AN14123

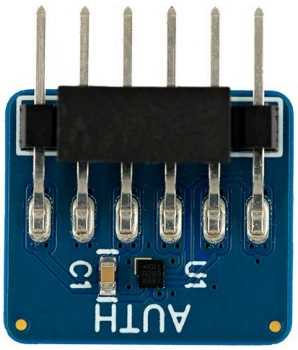
NX MW Documentation

Rev. 2.5.0 — 9 May 2025 **Application note**

1 Introduction

1.1 NX Middleware for EdgeLock A30 and NTAG-X-DNA Secure Authenticator

EdgeLock A30 and NTAG-X-DNA are secure authentication ICs for IoT platforms, electronic accessories and consumable devices such as home electronic devices, mobile accessories and medical supplies. EdgeLock A30 and NTAG-X-DNA supports on-chip ECC key generation to make sure that private keys are never exposed outside the IC. It performs cryptographic operations for security critical communication and control functions. EdgeLock A30 and NTAG-X-DNA are Common Criteria EAL 6+ security certified with AVA_VAN.5 on product level and supports a generic



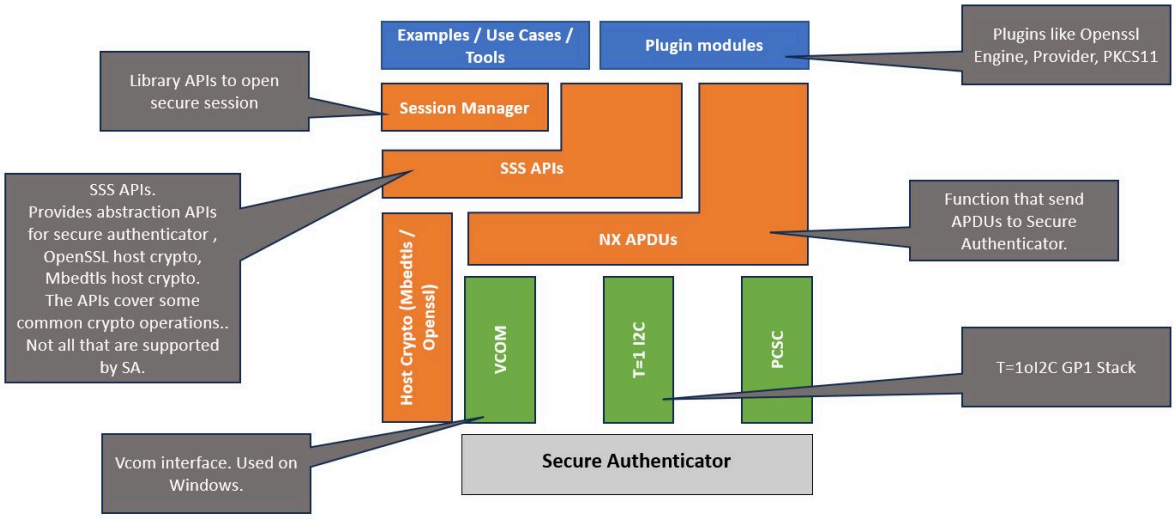
Crypto API providing AES, ECDSA, ECDH, SHA, HMAC and HKDF cryptographic functionality.

- Asymmetric cryptography features support 256-bit ECC over the NIST P-256 and brainpool P256r1 curves.
- Symmetric cryptography features support both AES-128 and AES-256.
- PKI-based mutual authentication based on the Sigma-I protocol.
- Symmetric three pass Mutual Authentication protocol compatible with NTAG42x and MIFARE DesFire EV2, DesFire EV3 and DesFire Light.
- Secure messaging channel using either AES-128 or AES-256 session encryption/decryption and MAC.

The Common Criteria security certification ensures that the IC security measures and protection mechanisms have been evaluated against sophisticated noninvasive and invasive attack scenarios.

- A30 supports an I2C contact interface and has two additional 2 GPIOs.
- A30 supports a low-power design, and consumes only 5 µA at Deep-Power-Down mode when an external VDD is supplied.

NX Middleware here provides the necessary interfaces, examples, demos for A30 Secure Authenticator (SA) IC. Following diagram shows different modules that are part of NX Middleware.




```
NOTE:
  1) It is recommended to use the latest version of this NX middleware software.
  2) VCOM and PC/SC interfaces (https://pcscworkgroup.com/) are provided
 for quick evaluation of the Secure authenticator examples and should not be
 deployed in a product.
```

To contact NXP or to report issues, please use **Support | NXP**

Refer **NX Middleware** for more details on the middleware.

1.1.1 Getting Started

Refer the following sections to build NX Middleware and run demos.

For Linux, refer **Getting started on Linux**

For Windows, refer **Getting Started on Windows**

For MCU Projects, refer **Getting Started on MCUs Using Standalone MCUXpresso Projects**

For MCU cmake build, refer **Getting Started on MCU cmake build**

1.1.2 Folder Structure

File/Folder	Content
binaries	Pre built binaries, session certificates and keys for testing
boards	Platform porting specific files
demos	Examples / Use cases / Access manager
doc	Documentation
ext	Openssl and PKCS11 dependencies required to build / run MW demos

File/Folder	Content
lib	Session Manager Code, SSS APIs, NX APIs to use NX SA
mcu_sdk	Files for cloning and building from MCU SDK
mcux_project	Standalone MCUXpresso projects for various supported MCUs
plugin	Standard plugins like OpenSSL Engine, OpenSSL provider, psa, pkcs11
scripts	Scripts for building the MW
tools	Temporary executables
ChangeLog.md	Change Log file
CMakeLists.txt	Root Cmake file
LICENSE.txt	License
README.md	Readme document
SCR.txt	Software Content Register

1.1.3 Useful Links

OpenSSL Provider - **OpenSSL Provider**

OpenSSL Engine - **OpenSSL Engine**

Platform Security Architecture - **Platform Security Architecture**

Access Manager for Linux - **Access Manager**

Command line tool - **NX CLI Tool**

1.2 Changelog

1.2.1 [v02.05.00]

- Support for Windows, LPC55s69, MCXN947, MCXA153 platforms added.
- For Windows, refer **Getting Started on Windows**
- For MCU Projects, refer **Getting Started on MCUs Using Standalone MCUXpresso Projects**
- For MCU cmake build, refer **Getting Started on MCU cmake build**
- MCUX projects for few examples added for LPC55s69, MCXN947, MCXA153. Refer **mcux_project**
- Pre-build binaries of vcom and NX CLI tool added. Refer **Binaries**
- New examples -
- Host co-processor example. Refer **Host co-processor demo**
- Qi Authentication example. Refer **Qi Authentication demo**
- ECC Standalone (session open from main). Refer **ECC Standalone example**
- VCOM example. Refer **VCOM example**
- New Plugins
- PKCS11 - Refer **PKCS11 library**
- PSA - Refer **PSA files**
- Mbed TLS ALT - Refer **Mbed TLS ALT files**
- OpenSSL Provider changes

- ECDSA functions handle different formats of SHA algorithm string (Example - "sha256" / "SHA256" / "SHA2-256")
- NX CLI tool changes
- Added write access condition option in setkey command
- New commands added - list-eckey, set-i2c_mgnt, set-cert_mgnt. For details refer - **NX CLI Tool**.
- AWS Cloud example extended for MCXN947 platform.
- Fixes for static analysis findings and memory leak issues.
- APDU buffers (of APDU functions of nx_apdu.c) are made global.

1.2.2 [v02.04.00]

- IMPORTANT: This is first version of GitHub release.
- Note: Windows, frdmK64 and LPC platform support / example will be added in subsequent releases.
- Previous versions of NX middleware are not released on GitHub
- Access Manager added (To support multiple client access SA)
- OpenSSL Provider changes
- CSR generation extended for all SHA algorithms now. (Get context functions - 'sss_rsa_signature_get_ctx_params' and 'sss_ecdsa_signature_get_ctx_params' updated to handle all SHA algorithms).
- Performance improvement : Provider is updated to store the client / server public key on host to avoid multiple secure authenticator reads during TLS connection.
- OSSL algorithms are updated with algorithm properties.
- Enable / Disable random number generation in nxp provider using compile time option - 'SSS_PROV_DISABLE_NX_RNG'. (Disabled by default)
- ECC key management import functions added - to handle reference keys. If the input key is not reference key, the function returns error to roll back on other available providers.
- ECC key management - match and duplicate functions added.
- ECDSA digest verify support added (function - sss_ecdsa_signature_digest_verify).
- nx cli tool updated to set and get binary data form nx secure authenticator.
- Fixes for static analysis findings.

1.2.3 [v02.03.00]

- lpc55s69 platform support added (cmake based build and mcuxpresso project).
- Mbedtls 3.x support added (Mbedtls version - 3.5.0). Using cmake option `-DNXMW_MBedTLS=`, NX middleware can be built with 2.x or 3.x version of Mbedtls.
- MCU SDK updated to version 2.14
- PSA APIs developed for NX secure authenticator. Refer - :ref:`psa-alt`.
- OpenSSL Provider - Provider updated to support reference keys in file format also.
- NX Cli tool - All provision scripts of demos (engine, provider, TLS examples) are updated to use nx cli tool.
- PKCS11 plugin improvements
- Code restructuring - Platform specific code is moved to board folder.
- Fixes for static analysis findings.

1.2.4 [v02.02.00]

- Fixes for static analysis findings.
- `nx_cli_tool` added. Refer - :ref:`nx-cli-tool`.

1.2.5 [v02.01.00]

- New Features
- OpenSSL engine for NX Secure authenticator. Refer - :ref:`intro-openssl-engine`.
- OpenSSL provider for NX Secure authenticator. Refer - :ref:`intro-openssl-provider`.
- PKCS11 module for NX Secure authenticator. Refer - :ref:`intro-pkcs11-lib`.
- TLS example updated for OpenSSL provider also. Refer - :ref:`tls-client-example`
- FreeRTOS support for k64 platform. (NXMW_RTOS:STRING=FreeRTOS)
- AWS cloud example for k64. Refer - :ref:`ksdk-demos-aws`
- Standalone MCUXpresso project for k64 added. Refer - :ref:`mcux_projects`.
- Default cmake option of sigma-i Curve type is changed to NIST_P (NXMW_Auth_Asymm_Host_ Curve:STRING=NIST_P),
- Default certificate type of sigma-i authentication is changed to x509
- Fixes for static analysis findings.
- Folder restructuring
  - SSS and HostLib modules are moved to `lib` folder
  - Platform specific contents are moved to `boards` folder
  - All plugin modules are moved to folder - `plugin`

1.2.6 [v02.00.00]

- Platforms supported - Windows, FRDM-k64 (native and vcom), Raspberry-pi

1.2.7 [v01.00.00]

- Platforms supported - Windows, FRDM-k64 (only vcom)
- Supports Nx by VCOM.
- APIs & enum/types Changes
  - Added support for SSS and nx APIs.

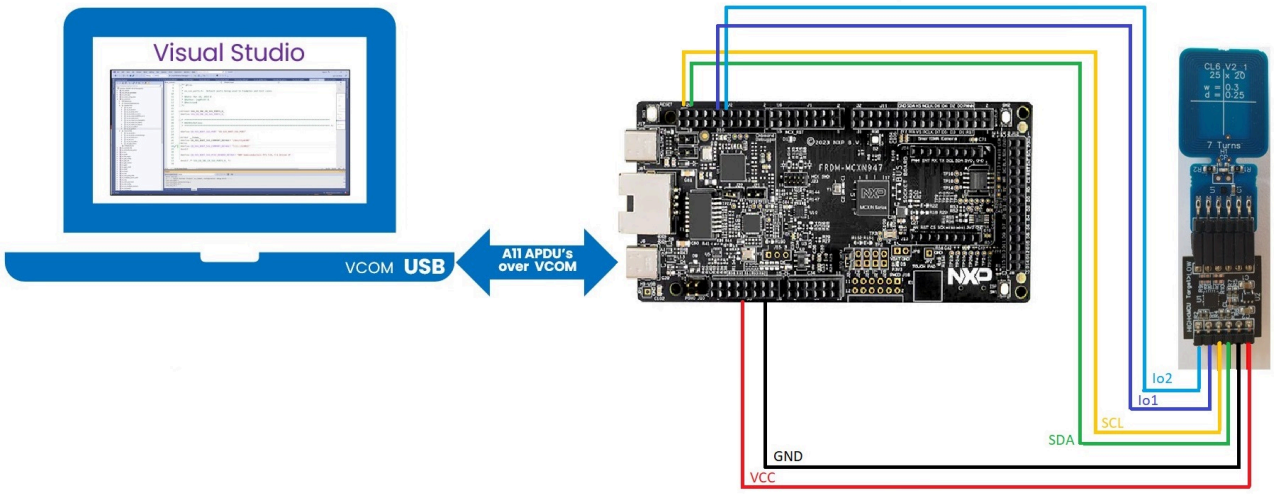
2 Build and Run

2.1 Windows

2.1.1 Getting Started on Windows

Building the NX MW on Windows enables to explore the MW stack and examples in a rich IDE. Only the low level communication (T=1oI2C) to the Secure Authenticator is done on a connected host MCU as interface to the Secure Authenticator.

Note: FRDM-MCXN947 is used as an Host MCU example in the following documentation.



2.1.1.1 Prerequisite

- Visual studio installed (>= 2015 version, or higher)
- Python installed (version 3.10 or above). Please follow the guideline at Python Download.
- West installed (version 1.2.0 or above). Please follow west setup to install west
- CMake installed (version 3.30.0 or above). Preferably at C:/opt/cmake/bin/cmake.exe

2.1.1.2 Getting the NX Middleware Source

- Follow the following steps to create a new workspace which downloads the NX middleware and the required files.

```
west init -m https://github.com/NXP/nxmw.git --mf mcu_sdk/west.yml workspace
OR
west init -m https://github.com/NXP/nxmw.git --mr <branch name> --mf mcu_sdk/
west.yml workspace

cd workspace
west update
```

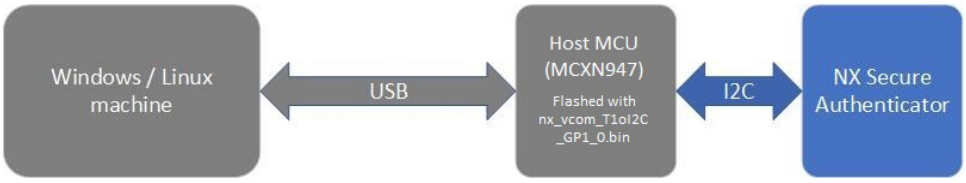
Note: The complete setup takes 10-15 minutes to download. Once downloaded you should have the NX middleware and all the required SDK files.

- If you have cloned the NX Middleware using git, and west setup is not done, you can follow the following steps

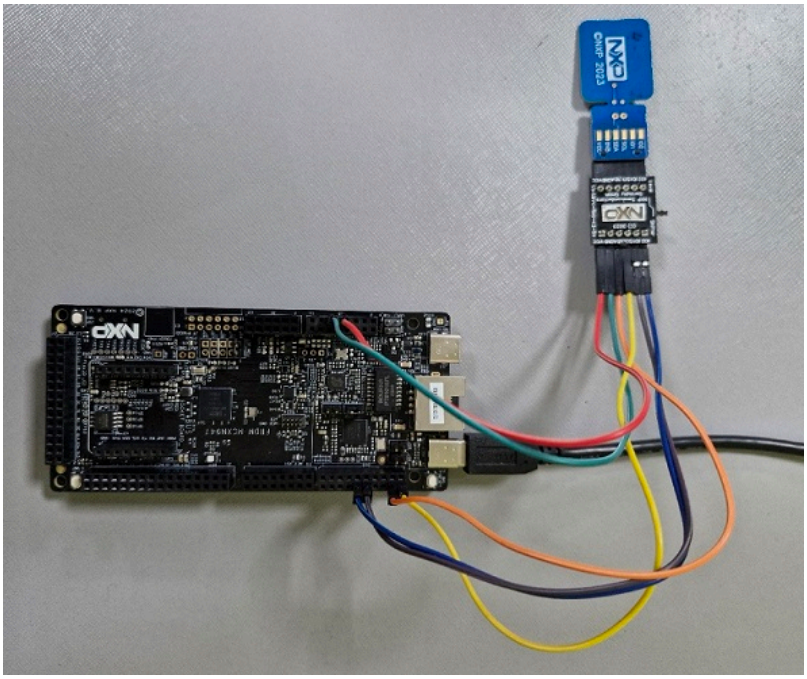
```
cd nxmw
west init -l --mf mcu_sdk/west.yml
cd ..
west update
```

2.1.1.3 Initial Setup

We use VCOM binary to achieve the low level communication (T=1oI2C) with the Secure Authenticator. This binary acts like a bridge between the windows host and the Secure Authenticator.



- Secure authenticator connected to a Host (MCXN947/MCXA153/LPC55S69). Please follow the pin diagram as shown.



- Connect Host MCU-Link port to Windows USB port.
- Flash MCXN947 with vcom binary. You can use the MCUX project provided in `nxmw/mcux_project/mcxn947/vcom` OR Use the precompiled binaries in binary folder `nxmw/binaries/nx_vcom`.
- Now connect the other USB port of MCXN947 to the PC
- The MCXN device will appear as a USB serial device. Note the COM port number with which it is connected to.

2.1.1.3.1 Create Build files

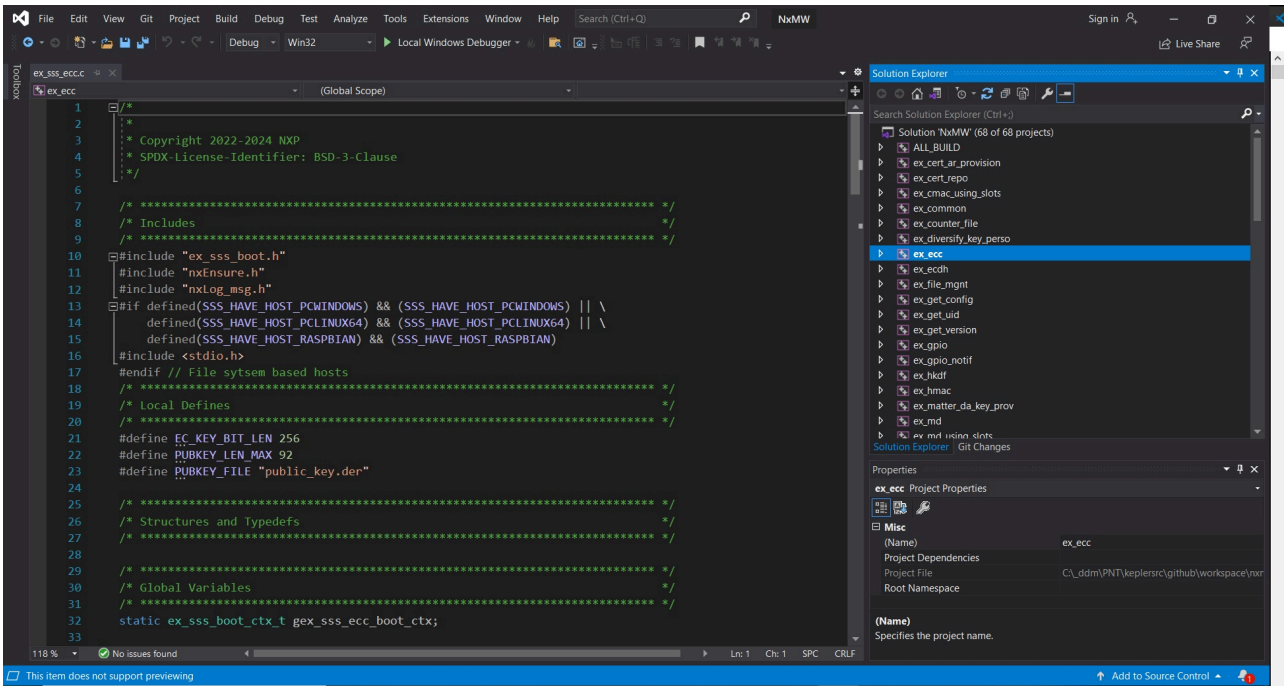
- Use `nxmw/scripts/create_cmake_projects.py` to generate the build files,

```
cd nxmw\scripts
env_setup.bat
python create_cmake_projects.py
```

Note: The env_setup.bat files defines the development tools environment. Depending on your tools (MCUXpresso, Visual Studio, Java, Python and CMake) file locations you may need to update the tools paths within the env_setup.bat file.

Note: Use Command Prompt to generate build files. env_setup.bat file may not work correctly with PowerShell.

- Build files are generated at `nxmw_build/`
- Use the visual studio solution at `nxmw_build/se_x86/NxMW.sln` to build the required example.



- By default the cmake options are set as below.

```
CMAKE_BUILD_TYPE:STRING=Debug
CMAKE_CONFIGURATION_TYPES:STRING=Debug;Release;MinSizeRel;RelWithDebInfo
CMAKE_INSTALL_PREFIX:PATH=C:/Program Files (x86)/NxMW
NXMW_All_Auth_Code:STRING=Enabled
NXMW_Auth:STRING=SYMM_Auth
NXMW_Auth_Asymm_CA_Root_Key_Id:STRING=0
NXMW_Auth_Asymm_Cert_Repo_Id:STRING=0
NXMW_Auth_Asymm_Cert_SK_Id:STRING=0
NXMW_Auth_Asymm_Host_Curve:STRING=NIST_P
NXMW_Auth_Asymm_Host_PK_Cache:STRING=Enabled
NXMW_Auth_Symm_App_Key_Id:STRING=0
NXMW_Auth_Symm_Diversify:STRING=Disabled
NXMW_Host:STRING=PCWindows
NXMW_HostCrypto:STRING=MBEDTLS
NXMW_Log:STRING=Default
NXMW_NX_Type:STRING=NX_R_DA
NXMW_OpenSSL:STRING=1_1_1
NXMW_RTOS:STRING=Default
NXMW_SA_Type=A30
NXMW_SMCOM:STRING=VCOM
NXMW_Secure_Tunneling:STRING=NTAG_AES128_EV2
```

```
    NXPInternal:BOOL=ON
    NXTST_FTR_TEST_COUNT:BOOL=OFF
    WithCodeCoverage:BOOL=OFF
    WithSharedLIB:BOOL=OFF
```

- Change the cmake options if required using the command `cmake-gui ..` For more details refer **CMake Options**

  Note: The Sigma Verifier/Prover demo requires to run the Personalization example once first. Refer **NX Personalization**.

2.1.1.3.1.1 Build / Run Examples

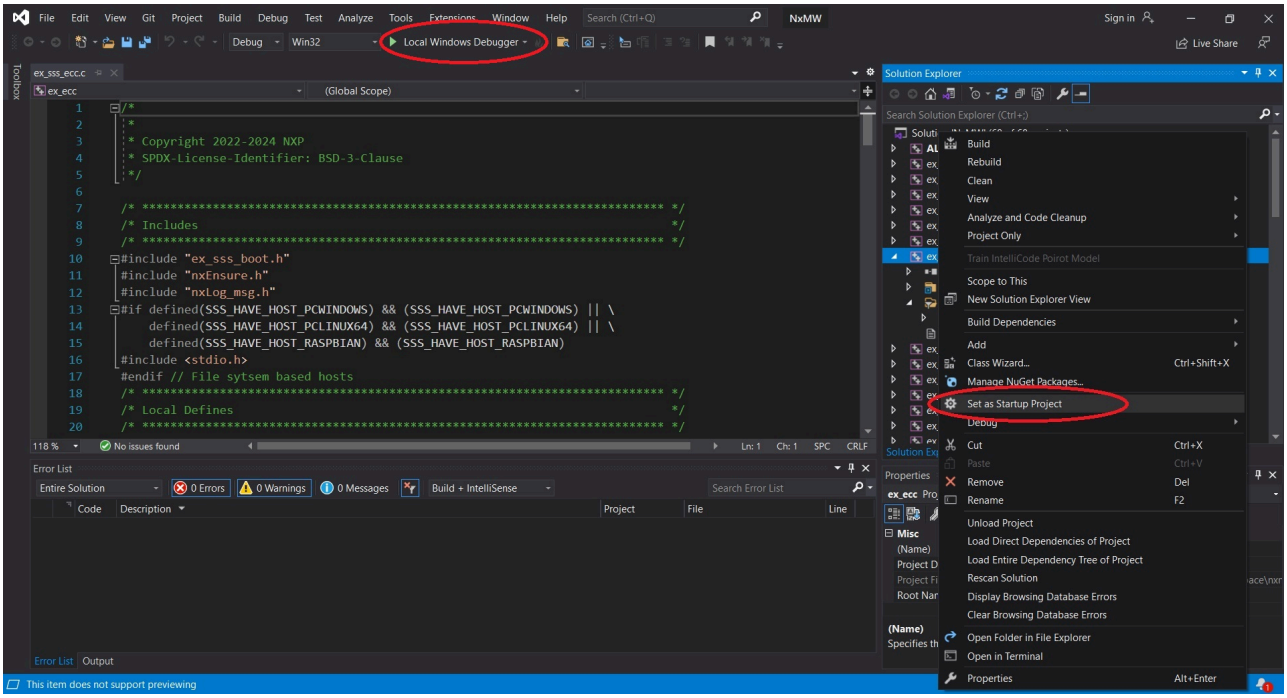
- To build the demo:

```
cd nxmw_build/se_x86
cmake --build . --target <example_name>
```

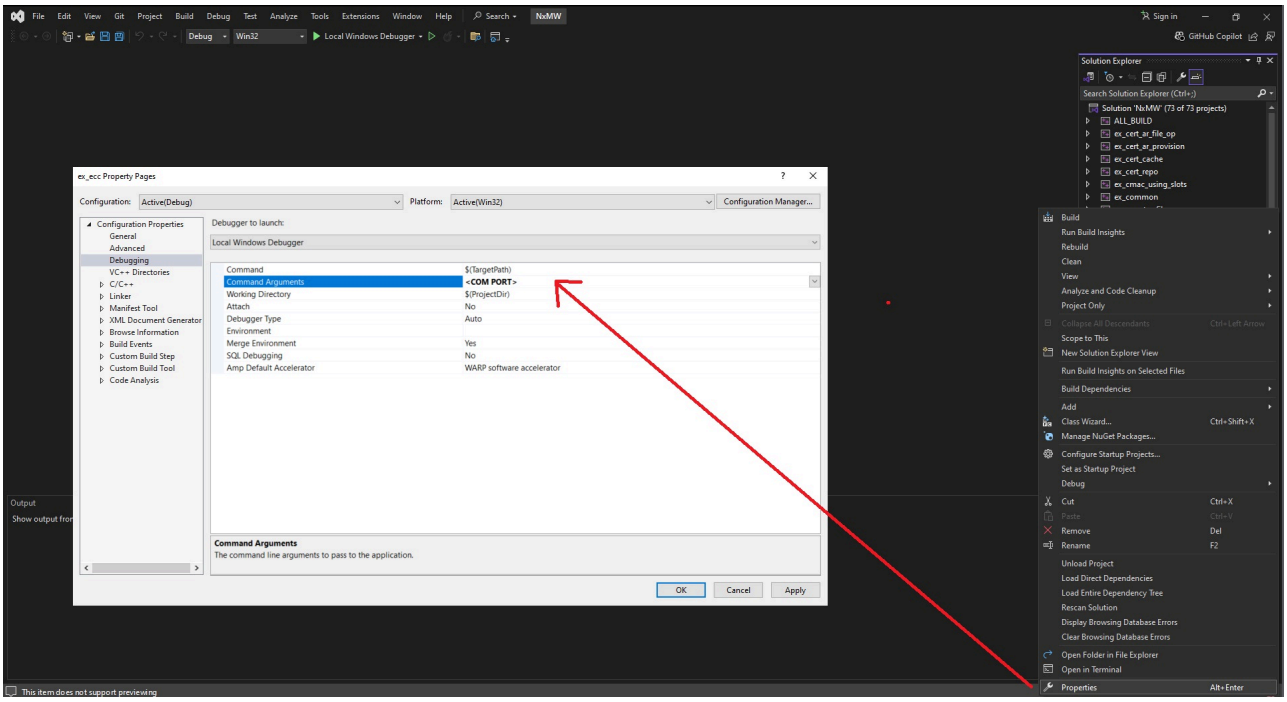
- Run demo as :

```
<example_name>.exe <COM_PORT>
```

- Debug demo
  – Set demo as startup project in Visual Studio and click on `Local Windows Debugger`



- Ensure that the com port is set in command line arguments



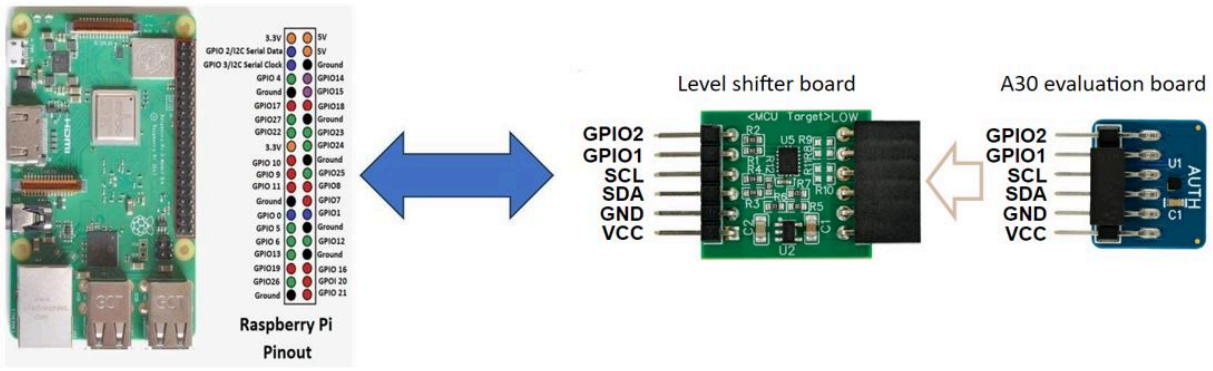
2.2 Raspberry Pi Build

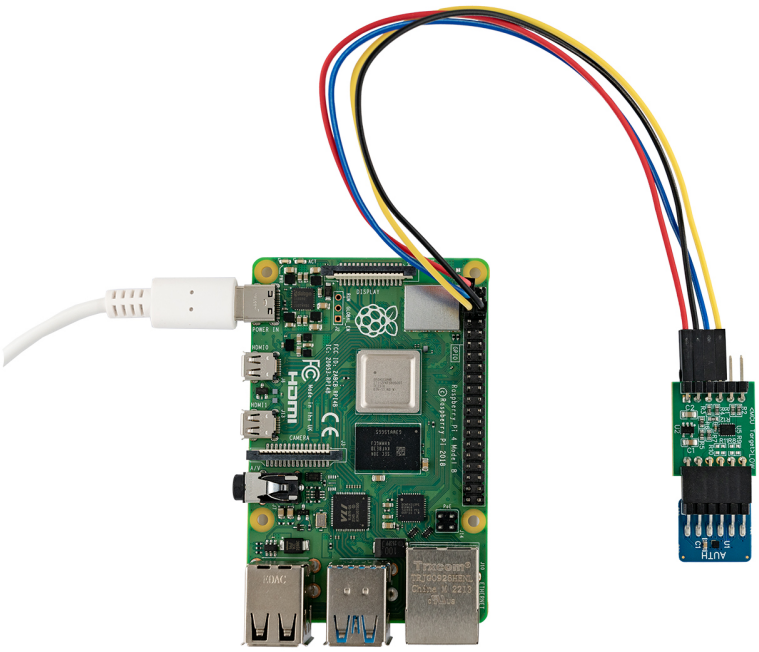
2.2.1 Getting started on Linux (Raspberry Pi 4)

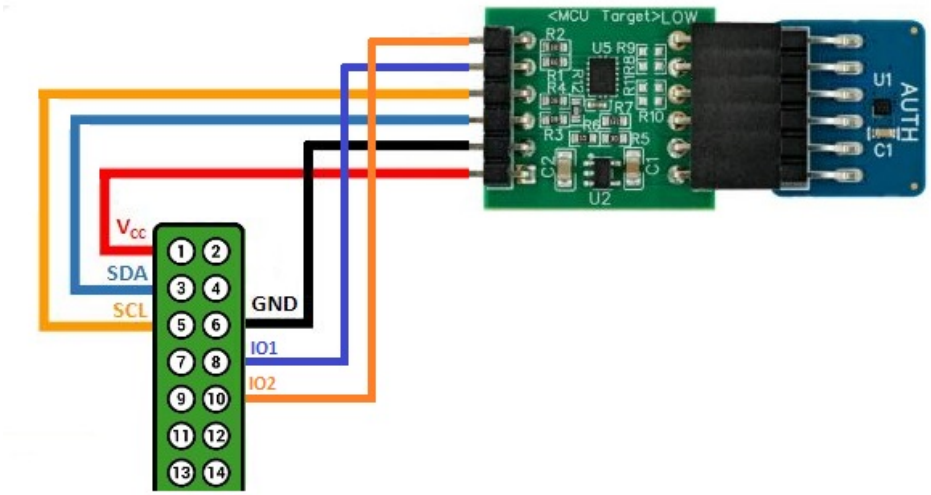
Run the following commands to build and run examples on Linux / Raspberry Pi 4

2.2.1.1 Prerequisites

- Linux should be running on the Raspberry Pi 4 development board (tested with Raspbian Buster (4.19.75-v7l +))
- Connect NX Secure Authenticator to I2C Linux of Linux machine (Example - consider Raspberry-Pi) as shown below,







- Enable I2C if not yet enabled on your board. (If `ls /sys/bus/i2c/devices` does not list `i2c-1`)
  - Run `sudo raspi-config`
  - Use the down arrow to select `Interface Options`.
  - Follow instructions and Enable I2C

- Install following packages required for the build

```
sudo apt-get install cmake cmake-curses-gui cmake-gui libssl-dev libsystemd-dev
```

2.2.1.2 Build Instructions

Execute the below commands to build and install the nx middleware

```
git clone  --recurse-submodules https://github.com/NXP/nxmw.git nxmw
cd nxmw/scripts
python create_cmake_projects.py
cd ../../nxmw_build/raspbian_native_nx_t1oi2c

# Change the cmake options if required by running ccmake .

cmake --build .
sudo make install
sudo ldconfig /usr/local/lib
```

If required cmake options can be changed and libraries can be rebuilt and installed as

```
cd nxmw_build/raspbian_native_nx_t1oi2c
ccmake .
```

Refer **CMake Options** for more details

Single demo (target) can be built using the following command:

```
cmake --build . --target <TARGET_NAME>
```

or

```
make <TARGET_NAME>
```

Run the required demo as:

```
./<TARGET_NAME>.bin
```

NOTE

```
A precondition for enabling Sigma-I authentication (setting the CMAKE option
 NXMW_Auth to SIGMA_I_Verifier or SIGMA_I_Prover) is to provision secure
 authenticator and the host with the required Sigma-I credentials using one of
 the following options:

1.Using of the NXP pre-provisioned A30 Sigma-I credentials.
- During manufacturing, A30 is trust-provisioned with a private EC key and the
 related A30 Leaf Certificate. (For further details see A30 application notes)

2. On-chip EC key generation and import of the related A30 certificates into the
 X.509 Certificate Repository.

3. Import of private EC key and import of the related A30 certificates
 into the X.509 Certificate Repository. (Refer the <xref href="../../demos/
nx/nx_Personalization/readme.md">**NX Personalization Example**</xref> or
```

```
 Personalization of SA Using NX CLI Tool section in <xref href="../../demos/nx/
nx_cli_tool/readme.md">**NX CLI Tool**</xref>).
```

2.3 CMake Options

2.3.1 NXMW_NX_Type: The NX Secure Authenticator Type

You can compile host library for different OS Applications of NX Secure Authenticator listed below.

`-DNXMW_NX_Type=None`: Compiling without any NX Type Support

`-DNXMW_NX_Type=NX_R_DA`: Application (DF name 0xD2760000850101)

`-DNXMW_NX_Type=NX_PICC`: MF (DF name 0xD2760000850100)

2.3.2 NXMW_Host: Host where the software stack is running

For e.g. Windows, PC Linux, Embedded Linux, FRDM-MCX like embedded platform

`-DNXMW_Host=PCWindows`: PC/Laptop Windows

`-DNXMW_Host=PCLinux64`: PC/Laptop Linux64

`-DNXMW_Host=lpcxpresso55s`: Embedded LPCXpresso55s

`-DNXMW_Host=Raspbian`: Embedded Linux on RaspBerry PI

`-DNXMW_Host=frdmmcxa153`: Embedded Freedom MCXA153

`-DNXMW_Host=frdmmcxn947`: Embedded Freedom MCXN947

2.3.3 NXMW_SMCOM: Communication Interface

How the host library communicates to the Secure Authenticator. This may be directly over an I2C interface on embedded platform. Or sometimes over Remote protocol like JRCP_V1_AM / VCOM from PC.

`-DNXMW_SMCOM=None`: Not using any Communication layer

`-DNXMW_SMCOM=VCOM`: Virtual COM Port

`-DNXMW_SMCOM=T1oI2C_GP1_0`: GP Spec

`-DNXMW_SMCOM=PCSC`: CCID PC/SC reader interface

`-DNXMW_SMCOM=JRCP_V1_AM`: Socket Interface Implementation. This is the interface used by the clients to connect from Host PC to access manager (which is run as a server in the Linux PC)

2.3.4 NXMW_HostCrypto: Counterpart Crypto on Host

What is being used as a cryptographic library on the host. As of now only OpenSSL / mbedTLS is supported

`-DNXMW_HostCrypto=MBEDTLS`: Use mbedTLS as host crypto

`-DNXMW_HostCrypto=OPENSSL`: Use OpenSSL as host crypto

`-DNXMW_HostCrypto=None`: No Host Crypto Note, the security of configuring NX to be used without HostCrypto needs to be assessed from system security point of view

2.3.5 NXMW_RTOS: Choice of Operating system

Default would mean nothing special. i.e. Without any RTOS on embedded system, or default APIs on PC/Linux

`-DNXMW_RTOS=Default`: No specific RTOS. Either bare metal on embedded system or native Linux or Windows OS

`-DNXMW_RTOS=FreeRTOS`: Free RTOS for embedded systems

2.3.6 NXMW_Auth: NX Authentication

This settings is used by examples to connect using various options to authenticate with the NX SE.

`-DNXMW_Auth=None`: Use the default session (i.e. session less) login

`-DNXMW_Auth=SIGMA_I_Verifier`: SIGMA I Verifier

`-DNXMW_Auth=SIGMA_I_Prover`: SIGMA I Prover

`-DNXMW_Auth=SYMM_Auth`: Symmetric Authentication

2.3.7 NXMW_Log: Logging

Set the logging level using this setting

`-DNXMW_Log=Default`: Default Logging

`-DNXMW_Log=Verbose`: Very Verbose logging

`-DNXMW_Log=Silent`: Totally silent logging

2.3.8 CMAKE_BUILD_TYPE

Refer: https://cmake.org/cmake/help/latest/variable/CMAKE_BUILD_TYPE.html

For embedded builds, this choices sets optimization levels. For MSVC builds, build type is selected from IDE As well

`-DCMAKE_BUILD_TYPE=Debug`: For developer

`-DCMAKE_BUILD_TYPE=Release`: Optimization enabled and debug symbols removed

`-DCMAKE_BUILD_TYPE=RelWithDebInfo`: Optimization enabled but with debug symbols

`-DCMAKE_BUILD_TYPE=`: Empty Allowed

2.3.9 NXMW_Secure_Tunneling: Secure Tunneling (Secure Messaging)

Successful Symmetric authentication and SIGMA-I mutual authentication results in the establishment of session keys and session IVs. These are used to encrypt and integrity protect the payloads to be exchanged.

`-DNXMW_Secure_Tunneling=None`: Plain Text

`-DNXMW_Secure_Tunneling=NTAG_AES128_AES256_EV2`: NTAG AES - 128 or AES - 256 (EV2) Secure Channel. Only valid for Sigma-I. Host supports both AES - 128 and AES - 256. The secure channel security strength is selected based on the SE configuration.

`-DNXMW_Secure_Tunneling=NTAG_AES128_EV2`: Only NTAG AES - 128 (EV2) Secure Channel

`-DNXMW_Secure_Tunneling=NTAG_AES256_EV2`: Only NTAG AES - 256 (EV2) Secure Channel

2.3.10 NXMW_Auth_Asymm_Host_PK_Cache: Host public key cache**

Support a cache of validated public keys and parent certificates on host. This is utilized to accelerate protocol execution time by removing the need to validate public key and certificates that have been previously verified.

Secure authenticator cache is enabled by Cmd.SetConfiguration. Refer **Enable Certificate Cache Example** for more information.

`-DNXMW_Auth_Asymm_Host_PK_Cache=Disabled`: Host's Public Key And Parent Certificates Cache Disabled

`-DNXMW_Auth_Asymm_Host_PK_Cache=Enabled`: Host's Public Key And Parent Certificates Cache Enabled

2.3.11 NXMW_Auth_Asymm_Cert_Repo_Id: Certificate Repository Id

Certificate Repository Id is used to identify certificate repository. Used in both personalization and demos with Sigma-I authentication. In personalization, it indicates repository to be initialized. In demos, it indicates repository to be used for Sigma-I authentication

`-DNXMW_Auth_Asymm_Cert_Repo_Id=0`: Certificate Repository 0

`-DNXMW_Auth_Asymm_Cert_Repo_Id=1`: Certificate Repository 1

`-DNXMW_Auth_Asymm_Cert_Repo_Id=2`: Certificate Repository 2

`-DNXMW_Auth_Asymm_Cert_Repo_Id=3`: Certificate Repository 3

`-DNXMW_Auth_Asymm_Cert_Repo_Id=4`: Certificate Repository 4

`-DNXMW_Auth_Asymm_Cert_Repo_Id=5`: Certificate Repository 5

`-DNXMW_Auth_Asymm_Cert_Repo_Id=6`: Certificate Repository 6

`-DNXMW_Auth_Asymm_Cert_Repo_Id=7`: Certificate Repository 7

2.3.12 NXMW_Auth_Asymm_Cert_SK_Id: Certificate Private Key Id

Id of ECC private key associated with this repository. Used in personalization for Sigma-I.

`-DNXMW_Auth_Asymm_Cert_SK_Id=0`: Certificate Private KeyId 0

`-DNXMW_Auth_Asymm_Cert_SK_Id=1`: Certificate Private KeyId 1

`-DNXMW_Auth_Asymm_Cert_SK_Id=2`: Certificate Private KeyId 2

`-DNXMW_Auth_Asymm_Cert_SK_Id=3`: Certificate Private KeyId 3

`-DNXMW_Auth_Asymm_Cert_SK_Id=4`: Certificate Private KeyId 4

2.3.13 NXMW_Auth_Asymm_CA_Root_Key_Id: Key ID of CA Root Public Key

Id of CA root public key associated with this repository. Used in personalization for Sigma-I.

`-DNXMW_Auth_Asymm_CA_Root_Key_Id=0`: CA Root KeyId 0

`-DNXMW_Auth_Asymm_CA_Root_Key_Id=1`: CA Root KeyId 1

`-DNXMW_Auth_Asymm_CA_Root_Key_Id=2`: CA Root KeyId 2

`-DNXMW_Auth_Asymm_CA_Root_Key_Id=3`: CA Root KeyId 3

`-DNXMW_Auth_Asymm_CA_Root_Key_Id=4`: CA Root KeyId 4

2.3.14 NXMW_Auth_Symm_App_Key_Id: Application Key ID

Indicate application key which is used in symmetric authentication.

`-DNXMW_Auth_Symm_App_Key_Id=0`: Application KeyId 0

`-DNXMW_Auth_Symm_App_Key_Id=1`: Application KeyId 1

`-DNXMW_Auth_Symm_App_Key_Id=2`: Application KeyId 2

`-DNXMW_Auth_Symm_App_Key_Id=3`: Application KeyId 3

`-DNXMW_Auth_Symm_App_Key_Id=4`: Application KeyId 4

2.3.15 NXMW_Auth_Asymm_Host_Curve: Host EC domain curve type

EC domain curve used for session key generation and session signature. Used in demos with Sigma-I authentication.

`-DNXMW_Auth_Asymm_Host_Curve=NIST_P`: EC Curve NIST-P

`-DNXMW_Auth_Asymm_Host_Curve=BRAINPOOL`: EC Curve Brainpool

2.3.16 NXMW_OpenSSL: For PC, OpenSSL version to pick up

On Linux based builds, this option has no impact, because the build system picks up the default available/ installed OpenSSL from the system directly. Also, this option has no impact if NXMW_HostCrypto is not selected as OPENSSL.

`-DNXMW_OpenSSL=1_1_1`: Use latest 1.1.1 version (Only applicable on PC)

`-DNXMW_OpenSSL=3_0`: Use 3.0 version (Only applicable on PC)

2.3.17 NXMW_MBedTLS: Which MBedTLS version to choose

This option has no impact if NXMW_HostCrypto is not selected as MBEDTLS.

`-DNXMW_MBedTLS=2_X`: Use 2.X version

`-DNXMW_MBedTLS=3_X`: Use 3.X version

2.3.18 NXMW_Auth_Symm_Diversify: Diversification of symmetric authentication key

When enabled, key used for symmetric authentication is diversification key derived from master key.

Otherwise master key is used.

`-DNXMW_Auth_Symm_Diversify=Disabled`: Symm Auth Key Diversification Disabled

`-DNXMW_Auth_Symm_Diversify=Enabled`: Symm Auth Key Diversification Enabled

2.3.19 NXMW_All_Auth_Code: Enable all authentication code

When enabled, all the authentication code is enabled in nx library.

`-DNXMW_All_Auth_Code=Disabled`: Enable only required authentication code (Based on NXMW_Auth Cmake option)

`-DNXMW_All_Auth_Code=Enabled`: Enable all authentication code

2.3.20 NXMW_mbedTLS_ALT: ALT Engine implementation for mbedTLS

When set to None, mbedTLS would not use ALT Implementation to connect to / use Secure Authenticator. This needs to be set to PSA for PSA example over SSS APIs

`-DNXMW_mbedTLS_ALT=SSS`: Enable SSS as ALT

`-DNXMW_mbedTLS_ALT=PSA`: Enable TF-M based on PSA as ALT

`-DNXMW_mbedTLS_ALT=None`: Not using any mbedTLS_ALT

NOTE: When this is set as PSA, cloud demos can not work with mbedTLS

2.3.21 NXMW_SA_Type: Enable host certificates of A30 for Sigma-I Authentication

When Secure Authenticator type is selected, respective host certificates are enabled in NX library.

`-DNXMW_SA_Type=A30`: Enable A30 host cert for Sigma-I authentication

`-DNXMW_SA_Type=NTAG_X_DNA`: Enable NTAG_X_DNA host cert for Sigma-I authentication

`-DNXMW_SA_Type=NXP_INT_CONFIG`: Enable NXP_INT_CONFIG host cert for Sigma-I authentication

`-DNXMW_SA_Type=Other`: Enable Other host cert for Sigma-I authentication

3 NX Middleware Stack

NX Middleware provides the necessary interfaces, examples, demos for NX Secure Authenticator (SA) IC. The **nxmw/lib** folder contains necessary files required to connect / use the NX secure authenticator. The middleware also has crypto examples (**demos**), plugins (OpenSSL engine, provider) (**plugin**)

Below section describe the secure session and the APIs exposed by the NX middleware stack to access NX secure authenticator. And finally the section describing on writing your own application.

3.1 NX Secure Authenticator session

Before starting any operation, the host is required to establish an authenticated session to NX Secure Authenticator.

Supported authenticated sessions are

```
- None
- Sigma-I-Verifier
- Sigma-I-Prover
- Symmetric Authentication
```

After the successful authentication, commands are required to be sent using any of the following Secure Messaging modes

```
- None
- NTAG_AES128_EV2.
- NTAG_AES256_EV2.
- NTAG_AES128_AES256_EV2 (Only for Sigma-I)
```

All of the above session authentication and command encryption modes can be selected using the cmake options. For more details on cmake options - refer **cmake options**,

Name	Value
CMAKE_BUILD_TYPE	Debug
CMAKE_CONFIGURATION_TYPES	Debug;Release;MinSizeRel;RelWithDebInfo
CMAKE_INSTALL_PREFIX	C:/Program Files (x86)/NxMW
NXMW_All_Auth_Code	Disabled
NXMW_Auth	SIGMA_I_Verifier
NXMW_Auth_Asymm_CA_Root_Key_Id	0
NXMW_Auth_Asymm_Cert_Repo_Id	0
NXMW_Auth_Asymm_Cert_SK_Id	0
NXMW_Auth_Asymm_Host_Curve	NIST_P
NXMW_Auth_Asymm_Host_PK_Cache	Enabled
NXMW_Auth_Symm_App_Key_Id	0
NXMW_Auth_Symm_Diversify	Disabled
NXMW_Host	PCWindows
NXMW_HostCrypto	MBEDTLS
NXMW_Log	Default
NXMW_MBedTLS	2_X
NXMW_NX_Type	NX_R_DA
NXMW_OpenSSL	1_1_1
NXMW_RTOS	Default
NXMW_SA_Type	A30
NXMW_SMCOM	VCOM
NXMW_Secure_Tunneling	NTAG_AES128_EV2
NXMW_mbedTLS_ALT	None
NXPInternal	☑
NXTST_FTR_TEST_COUNT	☐
WithCodeCoverage	☐
WithSharedLIB	☐

Following table shows valid combinations of session auth and secure tunneling modes.

AUTHENTICATION / SECURE TUNNELING	Sigma-I-Verifier	Sigma-I-Prover	None	Symm Auth
None	✗	✗	✓	✗
NTAG_AES128_AES256_EV2	✓	✓	✗	✗
NTAG_AES128_EV2	✓	✓	✗	✓
NTAG_AES256_EV2	✓	✓	✗	✓

To setup Sigma-I session, host will require and verify Secure Authenticator's certificate chain. Optionally, host can cache the public key in the leaf certificate and also the leaf certificate's value. This is utilized to accelerate protocol execution time by removing the need to validate public key certificates that have been previously verified.

The host can also cache Secure Authenticator's parent certificates. This can accelerate protocol execution time by removing the need to require parent certificate that have been previously verified from Secure Authenticator.

The cache feature can be enabled/disabled by cmake option NXMW_Auth_Asymm_Host_PK_Cache. (Maximum supported = 5).

Also, to setup Sigma-I session, host will provide its certificates chains to Secure Authenticator for verification. The certificates chains are stored based on the host OS type.

On Windows / Linux systems, the middleware will looks for certificate chain in following priority:

```
- Folder indicated by ENV variable "NX_AUTH_CERT_DIR"
- Folder "C:\nxp\configuration\cert_depth3_x509_rev1\"(Windows) or "/tmp/
configuration/cert_depth3_x509_rev1/" (Linux)
- Hard coded certificates defined in file: "nxmw/lib/sss/inc/
fsl_sss_nx_auth_keys.h"
```

Other micro-controllers:

```
- Hard coded certificates defined in file: "nxmw/lib/sss/inc/
fsl_sss_nx_auth_keys.h"
```

Note: The hard coded certificates and related private keys are stored in plain text for demonstration purposes only. During actual product deployment, customer has to adopt secure means as per their security needs (note that the potential issue is more with the private keys than certificates).

Note: T=1oI2C: When using the T=1oI2C smcom interface, at the beginning of session open, we do a dummy read (function - `phNxpEse_clearReadBuffer`) from the secure authenticator if any previous session data is pending and discard the data.

Note: When using Sigma I with supported MCUs, the operations take some time. Hence we need to disable the watchdog timer so that no error is thrown. For more information on this, (Refer: **Set Config Demo**).

3.2 NX Middleware APIs

3.2.1 Session Open APIs

Before starting any operation, the host is required to establish an authenticated session to NX Secure Authenticator.

Use the following APIs to open the session to NX secure authenticator.

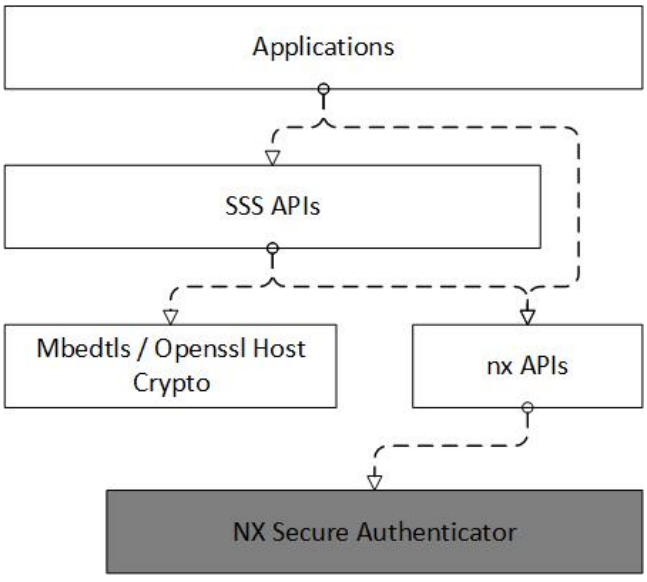
```
/*
Initialize authentication parameters for sigma auth.
*/
sss_status_t nx_init_conn_context_sigma_auth(nx_connect_ctx_t *nx_conn_ctx,
    nx_auth_type_t auth_type,
    nx_secure_symm_type_t secure_tunnel_type,
    sss_cipher_type_t host_cert_curve_type,
    sss_cipher_type_t host_ephem_curve_type,
    auth_cache_type_t cache_type,
    auth_compress_type_t compress_type,
    uint8_t se_cert_repo_id,
    uint16_t cert_ac_map);

OR

/*
    Initialize authentication parameters for symmetric auth.
*/
sss_status_t nx_init_conn_context_symm_auth(nx_connect_ctx_t *nx_conn_ctx,
    nx_auth_type_t auth_type,
```

```
    nx_secure_symm_type_t secure_tunnel_type,
    uint8_t key_no,
    bool pcdcap2_flag);

/*
    Initialize/ Prepare host crypto for authentication set up.
*/
sss_status_t nx_prepare_host_for_auth(
    sss_session_t *host_session, sss_key_store_t *host_ks, nx_connect_ctx_t
 *nx_conn_ctx);

/* Open session to NX Secure authenticator */

sss_status_t sss_session_open(sss_session_t *session,
sss_type_t subsystem,
uint32_t application_id,
sss_connection_type_t connection_type,
void *connectionData);

/* Create key store (Used in SSS APIs) */

sss_status_t sss_key_store_context_init(sss_key_store_t *keyStore, sss_session_t
 *session);

sss_status_t sss_key_store_allocate(sss_key_store_t *keyStore, uint32_t
 keyStoreId);
```

3.2.2 Crypto APIs

NX Middleware provides 2 set of crypto APIs to access the NX secure authenticator.

```
- SSS APIs
- nx APIs
```



3.2.2.1 SSS APIs

Not all NX Secure Authenticator features are covered using SSS APIs. Only the crypto functionality which can be switched between host and Nx secure authenticator are supported using SSS APIs. For all other use cases, use Nx APIs.

`SSS` is an acronym for **S**ecure **S**ub **S**ystem

The SSS APIs are functional APIs to abstract the access to various types of Cryptographic Sub Systems. Such secure subsystems could be (but not limited to):

```
- Secure Authenticators
- Software crypto
- Cryptographic HWs
```

When using SSS APIs, all the keys are referenced using the key Object. `sss_object_t`. Use the following APIs to create / initialize the key object.

```
sss_status_t sss_key_object_init(sss_object_t *keyObject, sss_key_store_t
 *keyStore);

sss_status_t sss_key_object_allocate_handle(sss_object_t *keyObject,
    uint32_t keyId,
    sss_key_part_t keyPart,
    sss_cipher_type_t cipherType,
    size_t keyByteLenMax,
    uint32_t options);
```

The key store parameter in the `sss_key_object_init` api will decide where the key has to be stored (Host or Nx Secure Authenticator).

Now, apart from the dedicated key-stores for asymmetric and symmetric keys, Nx Secure Authenticator also has two types of buffers- static and transient which can be used to store any data including keys. The cipherType parameter in `sss_key_object_allocate_handle` has been designed with the flexibility to incorporate this functionality too- the enum value kSSS_CipherType_BufferSlots for cipherType corresponds to the these internal buffers. In that case, the keyId would be interpreted as the unique Id for the internal buffer in the Nx Secure Authenticator. Other values for cipherType shall be interpreted as the natural key stores in Nx Secure Authenticator or the Host.

Demos like `ex-sss-ecdh`, `ex-sss-md-using-slots` can be referred to understand how the internal buffers are used.

Refer `fsl_sss_api.h` for complete set of SSS APIs supported.

3.2.2.2 NX APIs

These are the low level apis which create / encrypt / decrypt / transceive the APDUs of the NX secure authenticator.

Refer `nx_apdu.h` for complete set of Nx APIs supported.

3.2.3 Write Own Applications

A new application can be created in 2 ways.

3.2.4 Using ex_common code

Nx middleware has ex_common code implemented which does all the necessary session open task on behalf of the application.

Applications need to include the header file ex_sss_main_inc.h and start the application specific code from function - `ex_sss_entry`. (The authentication / connection parameters are decided by the cmake options / feature file contents)

```
#include <ex_sss_main_inc.h>

sss_status_t ex_sss_entry(ex_sss_boot_ctx_t *pCtx)
{
    // Application specific part
}
```

Library to be linked - **ex_common**.

All examples in demos folder use the above method.

OR

3.2.5 Use SSS APIs to open session

If the application needs better control of session, session open can be done at the start of the application.

Include the following header files

```
#include <fsl_sss_api.h>
#include <fsl_sss_nx_auth.h>
```

Refer **Session Open APIs** section above.

Library to be linked - **SSSAPIs**.

On successful session open, using SSS APIs or NX APIs, the required crypto operation can be performed using NX secure authenticator.

Following section shows the sequence of SSS APIs required for some common crypto operations.

3.2.6 Example crypto operations

3.2.7 Generating Asymmetric key

```
   sss_status_t sss_key_object_init(sss_object_t *keyObject, sss_key_store_t
*keyStore);

   sss_status_t sss_key_object_allocate_handle(sss_object_t *keyObject,
       uint32_t keyId,
       sss_key_part_t keyPart,
       sss_cipher_type_t cipherType,
       size_t keyByteLenMax,
       uint32_t options);

   sss_status_t sss_key_store_generate_key(sss_key_store_t *keyStore,
       sss_object_t *keyObject,
       size_t keyBitLen,
```

```
        void *options);
```

3.2.8 Set key

```
   sss_status_t sss_key_object_init(sss_object_t *keyObject, sss_key_store_t
*keyStore);

   sss_status_t sss_key_object_allocate_handle(sss_object_t *keyObject,
       uint32_t keyId,
       sss_key_part_t keyPart,
       sss_cipher_type_t cipherType,
       size_t keyByteLenMax,
       uint32_t options);

   sss_status_t sss_key_store_set_key(sss_key_store_t *keyStore,
       sss_object_t *keyObject,
       const uint8_t *data,
       size_t dataLen,
       size_t keyBitLen,
       void *options,
       size_t optionsLen);
```

4 Demos

4.1 DEMO List

SSS APIs Examples

Demo	Description
cert_access_right	cert_access_right Example
cert_cache	cert_cache Example
cert_repo	cert_repo Example
cmac_using_slots	cmac_using_slots Example
counter_file	counter_file Example
diversify_key_perso	diversify_key_perso Example
dual_interfaces	dual_interfaces Example
ecc	ecc Example
ecdh	ecdh Example
fileMgnt	fileMgnt Example
getCardUID	getCardUID Example
getConfig	getConfig Example
getVersion	getVersion Example
gpio	gpio Example
gpio_notif	gpio_notif Example
hkdf	hkdf Example

Demo	Description
hmac	hmac Example
md	md Example
md_using_slots	md_using_slots Example
multiple_symm_auth	multiple_symm_auth Example
nx_cli_tool	nx_cli_tool Example
nx_deep_pwr_down	nx_deep_pwr_down Example
nx_Minimal	nx_Minimal Example
nx_Personalization	nx_Personalization Example
nx_release_req_cmd	nx_release_req_cmd Example
nx_tool_setconfig	nx_tool_setconfig Example
originality_check	originality_check Example
rng	rng Example
sdm	sdm Example
setConfig	setConfig Example
symmetric	symmetric Example
tls_client	tls_client Example
update_key	update_key Example
usb_c	usb_c Example
nx_access_manager	nx_access_manager Example
vcom	vcom Example
ecc_standalone	ecc_standalone Example
host co_processor	host co-processor Example
sa_qi	qi authenticator Example
multiple sessions	multiple session Example
mbedtls_3_x_alt	mbedtls_3_x_alt Example

Cloud connectivity Examples

Demo	Description
aws with linux	aws with Linux
aws with freertos	aws with freertos

Openssl Engine Examples

Demo	Description
Openssl Engine	Openssl-Engine

Openssl provider Examples

Demo	Description
Openssl Provider	Openssl-Provider

Platform Security Architecture

Demo	Description
psa	psa

4.2 Utilities

4.2.1 NX Personalization Example

This project is used to provision keys / certificates for NX Secure Authenticator. These keys and certificates are used to setup Sigma-I authentication.

It must be noted that by this demo, the default symmetric keys are kept untouched. For actual deployment, customer should also provision the symmetric keys, or disable the symmetric authentication.

Provisioning for symmetric authentication is done by ex_update_key or ex_diversify_key_perso, explained in `ex-sss-update-key` and `diversify-key-perso`

The example will provision the below keys / certificates

- Write CA root public key
- Create certificate repository
- Write the device leaf key pair and device leaf certificate chain
- Write device certificate template and mapping table
- Activate the certificate repository

On Windows or Linux Hosts, the example will look for the certificates/keypair in following priority

- Folder indicated by ENV variable "NX_AUTH_CERT_DIR"
- Folder : "C:\nxp\configuration\cert_depth3_x509_rev1\"(Windows) OR "/tmp/configuration/cert_depth3_x509_rev1/" (Linux)

Hard coded certificates defined in `nxmw/demos/nx/nx_Personalization/nx_Personalization.h`

MCUs

Hard coded certificates defined in `nxmw/demos/nx/nx_Personalization/nx_Personalization.h`

```
**IMPORTANT**

The hard-coded certificates and related private keys are being stored in
plain text for demonstration purposes only. During actual product
deployment, customer has to adopt secure means as per their security
needs (note that the potential issue is more with the private keys than
certificates).
```

4.2.1.1 Building the example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
- Project - `nx_Personalization`

- Make sure to select "NXMW_Auth=SYMM_Auth" in cmake options.

NXMW_Secure_Tunneling should be selected according to the provisioned symmetric key (Refer `ex-sss-update-key`). It can be "NXMW_Secure_Tunneling=NTAG_AES128_EV2" or "NXMW_Secure_Tunneling=NTAG_AES256_EV2".

By default, the symmetric key would be AES-128 all zeros.

4.2.1.2 How to use

Run the tool as:

```
./nx_Personalization -c [{bp|nistp}] -m [AC bitmap] [port_name]

Default options:
- curve : "bp" (brainpool256)
- AC Bitmap : 0x3FFFF
- port_name : depending on the NXMW_SMCOM CMake option (COM7 | /dev/ttyACM0)
```

Note that all the parameters are optional here. If no parameter is supplied, it will take the default options as mentioned above.

4.2.2 NX-CLI Tool

This tool is a command line utility to evaluate the secure authenticator on Windows / Linux environment.

4.2.2.1 Build

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
  - Select CMake options:
    - `NXMW_All_Auth_Code=Enabled`
  - Project:`nxclitool`

4.2.2.2 About the Tool

This tool is a command line utility to evaluate the secure authenticator on Windows / Linux environment. The tool can be used to perform various cryptographic operations / personalization of SA.

It supports following commands:

Command	Description
connect	Connect to SA
disconnect	Disconnect from SA
get-uid	Get UID from SA
genkey	Generates ECC Key in SA and stores the public key to a file
get-ref-key	Generates a reference key
rand	Generate random numbers from SA
setkey	Set a private key inside SA
certrepo-*	Certificate repository commands
create-bin	Creates a standard data file inside SA

Command	Description
setbin	Set data to a standard data file inside SA
getbin	Get data from standard data file in SA
list-fileid	Fetches the list of file IDs inside SA
list-eckey	Fetches the list and properties of EC keys inside SA
set-i2c_mgnt	Set I2C configuration
set-cert_mgnt	Set certificate configuration

Refer individual command sections for more details.

Ensure that connect command is called before performing any crypto operations.

4.2.2.3 Connect/Disconnect Command

Connect command creates a temporary file with data related to connection context. (The actual connection to SA is done only in next crypto operation call. And after every crypto operation call, session is closed.) The disconnect command will delete the temporary file created during connect call.

Command Format:

```
nxclitool connect [OPTIONS] nxclitool disconnect
```

Options:

Common options required for all auth types:

- **-smcom**: Host device to connect to. Accepted values:
  - `pcsc`: To connect to the simulator via pcsc
  - `vcom`: To connect to the SA via vcom
  - `t1oi2c`: To connect to the SA via T=1oI2C
- **-port**: Port of the host device. Set the value to "default" to use the default port. If skipped, default port will be used
- **-auth**: Authentication type. Accepted values:
  - `none`
  - `sigma_i_verifier`
  - `sigma_i_prover`
  - `symmetric`
- **-sctunn**: Secure tunneling type. Accepted values:
  - `none`
  - `ntag_aes128_aes256_ev2`
  - `ccm_aes256`
  - `ntag_aes128_ev2`
  - `ntag_aes256_ev2`

Options required for symmetric auth types:

- **-keyid**: Key ID for symmetric auth. Accepted values: 0x00 - 0x04

Options required for sigma auth types:

- **-curve**: ECC Curve type for sigma auth type. Accepted values:
  - `brainpoolP256r1`

  - prime256v1
  - na
- **-repoid**: Repository ID in hex format for sigma auth type

Example

Command to connect with symmetric auth type:

```
./nxclitool connect -smcom t1oi2c -port "/dev/i2c-1" -auth symmetric -sctunn
 ntag_aes128_ev2 -keyid 0x00

<... Any Crypto Operation ...>

nxclitool disconnect
```

For sigma auth type:

```
./nxclitool connect -smcom t1oi2c -port "/dev/i2c-1" -auth sigma_i_verifier -
sctunn ntag_aes128_ev2 -repoid 0x01 -curve prime256v1

<... Any Crypto Operation ...>

nxclitool disconnect
```

4.2.2.4 Get UID Command

Get UID from Secure Authenticator. This command does not require any options.

Command Format

```
nxclitool get-uid
```

Example

```
./nxclitool connect -smcom t1oi2c -port "/dev/i2c-1" -auth symmetric -sctunn
 ntag_aes128_ev2 -keyid 0x00
./nxclitool get-uid
./nxclitool disconnect
```

4.2.2.5 List EC Key Command

Fetches the list and properties of EC keys inside Secure Authenticator. This command does not require any options.

Command Format

```
nxclitool list-eckey
```

Example

```
./nxclitool connect -smcom t1oi2c -port "/dev/i2c-1" -auth symmetric -sctunn
 ntag_aes128_ev2 -keyid 0x00
./nxclitool list-eckey
./nxclitool disconnect
```

4.2.2.6 Random Generation Command

Generates specified number of random bytes from SA.

Command Format

```
nxclitool rand -bytes [NO_OF_BYTES]
```

Example

```
./nxclitool connect -smcom t1oi2c -port "/dev/i2c-1" -auth symmetric -sctunn
 ntag_aes128_ev2 -keyid 0x00
./nxclitool rand -bytes 20
./nxclitool disconnect
```

4.2.2.7 Generate EC Key Command

Generates ECC key (curve type as input parameter). If an optional file path is provided, the public key will be stored in PEM format.

Command Format

```
nxclitool genkey [OPTIONS]
```

Options

- **-keyid**: Key ID for asymmetric keypair generation should be in HEX format. Range: **0x00 to 0x04**
- **-curve**: Curve type for keypair generation. Accepted values:
  - `brainpoolP256r1`: ECC curve type BRAINPOOL_256
  - `prime256v1`: ECC curve type NIST_P256
- **-out**: Store the public key to a file in PEM format (optional argument).

Example

```
./nxclitool connect -smcom t1oi2c -port "/dev/i2c-1" -auth symmetric -sctunn
 ntag_aes128_ev2 -keyid 0x00
./nxclitool genkey -keyid 0x02 -curve brainpoolP256r1 -out pub.key
./nxclitool disconnect
```

4.2.2.8 Set Key Command

Set a private key inside SA (Key ID and key file in PEM format as input parameter).

Command Format

```
nxclitool setkey [OPTIONS]
```

Options

- **-curve**: Curve type for keypair generation. Accepted values
  - `brainpoolP256r1`: ECC curve type BRAINPOOL_256
  - `prime256v1`: ECC curve type NIST_P256
- **-enable**: Operation to be performed by the key. Accepted values:
  - `none`
  - `ecdh`

  - `sign`
- **-in**: Path to the input certificate/key in PEM format
- **-keyid**: Key ID for asymmetric keypair generation should be in HEX format. Range: **0x00 to 0x04**

Example

```
./nxclitool connect -smcom t1oi2c -port "/dev/i2c-1" -auth symmetric -sctunn
 ntag_aes128_ev2 -keyid 0x00
./nxclitool setkey -keyid 0x02 -curve prime256v1 -in key.pem -enable sign
./nxclitool disconnect
```

4.2.2.9 Get Reference Key Command

Generates a reference key using key ID (where the private key is stored in SA) and the public key in PEM format (Key ID and public key file in PEM format as input parameter).

Note: As there is no communication involved with SA for this command, there is no need to use the connect/disconnect command.

Command Format

```
nxclitool get-ref-key [OPTIONS]
```

Options

- **-in**: Path to the public key in PEM format
- **-keyid**: Key ID for asymmetric keypair generation should be in HEX format. Range: **0x00 to 0x04**
- **-out**: Store the public key to a file in PEM format (optional argument)

Example

```
nxclitool get-ref-key -keyid 0x02 -in pub.key -out ref.key
```

4.2.2.10 Create Certificate Repository Command

Creates a certificate repository in the SA.

Command Format

```
nxclitool certrepo-create [OPTIONS]
```

Options

- **-keyid**: ECC private key ID associated with the repository
- **-repoid**: Certificate Repository ID
- **-wcomm, -rcomm, -kcomm**: Write, Read, Known communication modes respectively, required to write to/read from the repository. Accepted values:
  - `full`
  - `mac`
  - `na`
  - `plain`
- **-waccess, -raccess**: Write and Read Access Rights respectively, required to write to/read from the repository. Accepted values:
  - `0x00 to 0x0C` Auth required

- 0x0D Free over I2C
- 0x0E Free Access
- 0x0F No Access

Example

```
./nxclitool connect -smcom t1oi2c -port "/dev/i2c-1" -auth symmetric -sctunn
 ntag_aes128_ev2 -keyid 0x00
./nxclitool certrepo-create -repoid 0x01 -keyid 0x01 -wcomm full -rcomm full -
waccess 0x0 -raccess 0x0 -kcomm na
./nxclitool disconnect
```

4.2.2.11 Activate Certificate Repository Command

Activates the certificate repository.

Command Format

```
nxclitool certrepo-activate [OPTIONS]
```

Options

- **-kcomm**: Known Communication Mode, required to write to/read from the repository. Accepted values:
  - full
  - mac
  - na
  - plain
- **-repoid**: Certificate Repository ID

Example

```
./nxclitool connect -smcom t1oi2c -port "/dev/i2c-1" -auth symmetric -sctunn
 ntag_aes128_ev2 -keyid 0x00
./nxclitool certrepo-activate -repoid 0x01 -kcomm na
./nxclitool disconnect
```

4.2.2.12 Reset Certificate Repository Command

Resets the certificates/keys loaded in the certificate repository.

Command Format

```
nxclitool certrepo-reset [OPTIONS]
```

Options:

- **-repoid**: Certificate Repository ID. Default: 0x03
- **-waccess, -raccess**: Write, Read Access Rights respectively, required to write to/read from the repository. Accepted values:
  - 0x00 to 0x0C Auth required
  - 0x0D Free over I2C
  - 0x0E Free Access

  - `0x0F` No Access
- **-wcomm, -rcomm, -kcomm**: Write, Read, Known communication modes respectively, required to write to/read from the repository. Accepted values:
  - `full`
  - `mac`
  - `na`
  - `plain`

Example

```
./nxclitool connect -smcom t1oi2c -port "/dev/i2c-1" -auth symmetric -sctunn
 ntag_aes128_ev2 -keyid 0x00
./nxclitool certrepo-reset -repoid 0x01 -wcomm full -rcomm full -waccess 0x0 -
raccess 0x0 -kcomm na
./nxclitool disconnect
```

4.2.2.13 Manage Certificate Repository commands

certrepo-load-key / certrepo-load-cert / certrepo-load-mapping commands can be used to manage a certificate repository in the SA.

Command Format

```
./nxclitool certrepo-load-key [OPTIONS]
./nxclitool certrepo-load-cert [OPTIONS]
./nxclitool certrepo-load-mapping [OPTIONS]
```

Options

- **-certlevel**: Level of the certificate. Required with **certrepo-load-cert**and **certrepo-load-mapping**. Accepted values:
  - `leaf`
  - `p1`
  - `p2`
  - `root`
- **-kcomm** : Known Communication Mode, required to write to/read from the repository. Required with **certrepo-load-cert**and **certrepo-load-mapping**. Accepted values:
  - `full`
  - `mac`
  - `na`
  - `plain`
- **-repoid**: Certificate Repository ID in hex format. Required with **certrepo-load-cert**and **certrepo-load-mapping**
- **-curve**: ECC Curve type for the key. Required with **certrepo-load-key**. Accepted values:
  - `brainpoolP256r1`
  - `prime256v1`
  - `na`
- **-in**: Path to the input certificate/key. Required with **certrepo-load-cert**, **certrepo-load-mapping** and **certrepo-load-key**
- **-keyid**: Key ID for setting Root CA Key. Required with **certrepo-load-key**
- **-certtype**: Certificate wrapping. Required with **certrepo-load-key**. Accepted values:
  - `pkcs7`

  - `x509`
- **-waccess, -raccess**: Write, Read Access Rights respectively, required to write to/read from the repository. Required with **certrepo-load-key**. Accepted values:
  - `0x00 to 0x0C` Auth required
  - `0x0D` Free over I2C
  - `0x0E` Free Access
  - `0x0F` No Access
- **-keytype**: Type of key to be loaded in the repository. Required with **certrepo-load-key**. Accepted values:
  - `leaf`: Device Leaf Key
  - `rootca`: Root CA Key

Example

```
./nxclitool connect -smcom t1oi2c -port "/dev/i2c-1" -auth symmetric -sctunn
 ntag_aes128_ev2 -keyid 0x00
./nxclitool certrepo-load-key -keytype rootca -keyid 0x00 -curve prime256v1 -
certtype pkcs7 -in host_root_certificate.der
./nxclitool disconnect
```

```
./nxclitool connect -smcom t1oi2c -port "/dev/i2c-1" -auth symmetric -sctunn
./ntag_aes128_ev2 -keyid 0x00 nxclitool certrepo-load-cert -repoid 0x01 -
certlevel leaf -kcomm na -in device_leaf_certificate.der
./nxclitool disconnect
```

```
./nxclitool connect -smcom t1oi2c -port "/dev/i2c-1" -auth symmetric -sctunn
./ntag_aes128_ev2 -keyid 0x00 nxclitool certrepo-load-mapping -repoid 0x01 -
certlevel leaf -kcomm na -in device_leaf_certificate.der
./nxclitool disconnect
```

4.2.2.14 Read Certificate Repository

Reads a certificate from the repository using **certrepo-read-cert** or the repository metadata using **certrepo-read-metadata**.

Command Format

```
nxclitool certrepo-read-cert [OPTIONS]
nxclitool certrepo-read-metadata [OPTIONS]
```

Options

- **-out**: File path where the certificate will be stored. Required with **certrepo-read-cert**
- **-kcomm** : Known Communication Mode for the repository. Required with **certrepo-read-cert**. Accepted values:
  - `full`
  - `mac`
  - `na`
  - `plain`
- **-repoid**: Certificate Repository ID. Required with both commands
- **-certlevel**: Level of the certificate. Required with **certrepo-read-cert**. Accepted values:
  - `leaf`
  - `p1`

- p2
- root

Example

```
./nxclitool connect -smcom t1oi2c -port "/dev/i2c-1" -auth symmetric -sctunn
 ntag_aes128_ev2 -keyid 0x00
./nxclitool certrepo-read-cert -repoid 0x01 -certlevel p2 -kcomm na -out
 cert.der
./nxclitool disconnect
```

```
./nxclitool connect -smcom t1oi2c -port "/dev/i2c-1" -auth symmetric -sctunn
 ntag_aes128_ev2 -keyid 0x00
./nxclitool certrepo-read-metadata -repoid 0x01
./nxclitool disconnect
```

4.2.2.15 Create Binary File Command

Creates a standard data file inside SA of the specified length. Length in bytes should not be more than 1024.

Command Format

```
nxclitool create-bin [OPTIONS]
```

Options

- **-bytes**: File size in bytes
- **-caccess, -raccess, -rwaccess, -waccess**: Change, Read, Read-Write, Write Access Rights respectively. Accepted values:
  - 0x00 to 0x0C: Auth required
  - 0x0D: Free over I2C
  - 0x0E: Free Access
  - 0x0F: No Access
- **-fcomm**: File communication mode. Accepted values:
  - full
  - mac
  - na
  - plain
- **-id**: File ID in hex format

Example

```
./nxclitool connect -smcom t1oi2c -port "/dev/i2c-1" -auth symmetric -sctunn
 ntag_aes128_ev2 -keyid 0x00
./nxclitool create-bin -id 0x04 -bytes 100 -raccess 0x0F -waccess 0x0F -rwaccess
 0x0F -caccess 0x0F -fcomm full
./nxclitool disconnect
```

4.2.2.16 List File IDs Command

Fetches the list of standard data file IDs inside SA.

Command Format

```
nxclitool list-fileid
```

Note: **list-fileid** command does not require any additional arguments.

Example

```
./nxclitool connect -smcom t1oi2c -port "/dev/i2c-1" -auth symmetric -sctunn
 ntag_aes128_ev2 -keyid 0x00
./nxclitool list-fileid
./nxclitool disconnect
```

4.2.2.17 Set Binary Command

Sets data from a file to a standard data file inside SA.

Command Format

```
nxclitool setbin [OPTIONS]
```

Options

- **[-bytes]**: No. of bytes to store (optional). Default: No. of bytes from offset to EOF
- **-id**: File ID in hex format
- **-in**: Path to the file to read
- **[-offset]**: Offset of data to read from and store at in SA (optional). Default: 0

Example

```
./nxclitool connect -smcom t1oi2c -port "/dev/i2c-1" -auth symmetric -sctunn
 ntag_aes128_ev2 -keyid 0x00
./nxclitool setbin -id 0x04 -bytes 12 -offset 10 -in data_in.txt
./nxclitool disconnect
```

Note: If **-offset** option is not specified, default value used is 0 (zero).

Note: If **-offset** option is specified, CLI tool will read data from input file from the start and store it in standard data file inside SA at the offset value specified.

Note: If **-bytes** option is not specified, default value used is the size of file in bytes.

4.2.2.18 Get Binary Command

Gets data from a standard data file inside SA and stores it in a file in file system.

Command Format

```
nxclitool getbin [OPTIONS]
```

Options

- **[-bytes]**: No. of bytes to store (optional). Default: No. of bytes from offset to EOF
- **-id**: File ID in hex format
- **[-offset]**: Offset of data to read from SA standard file (optional). Default: 0
- **[-out]**: Stores the fetched data to a file on this path (optional)

Example

```
./nxclitool connect -smcom t1oi2c -port "/dev/i2c-1" -auth symmetric -sctunn
 ntag_aes128_ev2 -keyid 0x00
./nxclitool getbin -id 0x04 -bytes 12 -offset 10 -out data_out.txt
./nxclitool disconnect
```

Note: If **-offset** option is not specified, default value used is 0 (zero).

Note: If **-offset** option is specified, CLI tool will read data from standard data file in the SA from the offset specified and store it in the output file.

Note: If **-bytes** option is not specified, default value used is the number of bytes from offset value to EOF of standard data file

4.2.2.19 Personalization of SA Using NX CLI Tool

- Connect to the SA.

```
nxclitool connect -smcom <SMCOM> -port <PORT_NAME> -auth <AUTH_NAME> -sctunn
 <SECURE_TUNNELING> -keyid <KEY_ID>
```

- Load host root CA certificate.

```
nxclitool certrepo-load-key -keytype rootca -keyid <KEY_ID> -curve <CURVE> -
certtype <CERTIFICATE_TYPE> -in <FILE_NAME>
```

- Load device leaf key pair.

```
nxclitool certrepo-load-key -keytype leaf -keyid <KEY_ID> -curve <CURVE> -
certtype <CERTIFICATE_TYPE> -in <FILE_NAME>
```

- Create a certificate repository.

```
nxclitool certrepo-create -repoid <REPO_ID> -keyid <KEY_ID> -wcomm full -rcomm
 full -waccess 0x0 -raccess 0x0 -kcomm na
```

- Load LEAF, P1 and P2 level device certificates.

```
nxclitool certrepo-load-cert -repoid <REPO_ID> -certlevel leaf -kcomm na -in
 <FILE_NAME>
nxclitool certrepo-load-cert -repoid <REPO_ID> -certlevel p1 -kcomm na -in
 <FILE_NAME>
nxclitool certrepo-load-cert -repoid <REPO_ID> -certlevel p2 -kcomm na -in
 <FILE_NAME>
```

- If the certificates are PKCS7 type, load host certificate mappings for LEAF, P1 and P2 levels. Skip this step if certificates are X509 type.

```
nxclitool certrepo-load-mapping -repoid <REPO_ID> -certlevel leaf -kcomm na -in
 <FILE_NAME>
nxclitool certrepo-load-mapping -repoid <REPO_ID> -certlevel p1 -kcomm na -in
 <FILE_NAME>
nxclitool certrepo-load-mapping -repoid <REPO_ID> -certlevel p2 -kcomm na -in
 <FILE_NAME>
```

- Activate the certificate repository.

```
nxclitool certrepo-activate -repoid <REPO_ID> -kcomm na
```

- Disconnect from the SA.

```
./nxclitool disconnect
```

To verify the personalization, set the variable **NX_AUTH_CERT_DIR** to the path to the certificates and run the ex_ecc example.

```
set NX_AUTH_CERT_DIR=<CERT_FOLDER_PATH>
<PATH_TO_EXECUTABLE>\ex_ecc.exe
```

4.2.2.20 Set-Config I2C Management

set config i2c management like i2c support, i2c address and select protocol options

Command Format

```
nxclitool set-i2c_mgnt [OPTIONS]
```

Options

- **-i2csupport**: I2C disabled/enabled
- **-i2caddr**: The address used for the I2C target (default 0x20):
- **-protocoloptions**: The crypto protocols supported over I2C, refer NTAGECC_RefArch:

Example

```
./nxclitool connect -smcom t1oi2c -port "/dev/i2c-1" -auth symmetric -sctunn
 ntag_aes128_ev2 -keyid 0x00
./nxclitool set-i2c_mgnt -i2csupport 0x01 -i2caddr 0x20 -protocoloptions 0x1B85
./nxclitool disconnect
```

4.2.2.21 Set-Config Certificate Management

set config Certificate management like LeafCacheSize, IntermCacheSize, FeatureSelection and ManageCertRepo-AccessCondition

Command Format

```
nxclitool set-cert_mgnt [OPTIONS]
```

Options

- **-leafcachesize**: End Leaf certificate cache size 0x01...0x08
- **-intermcachesize**: Intermediate certificate cache size 0x02...0x08
- **-featureselection**: Enable SIGMA-I cache
  - `0x01`: Enable SIGMA-I cache
  - `0x00`: Disable SIGMA-I cache
- **ManageCertRepoAccessCondition [OPTIONS]**
  - **-wcomm**: ManageCertRepo communication mode. Accepted values:
    - `full`

  - `mac`
  - `plain`
  - `na`
- **-waccess**: Write Access Rights respectively. Accepted values:
  - `0x00 to 0x0C`: Auth required
  - `0x0D`: Free over I2C
  - `0x0E`: Free Access
  - `0x0F`: No Access

Example

```
./nxclitool connect -smcom t1oi2c -port "/dev/i2c-1" -auth symmetric -sctunn
 ntag_aes128_ev2 -keyid 0x00
./nxclitool set-cert_mgnt -leafcachesize 0x08 -intermcachesize 0x08 -
featureselection 0x01 -wcomm plain -waccess 0x00
./nxclitool disconnect
```

4.2.3 Set Configuration Example

This project demonstrates a set configuration operation using NX APIs. It will set GPIO1 to output mode and GPIO2 to input mode. The communication mode for managing and reading gpio is set to full protect. The access condition for managing and reading gpio is set to 1.

Refer **Set Configuration Example**

4.2.3.1 About the Example

This example does a set configuration operation.

It uses the following APIs and data types:

```
`nx_SetConfig_GPIOMgmt()`
`nx_SetConfig_WatchdogTimerMgmt()`
```

4.2.3.2 Building the example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
  - Project - `ex_set_config`
  - Select NXMW_Auth to SIGMA_I_Verifier, SIGMA_I_Prover or SYMM_Auth
  - NXMW_Secure_Tunneling to NTAG_AES128_EV2, NTAG_AES256_EV2 or NTAG_AES128_AES256_EV2 (only with SIGMA_I_Verifier or SIGMA_I_Prover)
  - The authentication must be setup with access condition 0. This means App Master key for symmetric authentication. For Sigma-I authentication, this mean AC0 of certificate access right. It's defined with CA root key and reader certificate (or certificate chain).
  - Set watch dog timer by enabling macro SET_CONFIG_WATCHDOG_TIMER_ENABLE 1. Adjust timer value 0-60sec using macro SET_CONFIG_WATCHDOG_TIMER_HWDTVALUE SET_CONFIG_WATCHDOG_TIMER_AWDT1VALUE SET_CONFIG_WATCHDOG_TIMER_AWDT2VALUE

Refer - **ex_sss_set_config.h**

4.2.3.3 How to use

Run the tool as:

```
./ex_set_config
```

4.2.3.4 Console output

If everything is successful, the output will be similar to:

```
sss   :INFO :Session Open Succeed
App   :INFO :Running Set Configuration Example ex_sss_set_config.c
App   :INFO :Set GPIO1 to output and GPIO2 to input.
App   :INFO :Set ManageGPIO access condition to full protection and access
 condition 1.
App   :INFO :Set Configuration successful !!!
App   :INFO :ex_sss_set_config Example Success !!!...
App   :INFO :ex_sss Finished
```

4.2.4 NX Set Config Tool

This project is used to configure the Nx Secure Authenticator.

Options supported:

1. **gpio1mode**: Set GPIO1 to disabled, input, output, input tag tamper or down-stream power out
2. **gpio2mode**: Set GPIO2 to disabled, input or output
3. **gpio1Notif**: Set GPIO1 notification on authentication. It can be disabled, enabled for authentication or enabled for presence of NFC field
4. **gpio2Notif**: Set GPIO2 notification on authentication. It can be disabled, enabled for authentication or enabled for presence of NFC field
5. **gpioMgmtCM**: Set ManageGPIO communication mode
6. **gpioReadCM**: Set ReadGPIO communication mode
7. **gpioMgmtAC**: Set ManageGPIO access condition
8. **gpioReadAC**: Set ReadGPIO access condition
9. **cryptoCM**: Set Crypto API communication mode
10. **cryptoAC**: Set Crypto API access condition
11. **keypairCM**: Set ManageKeyPair communication mode
12. **keypairAC**: Set ManageKeyPair access condition
13. **caRootKeyCM**: Set ManageCARootKey communication mode
14. **caRootKeyAC**: Set ManageCARootKey access condition

4.2.4.1 Prerequisites

- Run with access condition 0 (AppMasterKey)
- For Sigma-I, bit0 in CA root key bitmap should be set.
- For Symm authentication, this demo should compiled with APP_KEY_ID0

4.2.4.2 Building the example

- Build NX middleware stack on Linux. Refer **Linux build**.

- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
  - Project - `nx_tool_setconfig`
  - Select NXMW_Auth to SIGMA_I_Verifier or SIGMA_I_Prover or SYMM_Auth
  - NXMW_Secure_Tunneling to NTAG_AES128_EV2 or NTAG_AES128_EV2 or NTAG_AES128_AES256_EV2 (Only with Sigma-I)

4.2.4.3 How to use

Run the tool as:

```
./nx_tool_setconfig <option_list> <port_name>
```

option_list is a list of supported configuration options, where at least one option must be supplied. Here is the list of supported options and their values:

- [-gpio1mode]
- [-gpio2mode]
- [-gpio1Notif]
- [-gpio2Notif]
- [-gpioMgmtCM]
- [-gpioReadCM]
- [-gpioMgmtAC {0x0-0xF}]
- [-gpioReadAC {0x0-0xF}]
- [-cryptoCM]
- [-cryptoAC {0x0-0xF}]
- [-keypairCM]
- [-keypairAC {0x0-0xF}]
- [-caRootKeyCM]
- [-caRootKeyAC {0x0-0xF}]

For example, to set GPIO1 to be in the output mode, following command can be run:

```
./nx_tool_setconfig -gpio1mode output COM7
```

4.2.4.3.1 Console output

If everything is successful, the output will be similar to:

```
sss   :INFO :Session Open Succeed
hostLib:INFO :SET config Example Success !!!...
hostLib:INFO :ex_sss Finished
```

4.2.5 NX Minimal example

This project gets available memory from Secure Authenticator.

4.2.5.1 About the Example

This project gets available memory from Secure Authenticator.

It uses the following API: `nx_FreeMem()`

4.2.5.2 Building the Example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
-
- Project - `nx_Minimal`

Since this is a "minimal" example, it is expected to work over a plain session without setting up any authentication.

4.2.5.3 Console output

If everything is successful, the output will be similar to:

```
sss   :INFO :Session Open Succeed
App   :INFO :memSize=17728
App   :INFO :nx_Minimal Example Success !!!...
App   :INFO :ex_sss Finished
```

4.3 Crypto Examples

4.3.1 ECC Example

This project demonstrates Elliptic Curve Cryptography sign and verify operations using SSS APIs. The example will create a NIST P-256 key at location - **0x02**, with following policy

```
.freezeKUCLimit  = 0,
.cardUnilateralEnabled = 0,
.sdmEnabled      = 1,
.sigmaiEnabled   = 0,
.ecdhEnabled     = 0,
.eccSignEnabled  = 1,
.writeCommMode   = kCommMode_SSS_Full,
.writeAccessCond = Nx_AccessCondition_Auth_Required_0x0,
.kucLimit        = 0,
.userCommMode    = kCommMode_SSS_NA,
```

`userCommMode` is the communication mode provided by user. If it is valid, then Cmd.ManageKeypair will use it as commMode. Otherwise (when it is kCommMode_SSS_NA), MW will try to get commMode using Cmd.GetConfiguration.

The key will be used to sign and verify the data (digest). As the Secure Authenticator does not store the public part of the generated key pair (in this case, at location 0x002), this demo shows how one can extract and store that public key in a DER file (for the file system based hosts like Windows/RPi/Linux).

Refer **ECC Example**

4.3.1.1 About the Example

This example signs a digest data and verifies the generated signature.

It uses the following APIs and data types:

- `sss_asymmetric_context_init()`
- `kAlgorithm_SSS_SHA256` from :cpp`sss_algorithm_t`

- `kMode_SSS_Sign` from :cpp`sss_mode_t`
- `sss_asymmetric_sign_digest()`
- `kMode_SSS_Verify` from :cpp`sss_mode_t`
- `sss_asymmetric_verify_digest()`

4.3.1.2 Building the example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.

```
- Project : `ex_ecc`
```

The access condition of the authentication should match access condition configuration for Cmd.CryptoRequest and Cmd.ManageKeyPair which can be configured through Cmd.SetConfiguration Option 0x12 and 0x15.

4.3.1.3 Console output

If everything is successful, the output will be similar to:

```
nx_mw :INFO :Session Open Succeed
nx_mw :INFO :Running Elliptic Curve Cryptography Example ex_sss_ecc.c
nx_mw :INFO :Storing the generated public key in public_key.der (in the
 directory where this demo is run from.)
nx_mw :INFO :Do Signing
nx_mw :INFO :digest (Len=32)
      01 02 03 04    05 06 07 08    09 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
nx_mw :INFO :signature (Len=71)
      30 45 02 20    36 F6 12 62    EE 93 7A A7    DD DC AC A5
      1C 84 BE 9A    F9 C5 73 5F    88 74 D4 9E    2E B9 B7 A1
      BC E1 CE A9    02 21 00 D9    8F E7 8B 43    67 D2 F6 BA
      FE 47 83 BE    20 9A EF 40    39 E2 4E 8A    3B 4B CC 21
      E8 BA 11 86    B6 FC 45
nx_mw :INFO :Signing Successful !!!
nx_mw :INFO :Do Verification
nx_mw :INFO :digest (Len=32)
      01 02 03 04    05 06 07 08    09 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
nx_mw :INFO :signature (Len=71)
      30 45 02 20    36 F6 12 62    EE 93 7A A7    DD DC AC A5
      1C 84 BE 9A    F9 C5 73 5F    88 74 D4 9E    2E B9 B7 A1
      BC E1 CE A9    02 21 00 D9    8F E7 8B 43    67 D2 F6 BA
      FE 47 83 BE    20 9A EF 40    39 E2 4E 8A    3B 4B CC 21
      E8 BA 11 86    B6 FC 45
nx_mw :INFO :Verification Successful !!!
nx_mw :INFO :ex_sss_ecc Example Success !!!...
nx_mw :INFO :ex_sss Finished
```

4.3.2 ECDH Example

This project demonstrates generating a ECDH key using SSS APIs on NX Secure Authenticator.

Refer **Ecdh Example**

The example will create a key pair in nx Secure Authenticator and will set a pre-cooked public key on host key store.

The key pair and publicKey key objects will be the inputs for DH derive key.

The output ecdh keyObject can be created as

- Option 1 - ecdh key object on host (OpenSSL or MbedTLS) to hold the shared secret data.
- Option 2 - ecdh key object on nx Secure Authenticator with valid transient / static buffer slot id (cipher type = kSSS_CipherType_BufferSlots).
  - Valid slot numbers:
    - 0x80 - 0x87 (Transient buffer slots. Each slot of 16 bytes.)
    - 0xC0 - 0xCF (Static buffer slots. Each slot of 16 bytes.)

This example includes 4 cases:

1. Use static key-pair (key ID 0x3) and shared secret is output to host key object.
2. Use static key-pair (key ID 0x3) and shared secret is stored in internal transient buffer.
3. Use ephemeral key-pair (Key ID 0xFE for NIST P-256) and shared secret is output to host key object.
4. Use ephemeral key-pair (Key ID 0xFE for NIST P-256) and shared secret is stored in internal transient buffer.

Static key will be created with following policy

```
.freezeKUCLimit         = 0,
.cardUnilateralEnabled  = 0,
.sdmEnabled             = 0,
.sigmaiEnabled          = 0,
.ecdhEnabled            = 1,
.eccSignEnabled         = 0,
.writeCommMode          = kCommMode_SSS_Full,
.writeAccessCond        = Nx_AccessCondition_Auth_Required_0x1,
.kucLimit               = 0,
.userCommMode           = kCommMode_SSS_NA,
```

4.3.2.1 About the Example

This example generates an ECDH key.

It uses the following APIs and data types:

- `sss_derive_key_context_init()`
- `sss_key_object_init()`
- `sss_key_object_allocate_handle()`
- `sss_derive_key_dh_one_go()`
- `kAlgorithm_SSS_ECDH` from :cpp`sss_algorithm_t`
- `kMode_SSS_ComputeSharedSecret` from :cpp`sss_mode_t`

4.3.2.2 Building the example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
  - Project - `ex_ecdh`

The access condition of the authentication should match access condition configuration for Cmd.CryptoRequest and Cmd.ManageKeyPair which can be configured through Cmd.SetConfiguration Option 0x12 and 0x15.

4.3.2.3 Console output

If everything is successful, the output will be similar to:

```
sss   :INFO :Session Open Succeed
App   :INFO :

App   :INFO :ECDH with static keypair. Export shared secret to keyobject.
App   :INFO :ECDH own Keypair 3
App   :INFO :ECDH peer public Key (Len=91)
      30 59 30 13    06 07 2A 86    48 CE 3D 02    01 06 08 2A
      86 48 CE 3D    03 01 07 03    42 00 04 B0    62 84 30 F3
      42 A0 6A 15    F0 4C 61 EF    B4 47 45 9A    0C 43 D9 A9
      31 4F 09 AA    E6 52 0C 63    C8 63 8F E5    9F 8F A5 03
      4B 4B AB 01    6E 1F 86 6F    06 C4 47 89    E2 8E 49 1A
      AF 63 24 30    BE 40 91 FE    90 98 70
App   :INFO :ECDH successful !!!
App   :INFO :ECDH derive Key (Len=32)
      DA 30 73 7D    12 7F 18 17    C9 F8 47 AF    23 1C 8D 69
      62 06 DD F0    A5 BF AD EF    00 87 0D 23    0A 20 3B 95
App   :INFO :Plain text data:
  (Len=32)
      00 01 02 03    04 05 06 07    08 09 0A 0B    0C 0D 0E 0F
      10 11 12 13    14 15 16 17    18 19 1A 1B    1C 1D 1E 1F
App   :INFO :IV data:
  (Len=16)
      EB F5 22 DE    A7 BB D5 66    B8 F6 85 B7    AD C1 06 AB
App   :INFO :Encrypted data with ECDH derive key:
  (Len=32)
      E9 34 E2 30    A4 0B F6 5B    51 D4 CB CE    BD E8 3C A7
      EB F5 22 DE    A7 BB D5 66    B8 F6 85 B7    AD C1 06 AB
App   :INFO :ex_sss_ecdh Example Success !!!...
App   :INFO :

App   :INFO :ECDH with static keypair. Store shared secrect to internal buffer.
App   :INFO :ECDH own Keypair 4
App   :INFO :ECDH peer public Key (Len=91)
      30 59 30 13    06 07 2A 86    48 CE 3D 02    01 06 08 2A
      86 48 CE 3D    03 01 07 03    42 00 04 B0    62 84 30 F3
      42 A0 6A 15    F0 4C 61 EF    B4 47 45 9A    0C 43 D9 A9
      31 4F 09 AA    E6 52 0C 63    C8 63 8F E5    9F 8F A5 03
      4B 4B AB 01    6E 1F 86 6F    06 C4 47 89    E2 8E 49 1A
      AF 63 24 30    BE 40 91 FE    90 98 70
App   :INFO :ECDH successful !!!
App   :INFO :Plain text data:
  (Len=32)
      00 01 02 03    04 05 06 07    08 09 0A 0B    0C 0D 0E 0F
      10 11 12 13    14 15 16 17    18 19 1A 1B    1C 1D 1E 1F
App   :INFO :IV data:
  (Len=16)
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
App   :INFO :Encrypted data with internal derive key :
  (Len=32)
      D5 CE 48 09    68 63 4C 97    3A 10 25 7B    82 3A F9 88
      17 0F CC 5D    08 0A 57 A0    BB BF 9A D2    F2 CE 64 67
App   :INFO :ex_sss_ecdh Example Success !!!...
App   :INFO :

App   :INFO :ECDH with ephemeral keypair. Export shared secret to keyobject.
sss   :WARN :Keyid's 254 and 255 are used for ephemeral keys.
```

```
sss    :WARN :Key is already present at these locations. No new key is created
App    :INFO :ECDH own Keypair 254
App    :INFO :ECDH peer public Key (Len=91)
       30 59 30 13    06 07 2A 86    48 CE 3D 02    01 06 08 2A
       86 48 CE 3D    03 01 07 03    42 00 04 B0    62 84 30 F3
       42 A0 6A 15    F0 4C 61 EF    B4 47 45 9A    0C 43 D9 A9
       31 4F 09 AA    E6 52 0C 63    C8 63 8F E5    9F 8F A5 03
       4B 4B AB 01    6E 1F 86 6F    06 C4 47 89    E2 8E 49 1A
       AF 63 24 30    BE 40 91 FE    90 98 70
App    :INFO :ECDH successful !!!
App    :INFO :ECDH derive Key (Len=32)
       4D 1D D0 6E    EA F1 DC D3    4B 24 2A 1E    BD BB 91 59
       E2 11 CF C9    7F AF E6 A3    68 20 53 CE    25 84 3A 83
App    :INFO :Plain text data:
   (Len=32)
       00 01 02 03    04 05 06 07    08 09 0A 0B    0C 0D 0E 0F
       10 11 12 13    14 15 16 17    18 19 1A 1B    1C 1D 1E 1F
App    :INFO :IV data:
   (Len=16)
       1E 85 AD 3F    49 E3 6A EA    FE 80 83 D1    81 0D 7A BD
App    :INFO :Encrypted data with ECDH derive key:
   (Len=32)
       15 06 2D B4    96 2D 21 49    7D A3 81 1D    B8 A6 4E 89
       1E 85 AD 3F    49 E3 6A EA    FE 80 83 D1    81 0D 7A BD
App    :INFO :ex_sss_ecdh Example Success !!!...
App    :INFO :

App    :INFO :ECDH with ephemeral keypair. Store shared secrect to internal
 buffer.
sss    :WARN :Keyid's 254 and 255 are used for ephemeral keys.
sss    :WARN :Key is already present at these locations. No new key is created
App    :INFO :ECDH own Keypair 254
App    :INFO :ECDH peer public Key (Len=91)
       30 59 30 13    06 07 2A 86    48 CE 3D 02    01 06 08 2A
       86 48 CE 3D    03 01 07 03    42 00 04 B0    62 84 30 F3
       42 A0 6A 15    F0 4C 61 EF    B4 47 45 9A    0C 43 D9 A9
       31 4F 09 AA    E6 52 0C 63    C8 63 8F E5    9F 8F A5 03
       4B 4B AB 01    6E 1F 86 6F    06 C4 47 89    E2 8E 49 1A
       AF 63 24 30    BE 40 91 FE    90 98 70
App    :INFO :ECDH successful !!!
App    :INFO :Plain text data:
   (Len=32)
       00 01 02 03    04 05 06 07    08 09 0A 0B    0C 0D 0E 0F
       10 11 12 13    14 15 16 17    18 19 1A 1B    1C 1D 1E 1F
App    :INFO :IV data:
   (Len=16)
       00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
App    :INFO :Encrypted data with internal derive key :
   (Len=32)
       E8 09 89 6F    9B F9 3F E8    4A E5 8C 06    C1 96 80 D6
       AC 3B 9D 9A    8B C8 68 57    85 FA A7 EA    40 ED 3B 65
App    :INFO :ex_sss_ecdh Example Success !!!...
App    :INFO :ex_sss Finished
```

4.3.3 File Management Example

This project demonstrates basic File Management operations using Nx APIs.

Refer **File Management Example**

4.3.3.1 About the Example

This example creates a file, writes and reads data from it.

Note - The fileNo 1, 2 and 3 cannot be used, as there are some pre-provisioned static files present at these IDs.

It uses the following APIs and data types: `nx_GetFileIDs()` `nx_CreateStdDataFile()` `nx_WriteData()` `nx_ReadData()`

4.3.3.2 Building the example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
- Project - `ex_file_mgnt`
- Select NXMW_Auth to SIGMA_I_Verifier, SIGMA_I_Prover or Symm_Auth.-
- NXMW_Secure_Tunneling to NTAG_AES128_EV2, NTAG_AES256_EV2 or NTAG_AES128_AES256_EV2.

4.3.3.3 Console output

If everything is successful, the output will be similar to:

```
sss    :INFO :Session Open Succeed
App    :INFO :Running File Management Example ex_sss_file_mgnt.c
App    :INFO :Standard Data File creation successful !!!
App    :INFO :File write successful !!!
App    :INFO :File read successful !!!
App    :INFO :ex_sss_file_mgnt Example Success !!!...
App    :INFO :ex_sss Finished
```

4.3.4 RNG Example

This project demonstrates random number generation using SSS APIs.

Refer **Random Number Generation Example**

4.3.4.1 About the Example

This example generates a random number of given length.

It uses the following APIs: `sss_rng_context_init()` `sss_rng_get_random()` `sss_rng_context_free()`

4.3.4.2 Building the example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
- Project - `ex_rng`

The access condition of the authentication should match access condition configuration for Cmd.CryptoRequest which can be configured through Cmd.SetConfiguration Option 0x15.

4.3.4.3 Console output

If everything is successful, the output will be similar to:

```
sss   :INFO :Session Open Succeed
App   :INFO :Running Get Random Data Example ex_sss_rng.c
App   :INFO :Get Random Data successful !!!
App   :INFO :Random (Len=32)
      1C 40 F9 B2    68 C1 2B CC    04 50 AD 18    F9 D4 A5 B9
      A1 57 32 DF    00 CA 52 96    A7 C1 0B FC    03 8C D4 6C
App   :INFO :ex_sss_rng Example Success !!!...
App   :INFO :ex_sss Finished
```

4.3.5 HKDF Example

This project demonstrates an HMAC Key derivation operation based on info and salt using SSS APIs.

Refer **HKDF Example**

4.3.5.1 About the Example

This example does an HMAC Key derivation operation based on the info and salt.

It uses the following APIs and data types:

```
`sss_derive_key_context_init()`
`kAlgorithm_SSS_HMAC_SHA256` from :cpp`sss_algorithm_t`
`kMode_SSS_HKDF_ExtractExpand` from :cpp`sss_mode_t`
`kSSS_CipherType_AES`from :cpp`sss_cipher_type_t`
`sss_derive_key_one_go()`
```

```
NOTE:

In the provided example, the derived key is output to the host. For
actual product deployment, the security implications of doing this need
to be assessed. Alternatively the derived key can be kept within the Nx
Transient or Static Buffers, similarly as shown in the ECDH example
```

4.3.5.2 Building the example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
- Project - `ex_hkdf`

The access condition of the authentication should match access condition configuration for Cmd.CryptoRequest and Cmd.ChangeKey which can be configured through Cmd.SetConfiguration Option 0x15.

4.3.5.3 Console output

If everything is successful, the output will be similar to:

```
nx_mw :INFO :Session Open Succeed
nx_mw :INFO :Running HMAC Key Derivation Function Example ex_sss_hkdf.c
nx_mw :INFO :Do Key Derivation
```

```
nx_mw :INFO :salt (Len=32)
                AA 1A 2A E3     B2 76 15 4D     67 F9 D8 4C     B9 35 54 56
                BB 1B 2B 03     04 05 06 07     08 09 0A 0B     0C 0D 0E 0F
nx_mw :INFO :info (Len=80)
                00 01 02 03     04 05 06 07     08 09 0A 0B     0C 0D 0E 0F
                10 11 12 13     14 15 16 17     18 19 1A 1B     1C 1D 1E 1F
                20 21 22 23     24 25 26 27     28 29 2A 2B     2C 2D 2E 2F
                30 31 32 33     34 35 36 37     38 39 3A 3B     3C 3D 3E 3F
                40 41 42 43     44 45 46 47     48 49 4A 4B     4C 4D 4E 4F
nx_mw :INFO :Key Derivation successful !!!
nx_mw :INFO :hkdfOutput (Len=128)
                0A BB A2 EE     5F 25 04 CD     13 2E AF 2A     FC 60 76 5A
                DC 38 27 D4     58 7D EA 6C     FE C9 B7 0F     42 7C 66 82
                92 8C 2C A2     27 31 92 2B     36 28 23 7D     BA 3B FE 9C
                1B 08 F0 F9     19 BD 4E D0     51 90 2D 70     9A F5 35 7F
                21 80 1F EF     B4 48 98 BE     8F 2E C8 2D     75 A6 5A EB
                83 E0 C1 C0     5B EF CB 47     73 FC 81 27     03 8D 16 75
                10 43 65 67     F6 9D B8 E7     83 BA 32 6A     E0 90 E7 AE
                D5 85 80 75     68 6C 5C E7     38 4A 6D B7     4E A9 0E D2
nx_mw :INFO :ex_sss_hkdf Example Success !!!...
nx_mw :INFO :ex_sss Finished
```

4.3.6 HMAC Example

This project demonstrates an HMAC operation on a message using SSS APIs.

Refer **HMAC Example**

4.3.6.1 About the Example

This example does an HMAC operation on input data.

It uses the following APIs and data types:

- `sss_mac_context_init()`
- `kAlgorithm_SSS_HMAC_SHA256` from `sss_algorithm_t`
- `kMode_SSS_Mac` from `sss_mode_t`
- `sss_mac_one_go()`

4.3.6.2 Building the example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
  - Project - `ex_hmac`
  - The access condition of the authentication should match access condition configuration for Cmd.CryptoRequest and Cmd.ChangeKey which can be configured through Cmd.SetConfiguration Option 0x15.

4.3.6.3 Console output

If everything is successful, the output will be similar to:

```
sss   :INFO :Session Open Succeed
App   :INFO :Running HMAC (SHA256) Example ex_sss_hmac.c
App   :INFO :Do HMAC
```

```
App   :INFO :input (Len=10)
      48 65 6C 6C    6F 57 6F 72    6C 64
App   :INFO :hmac key (Len=16)
      48 65 6C 6C    6F 48 65 6C    6C 6F 48 65    6C 6C 6F 48
App   :INFO :HMAC (SHA256) successful !!!
App   :INFO :hmac (Len=32)
      68 7A 26 95    49 67 9D 6E    FA 11 19 5E    96 CB BA C2
      6B 50 A5 09    10 8A D1 48    B5 FC A0 94    2C BD 10 21
App   :INFO :ex_sss_hmac Example Success !!!...
App   :INFO :ex_sss Finished
```

4.3.7 Message Digest Example

This project demonstrates a Message Digest / hashing operation using SSS APIs.

Refer **MD Example**

4.3.7.1 About the Example

This example calculates the digest of a sample data and will compare with expected value.

It uses the following APIs and data types: `sss_digest_context_init()` `kAlgorithm_SSS_SHA256`from `sss_algorithm_t` `kMode_SSS_Digest` from `sss_mode_t` `sss_digest_one_go()`

4.3.7.2 Building the example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
- Project - `ex_md`

The access condition of the authentication should match access condition configuration for Cmd.CryptoRequest which can be configured through Cmd.SetConfiguration Option 0x15.

4.3.7.3 Console output

If everything is successful, the output will be similar to:

```
sss   :INFO :Session Open Succeed
App   :INFO :Running Message Digest Example ex_sss_md.c
App   :INFO :Do Digest
App   :INFO :input (Len=10)
      48 65 6C 6C    6F 57 6F 72    6C 64
App   :INFO :Message Digest successful !!!
App   :INFO :digest (Len=32)
      87 2E 4E 50    CE 99 90 D8    B0 41 33 0C    47 C9 DD D1
      1B EC 6B 50    3A E9 38 6A    99 DA 85 84    E9 BB 12 C4
App   :INFO :ex_sss_digest Example Success !!!...
App   :INFO :ex_sss Finished
```

4.3.8 Symmetric AES Example

This project demonstrates Symmetric Cryptography - AES encryption and decryption operations using an AES-128 key in the CBC mode.

Refer **Symmetric Example**

4.3.8.1 About the Example

This example does a symmetric cryptography AES encryption and decryption operation.

It uses the following APIs and data types:

```
`sss_symmetric_context_init()`
`kAlgorithm_SSS_AES_CBC` from :cpp`sss_algorithm_t`
`kSSS_CipherType_AES`from :cpp`sss_cipher_type_t`
`kMode_SSS_Encrypt`from :cpp`sss_mode_t`
`sss_cipher_one_go()`
`kMode_SSS_Decrypt`from :cpp`sss_mode_t`
`sss_symmetric_context_free()`
```

4.3.8.2 Building the example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
  - Project - `ex_symmetric`

The access condition of the authentication should match access condition configuration for Cmd.CryptoRequest and Cmd.ChangeKey which can be configured through Cmd.SetConfiguration Option 0x15.

4.3.8.3 Console output

If everything is successful, the output will be similar to:

```
sss   :INFO :Session Open Succeed
App   :INFO :Running AES Symmetric Example ex_sss_symmetric.c
App   :INFO :Do Encryption
App   :INFO :icv (Len=16)
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
App   :INFO :srcData (Len=16)
      48 45 4C 4C    4F 48 45 4C    4C 4F 48 45    4C 4C 4F 31
App   :INFO :Encryption successful !!!
App   :INFO :Encrypted data (Len=16)
      32 A6 04 88    C5 B3 FF 40    50 AF 56 A5    68 AE D1 05
App   :INFO :Do Decryption
App   :INFO :icv (Len=16)
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
App   :INFO :Encrypted data (Len=16)
      32 A6 04 88    C5 B3 FF 40    50 AF 56 A5    68 AE D1 05
App   :INFO :Decryption successful !!!
App   :INFO :Decrypted data (Len=16)
      48 45 4C 4C    4F 48 45 4C    4C 4F 48 45    4C 4C 4F 31
App   :INFO :ex_sss_symmetric Example Success !!!...
App   :INFO :ex_sss Finished
```

4.3.9 CMAC Example (Using slots in NX Secure Authenticator)

This project demonstrates a CMAC operation using NX APIs.

Refer **CMAC Example**

4.3.9.1 About the Example

This project demonstrates a CMAC operation using nx apis. The data is stored in static buffer 0 (kSE_CryptoDataSrc_SB0) and output is stored in transient buffer 0 (kSE_CryptoDataSrc_TB0). This allows to use the CMAC for e.g. key derivation, with keeping the derived key in the Nx secure authenticator.

It uses the following APIs and data types:

- `nx_CryptoRequest_Write_Internal_Buffer()`
- `nx_CryptoRequest_AES_CMAC_Sign()`

4.3.9.2 Building the example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
- project: `ex_cmac_using_slots`

The access condition of the authentication should match access condition configuration for Cmd.CryptoRequest which can be configured through Cmd.SetConfiguration Option 0x15.

4.3.9.3 Console output

If everything is successful, the output will be similar to:

```
sss   :INFO :Session Open Succeed
App   :INFO :Running CMAC Example (using Slots) ex_nx_cmac_using_slots.c
App   :INFO :Do CMAC
App   :INFO :input (Len=16)
      00 01 02 03    04 05 06 07    08 09 0A 0B    0C 0D 0E 0F
App   :INFO :CMAC output (Len=16)
      7B CF BB CA    7A 2E A6 8B    96 6F C5 39    9F 74 80 9E
App   :INFO :ex_nx_cmac_using_slots Example Success !!!...
App   :INFO :ex_sss Finished
```

4.3.10 Message Digest Example (Using slots in NX Secure Authenticator)

This project demonstrates a Message Digest / hashing operation using Nx apis. Nx Secure Authenticator has the option of storing the resultant hash in either an internal buffer (static/transient) or sending it out over a buffer. This example how to use the Nx API to store the result in an internal buffer.

Refer **MD Example Using Slots**

4.3.10.1 About the Example

This example calculates the digest of a sample data and output data (hash) is stored in static buffer 0 and 1. Note that, although we give kSE_CryptoDataSrc_SB0 (i.e. static slot no. 0) as the only input to the API, the output hash is actually stored in the buffers kSE_CryptoDataSrc_SB0 and kSE_CryptoDataSrc_SB1. This is because each buffer is 16 bytes in size and the SHA256 shall produce 32 bytes of hash. Internally, the buffers are aligned in a contiguous fashion, hence the output hash gets stored in both the buffers.

It uses the following APIs and data types:

- `nx_CryptoRequest_SHA_Oneshot()`
- `kSE_CryptoDataSrc_SB0`from :cpp`SE_CryptoDataSrc_t`

4.3.10.2 Building the example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
- Project - `ex_md_using_slots`

The access condition of the authentication should match access condition configuration for Cmd.CryptoRequest which can be configured through Cmd.SetConfiguration Option 0x15.

4.3.10.3 Console output

If everything is successful, the output will be similar to:

```
sss    :INFO :Session Open Succeed
App    :INFO :Running Message Digest Example ex_sss_md.c
App    :INFO :Do Digest
App    :INFO :input (Len=10)
       48 65 6C 6C    6F 57 6F 72    6C 64
App    :INFO :Digest will be written to static buffer 0 and 1
App    :INFO :ex_nx_digest Example Success !!!...
App    :INFO :ex_sss Finished
```

4.4 Management and Configuration Examples

4.4.1 Get Card UID Example

This project demonstrates getting card UID operation using Nx APIs.

Refer **Get Card UID Example**

4.4.1.1 About the Example

This example shows a simple operation to read the card UID.

It uses the following APIs and data types: -`nx_GetCardUID()`

4.4.1.2 Building the example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
- Project : `ex_get_uid`
- Select NXMW_Auth to SIGMA_I_Verifier, SIGMA_I_Prover or Symm_Auth.-
- NXMW_Secure_Tunneling to NTAG_AES128_EV2, NTAG_AES256_EV2 or NTAG_AES128_AES256_EV2.

4.4.1.3 Console output

If everything is successful, the output will be similar to:

```
sss    :INFO :Session Open Succeed
App    :INFO :Running Get Card UID Example ex_sss_get_uid.c
App    :INFO :Get Card UID
App    :INFO :Successful !!!
```

```
App   :INFO :Card UID (Len=7)
      00 01 02 03    04 05 06
App   :INFO :ex_sss_get_uid Example Success !!!...
App   :INFO :ex_sss Finished
```

4.4.2 Get Configuration Example

This project demonstrates get configuration operation using SSS APIs. It will read manufacturing product features and card configuration.

Refer **Get Configuration Example**

4.4.2.1 About the Example

This example does a get configuration operation.

It uses the following APIs and data types:

```
`nx_GetConfig_ManufactureConfig()`
`nx_GetConfig_PICCConfig()`
`nx_GetConfig_ATSUpdate()`
`nx_GetConfig_SAKUpdate()`
`nx_GetConfig_SMConfig()`
`nx_GetConfig_CapData()`
`nx_GetConfig_ATQAUpdate()`
`nx_GetConfig_SilentModeConfig()`
`nx_GetConfig_EnhancedPrivacyConfig()`
`nx_GetConfig_NFCMgmt()`
`nx_GetConfig_I2CMgmt()`
`nx_GetConfig_GPIOMgmt()`
`nx_GetConfig_EccKeyMgmt()`
`nx_GetConfig_CertMgmt()`
`nx_GetConfig_WatchdogTimerMgmt()`
`nx_GetConfig_CryptoAPIMgmt()`
`nx_GetConfig_LockConfig()`
```

4.4.2.2 Building the example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
  - Project - `ex_get_config`
  - Select NXMW_Auth to SIGMA_I_Verifier, SIGMA_I_Prover or SYMM_Auth.
  - NXMW_Secure_Tunneling to NTAG_AES128_EV2, NTAG_AES256_EV2 or NTAG_AES128_AES256_EV2 (only with SIGMA_I_Verifier or SIGMA_I_Prover).
  - The authentication must be setup with access condition 0. This means App Master key for symmetric authentication. For Sigma-I authentication, this mean AC0 of certificate access right. It is defined with CA root key and reader certificate (or certificate chain).

    Note: GetConfig Demo on MCXA153 currently supports Sigma Authentication with Mbedtls2. Symmetric Authentication is supported with Mbedtls 2 and Mbedtls 3.

4.4.2.3 Console output

If everything is successful, the output will be similar to:

```
nx_mw :INFO :Session Open Succeed
nx_mw :INFO :Running Get Configuration Example ex_sss_set_config.c

APDU  :DEBUG:GetConfiguration [manufacturer configuration]
nx_mw :INFO :=======================================
nx_mw :INFO :Get Manufacture Features successful !!!
nx_mw :INFO : Support ECC-based Unilateral Authentication: Enabled
nx_mw :INFO : Support Import of ECC Private Key: Enabled
... OTHER DETAILS ...
nx_mw :INFO :ex_sss_get_config Example Success !!!...
nx_mw :INFO :ex_sss Finished
```

4.4.3 Get Version Example

This project demonstrates a get HW/SW version using Nx APIs.

Refer **Get version Example**

4.4.3.1 About the Example

This example does a getting HW/SW version operation along with the manufacturing info.

It uses the following APIs and data types: `nx_GetVersion()`

4.4.3.2 Building the example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
- Project - `ex_get_version`
- Select NXMW_Auth to None, SIGMA_I_Verifier, SIGMA_I_Prover or SYMM_Auth.
- NXMW_Secure_Tunneling to None (only with NXMW_Auth None), NTAG_AES128_EV2, NTAG_AES256_EV2 or NTAG_AES128_AES256_EV2 (only with NXMW_Auth SIGMA_I_Verifier or SIGMA_I_Prover).

4.4.3.3 Console output

If everything is successful, the output will be similar to:

```
nx_mw :INFO :Session Open Succeed
nx_mw :INFO :Running Get Card Version Example ex_sss_get_version.c
nx_mw :INFO :Get Card Version
nx_mw :INFO :Successful !!!
nx_mw :INFO :HW Vendor ID: NXP Semiconductors
nx_mw :INFO :HW type: IoT
nx_mw :INFO :HW subtype: 17 pF, Tag Tamper
nx_mw :INFO :HW major version: NX (Zen-V)
nx_mw :INFO :HW minor version 0x0
nx_mw :INFO :HW storage size: 16 kB
nx_mw :INFO :HW protocol type: I2C and ISO/IEC 14443-4 support with Silent Mode
 support
nx_mw :INFO :SW Vendor ID: NXP Semiconductors
```

```
nx_mw :INFO :SW type: IoT
nx_mw :INFO :SW subtype: Standalone
nx_mw :INFO :SW major version: EV0
nx_mw :INFO :SW minor version 0xec
nx_mw :INFO :SW storage size: 16 kB
nx_mw :INFO :SW protocol type: I2C and ISO/IEC 14443-4 support with Silent Mode
 support
nx_mw :INFO :UIDFormat: 0x0
nx_mw :INFO :UIDLength: 0xa
nx_mw :INFO :Card UID (Len=10)
      00 01 02 2C    41 01 01 07    08 09
nx_mw :INFO :BatchNo: 0x0
nx_mw :INFO :FabKey identifier: 0x0
nx_mw :INFO :Calendar week of card production in BCD coding: 0x31
nx_mw :INFO :The year of production in BCD coding: 0x22
nx_mw :INFO :Fab Identifier: 0x32
nx_mw :INFO :ex_sss_get_version Example Success !!!...
nx_mw :INFO :ex_sss Finished
```

4.5 TLS Examples

4.5.1 OpenSSL Engine/Provider: TLS Client example

This section explains how to set-up a TLS link using the SSS OpenSSL Engine or Provider on the client side.

4.5.1.1 Summary

The TLS demo demonstrates setting up a mutually authenticated and encrypted link between a client and a server system. The keypair used to identify the client is stored in the Secure Authenticator. The keypair used to identify the server is simply available as a pem file.

The public keys associated with the respective key pairs are contained in respectively a client and a server certificate.

The CA is a self-signed certificate. The same CA is used to sign client and server certificate.

4.5.1.2 Secure Authenticator preparation (client side)

For the purpose of the demo one MUST inject the TLS client key pair into the Secure Authenticator and use a reference pem file (refer: **EC Reference Key Format**) referring to the provisioned key pair.

The python script **tlsProvision.py** (inside scripts folder) can be used to provision the SA with required key.

Generic command:

```
python tlsProvision_sa.py -smcom <SMCOM> -port <PORT_NAME> -curve <CURVE> -
keypath <PATH_TO_KEY>
```

The script will provision the client key at location 0x02 (for nist256) / 0x03 (for brainpool256p) with the following policies

```
.sdmEnabled      = 0,
.sigmaiEnabled   = 0,
.ecdhEnabled     = 0,
.eccSignEnabled  = 1,
.writeCommMode   = kCommMode_SSS_Full,
.writeAccessCond = Nx_AccessCondition_Auth_Required_0x1,
```

```
.userCommMode    = kCommMode_SSS_NA,
```

The required key is manually created / stored for the demo at located - `demos/nx/tls_client/credentials/<key_type>/tls_client_key.pem`

On RPI, you can use following command:

```
cd nxmw/demos/nx/tls_client/scripts
python tlsProvision.py -smcom t1oi2c -port /dev/i2c-1 -curve prime256v1 -
keypath ../credentials/prime256v1/tls_client_key.pem
```

Note: This command will store the key **tls_client_key.pem** inside the SA at key ID 0x02.

4.5.1.3 Start up the server

Note: The server can run e.g. on a PC. The server must be reachable over the TCP/IP network for the Client.

Server can be executed using the following generic command:

```
python tlsServer.py <ECDHE|ECDHE_SHA256|max> <prime256v1(default)|
brainpoolP256r1>
```

Execute the following command on the server platform to use the EC based server credentials, make a note of the IP address of the server:

```
cd nxmw/demos/nx/tls_client/scripts
python tlsServer.py ECDHE_SHA256 prime256v1
```

4.5.1.4 Establish a TLS link from the client to the server

4.5.1.5 Using OpenSSL Engine

- Build OpenSSL Engine (refer: **Building the OpenSSL engine**)
- Invoke the script **tlsSeClient.py** in a separate terminal using the IP address of the server as the first argument and ECDHE or ECDHE_SHA256 as the second argument (ECDHE corresponding to ECDH ephemeral) when connecting to a server using EC based credentials. Generic command:

```
python tlsSeClient.py <SERVER_IP_ADDRESS> <ECDHE|ECDHE_256>
```

Execute following command in a separate terminal to run TLS client:

```
cd nxmw/demos/nx/tls_client/scripts
python tlsSeClient.py 127.0.0.1 ECDHE_SHA256 prime256v1
```

Any message entered on the client terminal now will be received by server terminal and vice versa.

4.5.1.6 Using OpenSSL Provider

- Build OpenSSL Provider (refer: **Building the OpenSSL provider**)

- Invoke the script **tlsSeClient30.py** in a separate terminal using the IP address of the server as the first argument and ECDHE or ECDHE_SHA256 as the second argument (ECDHE corresponding to ECDH ephemeral) when connecting to a server using EC based credentials. Generic command:

```
python tlsSeClient30.py <SERVER_IP_ADDRESS> <ECDHE|ECDHE_256>
```

Execute following command in a separate terminal to run TLS client:

```
cd nxmw/demos/nx/tls_client/scripts
python tlsSeClien30.py 127.0.0.1 ECDHE_SHA256 prime256v1
```

Any message entered on the client terminal now will be received by server terminal and vice versa.

4.6 Cloud Examples

4.6.1 AWS Demo for RaspberryPi

This demo demonstrates connection to AWS IoT Console using pre-provisioned device credentials and publish/subscribe procedure using MQTT.

4.6.1.1 Prerequisites

- AWS account setup (Refer - https://docs.aws.amazon.com/iot/latest/developerguide/iot-gs-first-thing.html)
- Raspberry Pi with Raspbian OS, connected to the Internet
- OpensSSL 1.1 or OpenSSL 3.x installed

4.6.1.2 Secure Authenticator preparation (client side)

For the purpose of this demo one MUST inject the client key pair into the Secure Authenticator and use a reference pem file referring to the provisioned key pair.

The python script (`nxmw/demos/linux/aws/aws_provisioning_client/aws_provision_client.py`) can be used to provision the SA with required key

```
python aws_provision_client.py -smcom <SMCOM> -port <PORT NAME> -keypath <PATH
 TO KEY>
```

> **Note:** Path to the key to be provisioned - `nxmw/demos/linux/aws/aws_provisioning_client/credentials/nx_device_key.pem`
>
> **Note:** Build `nxclitool` before running provisioning script. Refer **Build nxclitool**.

The above commands will provision the client key at location 0x02 with the following policies.

```
.sdmEnabled      = 0,
.sigmaiEnabled   = 0,
.ecdhEnabled     = 0,
.eccSignEnabled  = 1,
.writeCommMode   = kCommModeSSSFull,
.writeAccessCond = NxAccessConditionAuthRequired0x1,
.userCommMode    = kCommModeSSSNA,
```

The required reference key is manually created / stored for the demo at location - `nxmw/demos/linux/aws/aws_provisioning_client/credentials/nx_device_reference_key.pem`

4.6.1.3 Build the OpenSSL engine (Required if using OpenSSL 1.x)

The OpenSSL engine uses the sss abstraction layer to access the crypto services of the Secure Authenticator. The following illustrates compiling the OpenSSL engine for Nx connected over I2C.

```
cd nxmw/scripts
python create_cmake_projects.py
cd ../../nxmw_build/raspbian_native_nx_t1oi2c
cmake -DNXMW_OpenSSL:STRING=1_1_1 .
cmake --build .
make install
ldconfig /usr/local/lib
```

4.6.1.4 Build the OpenSSL Provider (Required if using OpenSSL 3.x.x)

The OpenSSL provider uses the sss abstraction layer to access the crypto services of the Secure Authenticator. The following illustrates compiling the OpenSSL provider for Nx connected over I2C.

Note: comment the following lines in `nxmw/plugin/openssl_provider/provider/src/sssProvider_main.c` before building OpenSSL Provider

```
    //if (NULL == OSSL_PROVIDER_load(NULL, "default")) {
    //     sssProv_Print(LOG_FLOW_ON, "error in OSSL_PROVIDER_load \n");
    // }
```

```
cd nxmw/scripts
python create_cmake_projects.py
cd ../../nxmw_build/raspbian_native_nx_t1oi2c
cmake -DNXMW_Auth=None -DNXMW_OpenSSL:STRING=3_0 -DNXMW_Secure_Tunneling=None -
DNXMW_SMCOM=JRCP_V1_AM .
cmake --build .
make install
ldconfig /usr/local/lib
```

Note: If OpenSSL Provider is used to run the demo, Use Access Manager to establish connection with Secure Authenticator. For details, refer **Build and Run Access Manager**.

4.6.1.5 Run the example

1. Clone the aws-sdk :

```
cd nxmw/demos/linux/aws/
git clone https://github.com/aws/aws-iot-device-sdk-cpp.git
```

2. For the demo to build with openssl 3.x.x and to use sssProvider apply the patch to repo aws-iot-device-sdk-cpp

```
cd nxmw/demos/linux/aws/aws-iot-device-sdk-cpp
patch -p1 < ../sssProvider_Updates.patch
```

3. Modify the `CMakeLists.txt` file under `aws-iot-device-sdk-cpp/samples/PubSub` so it ensures `OPENSSL_LOAD_CONF` is defined (see excerpt below):

```
if (UNIX AND NOT APPLE)
    ADD_DEFINITIONS(-DOPENSSL_LOAD_CONF)
    # Prefer pthread if found
    set(THREADSPREFERPTHREADFLAG ON)
    set(CUSTOMCOMPILERFLAGS "-fno-exceptions -Wall -Werror")
```

```
elseif (APPLE)
    set(CUSTOMCOMPILERFLAGS "-fno-exceptions -Wall -Werror")
elseif (WIN32)
    set(CUSTOMCOMPILERFLAGS "/W4")
endif ()
```

Note: If curl is not installed - run `sudo apt-get install libcurl4-openssl-dev`

1. Set the permissions to `buildScript.sh` at `nxmw/demos/linux/aws/`:

```
chmod 777 buildScript.sh
```

2. Use 'buildScript.sh' script to build the mqtt application:

```
./buildScript.sh
```

3. Adapt the PubSub example specific configuration file so that it refers to the reference key and the device certificate.
   - Update the endpoint to match your AWS account
   - Ensure the AmazonRootCA1.pem certificate is in place (it is used by the rpi to validate the AWS IoT counterpart)
   - Update the configuration file (nxmw/demos/linux/aws/aws-iot-device-sdk-cpp/build/bin/config/SampleConfig.json) with endpoint, device\certificate\relative\path, device\private\key\relative\path (Ensure the value for "endpoint" matches your setup, you must replace "xxxxiukfoyyyy-ats.iot.eu-central-1.amazonaws.com")
   - Sample Json file :

```
{
"endpoint": "xxxxiukfoyyyy-ats.iot.eu-central-1.amazonaws.com",
"mqttport": 8883,
"httpsport": 443,
"greengrassdiscoveryport": 8443,
"rootcarelativepath": "certs/AmazonRootCA1.pem",
"devicecertificaterelativepath": "nxdevicecertificate.cer",
"deviceprivatekeyrelativepath": "nxdevicereferencekey.pem",
"tlshandshaketimeoutmsecs": 60000,
"tlsreadtimeoutmsecs": 2000,
"tlswritetimeoutmsecs": 2000,
"awsregion": "",
"awsaccesskeyid": "",
"awssecretaccesskey": "",
"awssessiontoken": "",
"clientid": "CppSDKTesting",
"thingname": "CppSDKTesting",
"iscleansession": true,
"mqttcommandtimeoutmsecs": 20000,
"keepaliveintervalsecs": 600,
"minimumreconnectintervalsecs": 1,
"maximumreconnectintervalsecs": 128,
"maximumackstowaitfor": 32,
"actionprocessingratehz": 5,
"maximumoutgoingactionqueuelength": 32,
"discoveractiontimeoutmsecs": 300000
}
```

4. To run the demo with OpenSSL Provider, search for `provider_sect` in `/nxmw/demos/linux/common/openssl30_sss_se050.cnf` file and set the nxp provider with high priority.

```
nxp_prov = nxp_prov_sec
default = default_sect
```

5. Based on OpenSSL version, select the appropriate configuration file in `/nxmw/demos/linux/common` directory::

```
openssl_sss_se050.cnf     ----- OpenSSL 1.x
openssl30_sss_se050.cnf   ----- OpenSSL 3.x.x
```

6. Upload the root certificate to AWS account.
7. Run access manager (Required only if using OpenSSL Provider)

```
cd nxmw_build/raspbian_native_nx_t1oi2c/bin
./accessManager
```

Note: Run Access Manager only when you use OpenSSL Provider.

8. Open New Terminal and set the OpenSSL config path as call:

```
$ export OPENSSL_CONF=nxmw/demos/linux/common/<appropriate-cnf-file>
```

9. Run the application

```
cd nxmw/demos/linux/aws/aws-iot-device-sdk-cpp/build/bin
export EX_SSS_BOOT_SSS_PORT=<port_name>
./pub-sub-sample
```

Note: When using Openssl Provider application will not connect to NX Secure Authenticator directly over i2c. It will connect to access manager which in turn will connect to SA over i2c. So export the port_name as `127.0.0.1:8040`.

Note: If OpenSSL Engine is used no need to export port since the application will directly connect to Secure Authenticator over i2c.

4.6.2 AWS Cloud Demo on FreeRTOS

This demo demonstrates connection to AWS IoT Console using pre-provisioned device credentials and publish/subscribe procedure using MQTT.

4.6.2.1 Prerequisites

- Active AWS account
- MCUXpresso installed (for running aws demo on MCXN947)
- Any Serial communicator(Tera Term)

4.6.2.2 Creating a device on AWS account

Refer - **AWS Developer guide**

Note: The device certificate needs to be activated and attached to a policy that allows usage of this certificate.

4.6.2.3 Creating and updating device keys and certificates to Secure Authenticator

- Before running the cloud_aws example the client key pair and certificate must be provisioned to the NX secure authenticator
- Build provisioning demo. Refer **cloud aws provisioning**.

4.6.2.4 Building the Demo

- Build NX middleware stack. Refer `Create Build files` section in **MCU_cmake build**.

- To get the AWS IoT MQTT broker endpoint for your account, go to the AWS IoT console and in the left navigation pane choose Settings. Copy the endpoint listed under the "Device data endpoint"
- In the `nxmw/demos/ksdk/aws_jitr/aws_client_credential_keys.h` file, update the endpoint in the macro "clientcredentialMQTT_BROKER_ENDPOINT" and thing name in the macro "clientcredentialIOT_THING_NAME"
- Update cmake options to use freeRTOS ::
  - `NXMW_RTOS:STRING=FreeRTOS`

```
- Project : `cloud_provisioning_aws`
```

4.6.2.5 Running the Demo

1. Open a serial terminal on PC for OpenSDA serial device with these settings:

```
- 115200 baud rate
- 8 data bits
- No parity
- One stop bit
- No flow control
```

2. Connect the ethernet port of the board to a router for internet access (IP address to the board is assigned by the DHCP server). Make sure the connection on port 8883 is not blocked.
3. Flash the program to the target board. Refer `Running the Example` section in **MCU_cmake build**

4.6.2.6 Console output

If everything is successful, the output will be similar to:

```
nx_mw :INFO :NX_PKG_v02.05.00_20250411
nx_mw :INFO :sss_ex_rtos_task Started.
nx_mw :INFO :cip (Len=22)
                01 04 63 07     00 93 02 08     00 02 03 E8     00 01 00 64
                04 03 E8 00     FE 00
nx_mw :INFO :Session Open Succeed
nx_mw :INFO :AWS subscribe publish example

Initializing PHY...
nx_mw :INFO :Getting IP address from DHCP ...

nx_mw :INFO :
 IPv4 Address     : 192.168.2.200

nx_mw :INFO :DHCP OK

[INFO] Create a TCP connection to a29rg0ytflhg6y.iot.eu-
central-1.amazonaws.com:8883.
[INFO] (Network connection 0x20017be0) TLS handshake successful.
[INFO] (Network connection 0x20017be0) Connection to a29rg0ytflhg6y.iot.eu-
central-1.amazonaws.com [INFO] MQTT connection established with the broker.
[INFO] MQTT connection successfully established with broker.

[INFO] A clean MQTT connection is established. Cleaning up all the stored
 outgoing publishes.

nx_mw :INFO :MQTT Connection successfully established
[INFO] Attempt to subscribe to the MQTT topic $aws/things/K64f_1/shadow/update/
accepted.
```

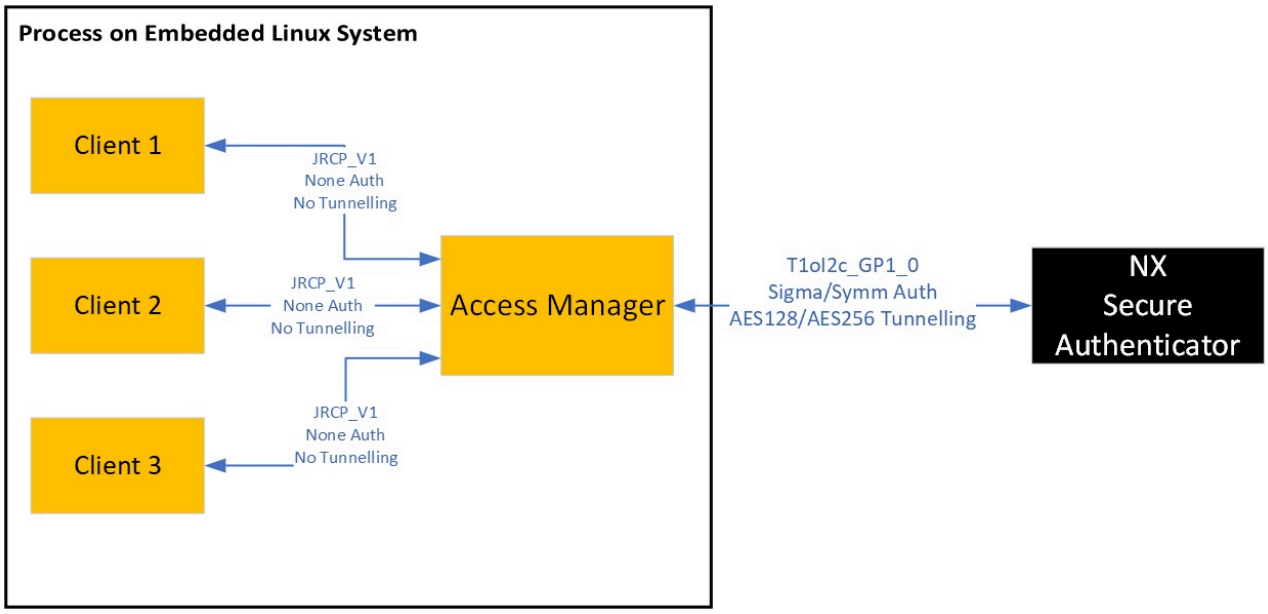
```
nx_mw :INFO :MQTT Subscribed
[INFO] the published payload:{"msg" : "hello from SDK QOS0 : 0"}

[INFO] PUBLISH sent for topic sdkTest/sub to broker with packet ID 2.

nx_mw :INFO :Published message {"msg" : "hello from SDK QOS0 : 0"}
nx_mw :INFO :AWS demo passed
nx_mw :INFO :Demo Example Finished
```

4.7 Access_Manager

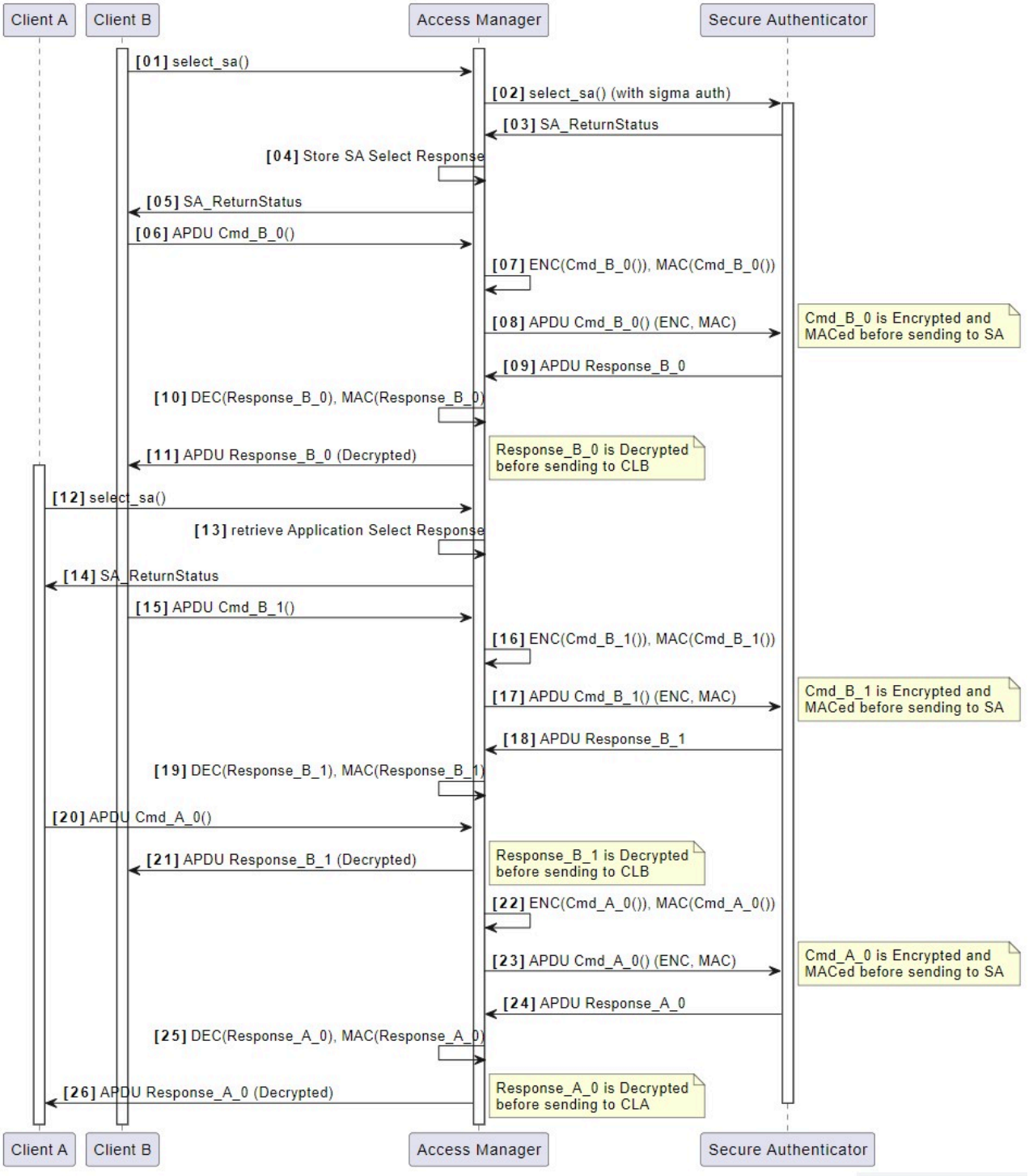
4.7.1 Access Manager

The Access Manager supports concurrent access from multiple processes to NX Secure Authenticator. Currently, Access Manager runs only in Linux environment but any process can connect to it via remote IP address. The Access Manager can establish a connection to the NX Secure Authenticator (SA) either as a plain connection, symmetric authentication or SIGMA_I authentication. Client processes can connect over the JRCP_V1_AM protocol to the Access Manager.

The following figure illustrates that the Access Manager is an independent process providing indirect access to the NX Secure Authenticator IC for client processes.



The following sequence diagram illustrates two processes connecting through the Access Manager to the Secure Element.



4.7.1.1 Supported Platforms

Access Manager can be used on linux and raspbian platforms with `NXMW_SMCOM` as **VCOM** or **T1oI2C_GP1_0**.

4.7.1.2 Build

Run the commands below to build access manager for NX secure authenticator. For details, refer **Linux build**.

```
git clone https://github.com/NXP/nxmw.git
cd nxmw/scripts
python create_cmake_projects.py
cd ../../nxmw_build/raspbian_native_nx_t1oi2c
cmake -DNXMW_Auth=SYMM_Auth -DNXMW_SMCOM=T1oI2C_GP1_0 -
DNXMW_Secure_Tunneling=NTAG_AES128_EV2 .
make accessManager
```

In case if the Access Manager is being built for SIGMA_I authentication, make sure that the SA is properly provisioned and that the environment variable `NX_AUTH_CERT_DIR` is set and pointing to the correct certificate repository before running the Access Manager. Refer **nx_Personalization**

4.7.1.3 Usage

Access manager accepts the optional arguments [ADDRESS_BINDING] and [PORT_NAME]. `ADDRESS_BINDING` determines whether the Access Manager accepts only localhost connections or allows remote connections. `PORT_NAME` determines the port through which Access Manager connects with the NX Secure Authenticator.

```
accessManager [ADDRESS_BINDING] [PORT_NAME] [-h] [--help]
```

ADDRESS_BINDING

- `local`: Only localhost connection accepted
- `remote`: Any supported connection accepted

PORT_NAME

- Any supported port name that access manager will use to connect to SA (like `/dev/i2c-1`)

If no arguments are given, localhost binding is used with default port name. The type of connection with SA (PLAIN, SYMMETRIC or SIGMA_I) is determined by the build configurations of Access Manager.

Example invocation:

```
bin/accessManager
bin/accessManager local
bin/accessManager remote /dev/i2c-1
```

To view help

```
bin/accessManager -h
bin/accessManager --help
```

PORT_NAME here **does not** refer to the port with which clients can connect to the Access Manager but rather the port with which the Access Manager connects to the SA.

Clients need to connect to the Access Manager using the port 8040 and this can be changed by changing the value of the macro `SERVERPORT`.

4.7.1.4 Demo: Concurrent access from 2 processes on RaspberryPi

- The demo requires an NX Secure Authenticator connected with a RaspberryPi. Interaction with the RaspberryPi is over three different shells. These shells can for example, be established via ssh from a PC on the same network or directly on the RaspberryPi.
- Build the Access Manager in a dedicated work area by following build instructions as above. Select static linking, enable **SYMM_Auth** on `NXMW_Auth` and use **T1oI2C_GP1_0** as `NXMW_SMCOM`.
- Build the NX_MW package in a dedicated work area by following build instructions as above. Select `NXMW_Auth` as **None**, `NXMW_Secure_Tunneling` to **None** and `NXMW_SMCOM` as **JRCP_V1_AM**.
- Start the Access Manager from a dedicated shell. Port name is not specified here and localhost binding is used for simplicity. Specific port name can be passed as last argument to accessManager:

```
./accessManager local
```

- Open another shell and run any example (such as ex_ecc) by providing IP address and port as connect string. Default port used is 8040 which can be changed by changing the value of macro `SERVERPORT`. IP address is 0.0.0.0 since it is hosted on the same machine:

```
./ex_ecc 0.0.0.0:8040
```

- Open another shell and run an example (such as ex_rng) similar to above example. To really see concurrent access, create two shell scripts to run the examples for a large number of times (say 100 times) and run them in separate shells.

User must make sure that one client process does not overwrite keys/data from another client process before it's completion. For example, if two instances of ex_ecc are ran in parallel, it may so happen that instance-1 creates a key at some ID and signs some data, after which instance-2 also creates another key at the same key ID. If instance-1 tries to verify its data using the same key ID, the verification will fail due to different keys.

4.7.1.5 NOTE:

- The Access Manager uses plain, symmetric or SIGMA_I authentication in the communication with the SA. Select the mode during building the accessManager.
- Client processes connect to the accessManager using the JRCP_V1_AM protocol.
- The Access Manager ensures APDU command/response pairs associated with a client process are executed without interference from another client process.
- The Access Manager does not connect to the SA at start up. It waits until a client process initiates a connection.
- If an APDU fails to be sent to SA or if the response for a particular APDU is an error code from the SA, the respective error code will be passed to the client process and session will be closed by the Access Manager. This is done because once an error code is sent by the SA, session needs to be restarted since SA does not accept further commands. Session will be reopened by the Access Manager on next client request. Some transient data may be lost due to the closing of the session, so client processes must check for the error codes sent by the Access Manager.
- Access Manager sends the APDUs to SA in the same sequence as they are received from clients. Hence, system integrator must ensure that execution of a command from one client does not affect the result of another command from another client (such as overwriting keys still in use by some other client).

4.7.1.6 Restrictions

- In case transmission of a command to the NX Secure Authenticator fails, the Access Manager will close the session. The Access Manager will not attempt to re-establish a broken connection until new connection request from client is received. To recognize and recover from a broken connection, a system integrator must

monitor failure to communicate to the NX Secure Authenticator by the client processes. As and if required the Access Manager must be restarted and the affected client processes must reconnect to the Access Manager.

- The Access Manager **does not**:
  - Handle power management
  - Keep track of NX Secure Authenticator resources

4.8 Other Examples

4.8.1 Certificate Access Right Demo

NTAGECC supports a private extension (ARExtension) for access right encoding within X.509 certificates. This project and another 2 demos nx_Personalization and ex_cert_ar_provision together show how the access right extension takes effect.

ex_cert_ar_provision will create a file (with file number 4) with FileAR.ReadWrite set to 1.

nx_Personalization will initialize CA Root Key with AC bitmap 0x0FFF which means access conditions 0x0 - 0xB are granted.

ex_cert_ar_file_op will setup Sigma-I authentication and read this file. It will fail when using certificate defined `demos/nx/cert_access_right/cert` because the certificate access right only grants access condition 0.

If using the certificates in `binaries/configuration/cert_depth3_PKCS7_rev1`, ex_cert_ar_file_op will pass because those certificates don't include access right extension. So access right is inherited from CA Root Key.

4.8.1.1 Pre-requisites

- `nx_Personalization` should run first with AC bitmap0 and bitmap1 set. Also certificates used are defined in `demos/nx/cert_access_right/cert`.
- `nx_Personalization` will initialize the certificates and keypairs for Sigma-I authentication.
- `ex_cert_ar_file_op` should run with certificates defined in `demos/nx/cert_access_right/cert`.

For Windows or Linux Like platform, it means set environment variable `NX_AUTH_CERT_DIR` to the certificate path.

For MCU like mcxn947, it means changing the certificates defined in `sss/ex/inc/ex_sss_nx_auth.h`

4.8.1.2 Building the Example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
- Project: `ex_cert_ar_file_op`
- Select NXMW_Auth to `NXMW_Auth=SIGMA_I_Verifier` or `NXMW_Auth=SIGMA_I_Prover`

4.8.1.3 Running the Example

If you have built a binary, flash the binary on to the board and reset the board.

```
If you have built an *exe* to be run from Windows, run as:

ex_cert_ar_file_op.exe <PORT NAME>
```

```
On Linux, run as:
```

```
./ex_cert_ar_file_op
```

4.8.1.4 Console output

If everything is successful, the output will be similar to:

```
nx_mw :INFO :Session Open Succeed
nx_mw :INFO :Running File Management Example ex_cert_ar_file_op.c
nx_mw :WARN :nxEnsure:'(ret == SM_OK) || (ret == SM_OK_ALT)' failed. At
 Line:3735 Function:sss_nx_TXn_AES_EV2
nx_mw :INFO :File read failed. Expected result if uses certificates of this
 demo !!!
nx_mw :INFO :ex_cert_ar_file_op Example Success !!!...
nx_mw :INFO :ex_sss Finished
```

4.8.2 Enable Certificate Cache Example

This project demonstrates the operation of the host and Secure Authenticator certificate cache, including enabling the Secure Authenticator certificate cache using Nx APIs. Enabling the certificate cache accelerates protocol execution by skipping the verification of certificates that have already been verified.

Refer **Certificate cache Example file**

4.8.2.1 Prerequisites

- Clear the host cache file.

4.8.2.2 About the Example

This example demonstrates enabling the certificate cache. If a certificate has been previously verified and is present in the cache, the verification process will be skipped. The steps in this example are as follows:

- Set up Sigma-I session 1 with CMake host cache enabled and exchange the certificate chain.
- Enable the Sigma-I certificate cache on the Secure Element (SE).
- Close session 1.
- Save the certificate in the host cache file.
- Set up Sigma-I session 2 with host certificate cache enabled. Since the Secure Authenticator's certificate hash and signature are cached, the host will not require certificates from the Secure Authenticator. Similarly, the Secure Authenticator will not require certificates from the host.
- Close session2 .

The example uses the following APIs and data types:

- `sss_session_close()`
- `nx_SetConfig_CertMgmt()`
- `sss_session_open()`

4.8.2.3 Building the Example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
- `NXMW_Auth=SIGMA_I_Verifier` or `NXMW_Auth=SIGMA_I_Prover`

- NXMW_Auth_Asymm_Host_PK_Cache=Enabled
- Project - ex_cert_cache

4.8.2.4 Console Output

If everything is successful, the output will be similar to:

```
sss    :INFO :Session Open Succeed
App    :INFO :Session 2 open succeed by certificate cache
App    :INFO :Close session 2
App    :INFO :ex_sss_cert_cache Example Success !!!…
App    :INFO :ex_sss Finished
```

4.8.3 Certificate Repository Example

This project demonstrates generating a certificate repository using SSS and Nx APIs. It will create repository, load certificate and mapping table and activate the repository.

Refer **Certificate Repo Example**

4.8.3.1 About the Example

This example creates a certificate repository and loads certificate chain into it. In detail:

- Set ECC private key associated with the repository.
- Create a certificate repository.
- Load certificate chains (Leaf, Parent and Grand Parent certificate)into repository.
- Load Mapping table into repository. This step is optional and only required when using host certificate wrapping e.g. PKCS#7. The wrapping basically provides a path, using ASN.1 encoding, to the start of the x.509 certificate.
- Activate repository.

The example uses the following APIs and data types:

- sss_key_object_init()
- sss_key_object_allocate_handle()
- sss_key_store_set_key()
- sss_key_object_free()
- nx_ManageCertRepo_CreateCertRepo()
- nx_ManageCertRepo_LoadCert()
- nx_ManageCertRepo_LoadCertMapping()
- nx_ManageCertRepo_ActivateRepo()

4.8.3.2 Building the example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
- Select NXMW_Auth to SIGMA_I_Verifier, SIGMA_I_Prover or SYMM_Auth
- NXMW_Secure_Tunneling to NTAG_AES128_EV2, NTAG_AES256_EV2 or NTAG_AES128_AES256_EV2 (only with SIGMA_I_Verifier or SIGMA_I_Prover)

- The authentication must be setup with access condition 0. This means App Master key for symmetric authentication. For Sigma-I authentication, this mean AC0 of certificate access right. It is defined with CA root key and reader certificate (or certificate chain).
- project - `ex_cert_repo`

4.8.3.3 Console output

If everything is successful, the output will be similar to:

```
sss    :INFO :Session Open Succeed
App    :INFO :Set ECC private key (3) associated with this repository.
App    :INFO :Load device leaf certificate Successful !!!
App    :INFO :Load device P1 certificate Successful !!!
App    :INFO :Load device P2 certificate Successful !!!
App    :INFO :Load leaf certificate mapping table for PKCS#7 wrapping
 Successful !!!
App    :INFO :Load P1 certificate mapping table for PKCS#7 wrapping
 Successful !!!
App    :INFO :Load P2 certificate mapping table for PKCS#7 wrapping
 Successful !!!
App    :INFO :Activate certificate repository Successful !!!
App    :INFO :ex_sss_cert_repo Example Success !!!...
App    :INFO :ex_sss Finished
```

4.8.4 Counter File Example

This project demonstrates counter file operations using Nx APIs.

Refer **Counter File Example**

4.8.4.1 About the Example

This example does following:

- Creates a counter file (file No is 5)
- Gets current counter value
- Increases counter by 3
- Gets current counter value

It uses the following APIs and data types:

- `nx_CreateCounterFile()`
- `nx_GetFileCounters()`
- `nx_IncrCounterFile()`

4.8.4.2 Building the Example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
- Project: `ex_counter_file`
- SIGMA_I_Verifier or SIGMA_I_Prover or SYMM_Auth should be used for NXMW_Auth.
- Due to the authentication requirement for nx_CreateCounterFile(), this demo should be run with access condition 0 (App Master Key)

4.8.4.3 Console output

If everything is successful, the output will be similar to:

```
sss    :INFO :Session Open Succeed
App    :INFO :Running File Management Example ex_sss_counter_file.c
App    :INFO :Counter File creation successful !!!
App    :INFO :Current counter value is 0x0
App    :INFO :Increase counter by 0x3
App    :INFO :Current counter value is 0x3
App    :INFO :ex_sss_counter_file Example Success !!!...
App    :INFO :ex_sss Finished
```

4.8.5 Diversification key perso Example

This project demonstrates diversification key personalization using SSS APIs.

Refer **Diversify Key Perso Example**

4.8.5.1 About the Example

This example derives the diversification key from master key and input parameters on host and injects the new key into Secure Authenticator.

The input parameter and master key can be set according to MCU type.

MCU without file system

1. Update the diversification key input parameters using macros EX_DIVERSIFY_INPUT_UID EX_DIVERSIFY_INPUT_AID EX_DIVERSIFY_INPUT_SID

Refer: `nxmw/lib/sss/ex/inc/ex_sss_nx_auth.h`

1. Update the master key using macro with `NXMW_Auth=SYMM_Auth`. case1 16-byte master key: EX_SYMM_AUTH_AES128_KEY case2 32-byte master key: EX_SYMM_AUTH_AES256_KEY
2. Update the master key using macro with `NXMW_Auth=SIGMA_I_Verifier` or `NXMW_Auth=SIGMA_I_Prover`. case1 16-byte master key: Define EX_SIGMA_I_AUTH_DEFAULT_AESKEY as EX_SYMM_AUTH_AES128_KEY Define EX_SIGMA_I_AUTH_DEFAULT_AESKEY_LEN as EX_SYMM_AUTH_AES128_KEY_SIZE case2 32-byte master key: Define EX_SIGMA_I_AUTH_DEFAULT_AESKEY as EX_SYMM_AUTH_AES256_KEY Define EX_SIGMA_I_AUTH_DEFAULT_AESKEY_LEN as EX_SYMM_AUTH_AES256_KEY_SIZE

Refer - `nxmw/lib/sss/ex/inc/ex_sss_nx_auth.h`

MCU with file system

1. Update the diversification key input parameters in configuration file. UID 00010203040506 AID 3042F5 SID 4E585020416275
   Refer - `nxmw/binaries/configuration/diversify_key_inputs/plain_dkey_input.txt`
2. Update the master key in configuration file.
   case1 16-byte master key : APPKEY 00000000000000000000000000000000 case2 32-byte master key : APPKEY 0000000000000000000000000000000000000000000000000000000000000000
   Refer - `nxmw/binaries/configuration/symmetric_keys/plain_appkey.txt`

To inject the newly generated key (keyID1), old key value is also required. Old key value is defined as MACRO

- case1 16-byte aes old key: EX_AES128_KEYID_1_OLD_KEY
- case2 32-byte aes old key: EX_AES256_KEYID_1_OLD_KEY
  Refer - `nxmw/lib/sss/ex/inc/ex_sss_nx_auth.h`

Note: This example implements the master key diversification in the host MCU for demonstration purposes. In real life use cases, depending on the security requirements of the system, master key storage and diversification should be done in a Secure Authenticator.

Construct diversification input as per AN10922.pdf and use host crypto library to get derived diversification key.

After generating the diversification key on host, this demo will setup Sigma-I/symmetric authentication session to inject diversification key into SE with KeyID 1. To build this project, use cmake option `NXMW_Auth_Symm_Diversify=Disabled` so that master key is used for authentication.

It uses the following APIs and data types: `sss_key_object_init()` `sss_key_object_allocate_handle()` `sss_key_store_set_key()` `sss_key_object_free()` `sss_mac_context_init()` `sss_mac_one_go()`

4.8.5.2 Building the Example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
  - Select CMake options:
    - `NXMW_Auth_Symm_Diversify=Disabled`
    - `NXMW_Auth=SIGMA_I_Verifier` or `NXMW_Auth=SIGMA_I_Prover` or `NXMW_Auth=SYMM_Auth`
    - `NXMW_Auth_Symm_App_Key_Id=0`
  - Project:`ex_diversify_key_perso`

4.8.5.3 Apply key diversification with symmetric authentication

After running this example, diversification key can be used for symmetric authentication which is used in other examples (e.g. nx_Minimal).

- Diversified key is used for symmetric authentication by `NXMW_Auth_Symm_Diversify=Enabled` in cmake options.

Build and run nx_Minimal example with cmake options.

- `NXMW_Auth_Symm_Diversify=Enabled`
- `NXMW_Auth=SYMM_Auth`
- `NXMW_Auth_Symm_App_Key_Id=1`

4.8.5.4 Console output

If everything is successful, the output will be similar to:

```
sss   :INFO :Session Open Succeed
App   :WARN :get inputs to derive dkey from:'C:\nxp\configuration
\plain_dkey_input.txt' (FILE=C:\nxp\configuration\plain_dkey_input.txt)
App   :INFO :uid (Len=7)
      00 01 02 03    04 05 06
App   :INFO :aid (Len=3)
      30 42 F5
App   :INFO :sid (Len=7)
      4E 58 50 20    41 62 75
App   :WARN :Using appkeys from:'C:\nxp\configuration\plain_appkey.txt' (FILE=C:
\nxp\configuration\plain_appkey.txt)
App   :INFO :masterKey (Len=16)
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
App   :INFO :diversifyKey (Len=16)
```

```
      76 92 01 0C    DA C5 22 63    FF 38 26 25    12 2E 48 3E
App   :INFO :successfully set diversifyKey into Nx
App   :INFO :ex_diversify_key_perso Example Success !!!...
App   :INFO :ex_sss Finished
```

4.8.6 Dual Interfaces Example

This project demonstrates NFC pause feature in a dual interfaces (NFC and I2C interfaces) case.

Refer **Dual Interface Example**

4.8.6.1 Prerequisites

- Hardware: NX SA should be connected to a supported MCU or Raspberry Pi.
- Nx middleware stack.
- NTAG configuration tool (e.g. RFIDDiscover) is available on NFC host.

4.8.6.2 About the Example

The NFC Pause feature allows to transfer control from an NFC Host to an MCU controlling NTAGECC as a master via the I2C interface. There are 2 ways to activate NFC pause:

- Cmd.ManageGPIO
- Cmd.ISOReadBinary/Cmd.ReadData

When NFC Pause is triggered, NTAGECC shall halt processing on the NFC interface until the Cmd.ManageGPIO is received on the I2C interface to release the NFC Pause.

NTAG configuration tool (e.g. RFIDDiscover) on NFC host is used to trigger NFC pause and this example will release NFC pause.

1. Initialize MCU GPIO as input pin. On FRDM-MCXN947, it is PTB3.
2. Read current GPIO status as initial GPIO status.
3. Read GPIO status at 1s interval and wait until status changes.
4. NFC host(RFIDDiscover) selects NTAGECC application from NFC interface.
5. NFC host(RFIDDiscover) setups symmetric authentication with Secure Authenticator.
6. NFC host(RFIDDiscover) configures GPIO2 for output or output with NFCPause file. If it is latter then NFC Pause file should also be configured. NFC pause file is supposed to be file 2 (can be changed by modifying EX_DUAL_INTERFACE_NFC_PAUSE_FILE_NO in ex_dual_interfaces.c).
7. NFC host(RFIDDiscover) calls Cmd.ManageGPIO with NFC Pause or Cmd.ReadData.
8. ex_dual_interfaces will detect GPIO status change.
9. ex_dual_interfaces will write to NFCPause file.
10. ex_dual_interfaces will call Cmd.ManageGPIO with NFC Pause Release.
11. NFC host(RFIDDiscover) gets response for Cmd.ManageGPIO or Cmd.ReadData.
12. NFC host(RFIDDiscover) calls Cmd.ManageGPIO with NFC Pause or Cmd.ReadData.
13. ex_dual_interfaces will close session.

It uses the following APIs and data types:

- `nx_WriteData()`
- `nx_ManageGPIO_Output()`

4.8.6.3 Building the example

- Build NX middleware stack on Linux. Refer **Linux build**.

- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
  - Project: `ex_dual_interfaces`
  - Select NXMW_Auth to None
  - NXMW_Secure_Tunneling to None

4.8.6.4 How to use

Run the tool as: ex_dual_interfaces.exe on windows or ./ex_dual_interfaces on Raspberry Pi.

4.8.6.5 Console output

If everything is successful, the output will be similar to:

```
nx_mw :WARN :Communication channel is Plain.
nx_mw :WARN :!!!Security and privacy must be assessed.!!!
nx_mw :INFO :Session Open Succeed
nx_mw :INFO :HOST GPIO (PTB3) Init Status: Low
nx_mw :INFO :HOST GPIO (PTB3) Status Changed: High
nx_mw :INFO :Write NDEF File Data (Offset 0x0, Length 0x8).
nx_mw :INFO :Read External I2C Sensor Could Be Called Here!
nx_mw :INFO :Toggle GPIO2 And Release NFC Pause.
nx_mw :INFO :Read HOST GPIO (PTB3): Low
nx_mw :INFO :Wait session close until NFC pause
nx_mw :INFO :HOST GPIO (PTB3) Status Changed: High
nx_mw :INFO :ex_dual_interfaces Example Success !!!...
nx_mw :INFO :ex_sss Finished
```

4.8.7 GPIO Example

This project demonstrates a GPIO output / input operation using Nx APIs.

Refer **GPIO Example**

4.8.7.1 Prerequisites

- Hardware: NX SA should be connected to FRDM-MCXN947 board or Raspberry Pi.
- GPIO1 should be already been configured to output and GPIO2 should be configured to input. **nx_tool_setconfig** or **ex_set_config** can be used for configuration.

4.8.7.2 About the Example

This example does a GPIO output / input operation.

It uses the following APIs and data types: (including the ones used to control host board GPIOs)

```
`nx_ManageGPIO_Output()`
`nx_ReadGPIO()`
`nx_host_GPIOInit()`
`nx_host_GPIORead()`
`nx_host_GPIOClear()`
`nx_host_GPIOSet()`
`nx_host_GPIOClose()`
`nx_host_GPIOClose()`
`GPIO_PinInit()`
`GPIO_PinRead()`
```

```
`GPIO_PortClear()`
`GPIO_PortSet()`
```

4.8.7.3 Building the example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
  - Project - `ex_gpio`
  - NXMW_Auth and NXMW_Secure_Tunneling should be selected according to simulator/IC configuration.

4.8.7.4 How to Run example

```
./nx_tool_setconfig -gpio1mode output -gpio2mode input -gpioMgmtCM full -
gpioReadCM full -gpioMgmtAC 0x0 -gpioReadAC 0x0
```

4.8.7.5 Console output

If everything is successful, the output will be similar to:

```
sss    :INFO :Session Open Succeed
App    :INFO :Clear GPIO1.
App    :INFO :Read HOST GPIO (PTB2): Low
App    :INFO :Set GPIO1.
App    :INFO :Read HOST GPIO (PTB2): High
App    :INFO :Toggle GPIO1.
App    :INFO :Read HOST GPIO (PTB2): Low
App    :INFO :Set HOST GPIO (PTB3): Low
App    :INFO :Read GPIO2 is low.
App    :INFO :Set HOST GPIO (PTB3): High
App    :INFO :Read GPIO2 is low.
App    :INFO :ex_sss_gpio Example Success !!!...
App    :INFO :ex_sss Finished
```

4.8.8 GPIO Notification Example

This project demonstrates a GPIO Notification operation using Nx APIs.

Refer **Get GPIO Notification Example**

4.8.8.1 Prerequisites

- Hardware: NX SA should be connected to FRDM-MCXN947 board or Raspberry Pi.
- GPIO1 should be already configured to output and also enable GPIO1 has GPIONotif. "nx_tool_setconfig"(Refer `nx_tool_setconfig`) or "ex_set_config" (GPIO1 notification should be enabled by setting macro SET_CONFIG_GPIO_NOTIFY_ENABLE=1. Refer `ex-sss-set-config`) can be used for configuration.

4.8.8.2 About the Example

This example does a GPIO Notif operation on authentication and read the status of GPIO.

The targeted GPIO will be HIGH, after successful execution of SIGMA-I mutual authentication, i.e. when a session is opened with SIGMA_I The targeted GPIO will be LOW, when losing authentication state, e.g. when a session is opened with none authentication.

It uses the following APIs and data types:

```
`sss_session_open()`
`nx_host_GPIOInit()`
`nx_host_GPIORead()`
`sss_session_close()`
```

4.8.8.3 Building the Example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
  - Project - `ex_gpio_notif`
  - `NXMW_All_Auth_Code=Enabled` and `NXMW_Auth` as either `SYMM_Auth`, `SIGMA_I_Verifier` or `SIGMA_I_Prover` should be selected according to IC configuration.

4.8.8.4 How to Run Example

```
./nx_tool_setconfig -gpio1mode output -gpio1Notif auth -gpioMgmtCM full -
gpioReadCM full -gpioMgmtAC 0x0 -gpioReadAC 0x0
```

4.8.8.5 Console output

If everything is successful, the output will be similar to:

```
sss   :INFO :Session Open Succeed
App   :INFO :Successfully opened SIGMA-session
App   :INFO :Read GPIONotif: High
Opening COM Port '\\.\COM10'

 Already  COM Port Open
 sss   :INFO :atr (Len=22)
      01 04 63 07    00 93 02 08    00 02 03 E8    00 01 00 64
      04 03 E8 00    FE 00
sss   :WARN :Communication channel is Plain.
sss   :WARN :!!!Not recommended for production use.!!!
sss   :INFO :Session Open Succeed
App   :INFO :Successfully opened Plain-session
App   :INFO :Read GPIONotif: Low
App   :INFO :ex_sss Finished
```

4.8.9 Multiple Symmetric Authentication Example

This project demonstrates Multiple Symmetric Authentication using SSS APIs

Refer **Multiple symmetric authentication**

4.8.9.1 About the Example

AES-based symmetric authentication is managed by Cmd.AuthenticateEV2First and Cmd.AuthenticateEV2-NonFirst. A First Authentication is done in state VCState.NotAuthenticated or in one of the authenticated states. The Non-First Authentication can only be applied after a First Authentication, i.e. in an authenticated state.

This example demonstrate how to setup 2 symmetric authentications with Cmd.AuthenticateEV2First and Cmd.AuthenticateEV2-NonFirst.

- Open session 1 with symmetric authentication i.e. cmd.authenticateEV2First. AES128 or AES256 session keys are generated which are used for EV2 secure messaging.
- Create and write file with session 1 EV2 secure tunneling.
- Bind session 2 to session 1. This allows cryptographically binding all messages within a transaction by using a transaction identifier TI and command counter CmdCtr.
- Open session 2 with symmetric authentication i.e. cmd.authenticateNonFirst. Session keys are re-generated.
- Read file with session 2 EV2 secure tunneling.

It uses the following APIs and data types: `sss_session_open()` `ex_sss_session_close()` `nx_CreateStdDataFile()` `nx_WriteData()` `nx_ReadData()`

4.8.9.2 Building the example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
  - Project - `ex_multiple_symm_auth`
  - `NXMW_Auth=SYMM_Auth`
  - `NXMW_Auth_Symm_App_Key_Id=0`

4.8.9.3 Console output

If everything is successful, the output will be similar to:

```
sss    :INFO :Session Open Succeed
App    :INFO :Standard Data File creation successful !!!
App    :INFO :File write successful !!!
App    :INFO :Bind session 1 to session 2
App    :INFO :Using default appkey. You can use appkeys from file by setting
 ENV=EX_SSS_BOOT_APPKEY_PATH to its path
App    :INFO :Trying to open session 2
sss    :INFO :Session Open Succeed
App    :INFO :File read successful !!!
App    :INFO :Unbind session 2
App    :INFO :Session 2 close
App    :INFO :ex_multiple_symm_auth Example Success !!!...
App    :INFO :ex_sss Finished
```

4.8.10 NX Deep Power down example

This project is used to demonstrate the deep power down on NX SA using T=1oI2C command.

The example will send a proprietary deep power down command (as described in the Datasheet) via I2C link to bring the device into the deep power down mode. The example will wait for 10 seconds before reopening the session which will also make the secure element wakeup from deep power down mode.

4.8.10.1 Building the Demo

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
- Project - `nx_deep_pwr_down`

4.8.10.2 Running the Example

If you have built a binary, flash the binary on to the board and reset the board.

If you have built an *exe* to be run from Windows using VCOM, run as:

```
nx_deep_pwr_down.exe <PORT NAME>
```

Where **<PORT NAME>** is the VCOM COM port.

On Raspberry-Pi or iMX board, run as:

```
./nx_deep_pwr_down
```

4.8.10.3 Console Output

If everything is successful, the output will be similar to:

```
nx_mw :INFO :Session Open Succeed
nx_mw :INFO :session_ctx->authType 0
nx_mw :INFO :Available free memory: 17632 bytes
nx_mw :INFO :Send deep power down command to the IC

nx_mw :INFO :Sleep for 10 seconds

nx_mw :INFO :cip (Len=22)
      01 04 63 07    00 93 02 08    00 02 03 E8    00 01 00 64
      04 03 E8 00    FE 00
nx_mw :WARN :Communication channel is Plain.
nx_mw :WARN :!!!Security and privacy must be assessed.!!!
nx_mw :INFO :Session Open Succeed
nx_mw :INFO :session_ctx->authType 0
nx_mw :INFO :Available free memory: 17632 bytes
nx_mw :INFO :nx_deep_pwr_down Example Success !!!...
nx_mw :INFO :ex_sss Finished
```

4.8.11 Update Key Example

This project demonstrates update aes key using sss apis.

4.8.11.1 About the Example

This project demonstrates update new aes key operation using sss apis.

- Update the new aes key use macro "EX_UPDATE_NEW_AESKEY".
- Update the old aes key use macro "EX_UPDATE_OLD_AESKEY".
- Update key ID use macro "EX_UPDATE_KEY_ID" (default it's 0).

Refer **Update Key Example code**

It uses the following APIs and data types:

```
`sss_key_object_init()`
`sss_key_object_allocate_handle()`
`sss_key_store_set_key()`
`sss_key_object_free()`
```

4.8.11.2 Building the Example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
  - Select CMake options:
    - `NXMW_Auth=SYMM_Auth`
    - `NXMW_Auth_Symm_App_Key_Id=0`
    - `NXMW_Secure_Tunneling=NTAG_AES128_EV2`
  - project - `ex_update_key`

4.8.11.3 Apply AES256 key with symmetric authentication

After running this example, AES256 key can be used for symmetric authentication which is used in other examples (e.g. nx_Minimal).

- AES256 key is used for symmetric authentication by `NXMW_Secure_Tunneling=NTAG_AES256_EV2` in cmake options.
- Host without file system - Update the AES key using macro with `NXMW_Auth=SYMM_Auth`. case1 16-byte AES key: EX_SYMM_AUTH_AES128_KEY case2 32-byte AES key: EX_SYMM_AUTH_AES256_KEY Refer **nxmw/lib/sss/ex/inc/ex_sss_nx_auth.h**
- Host with file system. - Update the AES key in configuration file. case1 16-byte AES key : APPKEY 00000000000000000000000000000000 case2 32-byte AES key : APPKEY 0000000000000000000000000000000000000000000000000000000000000000
- Build and run nx_Minimal example with cmake options. `NXMW_Auth=SYMM_Auth` `NXMW_Auth_Symm_App_Key_Id=0` `NXMW_Secure_Tunneling=NTAG_AES256_EV2`

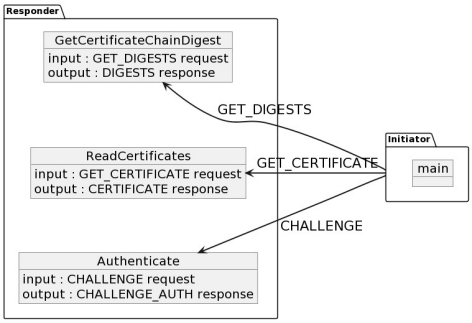
4.8.11.4 Console output

If everything is successful, the output will be similar to:

```
sss   :INFO :Session Open Succeed
App   :INFO :Old aes key (Len=16)
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
App   :INFO :New aes key (Len=32)
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
App   :INFO :Successfully aes key injected into nx
App   :INFO :ex_sss_update_key Example Success !!!...
App   :INFO :ex_sss Finished
```

4.8.12 Universal Serial Bus Type-C (USB-C) Authentication demo

This project is used to demonstrate the USB-C authentication flow between an Auth Responder and an Auth Initiator. The Auth Responder implements 3 functions for the 3 authentication requests an Auth Initiator can issue to the an Auth Responder : `GetCertificateChainDigest`, `ReadCertificates`, `Authenticate`.



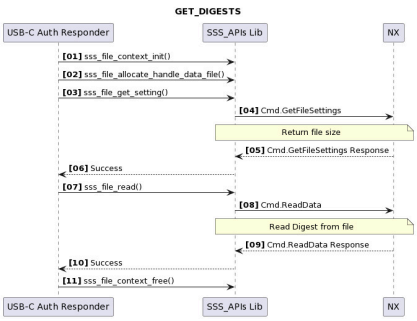
4.8.12.1 Pre-requisites

- The Secure Authenticator should be trust provisioned with correct keys and certificates for USB-C Authentication. Keys and certificates can be provisioned for test purpose by updating keys in `demos/nx/usb_c/usb_c_provisioning/usb_c_credentials.c` and running example `usb_c_provisioning`.
- By default USB-IF Root certificate is used in the certificate chain. If example `usb_c_provisioning` is run, you would need to disable macro `USE_ROOT_USBIF` in `usb_c_rootcert.c` to use test RootCA: usb_c_auth/usb_c_rootcert.c

> **Note:** If building and running this demo with SIGMA-I, make sure to personalize the SA on repo ID other than 0x00 as this ID is used by `usb_c_provisioning` to inject keys and certificates which will overwrite the certificate repository. For details on personalization, refer **NX Personalization**

4.8.12.2 GetCertificateChainDigest (GET_DIGESTS)

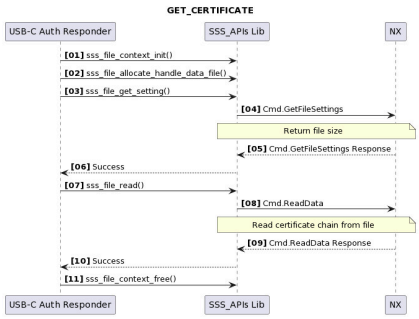
This function reads the digests of certificate chains stored inside the Secure Authenticator and returns all the digests as requested by the USB-C initiator.



Refer - `usb_c_responder_auth/usb_c_responder.c`

4.8.12.3 ReadCertificates (GET_CERTIFICATE)

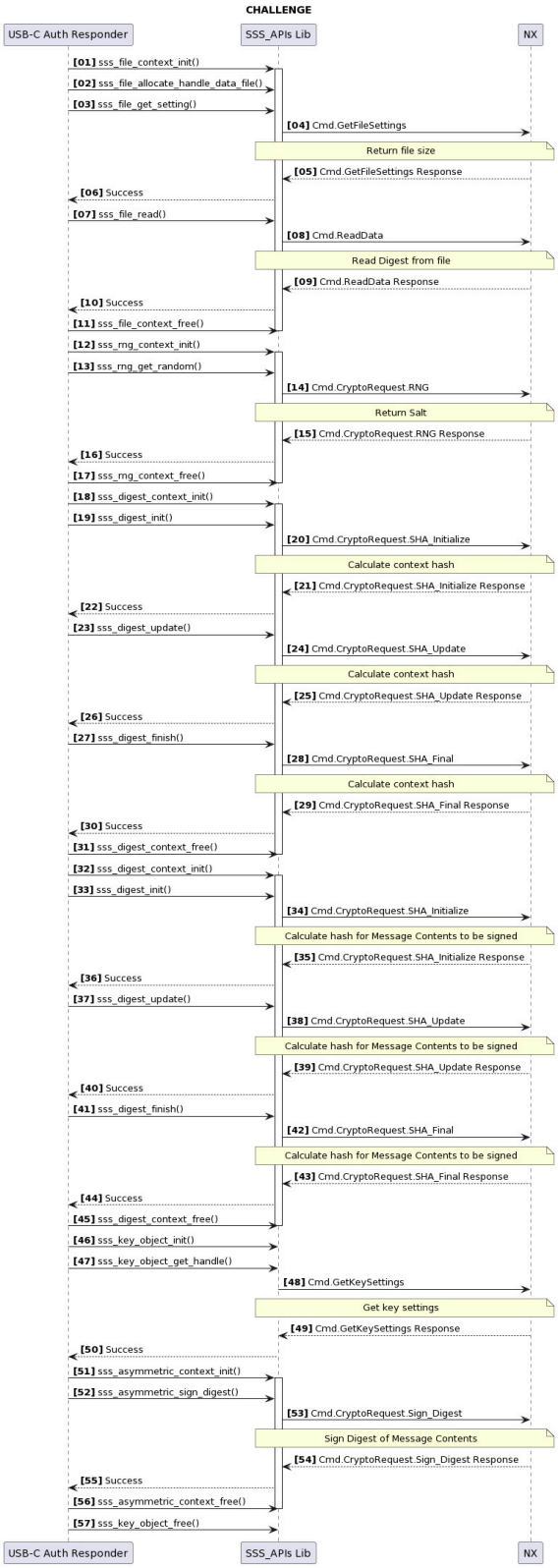
This function reads the certificate chain on the provided slot ID starting from the provided offset which starts from the beginning of the certificate chain and reading provided length bytes.



Refer - `usb_c_responder_auth/usb_c_responder.c`

4.8.12.4 Authenticate (CHALLENGE)

This function performs the CHALLENGE operation and returns the signature `R` and signature `S` values to the power receiver.



Refer - `usb_c_responder_auth/usb_c_responder.c`

4.8.12.5 Building the Demo

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
  - Project - `usb_c_auth`

Note: USB-C Demo on MCXA153 currently supports Sigma Authentication with Mbedtls2. Symmetric Authentication is supported with Mbedtls 2 and Mbedtls 3.

4.8.12.6 Running the Example

```
./usb_c_auth
```

4.8.12.7 Console output

On successful execution you should be able to see logs as:

```
sss   :INFO :Session Open Succeed
App   :INFO :Send command GET_DIGESTS
App   :INFO :Retrieved digest (Len=32)
      66 09 26 B6    CB 61 86 5C    60 78 1A 98    92 AB F4 B7
      C2 4A B6 27    7C 2A 69 84    8A C6 90 B4    1C 18 63 E1
App   :INFO :Send command GET_CERTIFICATE
App   :INFO :Certificate chain digest successfully verified
App   :INFO :Retrieved Leaf Certificate (Len=479)
      30 82 01 DB    30 82 01 81    A0 03 02 01    02 02 09 00
      81 9F 59 97    6B 38 47 91    30 0A 06 08    2A 86 48 CE
      3D 04 03 02    30 2B 31 15    30 13 06 03    55 04 0A 0C
      0C 4F 72 67    4E 61 6D 65    20 49 6E 63    2E 31 12 30
      10 06 03 55    04 03 0C 09    55 53 42 3A    31 61 30 61
      3A 30 22 18    0F 31 39 37    30 30 31 30    31 30 30 30
      30 30 30 5A    18 0F 39 39    39 39 31 32    33 31 32 33
      35 39 35 39    5A 30 42 31    15 30 13 06    03 55 04 0A
      0C 0C 4F 72    67 4E 61 6D    65 20 49 6E    63 2E 31 16
      30 14 06 03    55 04 03 0C    0D 55 53 42    3A 31 61 30
      61 3A 30 31    30 31 31 11    30 0F 06 03    55 04 05 13
      08 35 35 36    36 37 37 38    38 30 59 30    13 06 07 2A
      86 48 CE 3D    02 01 06 08    2A 86 48 CE    3D 03 01 07
      03 42 00 04    A9 B0 F8 66    B0 2D 91 2B    87 64 22 57
      D2 AE 0B 07    E1 FA 83 A6    8E B4 4F 3F    16 79 43 A3
      E5 D8 00 72    DC 0A D5 B3    00 A4 FB 8B    C0 05 3B 4D
      7C 9D 8D 48    BB AC 46 8C    E5 28 B7 5B    1C 5A FA 2E
      4D DA 38 45    A3 73 30 71    30 0C 06 03    55 1D 13 01
      01 FF 04 02    30 00 30 0B    06 03 55 1D    0F 04 04 03
      02 07 80 30    13 06 03 55    1D 25 01 01    FF 04 09 30
      07 06 05 67    81 11 01 01    30 3F 06 05    67 81 11 01
      02 04 36 00    02 40 00 01    04 00 00 12    34 02 16 02
      01 01 00 03    07 01 00 2A    0A 2A 0A 2A    0A 00 00 00
      01 2A 01 91    2C 05 06 00    00 00 55 1A    0A FD 04 54
      45 53 54 FE    04 1A 0A 12    34 30 0A 06    08 2A 86 48
      CE 3D 04 03    02 03 48 00    30 45 02 20    73 DB 58 34
      78 91 0C B9    C3 F5 6C 00    AB 6E 9C E0    E0 13 B1 A3
      41 D4 12 5F    3B E0 E9 31    A3 C3 6E D5    02 21 00 C6
      CF 1D BD 9D    0F B1 2C 5B    EF 17 4E 30    44 52 C3 27
      FA A2 FA 40    29 F5 49 65    31 21 05 C2    BB 2B 35
App   :INFO :Certificate chain successfully verified
```

```
App   :INFO :Retrieved Leaf certificate public key (Len=65)
      04 A9 B0 F8    66 B0 2D 91    2B 87 64 22    57 D2 AE 0B
      07 E1 FA 83    A6 8E B4 4F    3F 16 79 43    A3 E5 D8 00
      72 DC 0A D5    B3 00 A4 FB    8B C0 05 3B    4D 7C 9D 8D
      48 BB AC 46    8C E5 28 B7    5B 1C 5A FA    2E 4D DA 38
      45
App   :INFO :Send command CHALLENGE
App   :INFO :Challenge Signature (Len=64)
      D2 BF 7D C1    60 87 1B E9    26 39 C1 12    4B 2B BF 3A
      74 12 53 A7    34 B2 9E 9A    6C 4D DB CD    3F 5F 93 92
      98 40 7A 62    A4 FA F1 98    AB BF E5 9B    C0 E1 38 51
      76 B9 15 0D    79 C5 59 28    C9 1B 94 1C    53 BE 9E 5F
App   :INFO :Message Content (Len=140)
      01 83 00 00    30 61 B9 F0    7F 61 97 9B    90 5A DE 8F
      6C 80 23 74    42 AB C3 BD    5B 7B C9 84    CC 69 2E AE
      0F 62 D3 BA    01 03 00 01    01 01 01 00    66 09 26 B6
      CB 61 86 5C    60 78 1A 98    92 AB F4 B7    C2 4A B6 27
      7C 2A 69 84    8A C6 90 B4    1C 18 63 E1    EA CB 25 1A
      C4 B2 90 E2    3E 29 6E AD    5C 68 0E 79    86 50 44 C4
      79 93 55 4D    9E C9 4A 6C    56 9A F6 70    66 68 7A AD
      F8 62 BD 77    6C 8F C1 8B    8E 9F 8E 20    08 97 14 85
      6E E2 33 B3    90 2A 59 1D    0D 5F 29 25
App   :INFO :Challenge successfully verified
App   :INFO :usb_c_auth Example Success !!!...
App   :INFO :ex_sss Finished
```

4.8.12.8 Porting

The example allows porting of host verification and host RNG functions in case mbedTLS is not available. If you want to add your own implementation of these operations, update the following APIs in `usb_c_initiator_auth/usb_c_initiator_helper_port.c`:

4.8.13 Originality Check Example

During manufacturing, NTAGECC is trust-provisioned with an ECC-based key pair and related certificate to allow verification of the genuineness of the IC. The originality check is done by executing a card-unilateral authentication through a challenge-response protocol.

This project demonstrates originality check at PICC level using SSS/Nx APIs. It will do following

- Send random challenge to Secure Authenticator.
- Read originality check certificate.
- Verify Cert.Orig against the Originality CA Public key.
- Verify the signature received from Secure Authenticator with public key from certificate.

The Originality CA Public key is defined by macro EX_ORIG_CA_PUBLIC_KEY in ex_sss_originality_check.h User should update it with correct key value.

Refer **ex_sss_originality_check.c**

4.8.13.1 About the Example

This example does an originality check at PICC level by executing a card-unilateral authentication.

It uses the following APIs and data types: `nx_ISOInternalAuthenticate()` `nx_ReadData()` `sss_asymmetric_context_init()` `sss_asymmetric_verify_one_go()` `sss_asymmetric_context_free()`

4.8.13.2 Building the example

- Build NX middleware stack on Linux. Refer **Linux build**.
- Build NX middleware stack for Windows. Refer **Windows build**.
- Build NX middleware stack for supported MCUs. Refer **MCUX Cmake build**.
  - Project - `ex_originality_check`
  - Select NXMW_Auth to None
  - NXMW_Secure_Tunneling to None
  - NXMW_NX_Type to NX_PICC

4.8.13.3 How to use

Run the tool as:

If you have built a binary, flash the binary on to the board and reset the board.

```
If you have built an *exe* to be run from Windows, run as:

ex_originality_check.exe <PORT NAME>
```

```
On Linux, run as:

./ex_originality_check
```

4.8.13.4 Console output

If everything is successful, the output will be similar to:

```
sss   :INFO :Session Open Succeed
App   :INFO :Generate RndA (Len=16)
      C4 F4 F3 4B    46 DB 86 58    9E 9B 04 C3    2F DF 8E 51
App   :INFO :Use OptsA (Len=32)
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
App   :INFO :Send Cmd.ISOInternalAuthenticate.
App   :INFO :Rx RndB (Len=16)
      82 49 B5 1A    E7 6A E7 88    A1 14 60 3A    E8 2C 02 1F
App   :INFO :Rx Signature (Len=64)
      A3 F4 A3 A4    AA 3C 29 19    A1 4C EA 08    A5 9B 50 E5
      99 4A 96 B7    52 B5 ED DB    AD FC 97 E8    F9 BF 8D 12
      A1 3C 72 D6    47 EB 7E ED    BB C7 90 B6    A5 0D 3C EF
      41 83 1C 6D    E2 2C 9A BC    EC 45 1B 25    30 29 DE BC
App   :INFO :Read certificate.
App   :INFO :Verify Cert.Orig against the Originality CA Public key Success.
App   :INFO :Get public key from certificate.
App   :INFO :Public key (Len=91)
      30 59 30 13    06 07 2A 86    48 CE 3D 02    01 06 08 2A
      86 48 CE 3D    03 01 07 03    42 00 04 45    2D FB F7 98
      49 29 69 EF    BB 24 B6 58    E6 05 E5 26    71 37 70 F3
      4B E8 7A 57    9E 1C F1 34    79 1A A7 37    1D C8 09 39
      A4 D9 5C 1E    6C 7C 2C 52    D4 7A FF C3    E4 EC 46 28
      23 DB DE BB    AD 1B 28 9F    1D CF 29
App   :INFO :verify Signature Success.
App   :INFO :Originality Check Example Success !!!...
App   :INFO :ex_sss Finished
```

4.8.14 NX Release Request Command example

This project is used to demonstrate the release request command on NX using T=1oI2C API.

4.8.14.1 Build

Build NX middleware stack. Refer **Linux build**.

- Project - `nx_release_req_cmd`
- Set NXMW_Host as the appropriate platform

4.8.14.2 Running the Example

For board, flash the built binary on the board and reset.

On Windows (to run using VCOM), run the generated executable as:

```
nx_release_req_cmd.exe <PORT NAME>
```

Where **<PORT NAME>** is the VCOM COM port.

On Raspberry-Pi or iMX board, run the generated binary as:

```
./nx_release_req_cmd
```

4.8.14.3 Console Output

If everything is successful, the output will be similar to:

```
nx_mw :INFO :Session Open Succeed
nx_mw :INFO :Send release req command the IC

nx_mw :INFO :Sleep for 10 seconds

nx_mw :INFO :session_ctx->authType 5
nx_mw :INFO :Available free memory: 8672 bytes
nx_mw :INFO :nx_release_req_cmd Example Success !!!...
nx_mw :INFO :ex_sss Finished
```

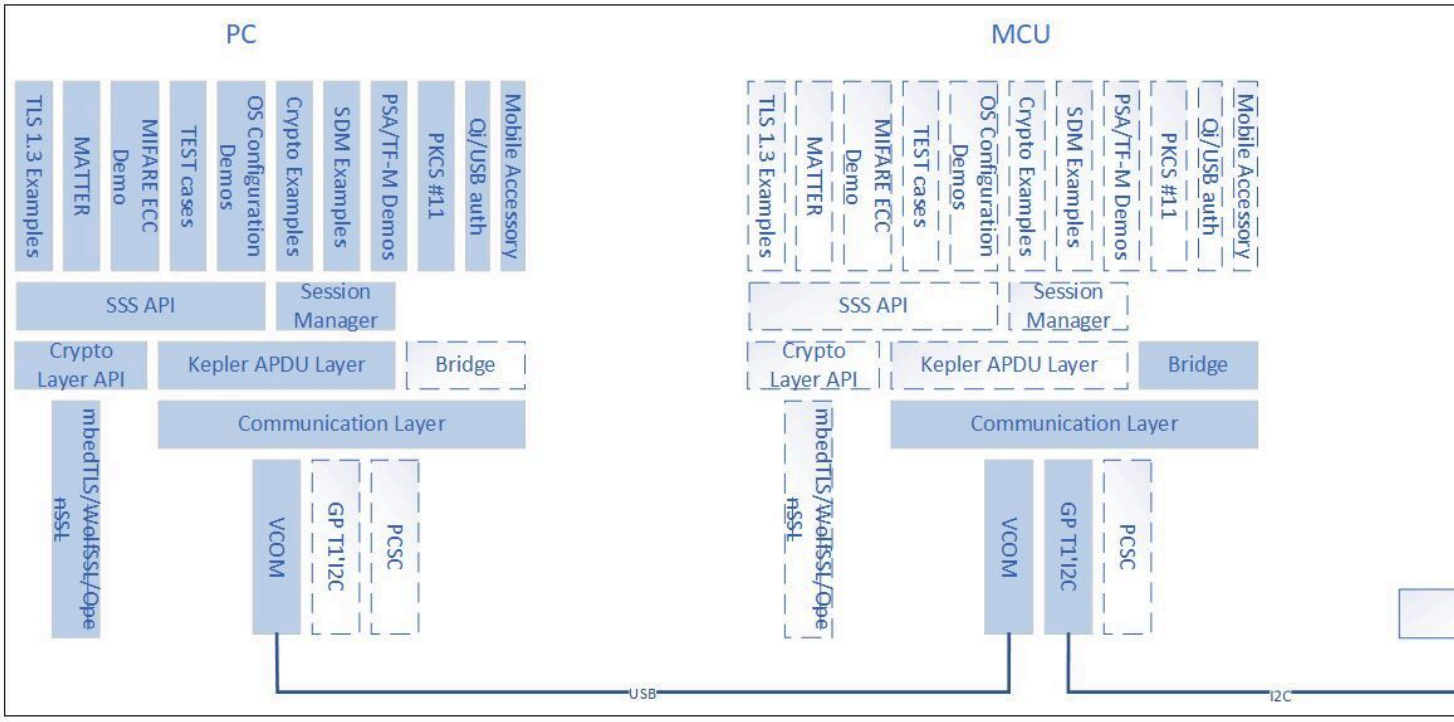
4.8.15 VCOM

Virtual COM Port interface can be used to connect to the SA from development PC/Laptop. This gives advantage of faster turn around time for development and feature experimentation.

Note: If the VCOM drivers are not installed on your PC, you would have to install it.

Embedded platforms supported: `LPCXpresso55s69` `frdmmcxn947` `frdmmcxa153`

The overall architecture looks like this:



4.8.15.1 Building the Demo

- Nx middleware stack. Refer **Build**.

Build project:

- Project: `vcom`

4.8.15.2 Steps to use VCOM for running the examples - PC & supported embedded platforms

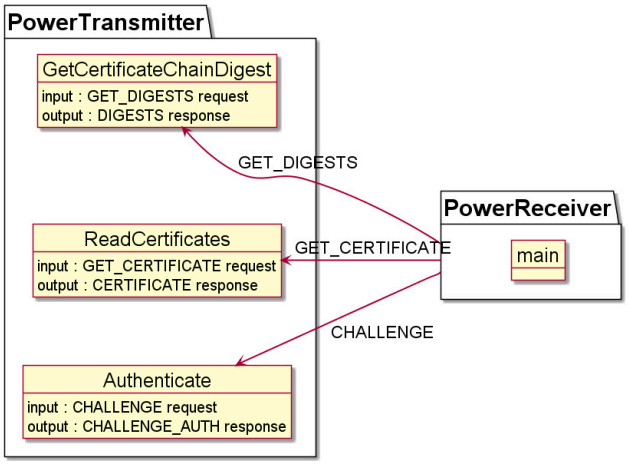
- Connect supported embedded platform with, SA attached to it on, the USB Interface (Not the MCU_LINK Debug).
- supported embedded platform should have the VCOM firmware flashed onto it which shall do the VCOM-I2C translation. (For more details, refer `vcom`).
- Open device manager, Under Ports (COM & LPT), look for the connected device.

- specify this COM port for connecting from PC. For e.g. `COM17` as per the above image.

  Note: It appears as "Virtual Com Port" or "USB Serial Device" based on the driver installed in the system.

4.8.16 Secure Authenticator (Qi) Authentication demo

This project is used to demonstrate the Qi authentication flow between a Power Transmitter and a Power Receiver. The Power Transmitter implements 3 functions for the 3 authentication requests a Receiver can issue to the Transmitter : `GetCertificateChainDigest`, `ReadCertificates`, `Authenticate`.

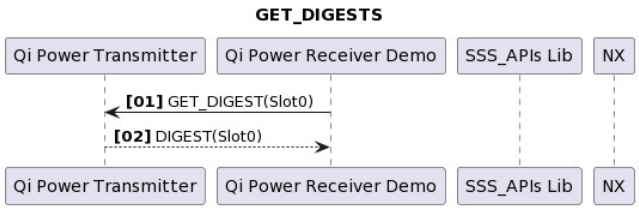


4.8.16.1 Pre-requisites

- Correct keys and certificates for Qi Authentication are recorded in the header. Keys and certificates can be updated `demos/nx/sa_qi/inc/sa_qi_common.h`
- By default WPC Root certificate is used in the certificate chain. You would need to disable macro `USE_ROOT_WPCCA` in `sa_qi_rx_rootcert.c` to use test RootCA:
  Refer - `sa_qi_receiver/sa_qi_rx_rootcert.c`

4.8.16.2 GetCertificateChainDigest (GET_DIGESTS)

This function reads the digest of certificate chains on the provided slot ID and returns all the digests as requested by the Power Receiver.



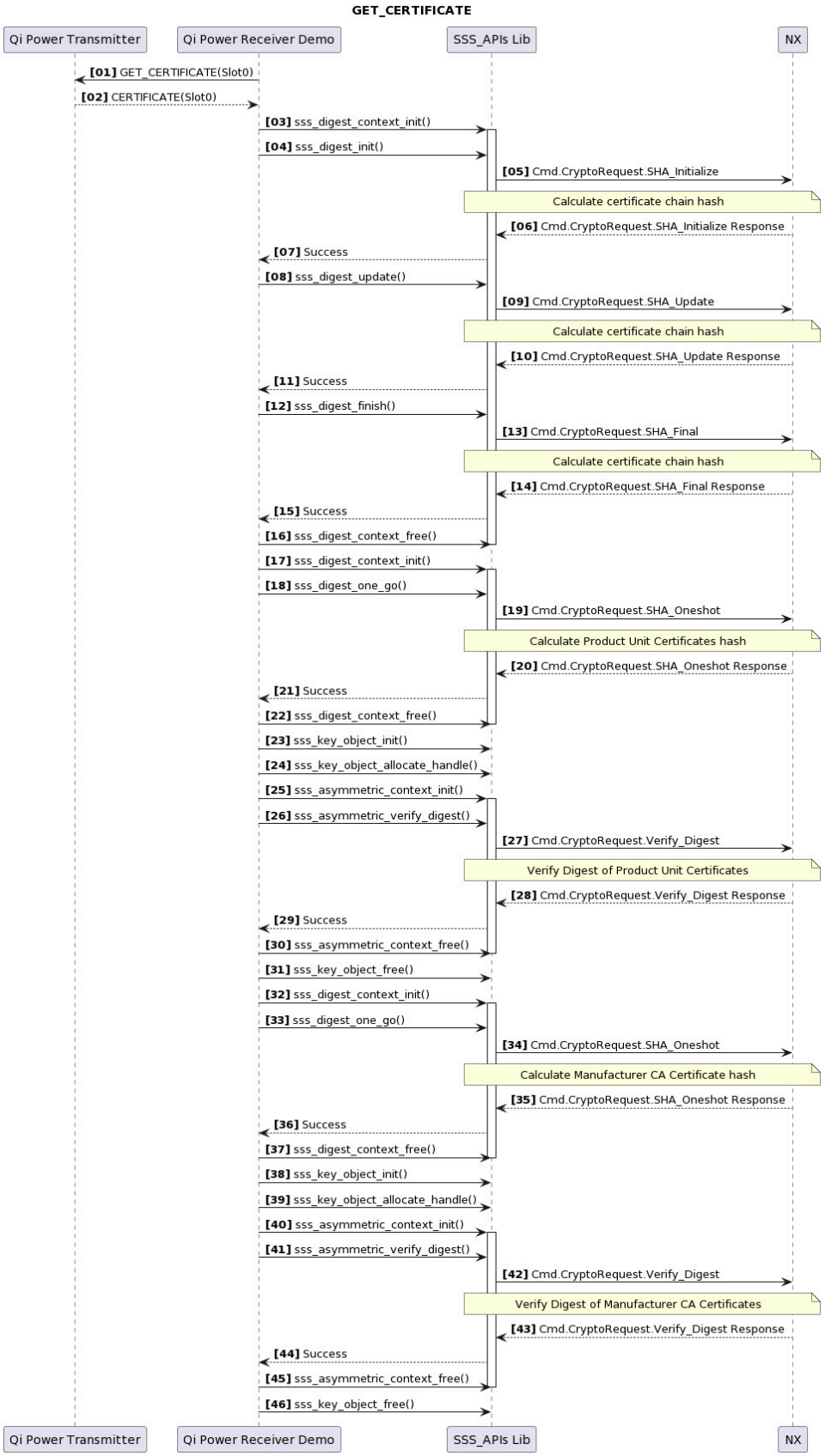
Refer - `qi_transmitter/qi_transmitter.c`

4.8.16.3 ReadCertificates (GET_CERTIFICATE)

This function reads the certificate chain on the provided slot ID starting from the provided offset and reading provided length bytes.

If the provided offset exceeds `0x600` then that indicates the power transmitter to offset from the Product Unit Certificate. Otherwise the offset starts from the beginning of the certificate chain.

Power Rx will receive certificate chain and verify Digest, PUC and manufacture certificate with Nx

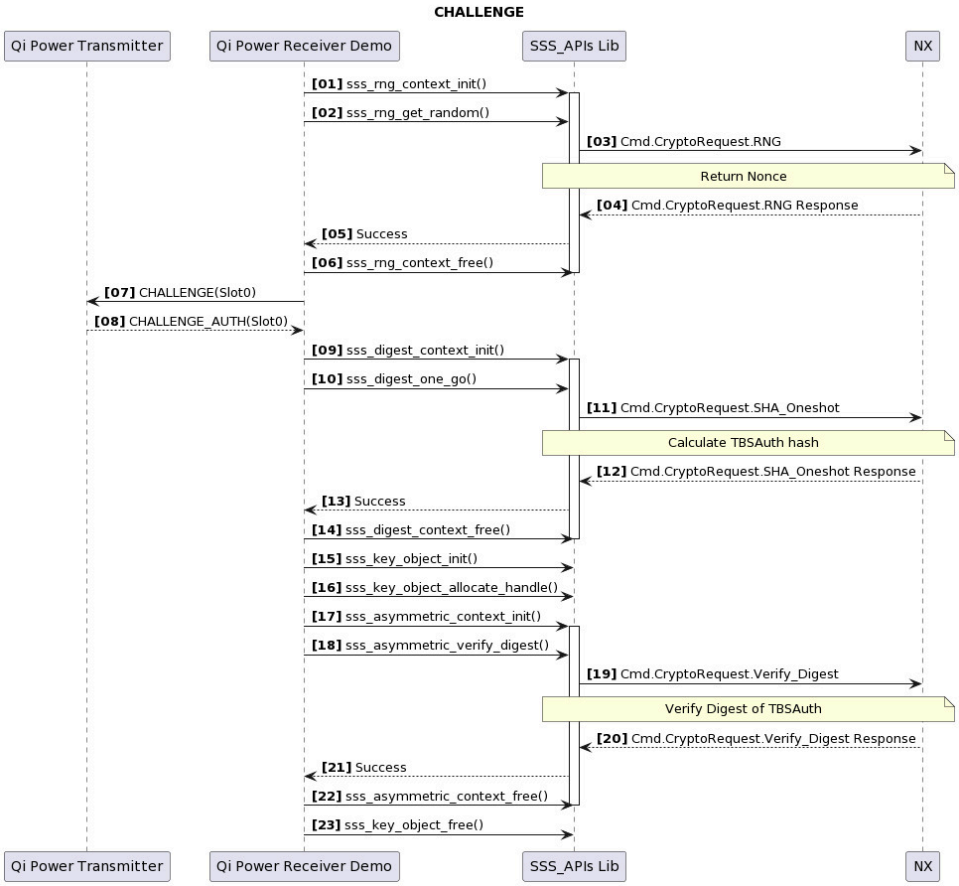


Refer - `qi_transmitter/qi_transmitter.c`

4.8.16.4 Authenticate (CHALLENGE)

This function performs the CHALLENGE operation and returns the signature `R` and signature `S` values to the power receiver.

Power Rx will receive 16B nonce from Nx to initiate challenge request The signature will be received at Rx side and Verify with Nx



Refer - `qi_transmitter/qi_transmitter.c`

4.8.16.5 Building the Demo

- Nx middleware stack. Refer **Build**.

Build project:

Select host crypto as Mbedtls.

- Project: `sa_qi_receiver`

Note: Qi Demo on MCXA153 currently supports only Symmetric Autentication.

4.8.16.6 Running the Example

If you have built a binary, flash the binary on to the board and reset the board.

If you have built an *exe* to be run from Windows using VCOM, run as:

```
sa_qi_receiver.exe <PORT NAME>
```

Where **<PORT NAME>** is the VCOM COM port.

4.8.16.7 Console output

On successful execution you should be able to see logs as:

```
App   :INFO :NX_PKG_v02.00.00_20231107
App   :INFO :Running sa_qi_receiver.exe
App   :INFO :Using PortName='COM10' (CLI)
App   :INFO :Using certificate/key from:'C:\nxp\configuration
\cert_depth3_PKCS7_rev1\' (Default path). You can specify certificates/keys file
 using ENV=NX_AUTH_CERT_DIR
App   :INFO :Read file from C:\nxp\configuration\cert_depth3_PKCS7_rev1\
\cert_and_key\brainpool\host_leaf_keypair.der
Opening COM Port '\\.\COM10'
sss   :INFO :atr (Len=22)
      01 04 63 07    00 93 02 08    00 02 03 E8    00 01 00 64
      04 03 E8 00    FE 00
seTunnel:INFO :Read file from C:\nxp\configuration\cert_depth3_PKCS7_rev1\
\cert_and_key\brainpool\device_root_certificate.der
seTunnel:INFO :Tx leaf cert request
seTunnel:INFO :Rx device certificate
seTunnel:INFO :Received hash match received leaf certificate.
seTunnel:INFO :Verify received ECC signature passed.
seTunnel:INFO :Verfiy X.509 certificate with root/cached CA certificate failed
seTunnel:INFO :Tx P1 cert request
seTunnel:INFO :Rx device certificate
seTunnel:INFO :Verfiy X.509 certificate with root/cached CA certificate failed
seTunnel:INFO :Tx P2 cert request
seTunnel:INFO :Rx device certificate
seTunnel:INFO :Verfiy X.509 certificate with certificate(C=NL, ST=Eindhoven,
 L=Eindhoven, O=NXP, CN=NXP Auth RootCAvE201) Passed
App   :INFO :Read file from C:\nxp\configuration
\cert_depth3_PKCS7_rev1\cert_cache\device_ca_cert_0.hex
seTunnel:INFO :Read file from C:\nxp\configuration\cert_depth3_PKCS7_rev1\
\cert_and_key\brainpool\host_leaf_certificate.der
seTunnel:INFO :Tx Leaf cert reply
seTunnel:INFO :Read file from C:\nxp\configuration\cert_depth3_PKCS7_rev1\
\cert_and_key\brainpool\host_leaf_certificate.der
seTunnel:INFO :Tx P1 cert reply
seTunnel:INFO :Read file from C:\nxp\configuration\cert_depth3_PKCS7_rev1\
\cert_and_key\brainpool\host_p1_certificate.der
seTunnel:INFO :Tx P2 cert reply
seTunnel:INFO :Read file from C:\nxp\configuration\cert_depth3_PKCS7_rev1\
\cert_and_key\brainpool\host_p2_certificate.der
sss   :INFO :Session Open Succeed
App   :INFO :Send command GET_DIGESTS
App   :INFO :Retrieved digest (Len=32)
      42 98 7B 59    E9 4D 30 BF    CA 0E 8C 89    CE 1A 9B 71
      19 0A D2 EA    48 8F AD 54    6A 7E 90 86    F3 94 18 CC
App   :INFO :Send command GET_CERTIFICATE
```

```
App   :INFO :Certificate chain digest successfully verified
App   :INFO :Retrieved PUC (Len=290)
      30 82 01 1E    30 81 C6 A0    03 02 01 02    02 09 04 32
      02 59 8B 89    90 00 00 30    0A 06 08 2A    86 48 CE 3D
      04 03 02 30    12 31 10 30    0E 06 03 55    04 03 0C 07
      30 30 32 41    2D 54 30 30    22 18 0F 32    30 32 32 30
      34 32 38 30    30 30 30 30    30 5A 18 0F    32 30 34 32
      30 34 32 33    30 30 30 30    30 30 5A 30    18 31 16 30
      14 06 03 55    04 03 0C 0D    30 30 36 33    38 36 2D 4D
      6F 64 65 6C    35 30 39 30    13 06 07 2A    86 48 CE 3D
      02 01 06 08    2A 86 48 CE    3D 03 01 07    03 22 00 02
      8B 9C DC 5E    24 F1 E1 39    B2 04 09 4A    47 B6 B6 FA
      30 0D 18 16    BC B6 53 6D    D8 7F 8F FC    C5 43 46 C0
      A3 1B 30 19    30 17 06 05    67 81 14 01    02 01 01 FF
      04 0B 04 09    02 21 18 95    62 64 00 00    02 30 0A 06
      08 2A 86 48    CE 3D 04 03    02 03 47 00    30 44 02 20
      11 DB FA D4    80 B5 BA 2A    82 09 9C FB    C9 45 CE FD
      C8 D2 5E CB    C5 F1 40 D4    7D EE F5 70    08 5B A2 3E
      02 20 21 07    A9 80 92 2E    19 ED 74 1E    EF AE 66 10
      DE 32 8C 38    68 B3 65 42    EB 2B 44 D2    4C 87 E3 CC
      0C 1F
App   :INFO :PUC successfully verified
App   :INFO :Manufacturer certificate successfully verified
App   :INFO :Certificate chain successfully verified
App   :INFO :Retrieved PUC public key (Len=65)
      04 8B 9C DC    5E 24 F1 E1    39 B2 04 09    4A 47 B6 B6
      FA 30 0D 18    16 BC B6 53    6D D8 7F 8F    FC C5 43 46
      C0 89 85 ED    B7 DD 74 17    A3 C1 2F CA    F5 3A B7 41
      54 D2 47 81    08 30 7C 50    AB 33 77 C2    89 94 48 70
      C0
App   :INFO :Send command CHALLENGE
App   :INFO :Challenge Signature (Len=64)
      87 86 44 39    15 F5 53 44    7B 42 77 50    06 D4 A1 3C
      E4 CE 9E 0C    D9 75 85 17    89 64 15 14    AC 96 40 17
      02 3E 10 36    48 38 C3 2A    9C F0 1C 90    5B 69 0B E8
      F1 83 A2 21    63 34 7D 79    D3 3E B1 1F    D1 01 FA D0
App   :INFO :TBSAuth (Len=54)
      41 42 98 7B    59 E9 4D 30    BF CA 0E 8C    89 CE 1A 9B
      71 19 0A D2    EA 48 8F AD    54 6A 7E 90    86 F3 94 18
      CC 1B 00 EF    6E 3F D2 33    3A B6 01 54    38 EE 1E 56
      E5 9A C6 13    11 CC
App   :INFO :Challenge successfully verified
App   :INFO :sa_qi_receiver Example Success !!!...
App   :INFO :ex_sss Finished
```

4.8.17 ECC Standalone Example

The example demonstrates the use of SSS APIs to open the session to secure authenticator. (Unlike other examples where the ex_common code is used for session open). The example will do board init of MCU and establish symmetric auth session with secure authenticator and perform ECDSA sign / verify crypto operations.

Refer ex_ecc_standalone

4.8.17.1 Prerequisites

• Build NX middleware stack. **Refer** MW stack

4.8.17.2 Building the example

Refer MCU Project Build

• Project: ecc_standalone

4.8.17.3 Console output

If everything is successful, the output will be similar to:

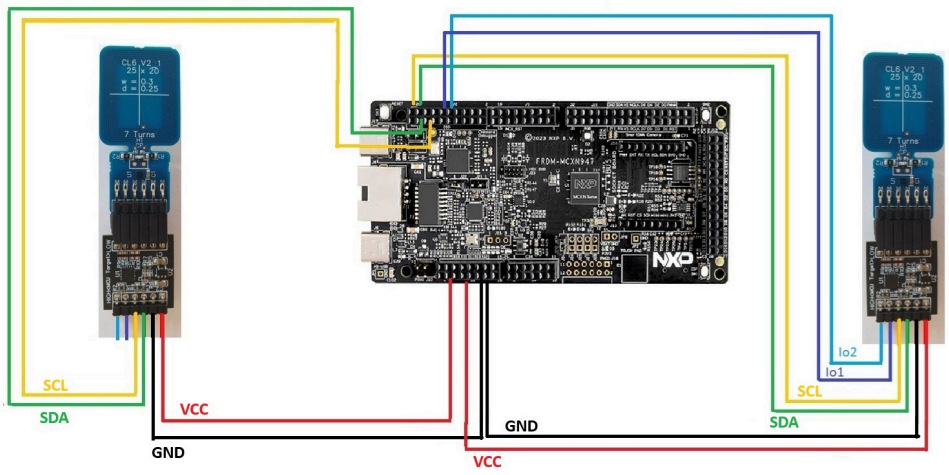
```
nx_mw:Session Open Succeed
nx_mw:Running Elliptic Curve Cryptography Example ex_sss_ecc.c
nx_mw:Do Signing
nx_mw:INFO (Len=361620)
     1 2 3 4 5 6 7 8 9 0 0 0 0 0 0 0
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
nx_mw:INFO (Len=361628)
     30 44 2 20 1C 89 D7 AD 5D E7 AF 2A D8 22 5 B2
     77 3 21 A 37 3C 1F 42 B4 75 14 57 7D 28 36 91
     B6 A0 1B 4A 2 20 1C 5B 13 7A AF C8 4A 31 3C B8
     66 E C6 7E AD FF 4A 28 72 92 34 3C 7E 6A F4 AE
     29 20 21 64 B2 D
nx_mw:Signing Successful !!!
nx_mw:Do Verification
nx_mw:INFO (Len=361620)
     1 2 3 4 5 6 7 8 9 0 0 0 0 0 0 0
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
nx_mw:INFO (Len=361628)
     30 44 2 20 1C 89 D7 AD 5D E7 AF 2A D8 22 5 B2
     77 3 21 A 37 3C 1F 42 B4 75 14 57 7D 28 36 91
     B6 A0 1B 4A 2 20 1C 5B 13 7A AF C8 4A 31 3C B8
     66 E C6 7E AD FF 4A 28 72 92 34 3C 7E 6A F4 AE
     29 20 21 64 B2 D
nx_mw:Verification Successful !!!
nx_mw:ECC-Standalone Example Success !!!...
```

4.8.18 NX Host co-processor example

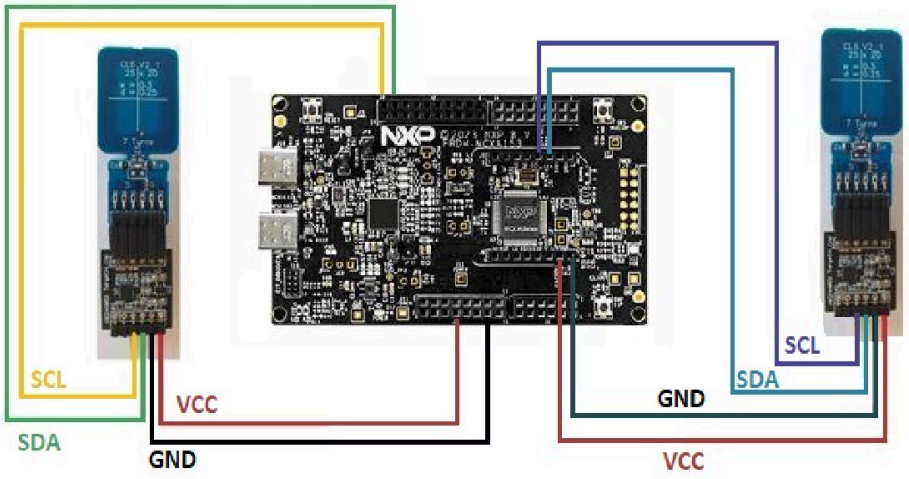
The project demonstrates use case where host doesn't have the capability to do crypto functions. 2 SAs are used in this case. One SA is used with host as a coprocessor which works as Verifier/Prover. Another SA is used independently as Prover/Verifier. Host will trigger authentication by message MSGR_START_PROTOCOL. Sigma-I protocol exchanging happens between verifier SA and prover SA. Host doesn't process the Sigma-I protocol messages and only works a bridge between 2 SAs.

Refer NX Host Co-processor Example

4.8.18.1 Board Setup



FRDM-MCXN947



FRDM-MCXA153

4.8.18.1.1 Prerequisites

- NX middleware stack. **Refer** **MW stack**
  - 2 SAs provision root CA public key and certificates
  - On SA Host co processor set configuration 0x10, I2C support, I2C address and ProtocolOptions of Controller Session Key.
  - run nxclitool for provision **Refer**nxclitool_scripts

4.8.18.1.2 About the Example

This is project is demostrates host coprocessor mutual authentication between two 2 secure authenticators. The secure messaging and response processing which is required for secure tunneling is offloaded to the coprocessor SA. In order to achieve this,

- Host sends the command to coprocessor SA by Cmd.ProcessSM_Apply.
- Coprocessor SA returns encrypted data and/or MAC to host.
- Host constructs C-APDU command with encrypted data and/or MAC and sends to SA.
- After receiving R-APDU from SA, host sends encrypted response and/or MAC to coprocessor SA by Cmd.ProcessSM_Remove.
- Coprocessor SA returns decrypted plaintext and/or MAC verification result to the host.

It uses the following APIs and data types:

- `nx_ProcessSM_Apply()`
- `nx_Freemem()`
- `nx_ProcessSM_Remove()`

4.8.18.1.3 Building the Demo

- Build NX middleware stack. **Refer**Cmake Build
- Project: `nx_host_coprocessor`

1. `NXMW_Auth=None`
2. `NXMW_Secure_Tunneling=None`
3. `NXMW_Host=frdmmcxn947`
4. `NXMW_SMCOM=T1oI2C_GP1_0`

4.8.18.1.4 Running the Example

After you built a binary, flash the binary on to the board and reset the board. **Refer** Running the Example

4.8.18.1.5 Console output

If everything is successful, the output will be similar to:

```
nx_mw :INFO :Select Device 1 host co processor
nx_mw :INFO :Select Device 2 NX
nx_mw :INFO :Wait for Mutual authentication
nx_mw :INFO :Mutual authentication is success
nx_mw :INFO :FreeMem Device 2
nx_mw :INFO :ProcessSM_Apply Device 1
nx_mw :INFO :ProcessSMRemove Device 1
nx_mw :INFO :Available free memory: 12480 bytes
nx_mw :INFO :nx_host_coprocessor Example Success !!!...
```

4.8.19 Multiple sessions (Plain and Sigma-I) Example

This project demonstrates opening and closing a plain session followed by a Sigma-I session out of the box.

Refer multiple_sessions

4.8.19.1 Prerequisites

- Nx middleware stack. **Refer**MW stack
- nx_Personalization example is built and run using the same curve type (NXMW_Auth_Asymm_Host_Curve) as is used in the example (cert_curve_type, ephem_curve_type).

4.8.19.2 About the Example

In this example, it is shown how can one open a plain session, perform an API call, close it and follow it the same for a Sigma-I session, **without** using the common header ex_sss_main_inc.h generally used in other examples.

It uses the following APIs and data types: -

- `sss_session_open` -
- `kSSS_ConnectionType_Plain`
- `sss_connection_type_t`
- `kSSS_ConnectionType_Encrypted`
- `sss_connection_type_t`
- `sss_session_close`
- `sss_host_session_close`

4.8.19.3 Building the example

- Build NX middleware stack. **Refer** Cmake_build.
  - Project - `ex_sss_multiple_sessions`

4.8.19.4 Console output

If everything is successful, the output will be similar to:

```
nx_mw :INFO :Open plain session

nx_mw :INFO :cip (Len=22)
                01 04 63 07     00 93 02 08     00 02 03 E8     00 01 00 64
                04 03 E8 00     FE 00
nx_mw :WARN :Communication channel is Plain.
nx_mw :WARN :!!!Security and privacy must be assessed.!!!
nx_mw :INFO :Session Open Succeed
nx_mw :INFO :session_ctx->authType 0
nx_mw :INFO :Available free memory = 8352
nx_mw :INFO :Close plain session

nx_mw :INFO :Init Host for Sigma-I session (AES128_NTAG)

nx_mw :WARN :mbedtls_entropy_func_3_X is a dummy implementation with hardcoded
 entropy. Mandatory to port it to the Micro Controller being used.
nx_mw :INFO :Using certificate/key from lib/sss/inc/fsl_sss_nx_auth_keys.h
 (cert_depth3_x509_rev1)
nx_mw :INFO :cip (Len=22)
                01 04 63 07     00 93 02 08     00 02 03 E8     00 01 00 64
                04 03 E8 00     FE 00
nx_mw :INFO :Using root certificate from lib/sss/inc/fsl_sss_nx_auth_keys.h
nx_mw :INFO :Verify X.509 certificate with root/cached CA certificate failed
nx_mw :INFO :Verify X.509 certificate with root/cached CA certificate failed
nx_mw :INFO :Verify X.509 certificate with certificate(C=NL, ST=Eindhoven,
 L=Eindhoven, O=NXP, CN=NXP Auth RootCAvE201) Passed
nx_mw :INFO :Session Open Succeed
nx_mw :INFO :Random bytes generated (Len=32)
                A8 46 04 39     12 03 0C A9     CB BB D1 BB     13 E1 CE CC
                B5 41 68 7B     22 2D 46 4B     36 18 90 CD     64 BD DA 65
nx_mw :INFO :Close sigma-i session
```

```
nx_mw :INFO :gex_sss_multiple_sessions Example Success !!!...
```

4.8.20 Mbed-TLS 3x Example

This project demonstrates ECDSA sing/verify, ECDH, RNG, ECB encryption/decryption operation on a message using SSS API and offloading the operations to NX SA using mbedtls alt file. **Refer** - **Mbed-TLS 3x Intro**

Refer - **Mbed-TLS 3x Example**

4.8.20.1 Prerequisites

The custom configuration file to enable required ALT operation is present in **Refer config file for Windows** - **mbedtls_sss_alt_config** **Refer config file for KSDK** **sss_ksdk_mbedtls_3x_alt_config** By default ECDSA Sign are enabled.

Use the below macros in the config file **Refer config file for Windows** - **mbedtls_sss_alt_config** **Refer config file for KSDK** **sss_ksdk_mbedtls_3x_alt_config** to enable / disable required crypto operation

- MBEDTLS_ECDSA_SIGN_ALT
- MBEDTLS_ECDSA_VERIFY_ALT
- MBEDTLS_AES_ENCRYPT_ALT
- MBEDTLS_AES_DECRYPT_ALT
- MBEDTLS_ECDH_COMPUTE_SHARED_ALT

4.8.20.2 About the Example

- ex_mbedtls3x_ecdsa_sign function does the following -
  1. Injecting the actual keypair on host.
  2. Sign using SSS APIs. Mbed-TLS alt will do a software rollback and uses host for ECDSA sign.
  3. Injecting actual key in Secure Authenticator.
  4. Injecting reference key on Host.
  5. Sign using SSS APIs, Mbed-TLS alt uses NX SA for sign.

     Note: In the default implementation, every time the control goes to ALT implementation, session open and close is performed. This will have all transient objects will be lost. To avoid the session open / close in ALT implementation, Use the sss_mbedtls_set_keystore_ecdsa_sign() / sss_mbedtls_set_keystore_ecdsa_verify()/ APIs to pass the key store.
- ex_mbedtls3x_ecdsa_verify function does the following -
  1. Injecting actual key in Secure Authenticator.
  2. Sign using SSS APIs. Mbed-TLS ALT will do software rollback and will use host for ECDSA Sign.
  3. Injecting public key on Host.
  4. Verify using SSS APIs. Mbed-TLS ALT will use NX SA for verify.

     Note: ECDSA Verify works only when the session authentication is Symmetric. Not with Sigma-I.
- ex_mbedtls3x_rng_gen function does the following -
  1. Generate random numbers using NX SA.

     Note: Random number generation is offloaded to NX Secure authenticator only if the key store is passed using sss_mbedtls_set_keystore_rng() API.
- ex_mbedtls3x_ecdh function does the following -
  1. Create a transient key in secure authenticator.
  2. Create and injecting the ref key on host.

3. Injecting the public key on host.
4. Derive ECDH using SSS APIs. Mbed-TLS alt use NX SA for ECDH.

   Note: ECDH key derivation is offloaded to NX Secure authenticator only if the key store is passed using sss_mbedtls_set_keystore_ecdh() API.

- ex_mbedtls3x_ebc , ex_mbedtls3x_cbc, ex_mbedtls3x_cmac functions does the following -
- ex_mbedtls3x_ebc
  1. Inject aes key on host.
  2. AES ECB encryption using SSS APIs. Mbed-TLS alt will do a software rollback since key store is not set.
  3. AES ECB decryption using SSS APIs. Mbed-TLS alt will do a software rollback since key store is not set.
  4. Set key store using sss_mbedtls_set_keystore_aes API.
  5. AES ECB encryption using SSS APIs, Mbed-TLS alt use SE for ECB encryption.
  6. AES ECB decryption using SSS APIs, Mbed-TLS alt use SE for ECB decryption.
- ex_mbedtls3x_cbc
  1. Inject aes key on host.
  2. AES CBC encryption using SSS APIs. Mbed-TLS alt will do a software rollback since key store is not set.
  3. AES CBC decryption using SSS APIs. Mbed-TLS alt will do a software rollback since key store is not set.
  4. Set key store using sss_mbedtls_set_keystore_aes API.
  5. AES CBC encryption using SSS APIs, Mbed-TLS alt use SE for CBC encryption.
  6. AES CBC decryption using SSS APIs, Mbed-TLS alt use SE for CBC decryption.
- ex_mbedtls3x_cmac
  1. Inject aes key on host.
  2. AES CMAC sign using SSS APIs. Mbed-TLS alt will do a software rollback since key store is not set.
  3. AES CMAC verify using SSS APIs. Mbed-TLS alt will do a software rollback since key store is not set.
  4. Set key store using sss_mbedtls_set_keystore_aes API.
  5. AES CMAC sign using SSS APIs, Mbed-TLS alt use SE for CMAC sign.
  6. AES CMAC verify using SSS APIs, Mbed-TLS alt use SE for CMAC verify.

Note: AES ECB/CBC encryption/decryption is offloaded to NX Secure authenticator only if the key store is passed using sss_mbedtls_set_keystore_aes() API. Only plain authentication mode (`NXMW_Auth=None`) is supported. set access condition plain mode for Cmd.CryptoRequest and which can be configured through Cmd.SetConfiguration Option 0x15 (Refer **nx_tool_setconfig**). Example:

```
nx_tool_setconfig.exe -cryptoCM plain -cryptoAC 0xE COM10
```

Note: To run ecdsa, ecdh and rng alt example on plain commMode or none authentication. set access condition plain mode for Cmd.CryptoRequest, Cmd.Managekeypair and which can be configured through Cmd.SetConfiguration Option 0x15 and 0x12 (Refer **nx_tool_setconfig**). Example:

```
nx_tool_setconfig.exe -cryptoCM plain -cryptoAC 0xE COM10
nx_tool_setconfig.exe -keypairCM plain -keypairAC 0xE COM10
```

4.8.20.3 Building the Example

- Project: `ex_mbedtls_3_x_alt`

1. `NXMW_HostCytpo=MBEDTLS`
2. `NXMW_MBedTLS=3_X`
3. `NXMW_mbedTLS_ALT=SSS`

Other options should be selected according to IC configuration.

4.8.21 sdm

4.8.21.1 Secure Dynamic Messaging (SDM) File Reading Demo and verify ECC signature

The Secure Dynamic Messaging (SDM) allows for confidential and integrity protected data exchange, without requiring a preceding authentication.

This project is used to demonstrate the SDM for reading file data, decryption and signature verification. In detail:

- Read out data from file 2.
- Decrypt PICCData and get VCUID and SDMCtr
- Host should maintain a SDM counter (SDMCtr) which will be used in decryption. This demo is assumed to run after `ex_sdm_provision`. So the SDMCtr should start with 0. If that is not the case, readout counter will be different from host SDMCtr and the readout value will be used. *This overwrite operation is only for demo purpose and should not be done in real case.*
- Generate session keys from KeyID.SDMFileReadKey (`EX_SSS_SDM_FILE_READ_AES_KEY`), VCUID and SDMCtr.
- Decrypt file data and output GPIO status
- Verify signature
- Read out data from file 2 again. SDMCtr will not increase because it targets the same file.
- Decrypt file data
- Verify signature
- Get free memory
- Read out data from file 2 for 3rd time. SDMCtr will increase by 1 because there is an different command (Cmd.FreeMem) before Cmd.ReadData
- Decrypt file data
- Verify signature

4.8.21.1.1 Pre-requisites

- set EX_SSS_ENABLE_SDM_ECC_SIGNATURE 1 in file ex_sdm_provision.c and build `ex_sdm_provision`
- `ex_sdm_provision` should run first. It will initialize the ECC key and change file setting for SDM.

4.8.21.1.2 Building the Demo

- Build NX middleware stack. Refer **Linux build**.
  - Project - `ex_sdm_file_read`
  - Select NXMW_Auth to None

4.8.21.1.3 Running the Example

```
./ex_sdm_file_read
```

4.8.21.1.4 Console output

If everything is successful, the output will be similar to:

```
nx_mw :WARN :!!!Security and privacy must be assessed.!!!
nx_mw :INFO :Session Open Succeed
```

```
nx_mw :INFO :Note: The demo is supposed to be run after Cmd.ChangeFileSettings.
 So SDMReadCtr is reset to 0x000000!
nx_mw :INFO :Read NDEF File (Len=256)
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      44 36 45 42    30 46 42 41    38 35 39 39    34 34 45 33
      35 31 31 34    41 38 43 31    33 34 45 37    37 43 37 41
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      37 36 32 38    31 34 46 34    37 44 39 38    32 43 37 34
      39 33 35 46    30 41 34 46    36 41 31 36    36 33 33 32
      39 31 42 43    36 43 32 44    35 46 44 31    38 36 37 38
      41 35 33 34    37 46 36 31    37 39 46 36    36 41 35 39
      46 45 36 45    39 32 35 32    36 43 46 41    45 46 31 32
      39 43 35 45    46 37 41 38    31 43 41 39    39 41 33 39
      33 41 46 30    43 30 44 46    33 38 33 30    39 36 41 44
      43 42 44 45    32 38 35 32    31 37 35 41    43 46 45 46
      42 38 32 31    45 31 41 45    34 33 33 31    39 30 37 37
      31 33 33 33    44 42 44 37    43 34 45 35    34 35 42 45
nx_mw :INFO :Decrypt Encrypted PICCData @0x20 (Length 0x20)
nx_mw :INFO :Decrypted PICC data in HEX (Len=16)
      C7 00 01 02    03 04 05 06    0E 00 00 73    0A 32 08 68
nx_mw :WARN :Readout SDMReadCtr(0xe) is different from host SDMReadCtr(0x1)!
nx_mw :WARN :Overwrite host SDMReadCtr. This is only for demo purpose and should
 not be done in real case!
nx_mw :INFO :Get VCUID from PICCData. (Len=7)
      00 01 02 03    04 05 06
nx_mw :INFO :Decrypt SDMENCFileData @0x60 (Length 0x20)
nx_mw :INFO :Decrypted file data (Len=16)
      00 00 00 00    00 00 00 00    49 49 49 00    00 00 00 00
nx_mw :INFO :GPIO Status @0x68: 0x49-0x49-0x49
nx_mw :INFO :verify Signature @0x80(Length 0x80) with data @0x10(Length 0x70))
nx_mw :INFO :Verify with ECC public key
nx_mw :INFO :Signature in ASN.1: (Len=71)
      30 45 02 21    00 91 BC 6C    2D 5F D1 86    78 A5 34 7F
      61 79 F6 6A    59 FE 6E 92    52 6C FA EF    12 9C 5E F7
      A8 1C A9 9A    39 02 20 3A    F0 C0 DF 38    30 96 AD CB
      DE 28 52 17    5A CF EF B8    21 E1 AE 43    31 90 77 13
      33 DB D7 C4    E5 45 BE
nx_mw :INFO :Verify signature passed.
nx_mw :INFO :Read NDEF File Again (Len=256)
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      44 36 45 42    30 46 42 41    38 35 39 39    34 34 45 33
      35 31 31 34    41 38 43 31    33 34 45 37    37 43 37 41
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      37 36 32 38    31 34 46 34    37 44 39 38    32 43 37 34
      39 33 35 46    30 41 34 46    36 41 31 36    36 33 33 32
      39 31 42 43    36 43 32 44    35 46 44 31    38 36 37 38
      41 35 33 34    37 46 36 31    37 39 46 36    36 41 35 39
      46 45 36 45    39 32 35 32    36 43 46 41    45 46 31 32
      39 43 35 45    46 37 41 38    31 43 41 39    39 41 33 39
      33 41 46 30    43 30 44 46    33 38 33 30    39 36 41 44
      43 42 44 45    32 38 35 32    31 37 35 41    43 46 45 46
      42 38 32 31    45 31 41 45    34 33 33 31    39 30 37 37
      31 33 33 33    44 42 44 37    43 34 45 35    34 35 42 45
nx_mw :INFO :Current SDMCtr 0xe
nx_mw :INFO :Decrypt SDMENCFileData @0x60 (Length 0x20)
nx_mw :INFO :Decrypted file data (Len=16)
```

```
      00 00 00 00    00 00 00 00    49 49 49 00    00 00 00 00
nx_mw :INFO :verify Signature @0x80(Length 0x80) with data @0x10(Length 0x70))
nx_mw :INFO :Verify with ECC public key
nx_mw :INFO :Signature in ASN.1: (Len=71)
      30 45 02 21    00 91 BC 6C    2D 5F D1 86    78 A5 34 7F
      61 79 F6 6A    59 FE 6E 92    52 6C FA EF    12 9C 5E F7
      A8 1C A9 9A    39 02 20 3A    F0 C0 DF 38    30 96 AD CB
      DE 28 52 17    5A CF EF B8    21 E1 AE 43    31 90 77 13
      33 DB D7 C4    E5 45 BE
nx_mw :INFO :Verify signature passed.
nx_mw :INFO :Get Free Memory
nx_mw :INFO :session_ctx->authType 0
nx_mw :INFO :Read NDEF File for 3rd time (Len=256)
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      45 34 43 38    42 46 31 35    44 46 37 31    41 32 30 35
      38 35 33 46    39 39 39 32    46 30 39 35    46 32 37 44
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      44 32 32 41    35 37 31 44    31 31 35 33    35 42 39 34
      37 39 45 36    38 36 43 38    34 43 41 33    32 45 36 36
      36 39 32 39    36 30 39 31    39 35 36 44    45 30 43 34
      31 46 36 31    38 44 38 46    32 36 37 43    44 46 43 38
      46 39 42 36    46 35 31 34    46 45 35 34    34 43 35 30
      42 38 35 32    42 32 34 37    37 43 34 38    34 44 38 36
      39 30 39 39    42 38 44 31    30 41 32 30    34 38 34 37
      37 32 41 38    38 42 42 32    46 42 36 35    45 37 42 30
      31 38 32 32    43 45 37 34    33 42 37 34    45 42 39 30
      41 38 33 39    38 39 45 37    36 39 44 41    41 33 44 42
nx_mw :INFO :Current SDMCtr 0xf
nx_mw :INFO :Decrypt SDMENCFileData @0x60 (Length 0x20)
nx_mw :INFO :Decrypted file data (Len=16)
      00 00 00 00    00 00 00 00    49 49 49 00    00 00 00 00
nx_mw :INFO :verify Signature @0x80(Length 0x80) with data @0x10(Length 0x70))
nx_mw :INFO :Verify with ECC public key
nx_mw :INFO :Signature in ASN.1: (Len=71)
      30 45 02 20    69 29 60 91    95 6D E0 C4    1F 61 8D 8F
      26 7C DF C8    F9 B6 F5 14    FE 54 4C 50    B8 52 B2 47
      7C 48 4D 86    02 21 00 90    99 B8 D1 0A    20 48 47 72
      A8 8B B2 FB    65 E7 B0 18    22 CE 74 3B    74 EB 90 A8
      39 89 E7 69    DA A3 DB
nx_mw :INFO :Verify signature passed.
nx_mw :INFO :SDM File Verify Example Success !!!...
nx_mw :INFO :ex_sss Finished
```

4.8.21.2 Secure Dynamic Messaging (SDM) File Reading Demo verify MAC

The Secure Dynamic Messaging (SDM) allows for confidential and integrity protected data exchange, without requiring a preceding authentication.

This project is used to demonstrate the SDM for reading file data, decryption and mac verification. In detail:

- Read out data from file 2.
- Decrypt PICCData and get VCUID and SDMCtr
- Host should maintain a SDM counter (SDMCtr) which will be used in decryption. This demo is assumed to run after `ex_sdm_provision`. So the SDMCtr should start with 0. If that is not the case, readout counter will be different from host SDMCtr and the readout value will be used. *This overwrite operation is only for demo purpose and should not be done in real case.*

- Generate session keys from KeyID.SDMFileReadKey (`EX_SSS_SDM_FILE_READ_AES_KEY`), VCUID and SDMCtr.
- Decrypt file data and output GPIO status
- Verify mac
- Read out data from file 2 again. SDMCtr will not increase because it targets the same file.
- Decrypt file data
- Verify mac
- Get free memory
- Read out data from file 2 for 3rd time. SDMCtr will increase by 1 because there is an different command (Cmd.FreeMem) before Cmd.ReadData
- Decrypt file data
- Verify mac

4.8.21.2.1 Pre-requisites

- set EX_SSS_ENABLE_SDM_ECC_SIGNATURE 0 in file ex_sdm_provision.c and build `ex_sdm_provision`
- `ex_sdm_provision` should run first. It will set change file setting for SDM.

4.8.21.2.2 Building the Demo

- Build NX middleware stack. Refer **Linux build**.
  - Project - `ex_sdm_mac`
  - Select NXMW_Auth to None

4.8.21.2.3 Running the Example

```
./ex_sdm_mac
```

4.8.21.2.4 Console output

If everything is successful, the output will be similar to:

```
nx_mw :WARN :Communication channel is Plain.
nx_mw :WARN :!!!Security and privacy must be assessed.!!!
nx_mw :INFO :Session Open Succeed
nx_mw :INFO :Note: The demo is supposed to be run after Cmd.ChangeFileSettings.
 So SDMReadCtr is reset to 0x000000!
nx_mw :INFO :Read NDEF File (Len=256)
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      32 45 31 43    45 36 46 44    35 46 37 39    42 32 45 31
      37 35 41 33    44 30 41 43    43 37 33 30    46 36 30 44
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      36 42 41 46    43 42 41 36    35 36 45 45    30 37 41 37
      42 42 35 34    39 39 42 33    31 31 37 46    38 34 30 46
      34 31 39 46    39 43 31 38    34 31 34 46    33 45 32 37
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
```

```
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
nx_mw :INFO :Decrypt Encrypted PICCData @0x20 (Length 0x20)
nx_mw :INFO :Decrypted PICC data in HEX (Len=16)
      C7 00 01 02    03 04 05 06    01 00 00 8F    32 7D 4F 5B
nx_mw :INFO :Get SDMCtr from PICCData 0x1. It's same to the SDMCtr stored on
 host
nx_mw :INFO :Get VCUID from PICCData. (Len=7)
      00 01 02 03    04 05 06
nx_mw :INFO :Decrypt SDMENCFileData @0x60 (Length 0x20)
nx_mw :INFO :Decrypted file data (Len=16)
      00 00 00 00    00 00 00 00    49 49 49 00    00 00 00 00
nx_mw :INFO :GPIO Status @0x68: 0x49-0x49-0x49
nx_mw :INFO :Verify SDM Mac @0x80(Length 0x10) with data @0x10(Length 0x70))
nx_mw :INFO :MAC verification passed
nx_mw :INFO :Read NDEF File Again (Len=256)
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      32 45 31 43    45 36 46 44    35 46 37 39    42 32 45 31
      37 35 41 33    44 30 41 43    43 37 33 30    46 36 30 44
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      36 42 41 46    43 42 41 36    35 36 45 45    30 37 41 37
      42 42 35 34    39 39 42 33    31 31 37 46    38 34 30 46
      34 31 39 46    39 43 31 38    34 31 34 46    33 45 32 37
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
nx_mw :INFO :Current SDMCtr 0x1
nx_mw :INFO :Decrypt SDMENCFileData @0x60 (Length 0x20)
nx_mw :INFO :Decrypted file data (Len=16)
      00 00 00 00    00 00 00 00    49 49 49 00    00 00 00 00
nx_mw :INFO :Verify SDM Mac @0x80(Length 0x10) with data @0x10(Length 0x70))
nx_mw :INFO :MAC verification passed
nx_mw :INFO :Get Free Memory
nx_mw :INFO :session_ctx->authType 0
nx_mw :INFO :Read NDEF File for 3rd time (Len=256)
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      34 36 39 44    31 30 43 37    31 31 44 44    31 33 43 42
      41 33 39 38    39 32 46 38    35 45 31 36    42 39 45 33
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      46 35 36 42    34 33 39 37    37 43 36 42    42 35 31 34
      34 45 39 31    30 37 39 33    39 36 45 45    34 36 31 32
      38 38 45 45    41 36 44 46    35 37 37 45    36 37 38 32
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
      00 00 00 00    00 00 00 00    00 00 00 00    00 00 00 00
nx_mw :INFO :Current SDMCtr 0x2
nx_mw :INFO :Decrypt SDMENCFileData @0x60 (Length 0x20)
nx_mw :INFO :Decrypted file data (Len=16)
      00 00 00 00    00 00 00 00    49 49 49 00    00 00 00 00
```

```
nx_mw :INFO :Verify SDM Mac @0x80(Length 0x10) with data @0x10(Length 0x70))
nx_mw :INFO :MAC verification passed
nx_mw :INFO :SDM File Verify Example Success !!!...
nx_mw :INFO :ex_sss Finished
```

4.8.21.3 Secure Dynamic Messaging (SDM) Provisioning Demo

The Secure Dynamic Messaging (SDM) allows for confidential and integrity protected data exchange, without requiring a preceding authentication.

NTAGECC supports two modes for integrity protection and authentication of the data:

- symmetric SDMMAC
- asymmetric SDMSIG

NTAGECC also support encryption based on symmetric cryptography for PICCData and generic file data.

This project is used to enable and configure SDM. After running this demo, ex_sdm_mac/ex_sdm_file_read can be used to read out data from file and decrypt the data and verify the mac/signature.

Warning This example is only for demonstration purpose. Maintaining and provisioning the keys/files should be done in a secure way.

In detail, this demo will do following:

- The AES Keys 1 are assumed to be default one (AES-128 of all '0') and this demo will change it. The new key values (KeyID.SDMMetaReadKey and KeyID.SDMFileReadKey) are defined in `demos/nx/sdm/ex_sdm_common.h`
- Set ECC key 2 with private key with `EX_SSS_SDM_ECC_PRIVATE_KEY` in `demos/nx/sdm/ex_sdm_provision/ex_sdm_provision.h`
- Change file 2 setting
  1. Enable Secure Dynamic Messaging and Mirroring
  2. Disable Deferred Configuration
  3. Enable VCUID mirroring
  4. Enable SDMReadCtr mirroring
  5. Disable SDMReadCtrLimit
  6. Enable SDM File Data Encryption with AppKey 1 (`EX_SSS_SDM_AES_KEY_ID`)
  7. Enable GPIO Status mirroring.
  8. Encrypt PICCData mirroring with AppKey 1 (`EX_SSS_SDM_AES_KEY_ID`)
  9. Enable SDM File Data Sign with ECC Key 2 (`EX_SSS_SDM_ECC_KEY_ID`) or
  10. Enable SDM File Data with MAC (`Nx_SDMFileRead_AccessCondition_No_SDM`)
  11. Access condition for Cmd.GetFileCounters is 0.
  12. SDM input data for signature start at offset 0x10 (`EX_SSS_SDM_SDMMACInputOffset`)
  13. PICCData is mapped to offset 0x20 (`EX_SSS_SDM_PICCDATA_OFFSET`)
  14. SDM encrypted data start at offset 0x60 (`EX_SSS_SDM_SDMENCOffset`) with length 0x20 (`EX_SSS_SDM_SDMENCLength`)
  15. GPIO Status is mapped to offset 0x68 (`EX_SSS_SDM_GPIOStatusOffset`)
  16. SDM signature start at offset 0x80 (`EX_SSS_SDM_SDMMACOffset`) or
  17. SDM mac start at offset 0x80 (`EX_SSS_SDM_SDMMACOffset`)

4.8.21.3.1 Building the Demo

- Build NX middleware stack. Refer **Linux build**.
  – project - `ex_sdm_provision`

4.8.21.3.2 Running the Example

```
./ex_sdm_provision
```

4.8.21.3.3 Console output

If everything is successful, the output will be similar to:

```
sss   :INFO :Session Open Succeed
App   :INFO :Set AES Key 1.
App   :INFO :Set ECC Private Key 2.
App   :INFO :Change File 2 Setting.
App   :INFO :SDM Enable: 1
App   :INFO :Defer Enable: 0
App   :INFO :VCUID Enable: 1
App   :INFO :SDMReadCtr Enable: 1
App   :INFO :SDMReadCtrLimit Enable: 0
App   :INFO :SDMENCFileData Enable: 1
App   :INFO :GPIOStatus Enable: 1
App   :INFO :SDMMetaRead: 0x1
App   :INFO :SDMFileRead: 0x1
App   :INFO :SDMFileRead2: 0x2
App   :INFO :SDMCtrRet: 0x0
App   :INFO :VCUIDOffset: 0x10
App   :INFO :SDMReadCtrOffset: 0x40
App   :INFO :PICCDataOffset: 0x20
App   :INFO :GPIOStatusOffset: 0x68
App   :INFO :SDMMACInputOffset: 0x10
App   :INFO :SDMENCOffset: 0x60
App   :INFO :SDMENCLength: 0x20
App   :INFO :SDMMACOffset: 0x80
App   :INFO :SDMReadCtrLimit: 0x0
App   :INFO :Defer SDM Encryption Enable: 0
App   :INFO :Defer Method: 0x0
App   :INFO :SDM File Setting Example Success !!!...
App   :INFO :ex_sss Finished
```

5 plugins / Add-ons

5.1 Introduction on Mbed TLS (3.x) ALT Implementation

Mbed TLS ALT implementation allows Mbed TLS stack use the secure authenticator access using SSS layer. Crypto operations performed using the secure authenticator.

Crypto operations supported -

1. ECDSA Sign. **Refer** ecdsa_sign_alt
2. ECDSA Verify. **Refer** ecdsa_sign_alt
3. Random number generation. **Refer** ctr_drbg_alt
4. AES ECB/CBC encryption/decryption. **Refer** aes_alt
5. ECDH **Refer** ecdh_alt

5.1.1 Example

Refer ex_mbedtls_3_x_alt

Note: 1. For ECDSA ALT implementation, every time the control goes to ALT implementation, session open and close is performed. This will have all transient objects will be lost. To avoid the session open / close in ALT implementation, Use the sss_mbedtls_set_keystore_ecdsa_sign() / sss_mbedtls_set_keystore_ecdsa_verify()/ APIs to pass the key store.

Note: 2. Use Reference key to refer to the actual key in NX Secure Authenticator for ECDSA Sign Operation. Using nxclitool you can generate / inject a key pair in SA and create reference key for the same. **Refer** **nxclitool_scripts** for example commands.

Note: 3. ECDSA Verify works only when the session authentication is Symmetric. Not with Sigma-I.

Note: 4. Random number generation is offloaded to NX Secure authenticator only if the key store is passed using sss_mbedtls_set_keystore_rng() API. (No session open done in alt implementation)

Note: 5. ECDH key derivation is offloaded to NX Secure authenticator only if the key store is passed using sss_mbedtls_set_keystore_ecdh() API. (No session open done in alt implementation)

Note: 6. AES ECB/CBC encryption/decryption is offloaded to NX Secure authenticator only if the key store is passed using sss_mbedtls_set_keystore_aes() API. (No session open done in alt implementation) For AES ALT, the key is always set at location 0x08 (kSE_CryptoDataSrc_TB0). And AES ALT works only in plain authentication mode(`NXMW_Auth=None`). The access conditions can be configured through Cmd.SetConfiguration Option 0x15. **Refer** **nx_tool_setconfig**

5.1.2 Key Management

Mbed TLS requires a key pair, consisting of a private and a public key, to be loaded before the cryptographic operations can be executed. This creates a challenge when Mbed TLS is used in combination with a secure authenticator as the private key cannot be extracted out from the Secure authenticator.

The solution is to populate the Key data structure with only a reference to the Private Key inside the Secure authenticator instead of the actual Private Key. The public key as read from the Secure authenticator can still be inserted into the key structure.

When the control comes to the ALT implementation, we check if the key is a reference key or not. In case of reference key, the key id is extracted and Secure authenticator is used to perform Sign operation. If the key is not a reference key, execution will roll back to software implementation.

5.1.3 Reference Key

Refer to **ec-reference-key-format**.

5.1.4 Build / Configuration of Mbed TLS ALT files

Mbed TLS library can be built with above ALT files using the below cmake options

CMake configurations (To be applied on top of a configured host build area):: - Select CMake options:

1. `NXMW_HostCrypto=MBEDTLS`
2. `NXMW_MBedTLS=3_X`
3. `NXMW_mbedTLS_ALT=SSS`

The custom configuration file to enable required ALT operation is present in **Refer config file for Windows** **mbedtls_sss_alt_config** **Refer config file for KSDK** **sss_ksdk_mbedtls_3x_alt_config** By default only ECDSA Sign is enabled.

Use the below macros in the configuration file **Refer** **mbedtls_sss_alt_config** to enable / disable required crypto operation

- MBEDTLS_ECDSA_SIGN_ALT

- MBEDTLS_ECDSA_VERIFY_ALT
- MBEDTLS_AES_ENCRYPT_ALT
- MBEDTLS_AES_DECRYPT_ALT
- MBEDTLS_ECDH_COMPUTE_SHARED_ALT

5.1.5 Secure Authenticator usage with ALT files in TLS handshake

SA is used for following operations during TLS handshake.

1. ECDSA Sign using provisioned client key.
2. Optional - All ECDSA verify operation.

5.1.6 Testing Mbed TLS ALT (Windows)

Mbed TLS client and server example can be used to test the Mbed TLS ALT implementation.

- Project: `mbedtls_3x_client` and `mbedtls_3x_server`
  - Build client example with ALT option enabled `-DPTMW_mbedTLS_ALT:STRING=SSS`
  - Build server example with ALT option disabled `-DPTMW_mbedTLS_ALT:STRING=None`

5.1.6.1 Running examples -

Directory `nxmw\plugin\mbedtls3x\scripts` contains test scripts for starting Mbed TLS server and client applications with different cipher suites. Before executing some test scripts, the secure authenticator must first be provisioned.

1. Provision secure authenticator using python scripts in directory `nxmw\plugin\mbedtls3x\scripts`.
   Build nxclitool (Refer :ref:_nx-cli-tool) with cmake options
   a. `NXMW_All_Auth_Code=Enabled`
   b. `NXMW_Auth=SYMM_Auth`
   c. `NXMW_Secure_Tunneling=NTAG_AES128_EV2`
   d. `NXMW_mbedTLS_ALT=none`

   To provision secure authenticator for ECC `create_and_provision_ecc_keys.py -auth_type symmetric -smcom vcom -port "COM3" -curve prime256v1 -waccess 0x0E`
   To see possible values of input arguments, run without any parameters `create_and_provision_ecc_keys.py`.
   .. note:: when running with vcom set port using EX_SSS_BOOT_SSS_PORT=<COM_PORT>.
2. Starting Mbed TLS SSL client and server applications::

```
python start_ssl2_server.py <ec_curve>
python start_ssl2_client.py <ec_curve> <cipher suite>
```

5.2 Introduction on OpenSSL engine

Starting with OpenSSL 0.9.6 an 'Engine interface' was added to OpenSSL allowing support for alternative cryptographic implementations. This Engine interface can be used to interface with external crypto devices. The key injection process is secure module specific and is not covered by the Engine interface.

Following functionality can be made available over the OpenSSL Engine interface:

- EC crypto
  - EC sign/verify
  - ECDH compute key
- Fetching random data

5.2.1 OpenSSL versions

The OpenSSL Engine is compatible with OpenSSL versions 1.1.1.

5.2.2 Platforms

The OpenSSL engine can be used on Raspberry Pi (running Raspbian).

5.2.3 Keys

5.2.3.1 Key Management

The cryptographic functionality offered by the OpenSSL engine requires a reference to a key stored inside the Secure Authenticator (exception is RAND_Method). These keys are typically inserted into the Secure Authenticator in a secured environment during production.

OpenSSL requires a key pair, consisting of a private and a public key, to be loaded before the cryptographic operations can be executed. This creates a challenge when OpenSSL is used in combination with a Secure Authenticator as the private key cannot be extracted out from the Secure Authenticator.

The solution is to populate the OpenSSL Key data structure with only a reference to the Private Key inside the Secure Authenticator instead of the actual Private Key. The public key as read from the Secure Authenticator can still be inserted into the key structure.

OpenSSL crypto API's are then invoked with these data structure objects as parameters. When the crypto API is routed to the Engine, the OpenSSL engine implementation decodes these key references and invokes the SSS API with correct Key references for a cryptographic operation. If the input key is not reference key, execution will roll back to OpenSSL software implementation

5.2.3.2 EC Reference key format

The following provides an example of an EC reference key. The value reserved for the private key has been used to contain:

- a pattern of `0x10..00` to fill up the data structure MSB side to the desired key length
- a 32 bit key identifier (in the example below `0x00000002`)
- a 64 bit magic number (always `0xA5A6B5B6A5A6B5B6`)
- a byte to describe the key class (`0x10` for Key pair and `0x20` for Public key)
- a byte to describe the key index (use a reserved value `0x00`)

```
Private-Key: (256 bit)
priv:
    10:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:
    00:00:00:00:00:00:02:A5:A6:B5:B6:A5:A6:B5:B6:
    kk:ii
pub:
    04:1C:93:08:8B:26:27:BA:EA:03:D1:BE:DB:1B:DF:
    8E:CC:87:EF:95:D2:9D:FC:FC:3A:82:6F:C6:E1:70:
    A0:50:D4:B7:1F:F2:A3:EC:F8:92:17:41:60:48:74:
    F2:DB:3D:B4:BC:2B:F8:FA:E8:54:72:F6:72:74:8C:
    9E:5F:D3:D6:D4
ASN1 OID: prime256v1
```

Note: The key identifier `0x00000002` (stored in big-endian convention) is in front of the magic number `0xA5A6B5B6A5A6B5B6`.

Note: The padding of the private key value and the magic number makes it unlikely a normal private key value matches a reference key.

Note: Ensure the value reserved for public key and ASN1 OID contain the values matching the stored key.

5.2.4 Building the OpenSSL engine

Refer **Linux build**

Select CMake options:

- `NXMW_HostCrypto=OPENSSL`
- `NXMW_OpenSSL=1_1_1`

Project: `sss_engine`

The cmake build system will create an OpenSSL engine for supported platforms. The resulting OpenSSL engine will be copied to the SW tree in directory `nxmw/plugin/openssl/bin`.

Ensure the following flag is defined when building an application that will be linked against the engine: `-DOPENSSL_LOAD_CONF`

5.2.5 Sample scripts to demo OpenSSL Engine

The directory `nxmw/plugin/openssl/scripts` (**here**) contains a set of python scripts. These scripts use the OpenSSL Engine in the context of standard OpenSSL utilities. They illustrate using the OpenSSL Engine for fetching random data and supported EC crypto operations. The scripts that illustrate EC crypto operations depend on prior provisioning of the Secure Authenticator.

The python script **openssl_provision.py** can be used to provision the SA with required key,

```
python openssl_provision.py -smcom <SMCOM> -port <PORT_NAME> -curve <CURVE> -
keypath <PATH_TO_KEY>
```

Some keys are placed in the path `nxmw/plugin/openssl/keys` for demonstration purpose. User can provision the SA for OpenSSL Engine using following command:

```
cd nxmw/plugin/openssl/scripts
python openssl_provision.py -smcom t1oi2c -port /dev/i2c-1 -curve prime256v1 -
keypath ../keys/prime256v1/ecc_key_kp.pem
```

Note: The provisioning script uses NX CLI Tool to provision the SA and assumes some of the inputs to the CLI tool for simplicity. For e.g. `NXMW_Secure_Tunneling` is set to `NTAG_AES128_EV2` and `NXMW_Auth` is set to `SYMM_Auth` by default and, these can be changed in the script as per user's preference.

Note: This script provisions the key with signing policy enabled at key ID 0x02 and the key with ECDH policy enabled at ID 0x03. User must make sure no other keys are present in these key IDs before provisioning.

To generate the random numbers, invoke following script:

```
cd nxmw/plugin/openssl/scripts
python3 openssl_rnd.py --connection_data none
```

The following set of commands invokes the OpenSSL Provider for ECDSA sign/verify operations and ECDH calculations:

```
python3 openssl_EccSign.py --key_type prime256v1
python3 openssl_Ecdh.py --key_type prime256v1
```

5.3 OpenSSL Provider for NX Secure Authenticator

A provider, in OpenSSL terms, is a unit of code that provides one or more implementations for various operations for diverse algorithms that one might want to perform.

Depending on the capabilities of the attached secure authenticator, the following functionality can be made available over the OpenSSL provider here (sssProvider).

- EC crypto
  - EC key generation
  - EC sign/verify
  - ECDH compute key
  - CSR
- Random generator

The OpenSSL provider is compatible with OpenSSL versions 3.0.x

OpenSSL provider is tested on i.MX (imx8mqevk, with yocto), Raspberry Pi (Raspberry Pi 4 Model B, Ubuntu 22.04.2 LTS)

5.3.1 Getting Started on Raspberry Pi

5.3.1.1 Prerequisite

- Raspberry pi
- cmake installed: `sudo apt-get install cmake`
- OpenSSL 3.0.x installed
- SA connected to Raspberry Pi on i2c port

5.3.1.2 Build

Run the commands below to build OpenSSL provider for NX secure authenticator

```
git clone https://github.com/NXP/nxmw.git
cd scripts
python create_cmake_projects.py
cd ../../nxmw_build/raspbian_native_nx_t1oi2c
cmake -DNXMW_OpenSSL=3_0 -DWithSharedLIB=OFF -DNXMW_All_Auth_Code=Enabled .
sudo make all
sudo make install
sudo ldconfig /usr/local/lib
```

Above commands will build the OpenSSL provider and NX CLI tool (used for provisioning the SA) and copy it in `plugin/openssl_provider/bin` and `binaries/tmp` folders respectively.

5.3.2 Testing OpenSSL Provider

5.3.2.1 Random Number Generation

```
openssl rand --provider /usr/local/lib/libsssProvider.so -hex 32
```

5.3.2.2 ECC (Nist256) Key Generation

```
mkdir output
openssl ecparam --provider /usr/local/lib/libsssProvider.so --provider default -
name prime256v1:0x02 -genkey -noout -out output/nx_prime256v1_ref.pem -propquery
 "provider=nxp_prov"
```

The above command will generate the key in secure authenticator at 0x02 keyid and the output `nx_prime256v1_ref.pem` is the reference to the key location of secure authenticator. Refer **"Referencing keys in the secure authenticator"** section on this page for more details.

The reference key can also be used to perform further crypto operation with secure authenticator.

Supported curves

- prime256v1 (secp256r1)
- brainpoolP256r1

> **Note:** The key will be generated with default policy i.e. only sign enabled.
>
> **Note:** Key generation on secure authenticator using nxp provider can be done only by loading nxp provider with highest priority.
>
> **Note:** Rest of the commands in this section which require an EC key will assume that a key is present at key ID 0x02 and "nx_prime256v1_ref.pem" is the corresponding reference key.

5.3.2.3 ECDSA - Sign Operation

```
mkdir input_data
echo "Hello Word" > input_data/input_data.txt
openssl pkeyutl --provider /usr/local/lib/libsssProvider.so --provider default
 -inkey nxp:0x02 -sign -rawin -in input_data/input_data.txt -out output/
signature.bin -digest sha256
```

In case the default provider is loaded first, ensure to pass the correct property query. Example -

```
openssl pkeyutl --provider default --provider /usr/local/lib/libsssProvider.so
 -inkey nxp:0x02 -sign -rawin -in input_data/input_data.txt -out output/
signature.bin -digest sha256 -propquery "?nxp_prov.signature.ecdsa=yes"
```

Refer - 'OSSL Algorithms property definitions' section for more details.

5.3.2.4 ECDSA - Verify Operation

```
#Extract public key from reference key
openssl ec -in output/nx_prime256v1_ref.pem -pubout -out output/pubkey.pem

openssl pkeyutl -verify --provider default -inkey output/pubkey.pem -pubin -
rawin -in input_data/input_data.txt -sigfile output/signature.bin -digest sha256
```

Note: Here verify operation is performed by host and not the SA as SA does not have public key required for this operation stored inside.

5.3.2.5 ECDH Operation

Provisioning of an EC key (with ECDH policy enabled) should be done first before performing ECDH operation using the python script **openssl_provision.py**. This script will inject an EC key inside SA at key ID 0x03 for ECDH operation. This script uses NX CLI tool to provision and assumes that the nxclitool binary is present at `binaries/tmp` (refer: **NX CLI Tool**).

```
cd plugin/openssl_provider/scripts
mkdir output
# Peer key is created on host
openssl ecparam -name prime256v1 -genkey -noout -out output/peer_key.pem

# Extract public key from peer key
openssl ec -in output/peer_key.pem -pubout -out output/peer_public_key.pem

# Assume off chip EC key generation
openssl ecparam -name prime256v1 -genkey -noout -out output/ec_key.pem

# Provisions ec_key.pem inside SA with ECDH policy enabled at Key ID 0x03
python openssl_provision.py -smcom t1oi2c -port /dev/i2c-1 -curve prime256v1 -
keypath output/ec_key.pem

openssl pkeyutl -derive --provider /usr/local/lib/libsssProvider.so --provider
 default -inkey nxp:0x03 -peerkey output/peer_public_key.pem -hexdump -out
 output/ecdh_key.bin
```

5.3.3 Referencing keys in the secure authenticator

The keys created inside secure authenticator can be referenced in 3 different ways

1. Reference Keys in file format
2. Labels with reference key. Example - nxp:"path to reference key file"
3. Labels with key id. Example - nxp:0x12345678

5.3.3.1 1. Reference Keys in file format

The cryptographic functionality offered by the OpenSSL provider requires a reference to a key stored inside the secure authenticator (exception is random generation).

OpenSSL requires a key pair, consisting of a private and a public key, to be loaded before the cryptographic operations can be executed. This creates a challenge when OpenSSL is used in combination with a secure authenticator as the private key cannot be extracted out from the secure authenticator.

The solution is to populate the OpenSSL Key data structure with only a reference to the private key inside the secure authenticator instead of the actual private key. The public key as read from the secure authenticator can still be inserted into the key structure.

OpenSSL crypto APIs are then invoked with these data structure objects as parameters. When the crypto API is routed to the provider, the NX OpenSSL provider implementation decodes these key references and invokes the secure authenticator APIs with correct key references for a cryptographic operation. If the input key is not a reference key, execution will roll back to OpenSSL software implementation.

Note: When using this method, the sss provider has to be loaded first. This will ensure that the sss provider can decode the key id information present in the reference key.

5.3.3.1.1 EC Reference Key Format

The following provides an example of an EC reference key. The value reserved for the private key has been used to contain:

- a pattern of `0x10..00` to fill up the data structure MSB side to the desired key length
- a 32 bit key identifier (in the example below `0x7DCCBBAA`)
- a 64 bit magic number (always `0xA5A6B5B6A5A6B5B6`)
- a byte to describe the key class (`0x10` for Key pair and `0x20` for Public key)
- a byte to describe the key index (use a reserved value `0x00`)

```
Private-Key: (256 bit)
priv:
    10:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:
    00:00:00:7D:CC:BB:AA:A5:A6:B5:B6:A5:A6:B5:B6:
    kk:ii
pub:
    04:1C:93:08:8B:26:27:BA:EA:03:D1:BE:DB:1B:DF:
    8E:CC:87:EF:95:D2:9D:FC:FC:3A:82:6F:C6:E1:70:
    A0:50:D4:B7:1F:F2:A3:EC:F8:92:17:41:60:48:74:
    F2:DB:3D:B4:BC:2B:F8:FA:E8:54:72:F6:72:74:8C:
    9E:5F:D3:D6:D4
ASN1 OID: prime256v1
```

- The key identifier `0x7DCCBBAA` (stored in big-endian convention) is in front of the magic number `0xA5A6B5B6A5A6B5B6`
- The padding of the private key value and the magic number make it unlikely a normal private key value matches a reference key.
- Ensure the value reserved for public key and ASN1 OID contain the values matching the stored key.

5.3.3.2 2. Labels with reference key.

In this method, the reference key file (described in previous section) with full path can be passed in string format with "nxp:" as prefix. Example - nxp:"path to reference key file".

> **Note:** When using this approach, there is no need to load the sss provider first. Default provider can have the higher priority.

5.3.3.3 3. Labels with key id.

In this method, the 4 byte key id of the Key created / stored in secure authenticator is passed as is in string format with "nxp:" as prefix. Example - nxp:0x12345678

> **Note:** When using this approach, there is no need to load the sss provider first. Default provider can have the higher priority.

5.3.4 OSSL Algorithms property definitions

Following properties definitions are added in nxp provider(see file ../provider/src/sssProvider_main.c),

- Random number generation - `nxp_prov.rand=yes`
- Key management - `nxp_prov.keymgmt=yes` (Required to offload the ECC operations to nxp provider when the keys are stored in NX secure authenticator).
- Signature - `nxp_prov.signature=yes` (Required to offload the ECC / Verify operations to nxp provider when the keys are stored in NX secure authenticator).

- ECDH - `nxp_prov.keyexch=yes` (Required only when the ephemeral keys are generated on NX).
- Key Store - `nxp_prov.store=yes` (Required when the keys are referenced using label (nxp:) or reference keys).
  - For keys passed with nxp: prefix - `nxp_prov.store.nxp=yes`
  - For keys passed with reference key format - `nxp_prov.store.file=yes`

5.3.5 Testing OpenSSL Provider

The directory `plugin/openssl_provider/scripts` contains a set of python scripts. These scripts use the OpenSSL provider in the context of standard OpenSSL utilities. They illustrate using the OpenSSL provider for fetching random data and EC crypto operations. Before using the scripts except openssl_EccGenKey.py, an EC key needs to be provisioned in SA. This can be done using **openssl_provision.py** which provisions the EC key at key ID 0x02 with signing policy enabled and at key ID 0x04 with ECDH policy enabled.

```
# ECC Key generation
python openssl_EccGenKey.py  --key_type prime256v1 --connection_data /dev/i2c-1

# ECC CSR and certificate creation
python openssl_EccCSR.py --key_type prime256v1 --connection_data /dev/i2c-1

# Provision EC key inside SA
python openssl_provision.py -smcom t1oi2c -port /dev/i2c-1 -curve prime256v1 -
keypath ../keys/prime256v1/ecc_key_kp.pem

# Random number generation
python openssl_rnd.py --connection_data /dev/i2c-1

# ECDSA Operations
python openssl_EccSign.py --key_type prime256v1 --connection_data /dev/i2c-1

# ECDH Key generation
python openssl_Ecdh.py --key_type prime256v1 --connection_data /dev/i2c-1
```

5.3.6 TLS Client example using provider

This section explains how to set-up a TLS link using the NX OpenSSL Provider on the client side. The TLS demo demonstrates setting up a mutually authenticated and encrypted link between a client and a server system.

The key pair used to identify the client is created / stored in the secure authenticator.

The key pair used to identify the server is simply available as a pem file.

The public keys associated with the respective key pairs are contained in respectively a client and a server certificate.

The CA is a self-signed certificate. The same CA is used to sign client and server certificate.

5.3.6.1 TLS1.2 / TLS1.3 client example using EC keys

Create client and server credentials as shown below

```
openssl ecparam -name prime256v1 -out prime256v1.pem

# Create Root CA key pair and certificate
openssl ecparam -in prime256v1.pem -genkey -noout -out tls_rootca_key.pem
```

```
openssl req -x509 -new -nodes -key tls_rootca_key.pem -subj /OU="NXP Plug Trust
 CA/CN=NXP RootCAvRxxx" -days 4380 -out tls_rootca.cer

# Create client key inside secure authenticator
openssl ecparam --provider /usr/local/lib/libsssProvider.so --provider default -
name prime256v1:0x02 -genkey -out tls_client_key_ref_0x02.pem

# Create Client key CSR. Use the provider to access the client key created in
 the previous file.
openssl req --provider /usr/local/lib/libsssProvider.so --provider default -
new -key tls_client_key_ref_0x02.pem -subj "/CN=NXP_NX_SA_TLS_CLIENT_ECC" -out
 tls_client.csr

# Create Client certificate
openssl x509 -req -sha256 -days 4380 -in tls_client.csr -CAcreateserial -CA
 tls_rootca.cer -CAkey tls_rootca_key.pem -out tls_client.cer

# Create Server key pair and certificate
openssl ecparam -in prime256v1.pem -genkey -noout -out tls_server_key.pem
openssl req -new -key tls_server_key.pem -subj "/CN=NXP_TLS_SERVER_ECC" -out
 tls_server.csr
openssl x509 -req -sha256 -days 4380 -in tls_server.csr -CAcreateserial -CA
 tls_rootca.cer -CAkey tls_rootca_key.pem -out tls_server.cer
```

Run Server as

```
openssl s_server -accept 8080 -no_ssl3 -named_curve prime256v1  -CAfile
 tls_rootca.cer  -cert tls_server.cer -key tls_server_key.pem -cipher ECDHE-
ECDSA-AES128-SHA256 -Verify 2 -state -msg
```

Run Client as

```
openssl s_client --provider /usr/local/lib/libsssProvider.so --provider default
 -connect 127.0.0.1:8080 -tls1_2 -CAfile tls_rootca.cer -cert tls_client.cer -
key tls_client_key_ref_0x02.pem -cipher ECDHE-ECDSA-AES128-SHA256 -state -msg
```

OR

```
openssl s_client --provider /usr/local/lib/libsssProvider.so --provider default
 -connect 127.0.0.1:8080 -tls1_3 -CAfile tls_rootca.cer -cert tls_client.cer -
key tls_client_key_ref_0x02.pem -state -msg
```

5.3.7 OpenSSL Configuration file

The provider can be loaded via OpenSSL configuration file also. Changes required in configuration file to load provider is shown below,

```
...

openssl_conf = openssl_init
config_diagnostics = 1

[openssl_init]
providers = provider_sect

[provider_sect]
nxp_prov = nxp_sect
```

```
default = default_sect

[nxp_sect]
identity = nxp_prov
module = <provider lib path>
activate = 1

[default_sect]
activate = 1

...
```

The order in which the providers are written in `[provider_sect]` section, defines the priority of the providers loaded. The one included first, will have the higher priority.

> **Note:** It is not recommended to modify the default OpenSSL config file. Create a new config file to load custom providers and set the OPENSSL_CONF env variable to config file path. Example: export OPENSSL_CONF=<CONFIG_FILE_PATH>

5.4 PKCS#11 Plugin

PKCS#11(v2.40) is a Public-Key Cryptography Standard for cryptographic data manipulation. It is mainly used with Hardware Security Modules and smart cards.

PKCS#11 standalone library is supported with NX for Linux based platforms.

5.4.1 PKCS#11 Label Handling

PKCS#11 library calculates keyId through `pkcs11_label_to_keyId`:

1. CKA_LABEL starts with `sss:`
   - Based on the most significant nibble keyID is being handled.
     - `sss:0x1xxxxxxx` EC keyID (0x00 to 0x04)
     - `sss:0x2xxxxxxx` Cert keyID (0x00 to 0x1F)
     - `sss:0x4xxxxxxx` Symm keyID (0x10 to 0x17)
   - keyID is generated by interpreting following string as hex value of the keyID. Example - If CKA_LABEL is `sss:0x10000001`, keyID is 0x01

     > **Note:** 0x1 to 0x3 is reserved keyIDs for certificate and cannot be used.

5.4.2 PKCS#11 specifications

Token Label
SSS_PKCS11

Pin
Not required

Supported Mechanisms

- Digest Mechanisms
  - CKM_SHA256
  - CKM_SHA384
- ECDSA Mechanisms
  - CKM_ECDSA

  - CKM_ECDSA_SHA256
- Key Generation Mechanisms
  - CKM_EC_KEY_PAIR_GEN
  - CKM_AES_KEY_GEN
- AES Mechanisms
  - CKM_AES_ECB
  - CKM_AES_CBC
- HMAC Mechanisms
  - CKM_SHA256_HMAC

Supported API

- General-purpose functions
  - C_Initialize
  - C_Finalize
  - C_GetInfo
  - C_GetFunctionList
- Slot and token management functions
  - C_GetSlotList
  - C_GetSlotInfo
  - C_GetTokenInfo
  - C_GetMechanismList
  - C_GetMechanismInfo
- Session management functions
  - C_OpenSession
  - C_CloseSession
  - C_GetSessionInfo
  - C_Login
  - C_Logout
- Object management functions
  - C_CreateObject
  - C_DestroyObject
  - C_GetAttributeValue
  - C_FindObjectsInit
  - C_FindObjects
  - C_FindObjectsFinal
- Encryption functions
  - C_EncryptInit
  - C_Encrypt
- Decryption functions
  - C_DecryptInit
  - C_Decrypt
- Message digesting functions
  - C_DigestInit
  - C_Digest
  - C_DigestUpdate
  - C_DigestFinal
- Signing and MACing functions
  - C_SignInit

  - C_Sign
  - C_SignUpdate
  - C_SignFinal
  - C_VerifyInit
  - C_Verify
  - C_VerifyUpdate
  - C_VerifyFinal
- Key management functions
  - C_GenerateKey
  - C_GenerateKeyPair
- Random number generation functions
  - C_SeedRandom
  - C_GenerateRandom

5.4.3 Building on Linux / Raspberry-Pi 4

PKCS#11 standalone shared library can be built on Linux / RaspberryPi platforms.

Build PKCS#11 library for Raspberry pi 4 with the following CMake configurations:

- `-DNXMW_RTOS=Default`
- `-DNXMW_HostCrypto=MBEDTLS` / `-DNXMW_HostCrypto=OPENSSL`
- Project: `sss_pkcs11`

> **Note:** While using PKCS#11 as a library on multi threaded systems, the application must ensure proper locking is used. Calling multiple APIs from the library from different threads without proper locks can lead to unexpected behavior.

5.4.4 Using with pkcs11-tool

Install `pkcs11-tool` by running:

```
sudo apt-get install opensc-pkcs11

Note: Tested with OpenSC 0.24.0 version
```

Set environment variable to the installed PKCS#11 shared library:

```
export PKCS11_MODULE=<NX_MW_PATH>/nxmw/plugin/pkcs11/bin/libsss_pkcs11.so
```

Generating new keypair:

```
pkcs11-tool --module $PKCS11_MODULE --keypairgen --key-type EC:prime256v1 --
label "sss:0x10000002"
```

Signing:

```
pkcs11-tool --module $PKCS11_MODULE --sign --id 10000002 -m ECDSA --input-file
 data.txt --output-file signature.sign
```

Hashing:

```
pkcs11-tool --module $PKCS11_MODULE --hash -m SHA256 --input-file in.der --
input-file hash.der
```

5.5 PSA (Platform Security Architecture)

The Platform Security Architecture (PSA) by ARM is a holistic set of threat models, security analyses, hardware and firmware architecture specifications, and an open source firmware reference implementation.

For details on PSA specification, refer to ARMmbed PSA Specification.

5.5.1 PSA SE Driver Interface

The SE Driver interface allows the user to register Secure Authenticator drivers for various cryptographic operations. It is not necessary that one driver should offer all cryptographic functionalities, we can register up to 4 drivers which may offer different functionalities.

5.5.2 PSA APIs in NX Middleware (supported with MbedTLS 3.X)

The PSA APIs for Secure authenticator are implemented in - plugin/psa/*.c.

To include the PSA plugin files in mbedTLS 3.x build enable PSA in mbedTLS_ALT cmake option (`mbedTLS_ALT=PSA`),

The following PSA features / APIs are supported in NX middleware using mbedTLS 3.x

1. Nist256 and Brainpool256 Generate Key / (API - `psa_generate_key`)
2. ECDSA Sign (Nist256 and Brainpool256) / (API - `psa_sign_hash`)
3. AES Key import / (API - `psa_import_key`)
4. AEAD Encrypt - one shot / (API - `psa_aead_encrypt`)
5. AEAD Decrypt - one shot / (API - `psa_aead_decrypt`)
6. HMAC - One shot / (API - `psa_mac_compute`)
7. HMAC Verify - One shot / (API - `psa_mac_verify`)

5.5.3 Building PSA Example

To test the above PSA APIs, refer the PSA example (plugin/psa/psa_example/psa_example.c). Use the below cmake options to build the example,

- Project: `psa_example`
- `Host=lpcxpresso55s69`
- `HostCrypto=MBEDTLS`
- `MBedTLS=3_X`
- `mbedTLS_ALT=PSA`

6 MCUXpresso Projects

6.1 Supported Platforms

Secure Authenticator MW is designed to run on several platforms including Windows, Linux and several MCUs including MCUXN947, LPC55S69 and MCXA153.

This folder contains the platform specific files and driver files

6.1.1 Build with Linux

To use the Secure Authenticator MW with Raspberry Pi, Refer **Linux Build**.

6.1.2 Build with Windows

To use the Secure Authenticator MW with Windows PC, Refer **Windows Build**.

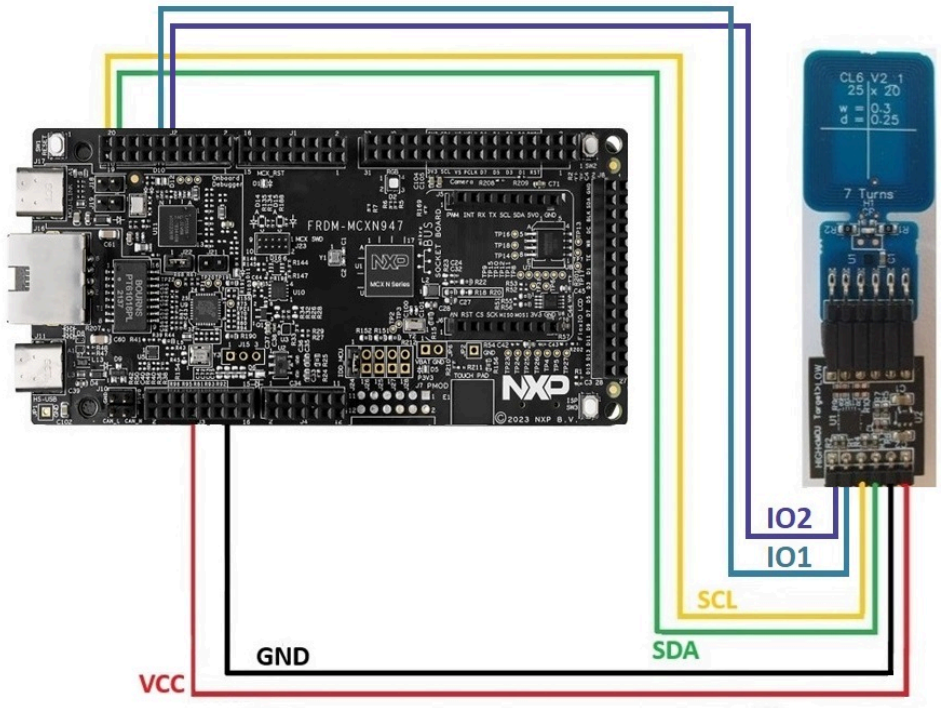
6.1.3 Build with MCUX Platforms

To build the Secure Authenticator MW using MCUXpresso for supported MCUs, Refer **MCUXpresso Build**.

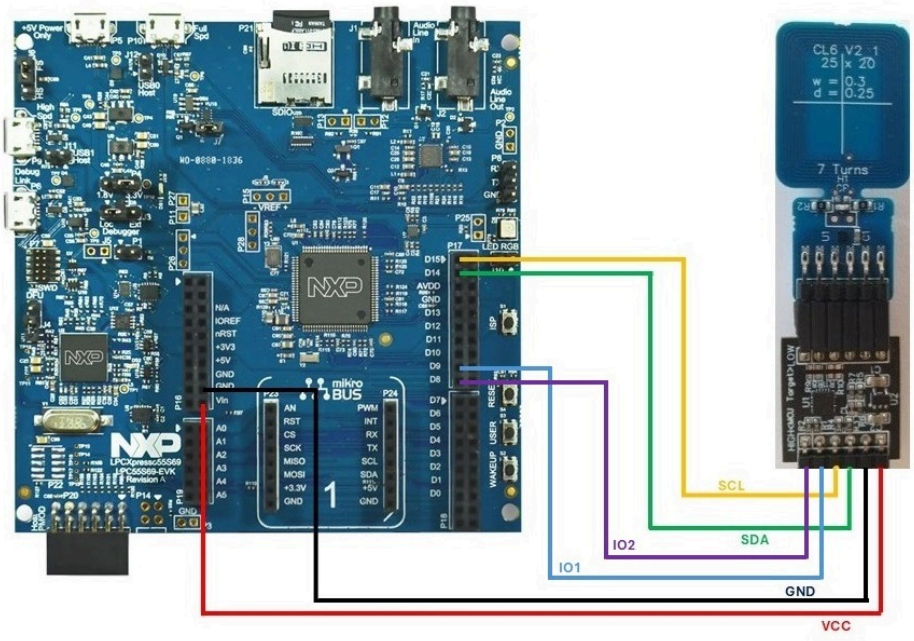
To build the Secure Authenticator MW using CMake build for supported MCUs, Refer **CMake Build**.

6.1.4 Secure Authenticator Connections with Host MCUs

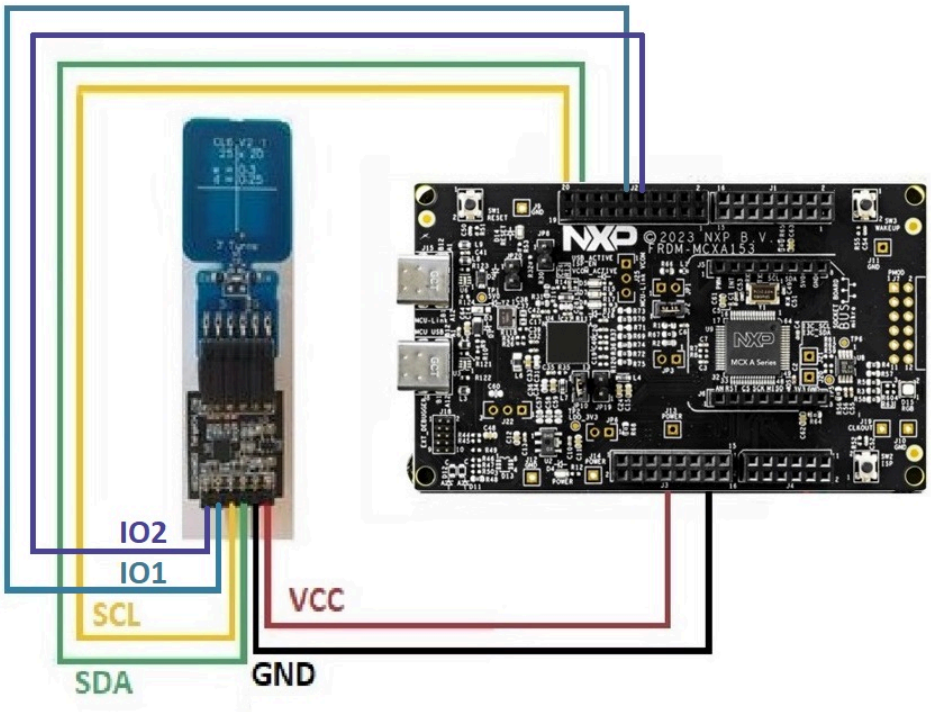
6.1.4.1 MCXN947 Pin Diagram



6.1.4.2 LPC55S69 Pin Diagram



6.1.4.3 MCXA153 Pin Diagram



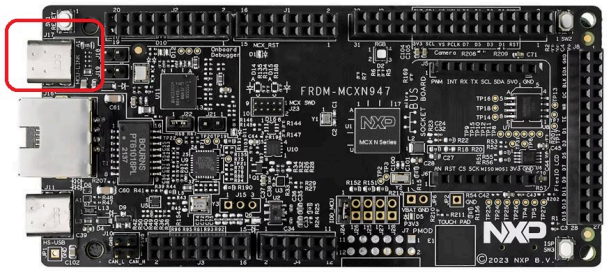
6.2 Getting started for MCUs Using Cmake Build

The NX middleware examples / demos can be cross compiled for MCXN947 / MCXA153 / LPC55S69 on Windows.

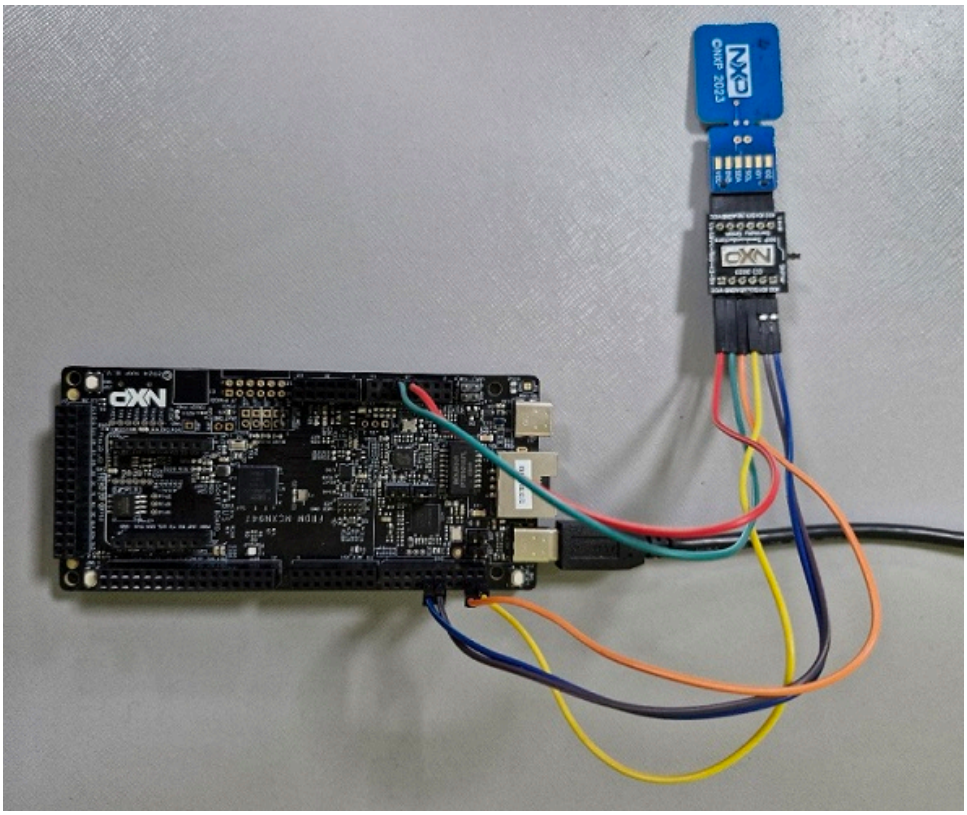
```
NOTE: MCXN947 is used as an Host MCU example in the following documentation.
```

6.2.1 Prerequisite

- MCUXpresso IDE 24.12 (https://www.nxp.com/support/:MCUXpresso-IDE) installed. (Update the installed path in file nxmw/scripts/env_setup.bat)
- SDKs installed for supported MCUs (V24.12.00 for MCXN947/LPC55S69/MCXA153) (To download SDK, refer **MCUXpresso SDK Dashboard**)
- Python installed (version 3.10 or above). Please follow the guideline at Python Download.
- West installed (version 1.2.0 or above). Please follow west setup to install west
- CMake installed (version 3.30.0 or above). Preferably at C:/opt/cmake/bin/cmake.exe
- Connect MCXN947/MCXA153/LPC55S69 using the debug link port to Windows



- Secure authenticator connected to MCXN947/MCXA153/LPC55S69 (Refer: **MCU Pin Diagram**)
- The overall set up will look as shown below.



6.2.2 Getting the NX Middleware Source

- Follow the following steps to create a new workspace which downloads the NX middleware and the required files.

```
west init -m https://github.com/NXP/nxmw.git --mf mcu_sdk/west.yml workspace
OR
west init -m https://github.com/NXP/nxmw.git --mr <branch name> --mf mcu_sdk/
west.yml workspace

cd workspace
west update
```

Note: The complete setup takes 10-15 minutes to finish.

- If you have cloned the NX Middleware using git, and west setup is not done, you can follow the following steps to download the MCU SDK,

```
cd nxmw
west init -l --mf mcu_sdk/west.yml
cd ..
west update
```

6.2.3 Create Build files

Use `nxmw/scripts/create_cmake_projects.py` to generate the build files -

```
cd nxmw/scripts
env_setup.bat
python create_cmake_projects.py
```

Note: The env_setup.bat files defines the development tools environment. Depending on your tools (MCUXpresso, Visual Studio, Java, Python and CMake) file locations you may need to update the tools paths within the env_setup.bat file.

Build files are generated at `nxmw_build/`

- Use the build environment `nxmw_build/nxmw-eclipse_arm` to cross compile for MCXN947/MCXA153/ LPC55S69 :

```
cd nxmw_build/nxmw-eclipse_arm
make
```

- By default the cmake options are set as below.

```
CMAKE_BUILD_TYPE:STRING=Debug
NXMW_ALL_Auth_Code:STRING=Enabled
NXMW_Auth:STRING=SYMM_Auth
NXMW_Auth_Asymm_CA_Root_Key_Id:STRING=0
NXMW_Auth_Asymm_Cert_Repo_Id:STRING=0
NXMW_Auth_Asymm_Cert_SK_Id:STRING=0
NXMW_Auth_Asymm_Host_Curve:STRING=NIST_P
NXMW_Auth_Asymm_Host_PK_Cache:STRING=Enabled
NXMW_Auth_Symm_App_Key_Id:STRING=0
NXMW_Auth_Symm_Diversify:STRING=Disabled
NXMW_Host:STRING=lpcxpresso55s
NXMW_HostCrypto:STRING=MBEDTLS
NXMW_Log:STRING=Default
NXMW_MBedTLS:STRING=2_X
NXMW_NX_Type:STRING=NX_R_DA
NXMW_OpenSSL:STRING=1_1_1
NXMW_RTOS:STRING=Default
NXMW_SA_Type=A30
NXMW_SMCOM:STRING=T1oI2C_GP1_0
NXMW_Secure_Tunneling:STRING=NTAG_AES128_EV2
NXMW_mbedTLS_ALT:STRING=None
```

- Change the cmake options if required. For more details on cmake Cmake Configurations - **Cmake Configurations**

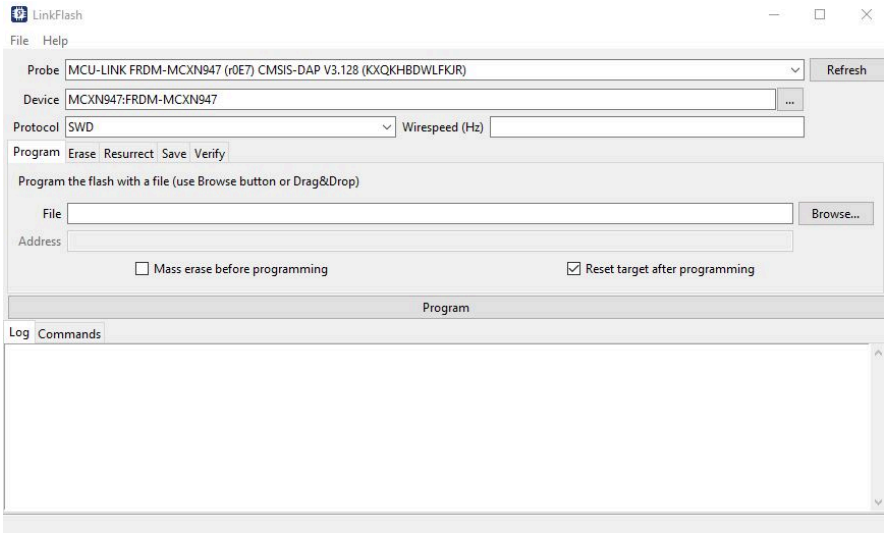
Note: The Sigma Verifier/Prover demo requires to run the Personalization example once first. Refer **Personalization**.

6.2.4 Setting up MCUXPresso IDE

1. Download MCUXpresso 24.12 from: https://www.nxp.com/support/developer-resources/software-development-tools/mcuxpresso-software-and-tools/mcuxpresso-integrated-development-environment-ide:MCUXpresso-IDE
2. For additional help please refer to: https://www.nxp.com/webapp/Download?colCode=MCUXPRESSO-UG

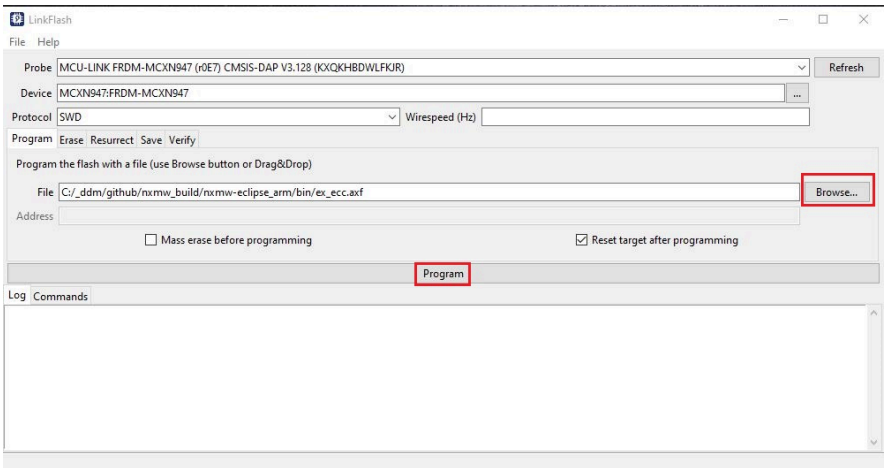
6.2.5 Running the Example

- Link Flash will get installed when installing the MCUXpresso IDE. If not installed, install it from: https://www.nxp.com/design/design-center/software/development-software/mcuxpresso-software-and-tools-/linkserver-for-microcontrollers:LINKERSERVER
- Open Link Flash to flash the binary.
- The probe and device will get detected automatically based on the MCU connected with PC.



- Click on browse and select the .axf or .bin file to be flashed and click on program to flash the file.

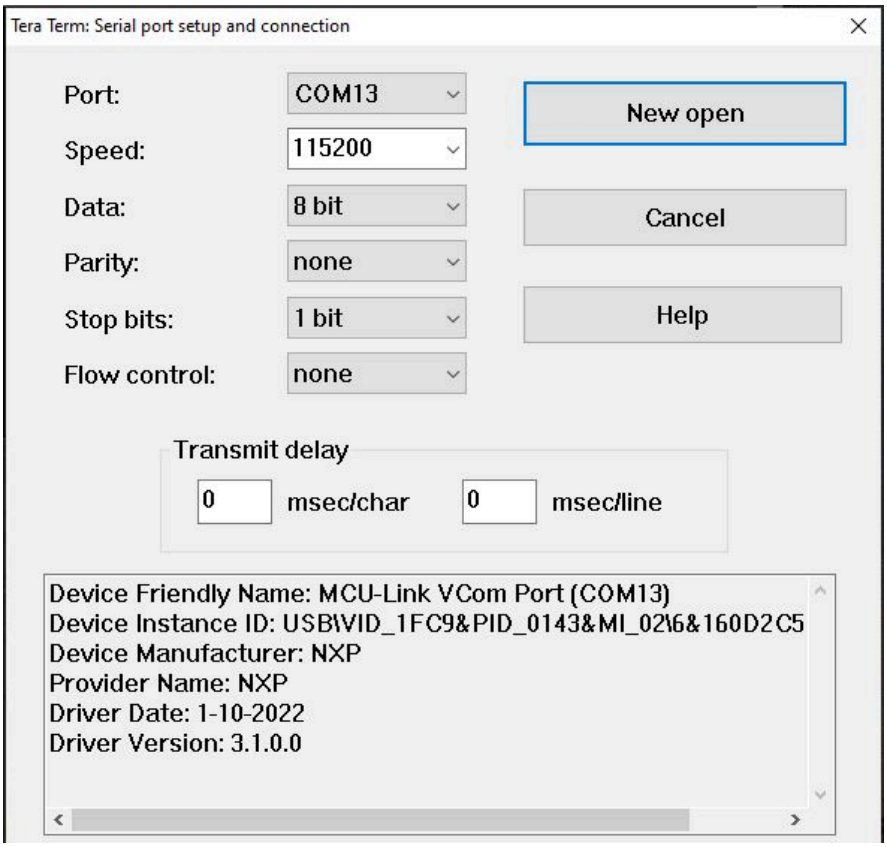
  Note: If .bin file is selected, specify the address (0x00 for MCXN947/MCXA153/LPC55S69).



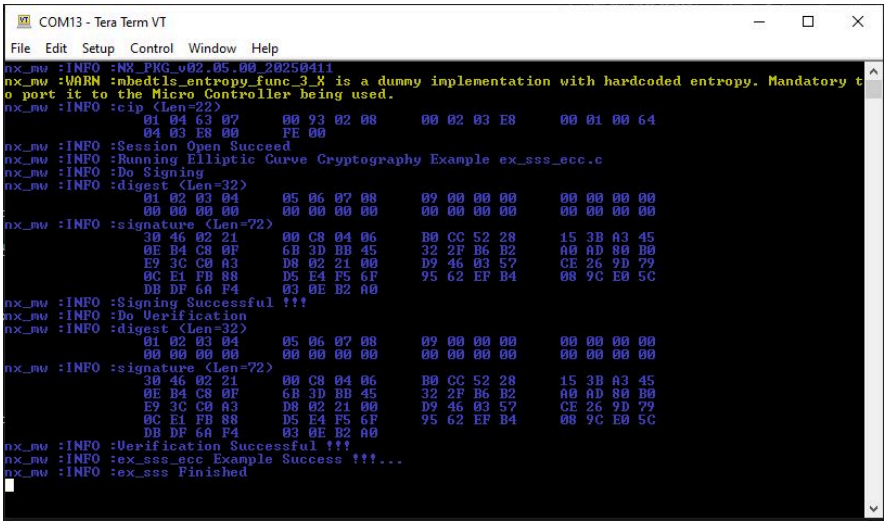
- Once flashed successfully, MCU will reset automatically and program execution will start. Logs will show-up on the terminal (logging in next section). To re-run the same program, press reset button on the MCU.

6.2.6 Logging on The Console

- Open Tera Term and connect serial port.



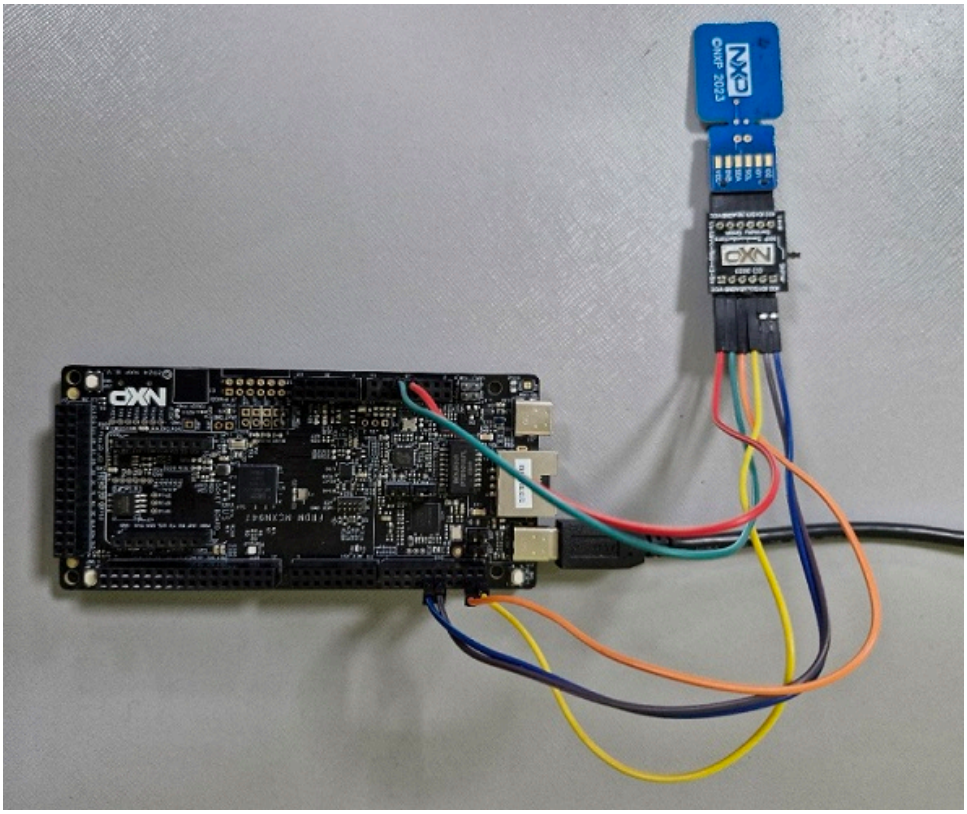
- Once the program execution begins, logs are printed on the terminal (Tera Term) indicating the status of execution.



6.3 Getting Started on MCUs Using Standalone MCUXpresso Projects

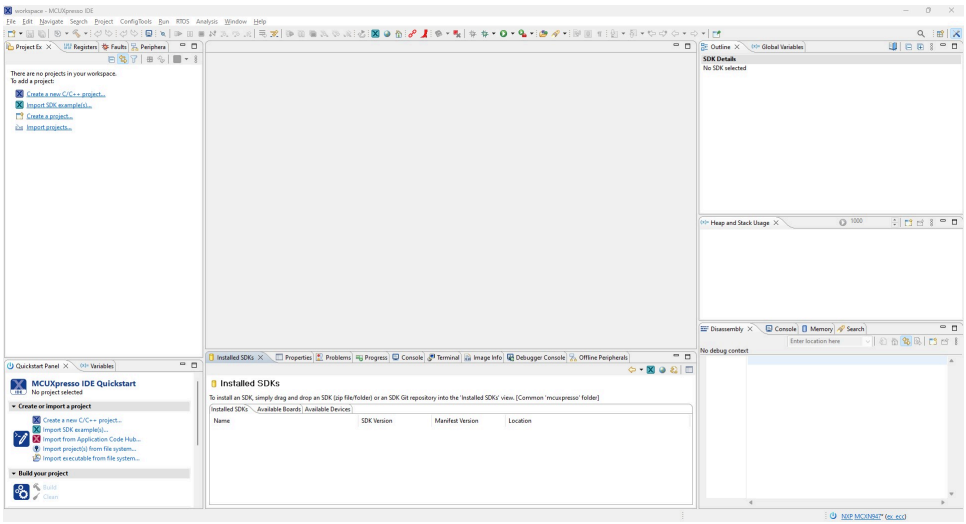
6.3.1 Prerequisites

1. Secure Authentication MW source is cloned from github. (Refer: **Getting the NX Middleware Source**).
2. MCUXpresso IDE v24.12.148 or above is installed (To download the IDE and user guide, refer: **MCUXpresso IDE**).
3. SDK installed for supported MCUs (V24.12.00 for MCXN947/LPC55S69/MCXA153) (To download SDK, refer **MCUXpresso SDK Dashboard**)
4. Secure Authenticator connected to Host MCU (Refer: **MCU Pin Diagram**).
5. The overall set up will look as shown below.

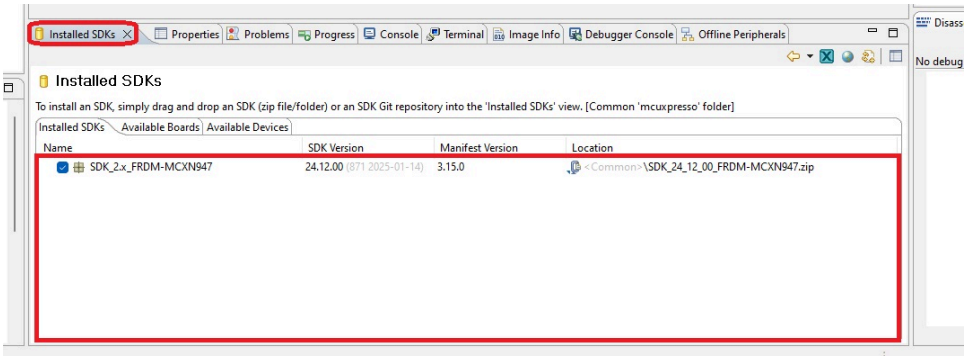


6.3.2 SDK Installation on MCUXpresso

- Open the MCUXpresso IDE and close the welcome page that shows on startup. The IDE should look like below image.

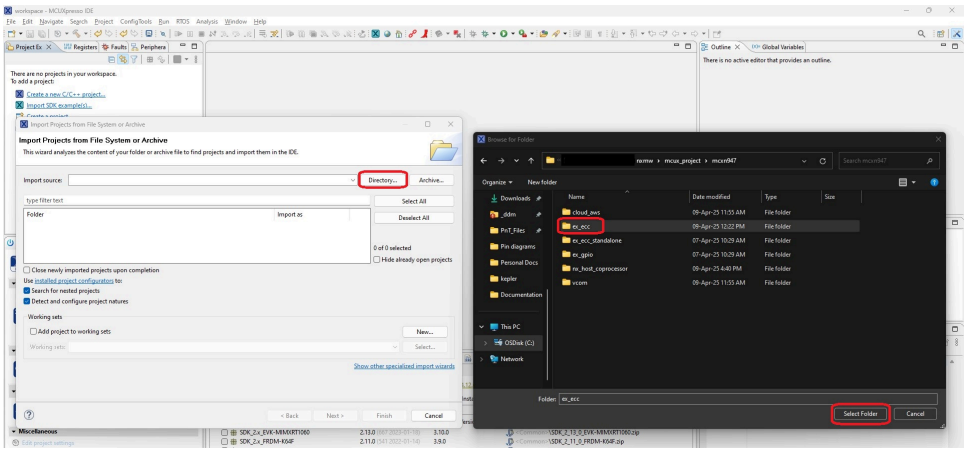


- Before importing a project, Software Development Kit (SDK) needs to be installed in the IDE corresponding to the board being used. To do this, go to **SDK Builder** on NXP website, select the required development board and build the SDK. Once built, download the SDK zip file.
- To install the SDK, drag the zipped SDK folder and drop in the installed SDK section highlighted in below image. If not done by default, select the SDK corresponding to the development board by clicking on the checkbox in front of the SDK.

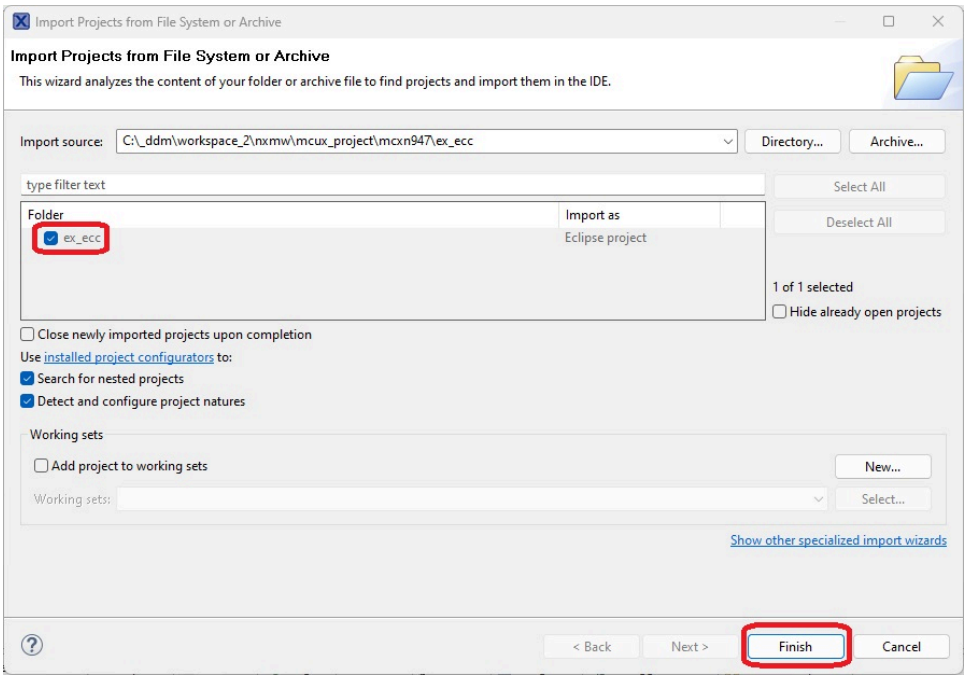


6.3.3 Importing The Project

- To import a project, go to **File** -> **Open Project from File System...**. In the dialog box that opens, click on **Directory...** and navigate to the target project in the folder corresponding to the required development board, select the required project and click on **Select Folder**.

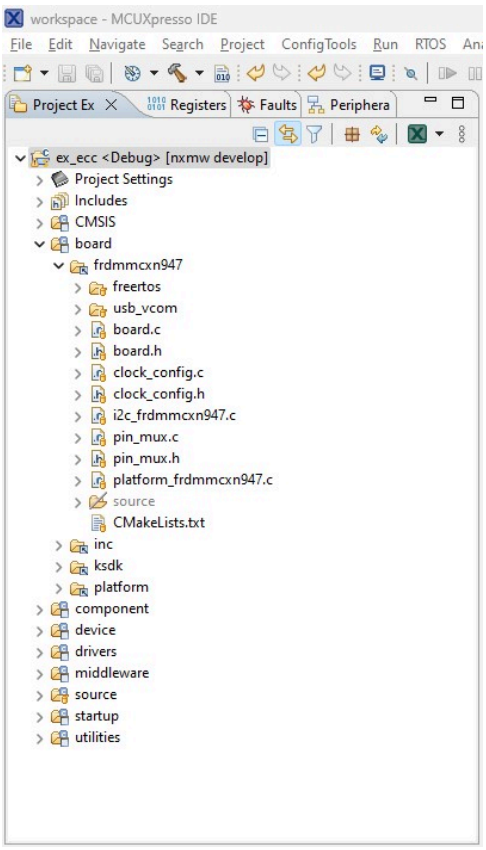


- Select the project name in the dialog box and click on **Finish** to import the project.



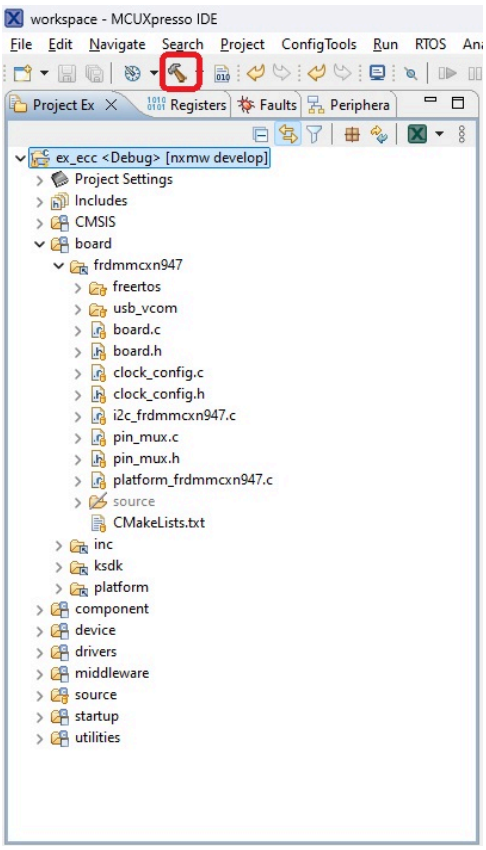
Note: It is not possible to import two projects with the same name in IDE. If another project of same name is to be imported, project suffix needs to be added.

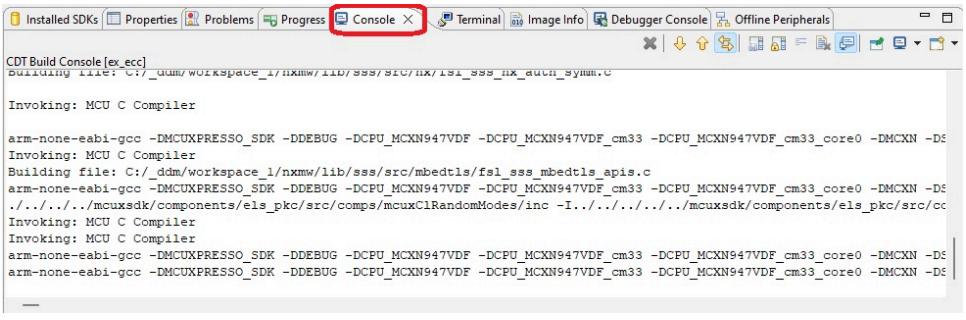
- The project can be viewed in the side panel under **Project Explorer** section. All the files linked to this project can be viewed here.



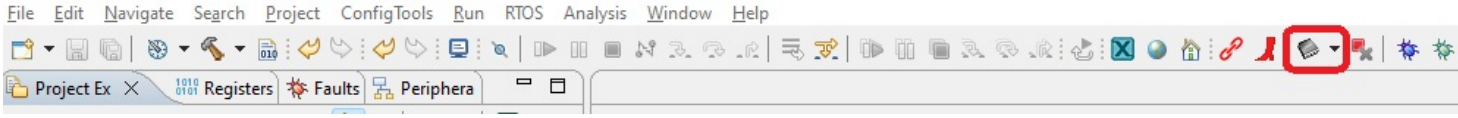
6.3.4 Build and Flash The Project

- To build the project, select the project in **Project Explorer** and click on the **Build** icon. The build logs will show up in the console below.

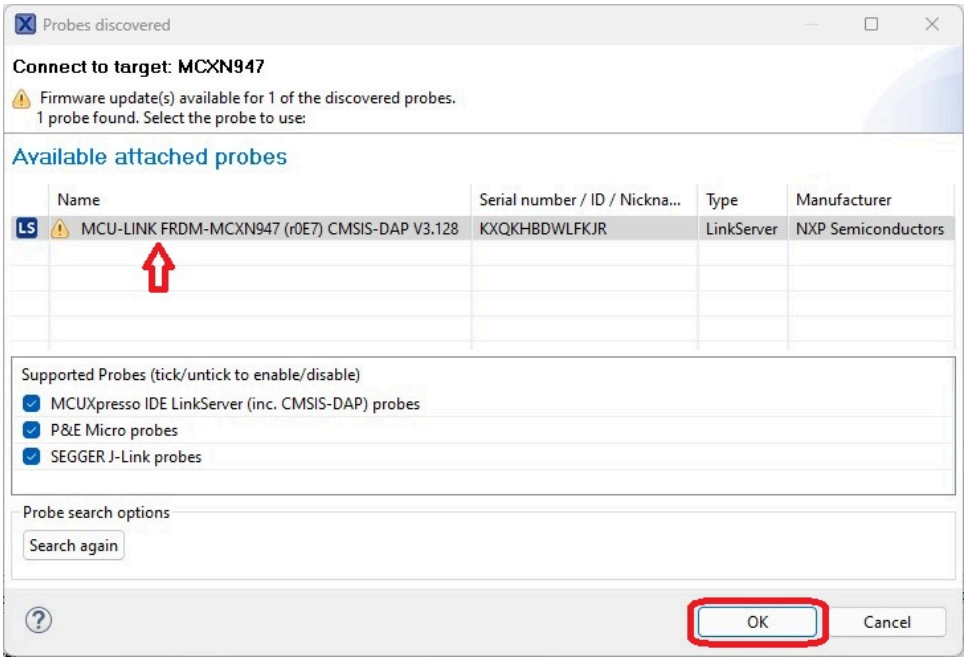




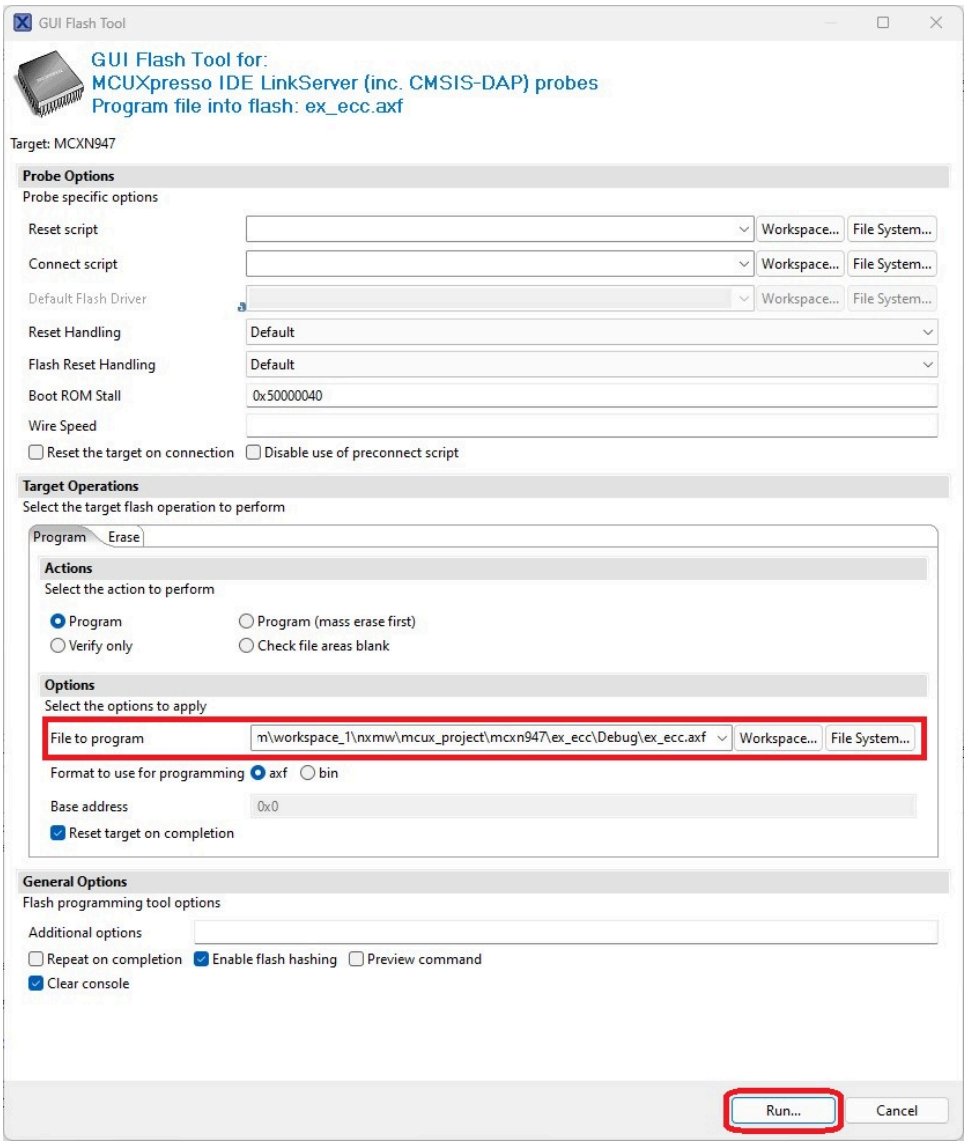
- Once the project is built successfully, the generated binary can be flashed using the GUI Flash Tool. Ensure that the development board is connected to the system and SA is properly connected to the board (refer to the end of this section for board setup). Select the project and click on the GUI Flash Tool icon as shown below.



- In the dialog box, select the corresponding probe (only one probe will show if one board is connected) and click on OK.

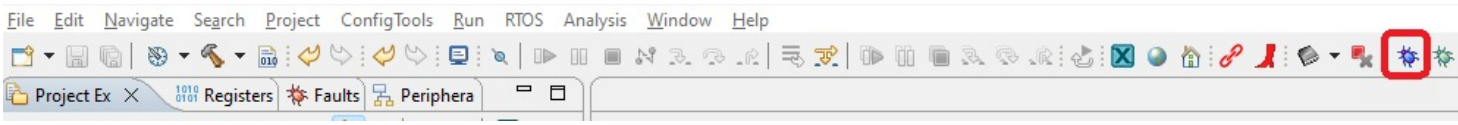


- In the next dialog box, in file to program, by default the latest generated binary (in `.axf` format) will be selected. Click **Run...** to flash the binary.



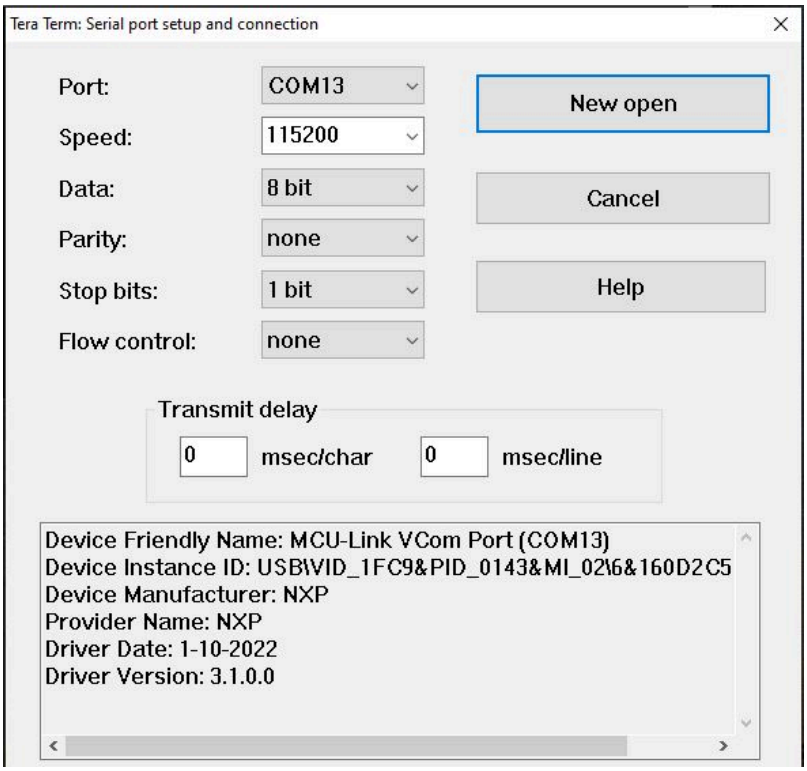
6.3.5 Debugging The Project

- To start debugging the project, select the project in Project Explorer and press the debug icon as shown below.

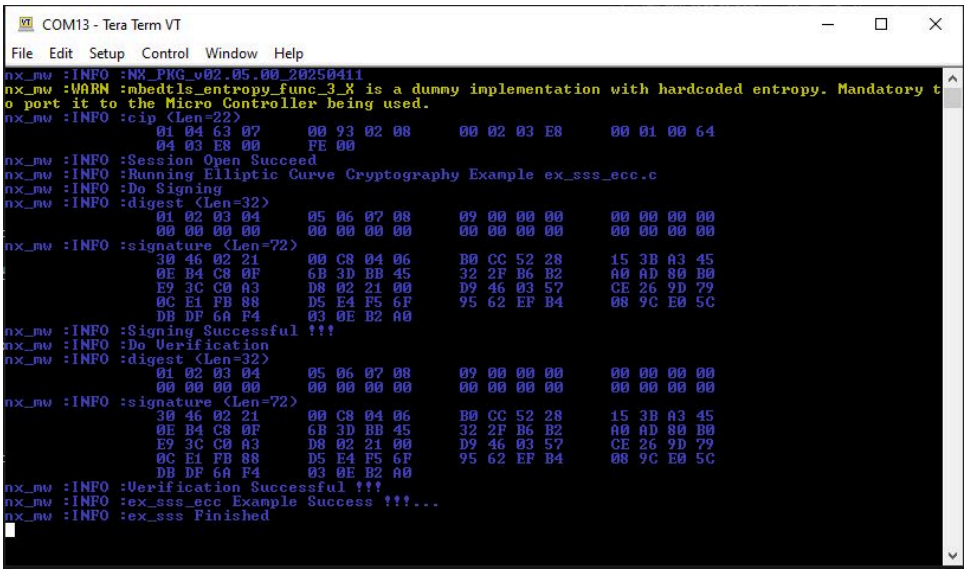


6.3.6 Running The Project

- Open Tera Term and connect serial port.



- Once the program execution begins, logs are printed on the terminal(Tera Term) indicating the status of execution.



6.4 MCUX Projects for NX SA

These projects demonstrate usage of MCUXpresso-IDE to build and run the demos. Projects for following boards are provided for reference:

- LPC55S69
- FRDM-MCXA153
- FRDM-MCXN947

This section will give a brief about the MCUX projects.

6.4.1 About The MCUX Projects

Demos for the supported boards are provided in respective MCUX folders. Every project contains MCUXpresso supported projects which can be used to build and debug the demos. The project folder contains the feature file `fsl_sss_ftr.h` with the necessary configurations for building the project.

> **Note:** All files you see in the projects are links to the middleware files. They are not copies. Only the fsl_sss_ftr.h file is different for different projects.
>
> **Note:** All projects are linked to particular versions 3rd party softwares. Most demos are linked to mbedtls 2. To use mbedtls 3, you need to remove mbedtls 2 from the sources and add mbedtls 3. Only then the macros present in `fsl_sss_ftr.h` can be updated.

For more information on the configurations (Refer: **SA Configs**).

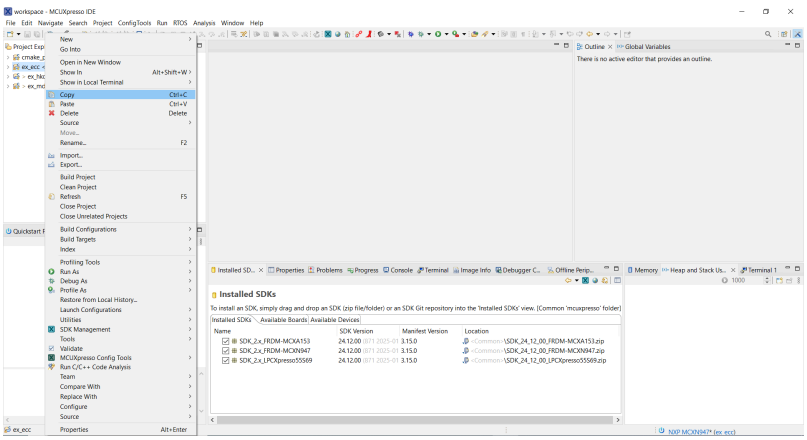
For building and running the projects, refer **Getting Started on MCUs Using Standalone MCUXpresso Projects**.

For details of Memory Configurations of MCXA153, refer **Memory Configs for MCXA153**.

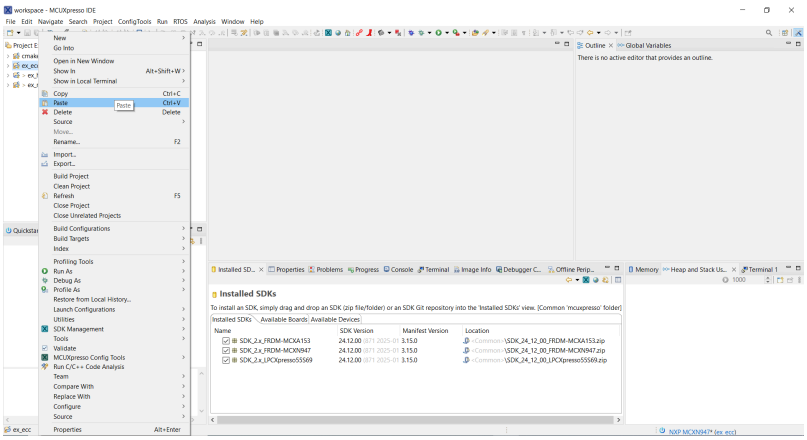
6.4.2 Creating New Standalone Projects

The Standalone projects structure allows you to create copies of existing projects which can then be modified according to any application.

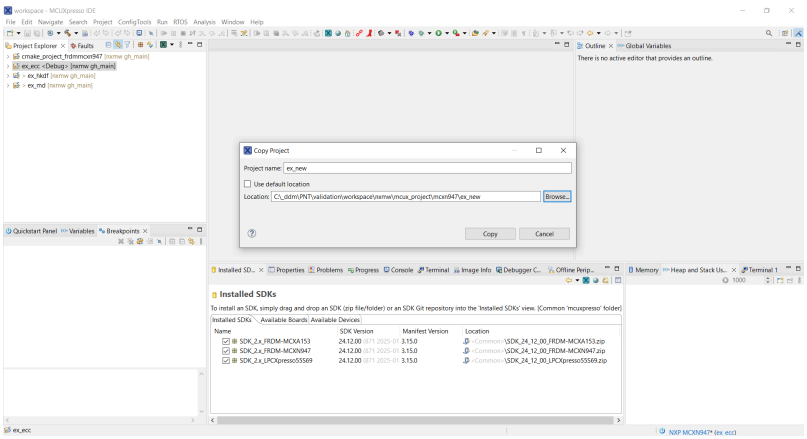
- Import a standalone project in MCUXpresso. Right click on the project and select Copy.



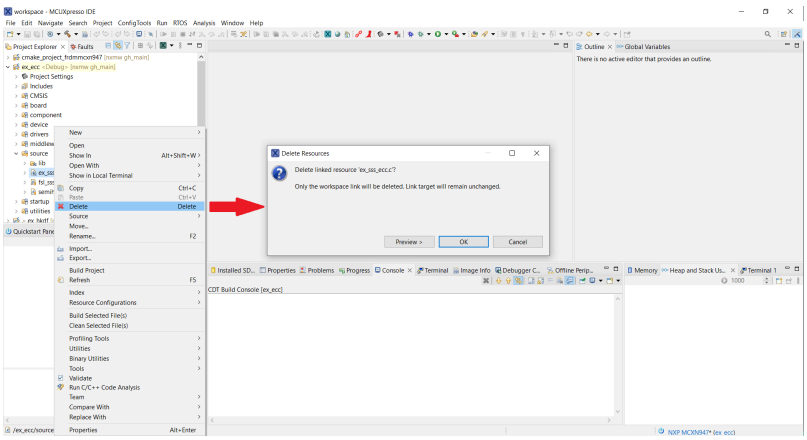
- Right click in the empty space and select Paste. You will see a copy project dialogue box.



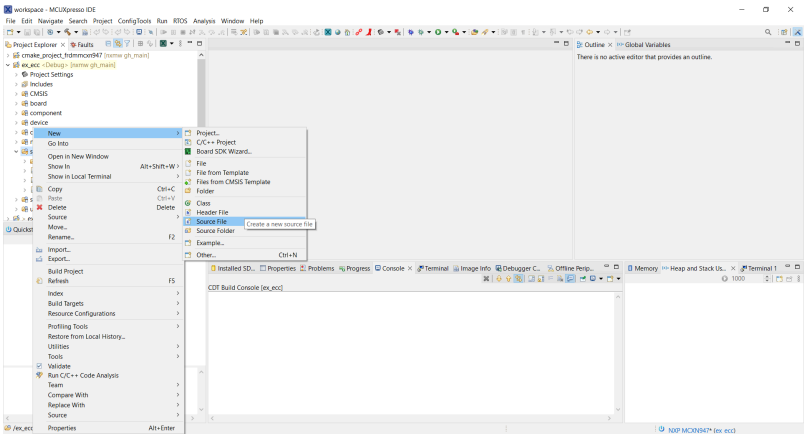
- Enter a new project name. Ensure that Default Location is not selected. In location, create a new project folder (preferably with same name as project) and and select that location.



- You will now see the new project in project explorer.
- From the new project, delete the existing demo file.



- Right click on the source folder and add new source files.



For more information on writing own applications Refer: **Write Own Applications**.

6.5 MCUX Projects for NX SA based on MCXA-153

6.5.1 Memory Configs MCUX Projects for MCXA153

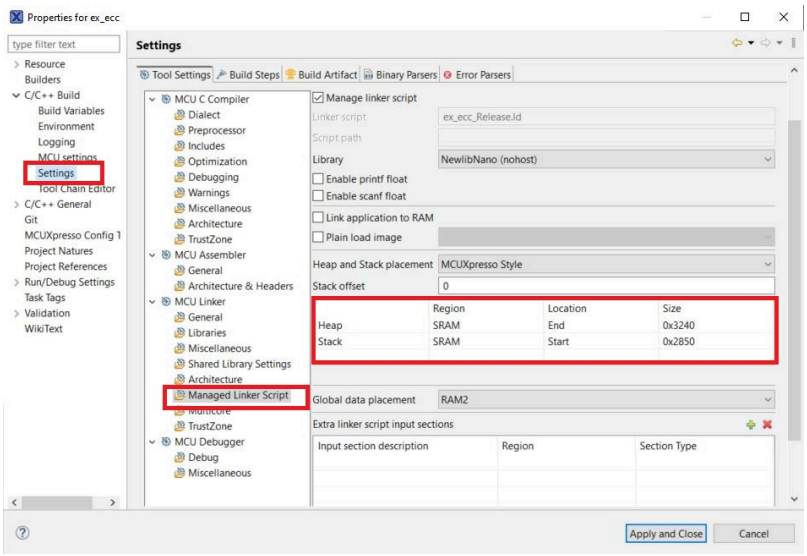
FRDM-MCXA153 is lower power MCU based on Arm® Cortex® M33

It has up to 128KB flash with up to 32 kB RAM with 8kB ECC

Due to its low memory capacity, Memory Configuration may need an update in Memory Configuration of MCUXpresso.

To update the memory configurations,

1. Goto Project Properties->Settings->Managed Linker Scripts
2. Update the Heap and Stack Size



```
Recommended Stack and Heap Sizes
Symmetric Authentication
Heap  - 0x3000
Stack - 0x2800

Sigma Authentication
Heap  - 0x3240
Stack - 0x2850
```

7 Porting Guide

7.1 New Platform Support Using Cmake Build

This sections explains the process of adding new platform support using cmake build system. The root of the middleware contains the **boards** folder and the new platform files need to be added in here,

```
nxmw
   |
   boards
       |
       ├── frdmmcxa153 ----- FRDM MCXA 153 specific files.
       |
       ├── frdmmcxn947 ----- FRDM MCXN 947 specific files.
       |
       ├── generic ------ Wrappers for timer implementation.
       |
       ├── inc
       |
       ├── ksdk --------- Common functions of KSDK.
       |
       ├── linux -------- Linux specific files.
       |
```

The following steps describe the process to add new micro-controller support.

Note: We will consider lpc55s69 as a new controller.

7.1.1 Add new host platform in cmake build options

Add an entry for imx-rt in the **cmake_options.py** file as shown

```
*** <ROOT-DIR>\nxmw\scripts\cmake_options.py ***

  LIST_HOST = [
    ("PCWindows", "PC/Laptop Windows", True),
    ("PCLinux64", "PC/Laptop Linux64", True),
    ("lpcxpresso55s", "Embedded LPCXpresso55s", True),
    ("Raspbian", "Embedded Linux on RaspBerry PI", True),
    ("frdmmcxa153", "Embedded frdmmcxa153", True),
    ("frdmmcxn947", "Embedded frdmmcxn947", True),
]
```

Run python cmake_options.py from the same location. This will update the cmake options with new host platform option ::

```
cd nxmw\scripts
python cmake_options.py
```

Rebuild the project (if already created one) as

```
  cd nxmw_build/nxmw-eclipse_arm
  cmake .
```

This will update the feature file with new platform macro -

```
/** Embedded LPCXpresso55s */
#define SSS_HAVE_HOST_LPCXPRESSO55S 0
```

Also do a manual update to :file:`nxmw/scripts/cmake_options.cmake` file to add an entry for new micro-controller. Example -

```
ELSEIF(SSS_HAVE_HOST_LPCXPRESSO55S)
  SET(SSS_HAVE_KSDK ON)
  SET(KSDK_BoardName "lpcxpresso55s69")
  SET(KSDK_CPUName "LPC55S69")
```

A new cmake file defining the linker options of new platform needs to be added. Example - ksdk_lpcxpresso55s.cmake.

Include the same in :file:`nxmw/scripts/ksdk.cmake`

7.1.2 Updating boards folder

Add new platform folder in boards directory (say lpcxpresso55s69). This should match the board name added in cmake_options.cmake file. Copy / implement the necessary files here. (Refer existing frdmmcxn947 folder).

Refer :file:`nxmw/boards/frdmmcxn947/CMakeLists.txt` and create a new cmake file - :file:`nxmw/boards/lpcxpresso55s69/CMakeLists.txt` to include the relevant files of new platform from the mcu-sdk and boards dir. Ensure that I2C driver is implemented properly. (Refer existing frdmmcxn947 folder for required files and implementation).

For more information on existing projects using cmake build refer: **CMake Projects for NX SA**.

7.2 New Platform Support Using MCUXpresso Project

This sections explains the process of adding new platform support code in NX Middleware. The following steps will describe the file / folders where changes would be required to add support to a new platform. (say lpc55s),

7.2.1 Updating boards folder

Add new platform folder in boards directory. This should match the board name added in cmake_options.cmake file. Copy/implement the necessary files here. Ensure that I2C driver is implemented properly. (Refer existing frdmmcxn947 folder for required files and implementation).

Refer :file:`nxmw/boards/frdmmcxn947/CMakeLists.txt` and create a new cmake file - :file:`nxmw/boards/<new-board>/CMakeLists.txt` to include the relevant files of new platform from the mcu-sdk and boards dir.

7.2.2 Create MCUXpresso Project

Use existing frdmmcxn947 project or any other platform as a reference and create a new project for new platform.

Remove the board specific file of old platform and add relevant files of new platform. To do do, simply delete the files which are not required and drag and drop the required files in the project explorer. Ensure that the mcu-sdk files of old platforms is replaced with new platform. Also ensure that all the indlude directories and preprocessor macros are updated in the project properties.

For more information of existing MCUX projects refer: **MCUX Projects for NX SA**.

7.3 Porting to New Host Crypto

NX middleware by default provides support for mbedTLS and OpenSSL host crypto via SSS APIs. The host crypto is used during session open phase and for any other crypto operations required later. The section describes the steps required for porting middleware to new host crypto.

7.3.1 OPTION-1

If the new host crypto is **only** required for session open phase, re-implement the functions in :file:`nxmw/lib/sss/src/nx/fsl_sss_nx_auth_host.c` file for new crypto. (The file is currently ported to OpenSSL and mbedTLS crypto).

7.3.2 OPTION-2

A new host crypto can be added by re-implementing the SSS APIs and updating the cmake options.

Follow the below steps -

Add an entry for new host crypto in the **cmake_options.py** file as shown

```
*** <ROOT-DIR>\nxmw\scripts\cmake_options.py ***

  LIST_HOSTCRYPTO = [
    ("MBEDTLS", "Use mbedTLS as host crypto", True),
    ("OPENSSL", "Use OpenSSL as host crypto", True),

    < NEW ENTRY>

    ("None",
     ("NO Host Crypto",
      "Note,  the security of configuring Nx to be used without HostCrypto",
      "needs to be assessed from system security point of view"

      ), True),
]
```

Run python cmake_options.py from the same location ::

```
cd nxmw/scripts
python cmake_options.py
```

Other files in the **\nxmw\scripts folder and feature file :file:`fsl_sss_ftr.h` are updated by this new option.**

Rebuild the project and the build folder's CMake would now reflect this newly added CMake variable. Example -

```
cd nxmw_build/nxmw-eclipse_arm
cmake .
```

Create new sss host crypto module in `sss` folder - :file:`nxmw/lib/sss/src/<NEW_CRYPTO>/<new_crypto>.c`

Include the same in cmake file - :file:`nxmw/lib/sss/CMakeLists.txt`

```
  PROJECT(SSS_APIs LANGUAGES C)

  FILE(
      GLOB
      API_FILES
      inc/*.h
      inc/*.h.in
      src/*.c
      src/nx/*.c
      src/mbedtls/*.c
      src/openssl/*.c
      *<NEW_CRYPTO>/<new_crypto>.c*
      src/keystore/*.c
      port/default/*.h
  )
```

Refer - :file:`nxmw/lib/sss/src/mbedtls/fsl_sss_mbedtls_apis.c` or :file:`nxmw/lib/sss/src/openssl/fsl_sss_openssl_apis.c`.

8 Revision history

Document ID	Release date	Description
AN14123 v2.5.0	09 May 2025	Corresponds to NX middleware version v02_05_00. See section “1.2 Changelog” for further details
AN14123 v1.0.0	15 jan 2025	Corresponds to NX middleware version v02_04_00. See section “1.2 Changelog” for further details

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.

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.

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.

Contents