AN14072

Accessing the HSE Using PKCS#11

Rev. 1.0 — 8 January 2024

Application note
CONFIDENTIAL

Document Information

Information	Content
Keywords	PKCS#11, HSE, S32G
Abstract	This application notes introduces accessing the HSE via the PKCS11 standard API. It shows examples of using OpenSSL and OpenSSL PKCS11 engine to access the HSE based on NXP Linux BSP software.

1 Acronyms

Table 1. Acronyms

Acronym	Description
API	Application Programming Interface
EC	Elliptic Curve
ECC	Elliptic Curve Cryptography
HSE	Hardware Security Engine
HSM	Hardware Security Module
MU	Messaging Unit, the host interface of HSE
NVM	Non-Volatile Memory
PKCS	Public-Key Cryptography Standards
UIO	Userspace I/O
URI	Uniform Resource Identifier

2 Introduction

This application note describes access schemes to the HSE using the PKCS#11 standard API. It shows examples of using OpenSSL and the OpenSSL PKCS11 engine to access the HSE under incorporation of the NXP Linux BSP software.

The Hardware Security Engine (HSE) is the security subsystem that is embedded in NXP S32 Platform Products. It provides cryptographic services to host CPUs and network accelerators among other functionalities. System (and software) designers can use HSE services, for example, secure key management, cryptography acceleration, and secure booting of the system. The host side communicates with the HSE through dedicated HSE-Host interfaces, called “Messaging Units”.

The PKCS#11 standard provides a standard Application Programming Interface (API) for software to access security devices like smart cards and Hardware Security Modules. Typically, these security devices are designed to provide some secure services such as secure key storage and cryptography algorithms acceleration. The standard API is called “Cryptoki”. Cryptoki specifies data types and functions available for applications. It supports functions for key object management and cryptographic operations which uses keys, such as data encryption, decryption, signature generation, and verification operations. See [1] and [2] of Section 7 for a detailed specification of PKCS#11 Cryptoki. More important is that Cryptoki isolates an application from the detailed implementation of security devices. Aim and target is that a Cryptoki application doesn’t need to be changed to interface with a different security device.

2.1 Cryptoki model

Figure 1 shows the simplified Cryptoki model. As defined by Cryptoki, a security device is represented as a so called Token. Cryptoki applications access a token via Slots. A slot corresponds to a device interface. For example, when accessing the HSE using Cryptoki, the token represents the HSE and the slot represents one MU interface of the HSE. A system can have multiple slots, and the application can connect to tokens using any one or all of these slots. Read [3] of Section 7 for further description of the Cryptoki model.

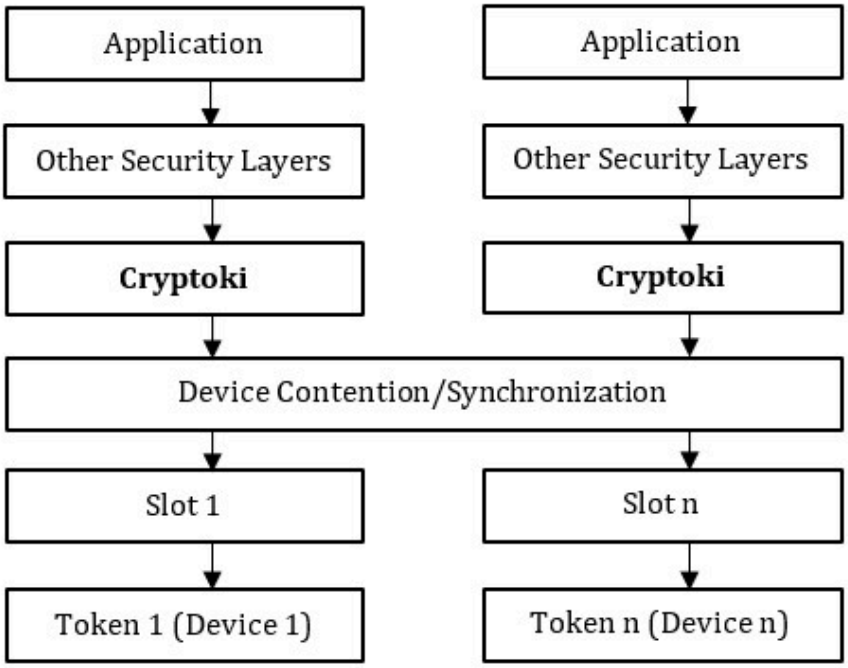

Figure 1. Cryptoki model

The Cryptoki's logical view of a token is a device that stores objects and can perform cryptographic functions. Cryptoki defines three classes of objects: data, certificates, and keys. A key object contains a cryptographic key and its attributes. The key may be a public key, a private key, or a secret key. Usually, a key object is required to perform a cryptographic function. Cryptoki provides object management APIs to create, destroy, or find an object within a token. The object is referenced in an application by using the object handle.

Note: *Data objects and certificate objects are not supported by the current implementation of Cryptoki for the HSE and are out of scope of this Application Note.*

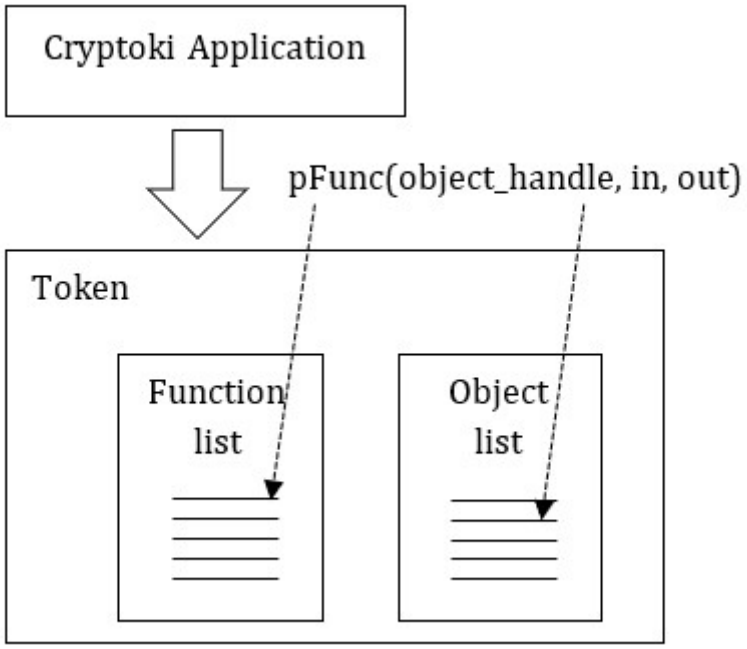

Figure 2. Logical view of a token

2.2 PKCS11-HSE

In the release of S32 Linux BSP software, Cryptoki is implemented to allow easier access to the HSE accelerator from Linux applications. The software is called PKCS11-HSE. The source code structure of PKCS11-HSE is as below:

```
├─examples
├─libhse
└─libpkcs
```

The PKCS11-HSE comprises two libraries and example applications. The LIBHSE is the HSE driver running in userspace of Linux OS. It operates the HSE UIO device that is set up by the Linux HSE UIO driver and maps the HSE interfaces to the userspace. The LIBPKCS is the implementation of Cryptoki on top of LIBHSE. See the user manual of Linux BSP for more information on how to enable PKCS11-HSE.

3 Access HSE using PKCS11-HSE

In this application note, three ways of using PKCS11-HSE for accessing the HSE are introduced, as shown in Figure 3:

1. The application accesses the HSE using the LIBHSE. The application initializes the HSE service descriptor and requests an HSE service via the LIBHSE. All services that the HSE supports are available for application.
2. The application accesses the HSE through Cryptoki.
3. The application accesses the HSE through OpenSSL and OpenSSL PKCS11 engine.

For variant (1), you can find examples in the 'example' folder of PKCS11-HSE. The example names start with ‘hse-’. Variants (2) and (3) are described further in the following chapters.

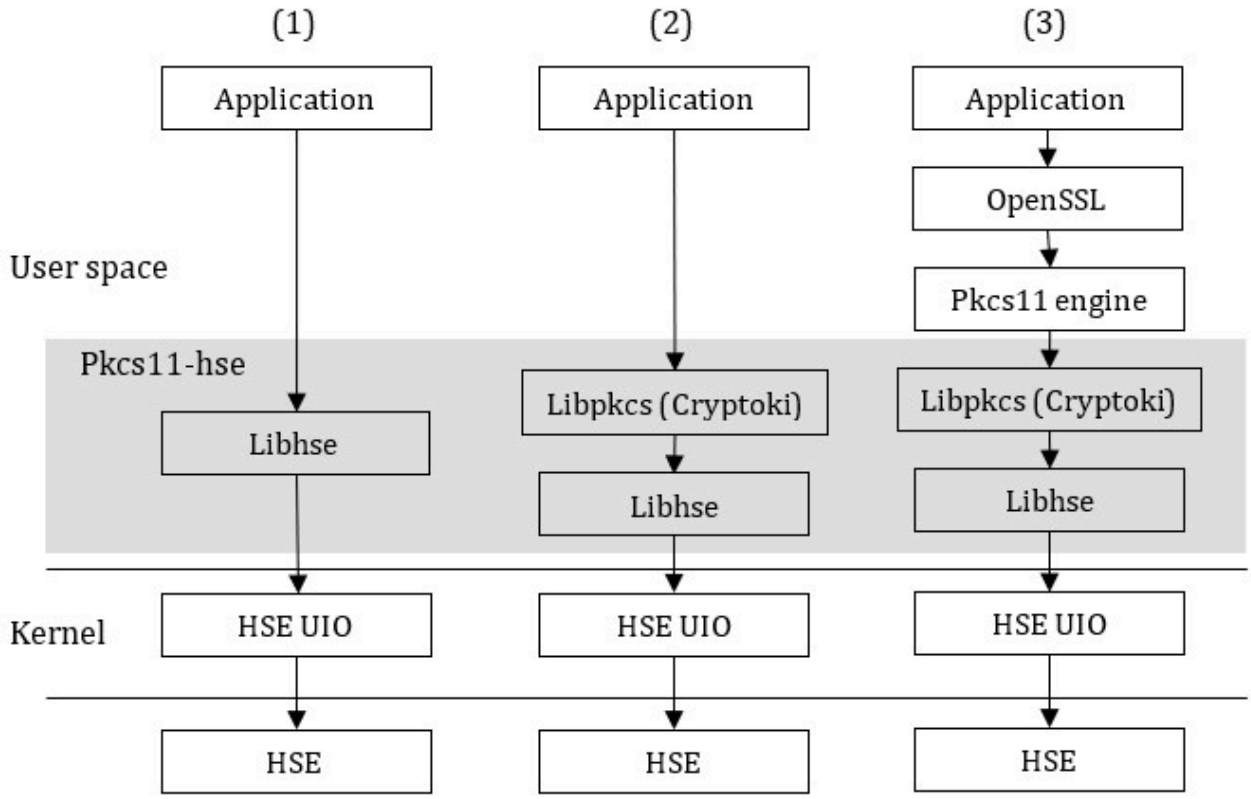

Figure 3. Use cases of PKCS11-HSE

4 Cryptoki application

An application becomes a Cryptoki application by calling the Cryptoki function C_Initialize from one of its threads. Examples can be found in the 'example' folder of PKCS11-HSE, starting with ‘pkcs-’.

Figure 4 shows the typical flow of a Cryptoki application.

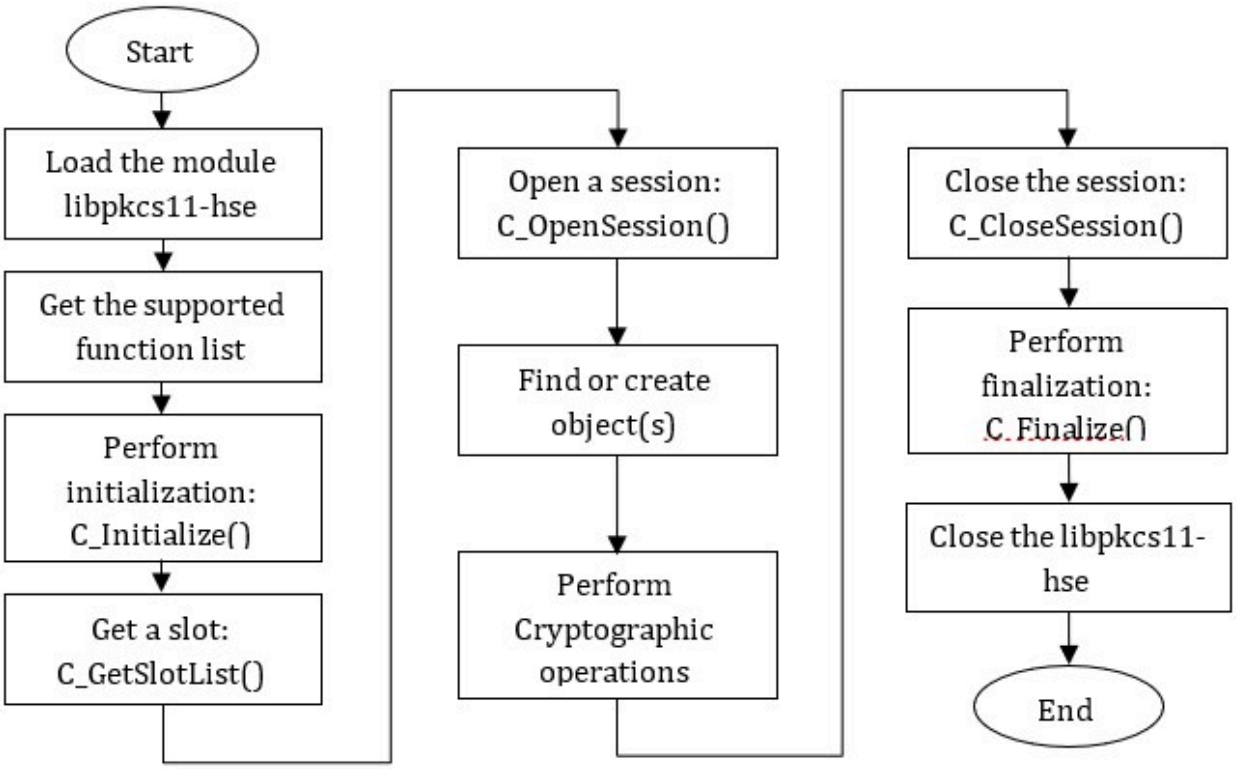

Figure 4. Cryptoki application flow

4.1 Storing objects in PKCS11-HSE, Notes

There are two physical places for for an NVM storage to contain a key object:

1. The file: `/etc/pkcs-hse-objs`. It stores the attributes of all available objects. For example, the label, the key ID, and so on.
2. HSE key slots. The key value of every key object is stored in the HSE key slots. The ID attribute of the object specifies the used key slot.

During initialization of Cryptoki, PKCS11-HSE tries building a list of objects in the memory. If there are no object records in the memory, it builds the object list based on the content of the file `pkcs-hse-objs`. On finalization of Cryptoki, the file `pkcs-hse-objs` gets updated to match to the object list in the memory.

The implemented object management functions update both the object list in the memory and the HSE key slots. For example, when creating a new key object, the new object is appended to the existing objects list and the key value of the new object is installed into the specified HSE key slot. The PKCS11-HSE does not update the HSE SYS_IMG which is stored in external NVM devices (MMC or NOR flash). This should be handled by the application instead.

When finding an object in the token, PKCS11-HSE searches for the object in the object list using the specified attributes as the keyword; for example, the label. It does not check the key availability in the HSE key slot. It assumes that the objects list in the memory and HSE key slots are synchronized.

5 OpenSSL PKCS11 engine

The engine framework of OpenSSL allows to integrate an alternative implementation of cryptographic primitives that the hardware security devices provide. OpenSSL provides engine APIs to implement a customized engine that can operate with a specific security device.

The open source project LIBP11 provides an implementation of the OpenSSL engine on top of Cryptoki. It can interface with the PKCS11-HSE. The following sections explain steps to enable multiple use cases of the OpenSSL PKCS11 engine.

6 Demo setups

In all described demos, the following software packages are used:

- OpenSSL 3.0.8
- OpenSC 0.23.0
- Libp11 0.4.12

Note: *In this Application Note, Linux BSP 38.0 for S32G is used. The board used is NXP-S32G-RDB3 with S32G399A Rev 1.1 silicon chip.*

Note: *In this section, the command line starting with the prompt '$' indicates the command runs on the host PC. The prompt '#' indicates the command runs on the targeting board.*

Note: *All patches and scripts mentioned in this section can be found in the repository: https://github.com/nxp-auto/AN14072-SW.*

6.1 Building OpenSSL and LIBP11

Since the software LIBP11 and OpenSC link to the OpenSSL, the OpenSSL should be compiled at first. By the default configuration of Yocto in Linux BSP 38.0, the OpenSSL version 3.0.8 is used. The same version is used here.

Note: *The host PC is with Ubuntu 20.04. The GCC toolchain used is GCC 11.3.0 for ARM64.*

Clone the OpenSSL repository.

```
$ cd $WORKSPACE
$ git clone https://github.com/openssl/openssl.git
$ cd openssl
$ git checkout openssl-3.0.8
```

Apply the below patch. The patch is a bug fix, which is required when running the TLS demo using the engine. Refer to [4] for more information.

```
$ git apply -v ../patch/openssl/0001-PR-20780-fix-20161.patch
```

Building of OpenSSL.

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

Note: *It's assumed `CROSS_COMPILE` is exported when building all software packages.*

```
$ mkdir ../openssl-aarch64
$ ./Configure linux-aarch64 --prefix=${WORKSPACE}/openssl-aarch64
$ make && sudo make install
```

After the building is finished, you can find the output in the folder `openssl-aarch64`. Clone the LIBP11 repository.

```
$ cd $WORKSPACE
$ git clone https://github.com/OpenSC/libp11.git
$ cd libp11
$ git checkout libp11-0.4.12
```

Apply the below three patches to add features to LIBP11.

1. Adding support of PKCS1 v1.5 encoding scheme for RSA signature algorithm:

```
$ git apply -v \
    ../patch/libp11/0001-Add-PKCS1-v1.5-encoding-for-rsa-sign.patch
```

2. Adding support of AES-128-CBC algorithm for the PKCS11 engine:

```
$ git apply -v \
    ../patch/libp11/0001-engine-support-for-aes-128-cbc-and-cmac.patch
```

3. Adding function of random number generation for the PKCS11 engine:

```
$ git apply -v \
    ../patch/libp11/0001-engine-support-for-random.patch
```

Building of LIBP11.

```
$ sudo apt install pkgconf libssl-dev
$ mkdir ../libp11-aarch64
$ ./bootstrap
$ ./configure \
    CC=${CROSS_COMPILE}gcc \
    --host=aarch64-none-linux-gnu \
    --prefix=$WORKSPACE/libp11-aarch64 \
    --with-enginesdir=$WORKSPACE/libp11-aarch64 \
    OPENSSL_CFLAGS="-I$WORKSPACE/openssl-aarch64/include" \
    LDFLAGS="-L$WORKSPACE/openssl-aarch64/lib"
$ make && sudo make install
```

After successful building of LIBP11, the output is stored in the folder `libp11-aarch64`.

6.2 Building PKCS11-tool

PKCS11-tool is a tool provided by the OpenSC. In the demo, use this tool for key objects management.

First, install the prerequisite packages on the host side.

```
$ sudo apt-get install pcscd libccid libpcsclite-dev \
libssl-dev libreadline-dev autoconf automake build-essential \
docbook-xsl xsltproc libtool pkg-config
```

Clone the OpenSC repository.

```
$ cd $WORKSPACE
$ git clone https://github.com/OpenSC/OpenSC.git
$ cd OpenSC
$ git checkout 0.23.0
```

Apply the two patches listed right after this text and build. The first patch is the bug fix for RSA private key parsing. The second patch is to set a template for the public key part of an ECC private key when creating a key object for the ECC key. This is needed because the HSE requires both public and private keys for importing an ECC private key (the corresponding key type for HSE is an ECC key pair).

```
$ git apply -v \
    ../patch/opensc/0001-fix-rsa-private-key-parser-error.patch
$ git apply -v \
    ../patch/opensc/0001-add-EC_POINTS-for-ECC-private-key-parsing.patch
```

Building of OpenSC.

```
$ mkdir ../opensc-aarch64
$ ./bootstrap
$ ./configure \
    --host=aarch64-linux --disable-strict \
    --prefix="${WORKSPACE}/opensc-aarch64" \
    --enable-openssl CC=${CROSS_COMPILE}gcc \
    LDFLAGS="-g -Wl,-rpath-link,${WORKSPACE}/openssl-aarch64/lib" \
    OPENSSL_LIBS="-lcrypto -L${WORKSPACE}/openssl-aarch64/lib" \
    OPENSSL_CFLAGS=-I${WORKSPACE}/openssl-aarch64/include
$ make && sudo make install
```

After successfully building, the output is stored in the folder `opensc-aarch64`.

6.3 Building PKCS11-HSE

There are two options for how to enable the HSE features and PKCS11-HSE in Linux BSP.

6.3.1 Build PKCS11-HSE using Yocto

The default configuration of Yocto does not build the PKCS11-HSE. To enable it (as an example), update the `conf/local.conf` file in the Yocto build directory with these lines:

```
DISTRO_FEATURES:append = " hse"
NXP_FIRMWARE_LOCAL_DIR = "/path/to/hse/firmware/deliverables"
HSE_VERSION = "0_2_22_0"
HSE_SOC_REV = "rev1.1"
HSE_LIC = "license.rtf"
HSE_LIC_MD5 = "0474bb8a03b7bc0ac59e9331d5be687f"
```

NXP_FIRMWARE_LOCAL_DIR must be set to the folder which contains the HSE firmware deliverables. The name of the folder of HSE firmware deliverables must be in the pattern of HSE_FW_<SoC>_<Version>, for example, HSE_FW_S32G3_0_2_16_1.

Then, you can build the Linux BSP using Yocto per instructions in the Linux BSP user manual.

6.3.2 Manually build PKCS11-HSE

```
$ cd $WORKSPACE
$ git clone https://github.com/nxp-auto-linux/pkcs11-hse.git
$ cd pkcs11-hse
$ git checkout bsp38.0
```

Apply the patch below. The patch adds an example application PKCS-engine. The application is used for AES-128-CBC encryption/decryption and CMAC generation/verification using the engine.

```
$ git apply -v \
    ../patch/pkcs11-hse/0001-openssl-engine-example-for-AES-CBC.patch
```

Build the PKCS11-HSE.

```
$ export HSE_FWDIR=/path/to/hse/firmware/deliverables
$ export LIBP11_DIR=${WORKSPACE}/libp11-aarch64
$ export OPENSSL_DIR=${WORKSPACE}/openssl-aarch64
$ make install
```

After successfully building, libraries and example applications are installed in the `Out` folder.

Note: *In order to use the PKCS11-HSE, the HSE features of the Linux BSP must be enabled. To enable the HSE features manually, the u-boot and the Arm Trusted Firmware (TF-A) needs to be built. Refer to instructions described in the below sections of [6] to build u-boot and TF-A with support of HSE features:*

- *10.5.1.1 Building U-Boot with support for HSE features*
- *10.5.1.2 Building TF-A FIP with support for HSE features*

6.4 Deployment on S32G-VNP-RDB3

Transfer PKCS11-tool and libraries to the target board. It's assumed the board is up and is connected to your local network. The IP address is IP-ADDR. We use the program 'scp' to transfer the file from the host PC to the board.

```
$ cd $WORKSPACE
$ scp opensc-aarch64/bin/pkcs11-tool root@IP-ADDR:/home/root
$ scp opensc-aarch64/lib/libopensc.so.8.1.0 root@IP-ADDR:/usr/lib
```

Transfer LIBP11 LIBS to the target board.

```
$ scp libp11-aarch64/libpkcs11.so root@IP-ADDR:/usr/lib/engines-3
$ scp libp11-aarch64/lib/libp11.so.3.5.0 root@IP-ADDR:/usr/lib
```

Transfer output of PKCS11-HSE to the target board.

```
$ scp -r pkcs11-hse/out root@IP-ADDR:/home/root
```

Deployment on targeting board.

```
# cd ~
# cp out/lib/libhse.so.2.1 /usr/lib
# cp out/lib/libpkcs-hse.so.1.0 /usr/lib
# ldconfig -l /usr/lib/libopensc.so.8.1.0
# ldconfig -l /usr/lib/libp11.so.3.5.0
# ldconfig -l /usr/lib/libhse.so.2.1
# ldconfig -l /usr/lib/libpkcs-hse.so.1.0
```

6.5 Use cases

After deployment of required software, we can now continue to run several use cases using the PKCS11-HSE. The PKCS#11 tool `pkcs11-tool` is used to generate key objects for the token. And then use the OpenSSL and the PKCS11 engine to perform cryptographic operations making use of the installed key objects.

First, generate public and private keys for testing.

```
# cd ~/workspace
# mkdir keys
# cd keys
```

Use the OpenSSL application to generate a RSA-2048 key pair (private and public keys):

```
# openssl genrsa -out rsa2048_private.pem 2048
# openssl rsa \
    -in rsa2048_private.pem \
    -pubout -outform pem \
    -out rsa2048_public.pem
```

Note: *In some case, the length of private exponent of the generated RSA private key (`rsa2048_private.pem`) is 255 bytes. This is not acceptable by the HSE. Use the below command to check the private exponent of the generated RSA private key:*

```
openssl rsa -in rsa2048_private.pem -noout -text
```

Then, check the length of the parameter privateExponent in the output log. If the length of privateExponent is not 256 bytes, try to re-run the above commands to re-generate the RSA-2048 key pair.

Generate an ECC key pair:

```
# openssl ecparam -list_curves
# openssl ecparam -name secp256r1 -genkey -out ecc_private.pem
# openssl ec -in ecc_private.pem -pubout -out ecc_public.pem
```

Generate an AES-128 key with the same key value with the key used in the PKCS-engine example application:

```
# echo -e -n \
    "\x12\x34\x56\x78\x90\xab\xcd\xef\x12\x34\x56\x78\x90\xab\xcd\xef" \
    > aes_128.key
```

Next, modify OpenSSL configuration file for the PKCS11 engine. Back it up before making modifications.

```
# cp /etc/ssl/openssl.cnf /etc/ssl/openssl.cnf.default
# vi /etc/ssl/openssl.cnf
```

Edit the `openssl.cnf` as below:

- In the section `[openssl_init]`, append the below line

```
engines = engine_section
```

- Append the below lines at the end of the file.

```
[engine_section]
pkcs11 = pkcs11_section
[pkcs11_section]
```

```
engine_id = pkcs11
dynamic_path = /usr/lib/engines-3/libpkcs11.so
MODULE_PATH = /usr/lib/libpkcs-hse.so.1
init=0
```

Test the PKCS11 engine. If everything is set up correctly, a similar log to the one provided below should be observed.

```
# openssl engine pkcs11 -t
(pkcs11) pkcs11 engine
     [ available ]
```

To import keys into HSE key catalogs, the first step is to format HSE key catalogs. This can be done using the example application 'hse-secboot'.

```
# ~/out/bin/hse-secboot -f -o -d /dev/mmcblk0
[INFO] Formatting HSE key catalog hse: device initialized, status 0x6b20
[INFO] Retrieving IVT from device /dev/mmcblk0
[INFO] Enabling MUs
[INFO] Formatting NVM and RAM key catalogs
[INFO] Retrieving SYSIMG size
[INFO] Publishing SYSIMG
[INFO] Writing SYSIMG to /dev/mmcblk0
```

Then, try to remove the file '`pkcs-hse-objs`' which is used for storing PKCS11 objects. To ensure both the HSE key catalogs and storage of PKCS#11 objects are cleaned.

```
# rm /etc/pkcs-hse-objs
```

Install the generated RSA private key to the token using the PKCS11-tool. This creates a PKCS#11 key object and import the key into the HSE key slot.

```
# ~/pkcs11-tool \
    --module /usr/lib/libpkcs-hse.so.1 \
    --write-object keys/rsa2048_private.pem \
    --type privkey \
    --id 000601 \
    --label "HSE-RSAPRIV-KEY"
```

Note: *The option '--id 000601' specifies the key slot (i.e. the key handle) used by the HSE to store the key value: key catalog 0x01 (that is, NVM), key group 0x06, key slot 0x00.*

Continue installing other keys:

```
# ~/pkcs11-tool \
    --module /usr/lib/libpkcs-hse.so.1 \
    --write-object keys/rsa2048_public.pem \
    --type pubkey \
    --id 000701 \
    --label "HSE-RSAPUB-KEY"
# ~/pkcs11-tool \
    --module /usr/lib/libpkcs-hse.so.1 \
    --write-object keys/ecc_private.pem \
    --type privkey \
    --id 000301 \
    --label "HSE-ECCPRIV-KEY"
# ~/pkcs11-tool \
    --module /usr/lib/libpkcs-hse.so.1 \
```

```
    --write-object keys/ecc_public.pem \
    --type pubkey \
    --id 000401 \
    --label "HSE-ECCPUB-KEY"
# ~/pkcs11-tool \
    --module /usr/lib/libpkcs-hse.so.1 \
    --write-object keys/aes_128.key \
    --type secrkey \
    --key-type AES:128 \
    --id 000101 \
    --label "HSE-AES-128-KEY"
# ~/pkcs11-tool \
    --module /usr/lib/libpkcs-hse.so.1 \
    --write-object keys/aes_128.key \
    --type secrkey \
    --key-type AES:128 \
    --id 010101 \
    --label "HSE-AES-128TEST"
```

Then, to check the installation, list and check the objects that were installed:

```
# ~/pkcs11-tool \
    --module /usr/lib/libpkcs-hse.so.1 \
    --list-object
```

To delete a key object that was installed before, you can do as in the below example. Both the PKCS11 object and the key in the HSE key slot are deleted.

```
# ~/pkcs11-tool \
    --module /usr/lib/libpkcs-hse.so.1 \
    --delete-object \
    --type secrkey --id 010101
```

Note: *When using the PKCS11-tool to list the installed objects, only the PKCS11 objects are counted. It does not check the key availability in the HSE key slots.*

After key installation, use the installed keys for cryptography operations.

6.5.1 Signature generation and verification

First, generate a plain text file for testing.

```
# cd ~/workspace
# echo "The quick brown fox jumps over the lazy dog" > plain.txt
```

Perform the RSA signature generation and verification with PKCS1 v1.5 encoding scheme.

```
# openssl dgst -engine pkcs11 \
    -keyform engine \
    -sign "pkcs11:token=NXP-HSE-Token;object=HSE-RSAPRIV-KEY" \
    -out rsa.sig -sha512 \
    plain.txt
```

The above command calls for the OpenSSL ‘dgst’ program to generate an RSA signature for the input file. In the low level it calls for the HSE signature generation service to sign the input using the key in the HSE key slot. The key used is specified using the PKCS#11 Uniform Resource Identifier (URI) scheme. See [5] for the specification of the PKCS#11 URI scheme. In the above URI example, the token and the object are

specified by their label (string) each. The used object can also be specified using its ID attribute, for example `"pkcs11:token=NXPHSE-Token;id=%00%06%01"`.

To verify the generated signature, the below command can be used:

```
# openssl dgst -engine pkcs11 \
    -keyform engine \
    -verify "pkcs11:token=NXP-HSE-Token;object=HSE-RSAPUB-KEY" \
    -signature rsa.sig -sha512 \
    plain.txt

hse: device initialized, status 0x6b20 Verified OK
```

Perform the RSA signature generation and verification with PSS encoding scheme.

Signature generation:

```
# openssl dgst -engine pkcs11 \
    -keyform engine \
    -sign "pkcs11:token=NXP-HSE-Token;object=HSE-RSAPRIV-KEY" \
    -sigopt rsa_padding_mode:pss \
    -sigopt rsa_pss_saltlen:20 \
    -sigopt rsa_mgf1_md:sha512 \
    -out rsa.sig -sha512 \
    plain.txt
```

Signature verification:

```
# openssl dgst -engine pkcs11 \
    -keyform engine \
    -verify "pkcs11:token=NXP-HSE-Token;object=HSE-RSAPUB-KEY;type=public" \
    -sigopt rsa_padding_mode:pss \
    -sigopt  rsa_pss_saltlen:20 \
    -sigopt rsa_mgf1_md:sha512 \
    -signature rsa.sig -sha512 \
    plain.txt

Engine "pkcs11" set. hse: device initialized, status 0x6b20
Verified OK
```

ECDSA signature generation and verification:

```
# openssl dgst -engine pkcs11 \
    -keyform engine \
    -sign "pkcs11:token=NXP-HSE-Token;object=HSE-ECCPRIV-KEY" \
    -out ecc.sig -sha512 \
    plain.txt
# openssl dgst -engine pkcs11 \
    -keyform engine \
    -verify "pkcs11:token=NXP-HSE-Token;object=HSE-ECCPUB-KEY" \
    -signature ecc.sig -sha512 \
    plain.txt

Engine "pkcs11" set. hse: device initialized, status 0x6b20
Verified OK
```

6.5.2 RSA encryption and decryption

```
# openssl rsautl \
    -engine pkcs11 \
    -keyform engine \
    -encrypt -pkcs -pubin \
    -inkey "pkcs11:token=NXP-HSE-Token;object=HSE-RSAPUB-KEY;type=public" \
    -in plain.txt \
    -out rsa-cipher.bin
# openssl rsautl \
    -engine pkcs11 \
    -keyform engine \
    -decrypt -pkcs \
    -inkey "pkcs11:token=NXP-HSE-Token;object=HSE-RSAPRIV-KEY;type=private" \
    -in rsa-cipher.bin \
    -out rsa-decrypted.txt
```

Check the decrypted result:

```
# cat rsa-decrypted.txt
The quick brown fox jumps over the lazy dog
```

6.5.3 AES encryption and decryption and CMAC generation and verification

```
# ~/out/bin/pkcs-engine
```

The example application `pkcs-engine` calls for OpenSSL APIs to perform AES-128-CBC and CMAC operations using the pkcs11 engine.

6.5.4 Random number generation

```
# openssl rand -engine pkcs11 -hex 16
```

6.5.5 TLS handshaking

The OpenSSL engine can be used in the TLS handshake processes for signature generation and random number generation. For this case, the patched OpenSSL lib must be used. On the host side, transfer the built libcrypto.so.3 to the board:

```
$ scp openssl-aarch64/lib/libcrypto.so.3 root@IP-ADDR:/tmp
```

Then, copy the lib to replace the current one:

```
# cp /tmp/libcrypto.so.3 /usr/lib
```

Note: *After the lib is replaced, the SSH connection is lost and needs to be reestablished.*

```
# ldconfig -l /usr/lib/libcrypto.so.3
# export OPENSSL_CONF=/etc/ssl/openssl.cnf
```

Note: *After replacing the libcrypto.so, $OPENSSL_CONF must be set in order to use the former configuration.*

Run the below scripts to generate keys and certificates used in the TLS communication.

```
# cd tls-demo/scripts
# export OPENSSL_CONF=/etc/ssl/openssl.cnf.default
# ./tlsCreateCredentialsRunOnce.sh
```

Then, install the generated ECC private key for the TLS client into the HSE.

```
# export OPENSSL_CONF=/etc/ssl/openssl.cnf
# ~/pkcs11-tool \
    --module /usr/lib/libpkcs-hse.so.1 \
    --write-object ../ecc/tls_client_key.pem \
    --type privkey \
    --id 010301 \
    --label "HSE-ECCPRIV-TLS"
```

Open another SSH session connecting to your board. The session runs the TLS server. Start the TLS server by executing the scripts below.

```
# ./tlsServer.sh ECDHE
```

On the client side, start the TLS client and connect to the server (localhost).

```
# ./tlsClient.sh localhost ECDHE
```

7 References

1. PKCS #11 Cryptographic Token Interface Base Specification Version 3.0
2. PKCS #11 Cryptographic Token Interface Current Mechanisms Specification Version 3.0
3. PKCS #11 Cryptographic Token Interface Usage Guide Version 2.40. https://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html
4. OpenSSL 3.0: SSL ECDHE Kex fails when OpenSSL Engine with EC methods is set in the config file https:// github.com/openssl/openssl/issues/20161
5. RFC 7512 The PKCS #11 URI Scheme. https://www.rfc-editor.org/rfc/rfc7512.txt
6. Linux BSP 38.0 User Manual for S32G3 platforms

8 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT

SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

9 Revision history

This table summarizes the revisions to this document.

Table 2. Revision history

Document ID	Release date	Description
AN14072 v.1.0	8 January 2024	Initial release

10 Legal information

10.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

10.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Suitability for use in automotive applications — This NXP product has been qualified for use in automotive applications. If this product is used by customer in the development of, or for incorporation into, products or services (a) used in safety critical applications or (b) in which failure could lead to death, personal injury, or severe physical or environmental damage (such products and services hereinafter referred to as "Critical Applications"), then customer makes the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, safety, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. As such, customer assumes all risk related to use of any products in Critical Applications and NXP and its suppliers shall not be liable for any such use by customer. Accordingly, customer will indemnify and hold NXP harmless from any claims, liabilities, damages and associated costs and expenses (including attorneys' fees) that NXP may incur related to customer's incorporation of any product in a Critical Application.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. - NXP B.V. is not an operating company and it does not distribute or sell products.

10.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Tables

Figures

Contents

For more information, please visit: http://www.nxp.com

Date of release: 8 January 2024
Document identifier: AN14072