~/
   # cp pkcs11-tool ~/
   ```

6. Unmount the mounted directory

   ```
   #umount pkcs
   ```

7. Configure dynamic linker at run-time

   ```
   #ldconfig -l /usr/lib/libp11.so.3.4.3
   #ldconfig -l /usr/lib/libcrypto.so.1.1
   #ldconfig -l /usr/lib/libopensc.so.7.0.0
   #ldconfig -l /usr/lib/libhse.so.1.0
   ```

9. PKCS test execution on target

1. Run the application pkcs-keyop

   ```
   # ./pkcs-keyop /home/root/pkcs11-hse.so
   ```

   Pkcs-keyop application demonstrates the creating and removal of the keys using the PKCS API's called from the library libpkcs-hse.

   ```
   hse: device initialized, status 0x6b20
   [  932.669666] uio_hse 40211000.mu1b: device hse-uio open
   1 slots available
   Using token:
   Manufacturer......: NXP-Semiconductors
   Description.......: NXP-HSE-Slot
   Token label.......: NXP-HSE-Token
   Key pair[  933.305201] uio_hse 40211000.mu1b: device hse-uio released
    stored
   Keys available: 1
   Enumerated key label: HSE Key Pair
   Key removed
   ```

2. Run pkcs11-tool to load RSA keys (public/pair), EC keys (public) and AES keys. The -id switch corresponds to the key's number (00), slot (06) and catalog (01), in hexadecimal, from the HSE Key Catalog. Some examples.

   NOTE

   To run this example. You need RSA Key pair, ec Key pair and aes and can be generated using openSSL.

   ```
   #./pkcs11-tool --module ~/libpkcs-hse.so --write-object /<path>/rsa_keypair.der \
   --type privkey --id 000601 --label "HSE-RSAPRIV-KEY"
   #./pkcs11-tool --module ~/libpkcs-hse.so --write-object /<path>/rsa_keypub.der \
   --type pubkey --id 000701 --label "HSE-RSAPUB-KEY"
   #./pkcs11-tool --module ~/libpkcs-hse.so --write-object /<path>/ec_keypub.der \
   --type pubkey --id 000401 --label "HSE-ECPUB-prime256v1-KEY"
   #./pkcs11-tool --module ~/libpkcs-hse.so --write-object /<path>/aes.key \
   --type secrkey --key-type AES:256 --id 000101 --label "HSE-AES-256-KEY"
   ```

   Below is the displayed message if the key import operation is successful

   ```
   hse: device initialize[ 1293.011266] uio_hse 40211000.mu1b: device hse-uio open
   ```

```
d, status 0x6b20
Using slot 0 with a present token (0x0)
Created private key:
[ 1293.061793] uio_hse 40211000.mu1b: device hse-uio released
  label:      HSE-RSAPRIV-KEY
  ID:         000601
  Usage:      none
  Access:     none
```

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number: AN13495
Rev. 0
03/2022