AN14471

Using Hardware Security Module for Code Signing

Rev. 1.0 — 4 November 2024

Application note

Document information

Information	Content
Keywords	AN14471, HSM, CST, SPSDK
Abstract	This application note describes how to generate the final signed image using tool CST or spsdk with an HSM.

1 Introduction

NXP i.MX RT processors provide the secure boot feature, which makes the hardware to have a mechanism to ensure that the software can be trusted. The secure boot feature is also known as a high-assurance boot (HAB).

The secure boot feature is based on public key infrastructure. The OEM can use it to make their product reject any system image, which is not authorized to run. For high levels of security, the OEM needs to control access and limit the risk to the sensitive private keys. A hardware security module (HSM) protects their private keys and handles cryptographic operations without exposing private keys.

This application note demonstrates how to generate the final signed image using the Code Signing Tool (CST) or spsdk with an HSM.

This document targets for the i.MX RT1170 hardware platform. The attached package for the i.MX RT1170 hardware platform is used as an example, although these steps can be applied to other i.MX RT platforms.

1.1 CST

The CST provides support to sign and encrypt images for use with high assurance boot (HAB) and advanced high assurance boot (AHAB) enabled NXP processors. The signatures generated by the CST can then be included as part of the end-product software image.

The CST accesses the keys used for signatures locally by default. If a user can run CST locally in the HSM server, we can use the elftosb tool to generate a bootable signed image for i.MX RT devices. The elftosb calls the CST to generate the signatures and pack everything to the boot ROM expects the boot image. For more information, refer to *How to use i.MXRT Security Boot* (document AN12079) or *How to use HAB secure boot in i.MX RT10xx* (document AN12681).

Referring to **Appendix B, Replacing the CST Backend Implementation** of the Code-Signing Tool User's Guide, NXP has architected the Code-Signing Tool in two parts: front-end and back-end. The front-end contains all the NXP proprietary operations, while the back-end containing all standard cryptographic operations. For a back-end replacement to interface with a PKCS#11 enabled HSM, refer to *Using Code-Signing Tool with Hardware Security Module* (document AN12812).

Sometimes, the OEM may want to sign the firmware from an HSM or server. Pass the signature to CST and run CST locally. The CST tool has been slightly modified to allow such an asynchronous operation by extracting the digests to be signed first and embedding the signatures back into the binary in the second step. It demonstrates the steps for this case in Section 2.

Note: *The signature CST tool request is CMS format.*

1.2 SPSDK tool

Secure Provisioning SDK (SPSDK) is a unified, reliable, and easy to use Python SDK library working across the NXP MCU portfolio. It enables connection and communication with a target device for generation of secure bootable files image, security features configuration, and generation and management of cryptographic keys and certificates.

SPSDK allows the users to install plugins and integrate them with SPSDK functionality. A plugin signature provider allows using a custom provider for the authentication instead of keys stored on a local machine. Signature provider requires a custom implementation of an HTTP server with a simple API providing the authentication. SPSDK uses a simple flask REST API service representing the remote HSM machine. It describes the process of setting up signature provider and building an image signed by the signature provider in Section 3. User can refer to the online document for the usage of a signature provider.

In this work, we are using SPSDK v2.1.0.

2 Using CST

This section describes how to change CST to generate the hash of an image, which can be used to generate the signature by HSM and then embedding the CMS signatures back into the binary.

2.1 CST workflow

The original CST workflow is described in Figure 1.

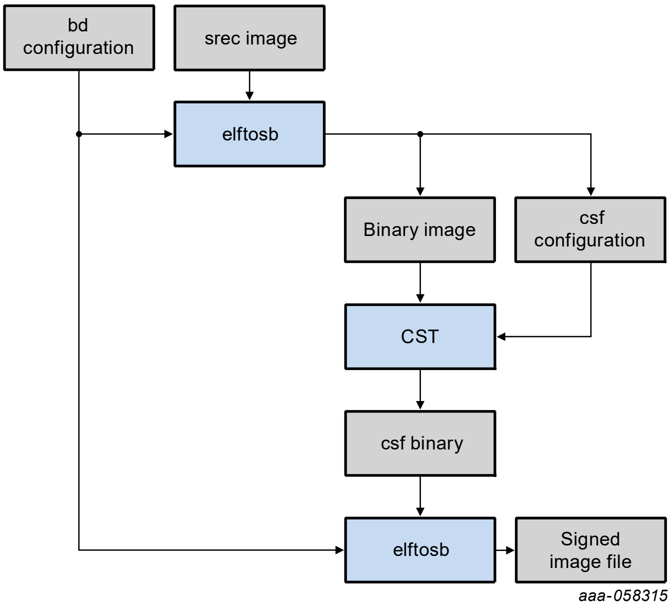


Figure 1. Original CST workflow

The CST tool is slightly modified to extract the digests to be signed first and embedding the signatures back into the binary in a second calling elftosb.

The modified tool workflow is described in Figure 2.

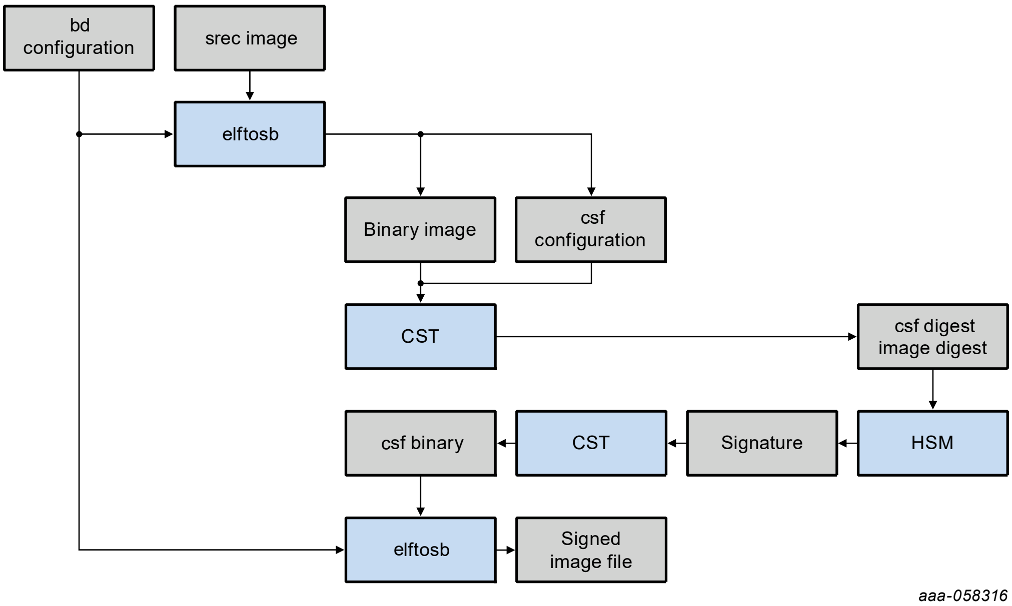


Figure 2. Modified CST workflow

2.2 Get sources

User can search CST and download the Code Signing Tool package from http://www.nxp.com. The latest version is CST v3.4.0.

2.3 Change sources

The CST tool has a front-end supporting the NXP proprietary operations. The back-end perform all cryptographic operations related to digital signature generation and encryption and accesses key material directly in the filesystem. The default CST backend uses OpenSSL to perform signature generation and data encryption.

The default CST backend implementation is located at `cst-3.4.0\code\back_end-ssl\src`. The function `gen_sig_data_cms` in the file `adapt_layer_openssl.c` is used to sign the data or image. Users can slightly modify this function to allow for getting the hash of data to be signed and embedding the signatures back into the final binary.

The modified function `gen_sig_data_cms` is as below:

```
int32_t gen_sig_data_cms(const char *in_file, X509 *cert,
    EVP_PKEY *key, hash_alg_t hash_alg,
    uint8_t *sig_buf, size_t *sig_buf_bytes)
{
    FILE *sig_fp = NULL;
    int size = 0;
    char cms_sig_path[] = "./csf_sig.bin";
    int32_t err_value = CAL_SUCCESS;

    if (strcmp(in_file, "csfsig.bin") == 0)
    {
        FILE *hashFile = NULL;
        uint8_t *hash;
        int hash_bytes = HASH_BYTES_MAX;;
```

```
        hash = OPENSSL_malloc(HASH_BYTES_MAX);
        /* Generate hash of data from in_file */
        err_value = calculate_hash(in_file, hash_alg, hash, &hash_bytes);
        if (err_value != CAL_SUCCESS) {
            return err_value;
        }
 /* Save hash value of CSF data */
        hashFile = fopen("csfhash.bin", "wb");
        fwrite(hash, sizeof(uint8_t), hash_bytes, hashFile);
        fclose(hashFile);
        OPENSSL_free(hash);

        printf("Waiting for signature of CSF\r\n");
    }
    else if (strcmp(in_file, "imgsig.bin") == 0)
    {
        FILE *hashFile = NULL;
        uint8_t *hash;
        int hash_bytes = HASH_BYTES_MAX;

        hash = OPENSSL_malloc(HASH_BYTES_MAX);

        /* Generate hash of data from in_file */
        err_value = calculate_hash(in_file, hash_alg, hash, &hash_bytes);
        if (err_value != CAL_SUCCESS) {
            return err_value;
        }
/* Save image hash value */
        hashFile = fopen("imghash.bin", "wb");
        fwrite(hash, sizeof(uint8_t), hash_bytes, hashFile);
        fclose(hashFile);
        OPENSSL_free(hash);

        strcpy(cms_sig_path,"./img_sig.bin");
        printf("Waiting for signature of image\r\n");
    }
    else
    {
        printf("Unknown in_file!");
    }
    /* Wating for the HSM signature */
    while ((sig_fp = fopen(cms_sig_path, "rb")) == NULL)
    {
        system("pause");
    }

    fseek(sig_fp, 0, SEEK_END);
    size = ftell(sig_fp);
    rewind(sig_fp);
    fread(sig_buf, sizeof(uint8_t), size, sig_fp);
    fclose(sig_fp);

    *sig_buf_bytes = size;

    return CAL_SUCCESS;
```

The parameter `in_file` keeps the data to be signed. The updated function generates the hash value and saves it as a file, then waits for the signature generated by HSM.

2.4 Compile sources

The README in the CST package explains how to create a build environment using the Dockerfile and build the CST source code.

In this work, it creates a build environment using the **MSYS2** console on the Windows platform with the below steps:

1. Download and install MSYS2.
2. Run MSYS2 MINGW32 and install software gcc, bison, and flex with the below commands:
   ```
   pacman -S bison flex mingw-w64-i686-gcc
   ```
3. Change `byacc` to `yacc` in file `cst-3.4.0\code\build\make\init.mk`.
4. Run the below command under the CST root folder to initiate the build process:
   ```
   OSTYPE=mingw32 make install
   ```

The CST makefile downloads, unpack, configure, and build OpenSSL version 3.2.0 in the current directory. User can locate Openssl with environment variable `OPENSSL_PATH` in the next build as below:

```
OSTYPE=mingw32 make install OPENSSL_PATH="./openssl-mingw32"
```

The build result `cst.exe` is located in the directory `cst-3.4.0\build\mingw32\bin`.

2.5 Prepare the secure boot image

A command sequence description file is parsed and processed by the CST application. It generates a binary file containing the command sequence file commands (valid only for HAB), certificates, and signatures, which are interpreted by ROM. The user can use CST directly. Here, we use the elftosb tool instead of CST. The elftosb can generate the description file, then pass this file to CST, and call CST to get the final signed image.

It takes the RT1170 as an example. To generate a signed bootable image using the elftosb, the steps are as below:

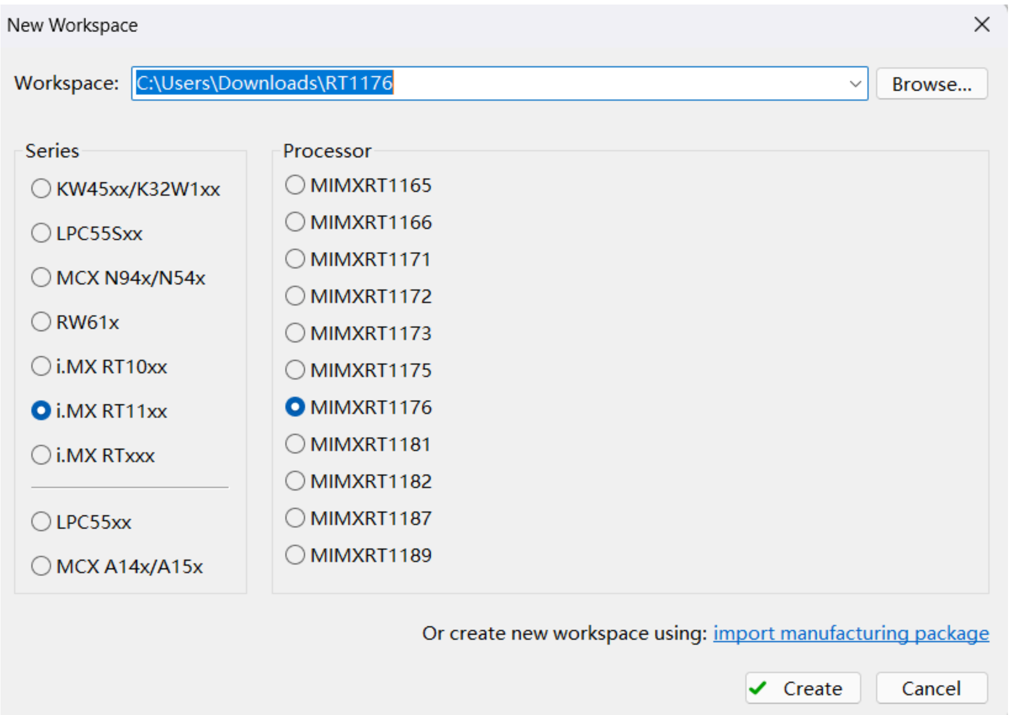
2.5.1 Building application executable

To build an SREC format application executable, refer to section 4.3.1 of *How to use HAB secure boot in i.MX RT10xx* (document AN12681). For the user of MCUXpressoIDE, change the macro `XIP_BOOT_HEADER_ENABLE` to 0 in Settings > Preprocessor in the project. For more details about building an application executable with MCUXpresso, see *Getting Started with MCUXpresso SDK for MIMXRT1170-EVK.pdf* in SDK.

2.5.2 Keys and certificate generation

Users must generate keys and certificates. To generate ecc p256 keys and certificates using the MCUXpresso Secure Provisioning tool, perform the following steps:

1. Retrieve and install the MCUXpresso Secure Provisioning tool from the NXP website.
2. Run this tool, click the button to switch the processor, select MIMXRT11xx. To select a processor from a different family, the user must create a workspace.
   Note: *The user can open the MCUXpresso Secure Provisioning tool with administrator mode. Otherwise, some important material is not generated.*



New Workspace

Workspace: C:\Users\Downloads\RT1176

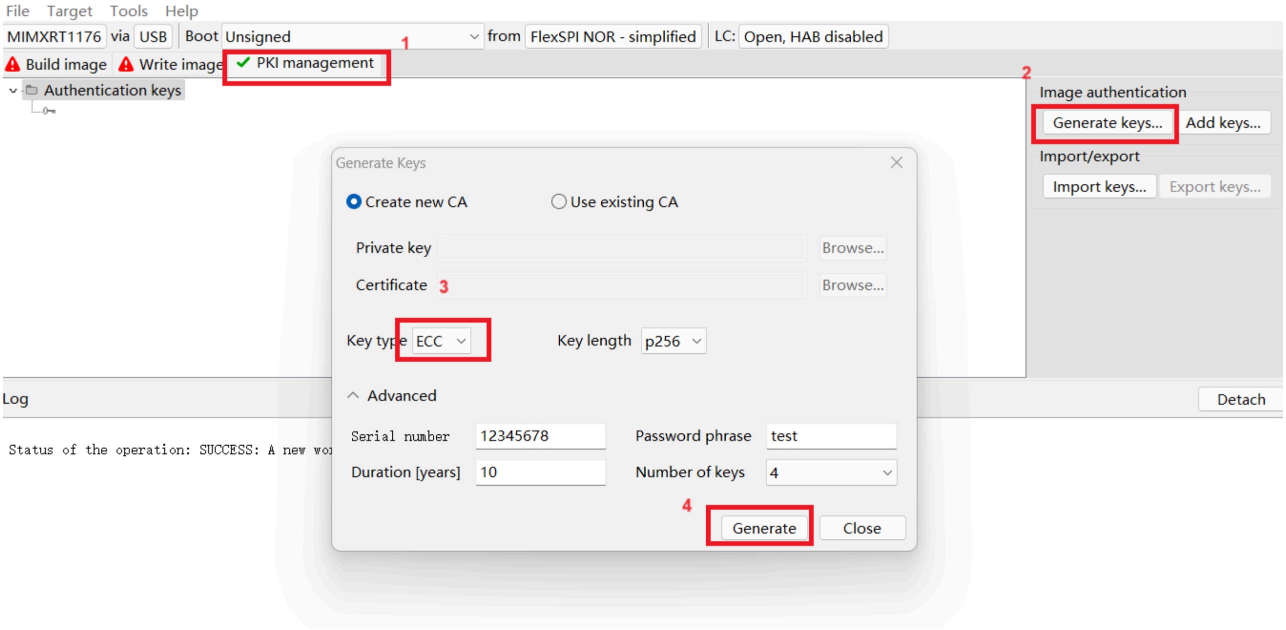
Series
○ KW45xx/K32W1xx
○ LPC55Sxx
○ MCX N94x/N54x
○ RW61x
○ i.MX RT10xx
● i.MX RT11xx
○ i.MX RTxxx
○ LPC55xx
○ MCX A14x/A15x

Processor
○ MIMXRT1165
○ MIMXRT1166
○ MIMXRT1171
○ MIMXRT1172
○ MIMXRT1173
○ MIMXRT1175
● MIMXRT1176
○ MIMXRT1181
○ MIMXRT1182
○ MIMXRT1187
○ MIMXRT1189

Or create new workspace using: import manufacturing package

Figure 3. Create a workspace

3. In the **Keys Management** view, click the **Generated keys** button, then specify all parameters in this menu.



MIMXRT1176 via USB Boot Unsigned from FlexSPI NOR - simplified LC: Open, HAB disabled

Generate Keys

● Create new CA ○ Use existing CA

Private key

Certificate

Key type ECC

Key length p256

Advanced

Serial number 12345678

Password phrase test

Duration [years] 10

Number of keys 4

```
Status of the operation: SUCCESS: A new wo
```

Figure 4. Generate ecc keys and certs

4. The user can find generated keys and certificates in the "keys" and "crts" folder in the workspace directory.

2.6 Create BD file

An example *.bd file for ecc p256 keys and certificates is as below:

```
options {
    flags = 0x08;
    startAddress = 0x30000000;
    ivtOffset = 0x1000;
    initialLoadSize = 0x2000;
}

sources {
    elfFile = extern(0);
}

constants {
    SEC_CSF_HEADER              = 20;
    SEC_CSF_INSTALL_SRK         = 21;
    SEC_CSF_INSTALL_CSFK        = 22;
    SEC_CSF_INSTALL_NOCAK       = 23;
    SEC_CSF_AUTHENTICATE_CSF    = 24;
    SEC_CSF_INSTALL_KEY         = 25;
    SEC_CSF_AUTHENTICATE_DATA   = 26;
    SEC_CSF_INSTALL_SECRET_KEY  = 27;
    SEC_CSF_DECRYPT_DATA        = 28;
    SEC_NOP                     = 29;
    SEC_SET_MID                 = 30;
    SEC_SET_ENGINE              = 31;
    SEC_INIT                    = 32;
    SEC_UNLOCK                  = 33;
}

section (SEC_CSF_HEADER;
    Header_Version="4.2",
    Header_HashAlgorithm="sha256",
    Header_Engine="ANY",
    Header_EngineConfiguration=0,
    Header_CertificateFormat="x509",
    Header_SignatureFormat="CMS"
    )
{
}

section (SEC_CSF_INSTALL_SRK;
    InstallSRK_Table="gen_hab_certs/SRK_hash.bin", // "valid file path"
    InstallSRK_SourceIndex=0
    )
{
}

section (SEC_CSF_INSTALL_CSFK;
    InstallCSFK_File="crts/CSF1_1_sha256_p256_v3_usr_crt.pem", // "valid file
 path"
    InstallCSFK_CertificateFormat="x509" // "x509"
    )
{
}

section (SEC_CSF_AUTHENTICATE_CSF)
{
```

```
}

section (SEC_CSF_INSTALL_KEY;
    InstallKey_File="crts/IMG1_1_sha256_p256_v3_usr_crt.pem",
    InstallKey_VerificationIndex=0, // Accepts integer or string
    InstallKey_TargetIndex=2) // Accepts integer or string
{
}

section (SEC_CSF_AUTHENTICATE_DATA;
    AuthenticateData_VerificationIndex=2,
    AuthenticateData_Engine="ANY",
    AuthenticateData_EngineConfiguration=0)
{
}

section (SEC_SET_ENGINE;
    SetEngine_HashAlgorithm = "sha256", // "sha1", "Sha256", "sha512"
    SetEngine_Engine = "CAAM", // "ANY", "SAHARA", "RTIC", "DCP", "CAAM" and
 "SW"
    SetEngine_EngineConfiguration = "0") // "valid engine configuration values"
{
}
```

2.7 Generate signed image

To generate a signed bootable image using elftosb, perform the following steps:

1. Retrieve the elftosb package from the NXP website.
2. Copy the SREC application image into the same folder that holds the elftosb executable.
3. Copy the compiling "cst" executable, "crts", "gen_hab_certs", and "keys" folders from the MCUXpresso Secure Provisioning RT1176 workspace to the same folder that holds the elftosb executable.
4. Generate a bootable image using elftosb.

```
elftosb -f imx -V -c .\imx-flexspinor-normal-signed.bd -o .\iled_blinky.bin.
\iled_blinky.srec
```

As we change the CST source, the tool generates `imghash.bin` and wait for the signature of the image first. After we provide a signature binary file `img_sig.bin`, press any key to go ahead. The CST generates the hash value of CSF data `csfhash.bin` and wait for the signature of it. The procedure completes after providing CSF signature data `csf_sig.bin`.

After the above operation, there are two bootable images generated by elftosb.

- The first one is `iled_blinky.bin`. The memory region from 0 until `ivt_offset` is fill with padding bytes (all 0x00s).
- The second one is `iled_blinky_nopadding.bin`, which starts from `ivt_offset` directly.

For test, user can use the below openssl command to generate the signature.

```
openssl cms -sign -nosmimecap -nocerts -partial_chain -digest <hash
 value> -passin file:keys/key_pass.txt -inform pem -outform der
 -signer ./crts/IMG1_1_sha256_p256_v3_usr_crt.pem -inkey ./keys/
IMG1_1_sha256_p256_v3_usr_key.pem -out img_sig.bin
```

```
openssl cms -sign -nosmimecap -nocerts -partial_chain -digest <hash
 value> -passin file:keys/key_pass.txt -inform pem -outform der
```

```
 -signer ./crts/CSF1_1_sha256_p256_v3_usr_crt.pem -inkey ./keys/
CSF1_1_sha256_p256_v3_usr_key.pem -out csf_sig.bin
```

Note: *Openssl parameter digest provides the hash value in hexadecimal form and openssl provides `-digest` starting from v3.2.*

3 Using SPSDK

This section describes how to set up signature provider and build an image signed by the signature provider with the "nxpimage" tool. The nxpimage in spsdk is a tool for generating TrustZone, master boot image, and secure binary images. This tool is used to create various kinds of NXP images. The configuration file for nxpimage contains all possible configuration settings.

SPSDK uses a simple flask REST API service representing the remote HSM machine. However, in the real world, it is expected that the implementation is changed by communication with hardware HSM module or custom HTTPS communication to a server. We have used the `secp256r1` key type in this work.

3.1 Signature provider workflow

Figure 5 describes a workflow for signing an image with a signature provider. The nxpimage tool sends requests to the signature provider. The signature provider should pass the request to an HSM or server and then pass the response back to the spsdk tool.

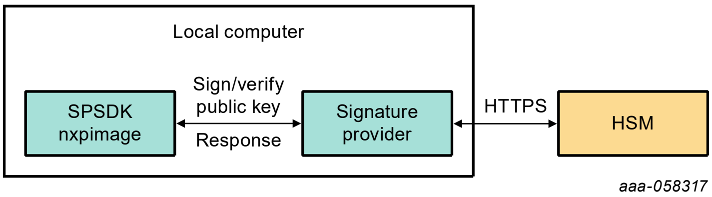


Figure 5. Signature provider workflow

3.2 Set up config file

The configuration file is needed for successful generation of signed image using the nxpimage application.

Two configuration settings, *signPrivateKey* and *signProvider*, are used to control if nxpimage signs the image with a local private key or using remote signing. These two configuration values are mutually exclusive, so only one can be chosen.

In the config file, the field `AuthenticateData_SignProvider` is used to set image private key file. The field `AuthenticateCsf_SignProvider` sets private key file for CSF data.

The configuration value format is "type=<sp_type>;<key1>=<value1>;<key2>=<value2>;…"

The `sp_type` is super awesome signature provider (sasp), which is defined in the custom signature provider (`plugins/sasp.py`).

The example for `AuthenticateData_SignProvider` is as below:

```
AuthenticateData_SignProvider:
type=sasp;key_number=1;key_type=secp256r1
```

There is an RT1176 HSM example project that includes an example config file `rt1176_xip_signed.yaml`. User can unzip the project and copy folder RT1176 to folder `spsdk\examples\signature_provider`.

3.3 Install SPSDK

For SPSDK, ensure to have Python 3.9+ installed.

To install the SPSDK from source code, create a virtual Python environment using the console command window:

1. Create a virtual environment (for example, venv):

   ```
   python -m venv <name>
   ```

2. Activate the virtual environment (for windows):

   ```
   <name>\Scripts\activate
   ```

3. Download the SPSDK source code:

   ```
   git clone https://github.com/nxp-mcuxpresso/spsdk.git
   ```

4. Enter the folder spsdk and install *SPSDK* from the source code.

   ```
   pip install -U -e
   ```

For other platform, refer to the installation guide available at: https://spsdk.readthedocs.io/en/latest/usage/installation.html.

3.4 Install jupyter

Jupyter notebook is a web-based interactive development environment. We provide jupyter notebooks as an interactive documentation.

Install additional development requirements using the below command to run jupyter notebooks:

```
pip install spsdk[examples]
```

3.5 Set up HSM

HSM example in SPSDK does not enable `secp256r1` key type and the private keys are encrypted in the procedure of generation. So, there are two changes in the file `spsdk\examples\signature_provider\common\hsm\sahsm.py`.

The first change is to enable `secp256r1` key type in the HSM demo as below:

```
SUPPORTED_KEY_TYPES = ["rsa2048", "secp256r1", "secp384r1"]
```

The second change is to set a password to decrypt private keys correctly. Change is in the function **_load_private_key**.

```
private_key = PrivateKey.load(private_key_file, 'test')
```

In the HSM example, the private key name format is `hsm_k{num}_{key_type}.pem` (for example, `hsm_k0_secp384r1.pem`).

The signature provider passes 'num' and 'key type' parameters to HSM. So, a user should rename private keys and copy them to the folder `spsdk\examples\signature_provider\common\hsm`.

For example, rename `CSF1_1_sha256_p256_v3_usr_key.pem` to `hsm_k0_secp256r1.pem` and rename `IMG1_1_sha256_p256_v3_usr_key.pem` to `hsm_k1_secp256r1.pem`. The keys are generated in Section 2.5.2.

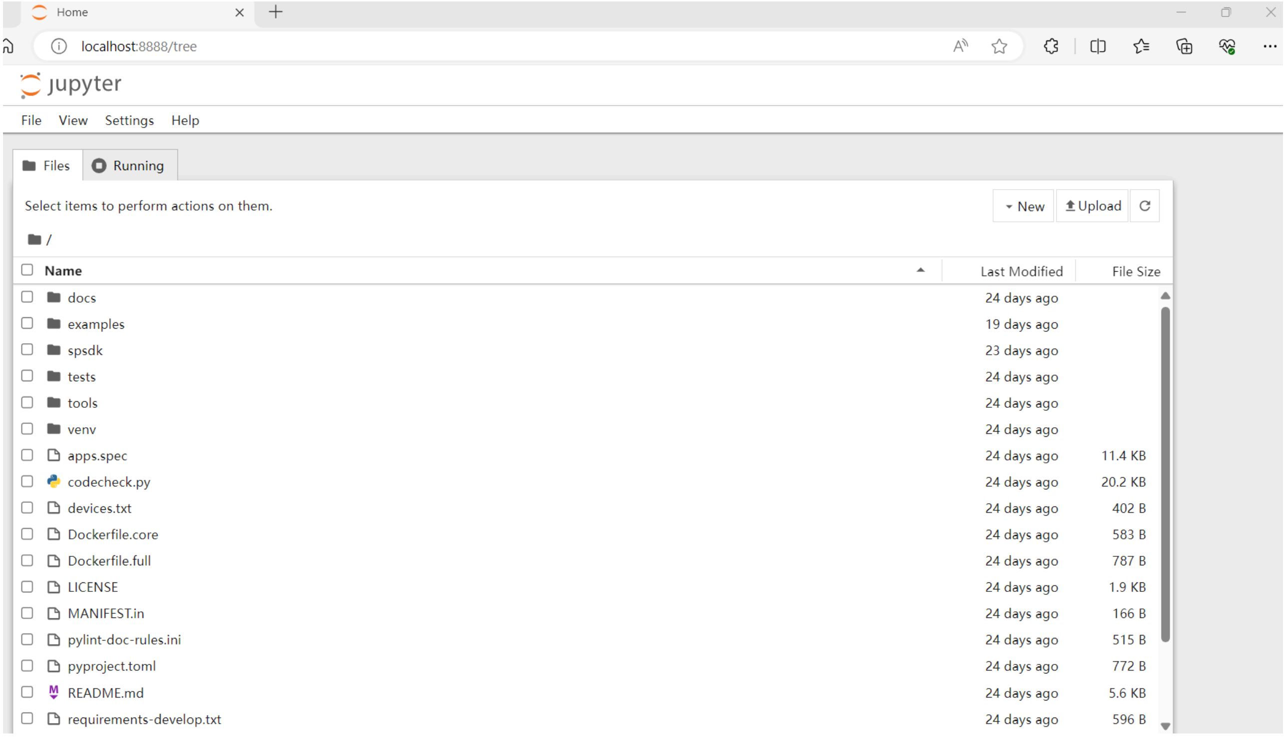
3.6 Generate signed image

This section describes how to use a custom remote signing service for generating a signed image using the nxpimage tool.

To generate a signed bootable image, perform the following steps:

1. Copy user demo image to `spsdk\examples\signature_provider\RT1176\data_img\`. The attached project provides a demo project `evkbmimxrt1170_hello_world_demo_cm7.s19`.
2. Copy the 'crts' and 'gen_hab_certs' folders from the MCUXpresso Secure Provisioning RT1176 workspace to the folder `spsdk\examples\signature_provider\RT1176\data_img\`.
3. Launch jupyter notebook in python virtual environment with below command:

```
jupyter notebook
```



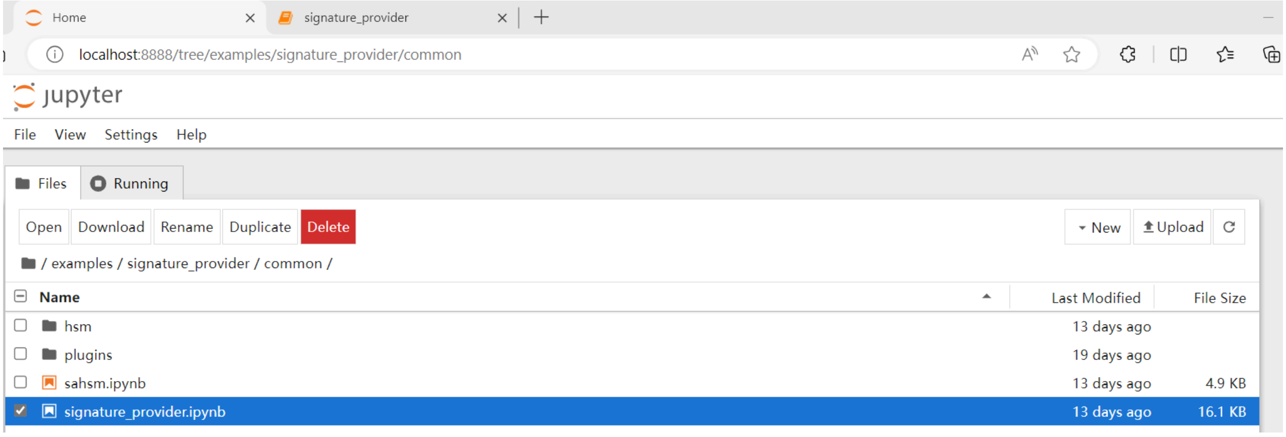
Select items to perform actions on them.

/

☐ Name	Last Modified	File Size
☐ docs	24 days ago	
☐ examples	19 days ago	
☐ spsdk	23 days ago	
☐ tests	24 days ago	
☐ tools	24 days ago	
☐ venv	24 days ago	
☐ apps.spec	24 days ago	11.4 KB
☐ codecheck.py	24 days ago	20.2 KB
☐ devices.txt	24 days ago	402 B
☐ Dockerfile.core	24 days ago	583 B
☐ Dockerfile.full	24 days ago	787 B
☐ LICENSE	24 days ago	1.9 KB
☐ MANIFEST.in	24 days ago	166 B
☐ pylint-doc-rules.ini	24 days ago	515 B
☐ pyproject.toml	24 days ago	772 B
☐ README.md	24 days ago	5.6 KB
☐ requirements-develop.txt	24 days ago	596 B

Figure 6. Jupyter environment

4. Navigate to `examples\signature_provider\common` and open `signature_provider.ipynb`.

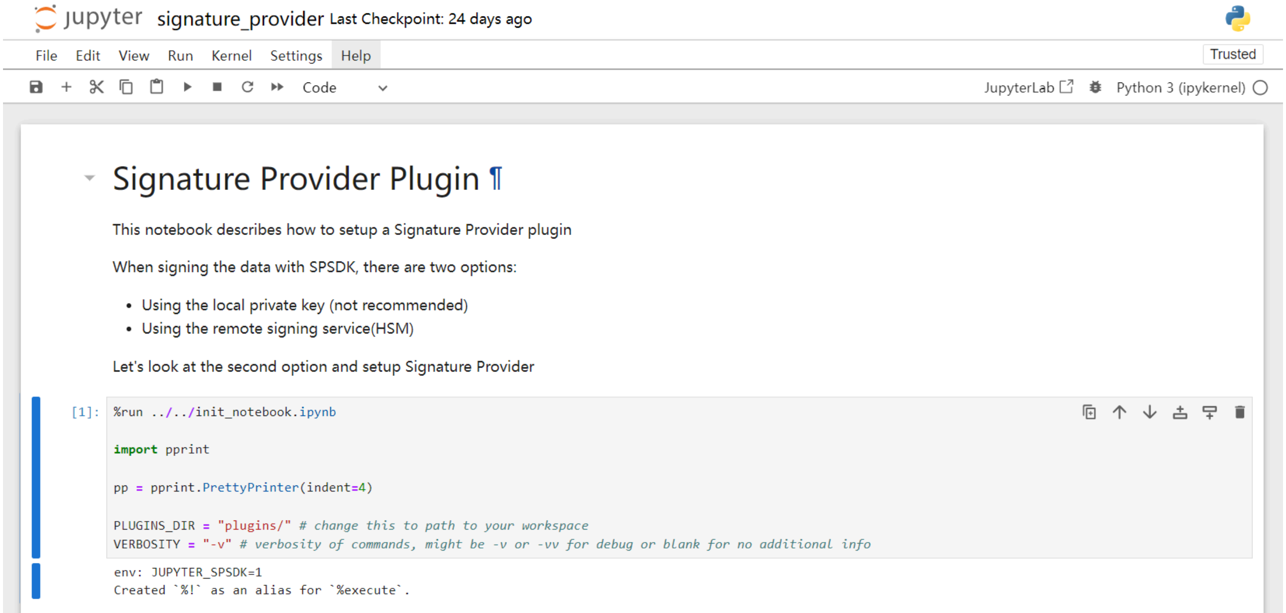


/ examples / signature_provider / common /

☐	Name	Last Modified	File Size
☐	hsm	13 days ago	
☐	plugins	19 days ago	
☐	sahsm.ipynb	13 days ago	4.9 KB
☑	signature_provider.ipynb	13 days ago	16.1 KB

Figure 7. Open Signature Provider

5. Press shift+enter to run the first cell to set up the Signature Provider plugin.



Signature Provider Plugin ¶

This notebook describes how to setup a Signature Provider plugin

When signing the data with SPSDK, there are two options:

- Using the local private key (not recommended)
- Using the remote signing service(HSM)

Let's look at the second option and setup Signature Provider

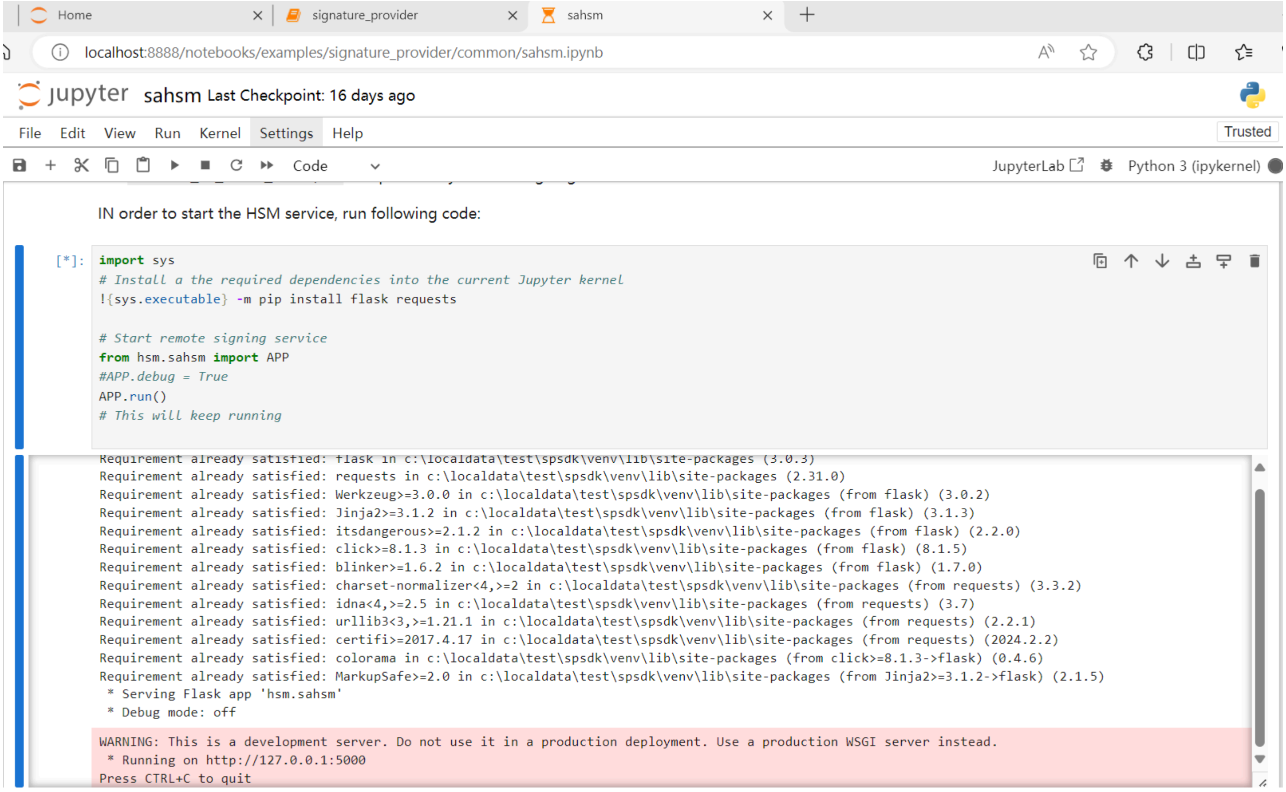
[1]:

```
%run ../../init_notebook.ipynb

import pprint

pp = pprint.PrettyPrinter(indent=4)

PLUGINS_DIR = "plugins/" # change this to path to your workspace
VERBOSITY = "-v" # verbosity of commands, might be -v or -vv for debug or blank for no additional info
```

```
env: JUPYTER_SPSDK=1
Created `%!` as an alias for `%execute`.
```

Figure 8. Setup signature provider plugin

6. Open `examples\signature_provider\common\sahsm.ipynb` and start the custom HSM by running the cell.



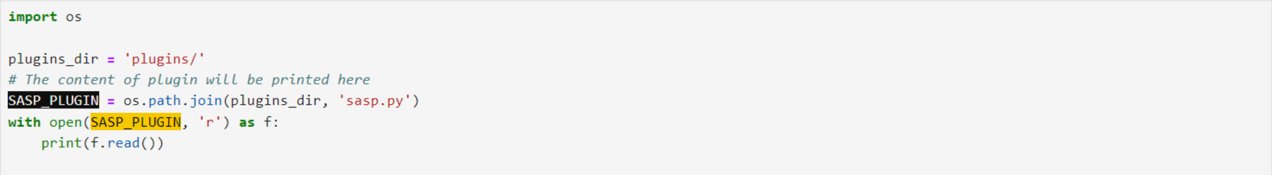
IN order to start the HSM service, run following code:

```
[*]: import sys
     # Install a the required dependencies into the current Jupyter kernel
     !{sys.executable} -m pip install flask requests

     # Start remote signing service
     from hsm.sahsm import APP
     #APP.debug = True
     APP.run()
     # This will keep running
```

```
Requirement already satisfied: flask in c:\localdata\test\spsdk\venv\lib\site-packages (3.0.3)
Requirement already satisfied: requests in c:\localdata\test\spsdk\venv\lib\site-packages (2.31.0)
Requirement already satisfied: Werkzeug>=3.0.0 in c:\localdata\test\spsdk\venv\lib\site-packages (from flask) (3.0.2)
Requirement already satisfied: Jinja2>=3.1.2 in c:\localdata\test\spsdk\venv\lib\site-packages (from flask) (3.1.3)
Requirement already satisfied: itsdangerous>=2.1.2 in c:\localdata\test\spsdk\venv\lib\site-packages (from flask) (2.2.0)
Requirement already satisfied: click>=8.1.3 in c:\localdata\test\spsdk\venv\lib\site-packages (from flask) (8.1.5)
Requirement already satisfied: blinker>=1.6.2 in c:\localdata\test\spsdk\venv\lib\site-packages (from flask) (1.7.0)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\localdata\test\spsdk\venv\lib\site-packages (from requests) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\localdata\test\spsdk\venv\lib\site-packages (from requests) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\localdata\test\spsdk\venv\lib\site-packages (from requests) (2.2.1)
Requirement already satisfied: certifi>=2017.4.17 in c:\localdata\test\spsdk\venv\lib\site-packages (from requests) (2024.2.2)
Requirement already satisfied: colorama in c:\localdata\test\spsdk\venv\lib\site-packages (from click>=8.1.3->flask) (0.4.6)
Requirement already satisfied: MarkupSafe>=2.0 in c:\localdata\test\spsdk\venv\lib\site-packages (from Jinja2>=3.1.2->flask) (2.1.5)
 * Serving Flask app 'hsm.sahsm'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Figure 9. Start HSM

7. Switch to `signature_provider.ipynb`. Run below cell to set the environment variable `SASP_PLUGIN` for Signature Provider plugin.

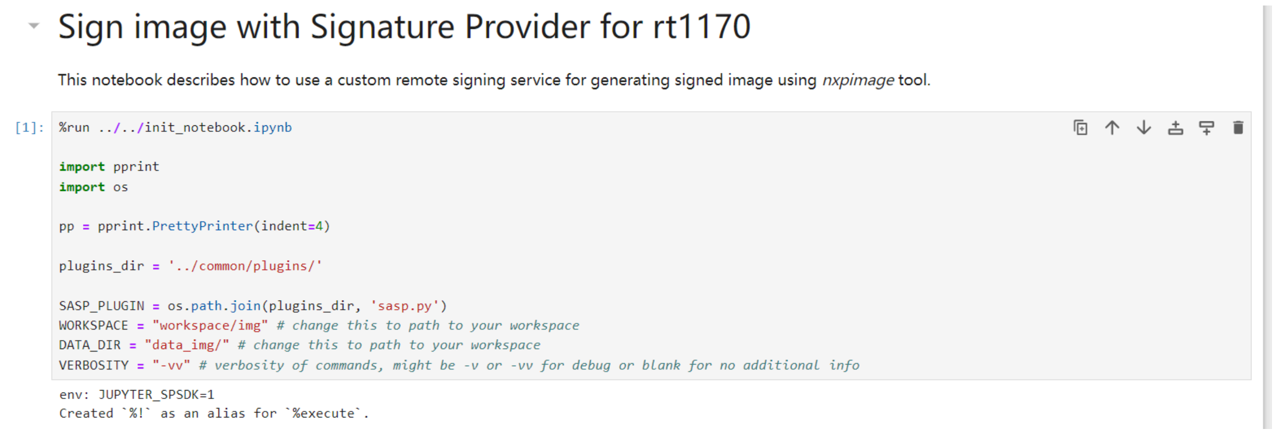


```
import os

plugins_dir = 'plugins/'
# The content of plugin will be printed here
SASP_PLUGIN = os.path.join(plugins_dir, 'sasp.py')
with open(SASP_PLUGIN, 'r') as f:
    print(f.read())
```

Figure 10. Set Signature Provider plugin env variable

After changing `_load_private_key` function in Section 3.5, the test for the functionality of HSM in this notebook is failed as the SPSDK test private keys is not encrypted.

8. Navigate to `examples\signature_provider\RT1176` and open `rt1170_hab.ipynb`.
9. In `rt1170_hab.ipynb`, run the first cell to initiate the notebook.



Sign image with Signature Provider for rt1170

This notebook describes how to use a custom remote signing service for generating signed image using *nxpimage* tool.

```
[1]: %run ../../init_notebook.ipynb

     import pprint
     import os

     pp = pprint.PrettyPrinter(indent=4)

     plugins_dir = '../common/plugins/'

     SASP_PLUGIN = os.path.join(plugins_dir, 'sasp.py')
     WORKSPACE = "workspace/img" # change this to path to your workspace
     DATA_DIR = "data_img/" # change this to path to your workspace
     VERBOSITY = "-vv" # verbosity of commands, might be -v or -vv for debug or blank for no additional info
```

```
env: JUPYTER_SPSDK=1
Created `%!` as an alias for `%execute`.
```

Figure 11. Initiate the notebook

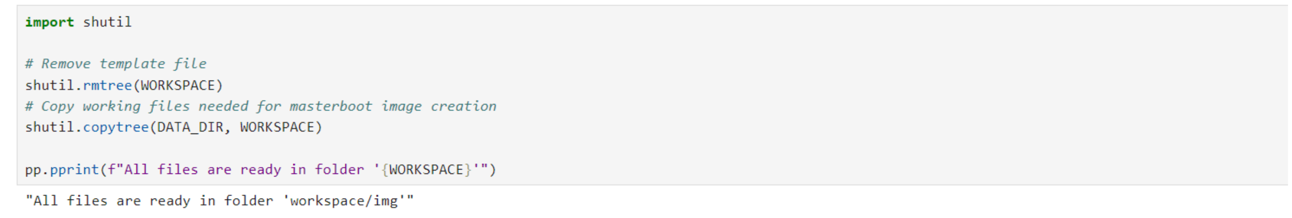
10. Run the second cell to generate a configuration template and set the path for the demo configuration file `rt1176_xip_signed.yaml`.

```
import os
from spsdk.utils.misc import load_file

%! nxpimage $VERBOSITY hab get-template --output $WORKSPACE/rt1176_xip_signed.yaml --force

CONFIG_PATH = os.path.join(WORKSPACE, "rt1176_xip_signed.yaml")
# just for verification that the template was generated
assert os.path.exists(CONFIG_PATH)

config_content = load_file(CONFIG_PATH)
print(config_content)
```

Figure 12. Show template file

Note: *Template is available at* `workspace\img` *folder. The configuration file* `data_img/rt1176_xip_signed.yaml` *is used in this example.*

11. Run the third cell to copy all the working files from folder `data_img` to folder `workspace\img`.



```
import shutil

# Remove template file
shutil.rmtree(WORKSPACE)
# Copy working files needed for masterboot image creation
shutil.copytree(DATA_DIR, WORKSPACE)

pp.pprint(f"All files are ready in folder '{WORKSPACE}'")
```

```
"All files are ready in folder 'workspace/img'"
```

Figure 13. Copy working files

Note: *User may need to update the item* `entryPointAddress` *in the configuration yaml file for the user image.*

12. Run the last cell to generate the signed image.

```
%! nxpimage $VERBOSITY hab export --plugin $SASP_PLUGIN -c $CONFIG_PATH -o $WORKSPACE/hello_world_demo_cm7_hab.bin

# check if the signed image exists
output_file = os.path.join(WORKSPACE, "hello_world_demo_cm7_hab.bin")
assert os.path.exists(output_file)
```

```
nxpimage -v hab export --plugin ../common/plugins/sasp.py -c workspace/img\rt1176_xip_signed.yaml -o workspace/img/hello_world_demo_cm7_hab.bin
Success. (HAB container: C:\LocalData\Doc\vendor\NXP\working\spsdk\examples\signature_provider\RT1176\workspace\img\hello_world_demo_cm7_hab.bin create
d.)
```

Figure 14. Generate the signed image

As the command line shows, the output image is `hello_world_demo_cm7_hab.bin` at the folder workspace.

4 References

- Code-Signing Tool User’s Guide
- How to use i.MXRT Security Boot (document AN12079)
- How to use HAB secure boot in i.MX RT10xx (document AN12681)
- SPSDK Signature Provider doc: https://spsdk.readthedocs.io/en/latest/examples/signature_prov.html

5 Note about the source code in the document

The 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 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.

6 Revision history

Table 1 summarizes the revisions to this document

Table 1. Revision history

Document ID	Release date	Description
AN14471 v.1.0	4 November 2024	Initial public release

Legal information

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.

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.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

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.

Limiting values — Stress above one or more limiting values (as defined in the Absolute Maximum Ratings System of IEC 60134) will cause permanent damage to the device. Limiting values are stress ratings only and (proper) operation of the device at these or any other conditions above those given in the Recommended operating conditions section (if present) or the Characteristics sections of this document is not warranted. Constant or repeated exposure to limiting values will permanently and irreversibly affect the quality and reliability of the device.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://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.

No offer to sell or license — Nothing in this document may be interpreted or construed as an offer to sell products that is open for acceptance or the grant, conveyance or implication of any license under any copyrights, patents or other industrial or intellectual property rights.

Quick reference data — The Quick reference data is an extract of the product data given in the Limiting values and Characteristics sections of this document, and as such is not complete, exhaustive or legally binding.

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.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

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.

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.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, μVision, Versatile — are trademarks and/or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

Contents