AN14861

Secure OEM Key Provisioning with EdgeLock Secure Enclave on RT1180

Rev. 1.0 — 13 November 2025 Application note

Document information

Information	Content
Keywords	AN14861, EdgeLock Secure Enclave, HSM, SPSDK, RT1180, key provisioning
Abstract	This application note describes how to provision the OEM key using EdgeLock Secure Enclave HSM services.

1 Introduction

In secure IoT devices, OEM cryptographic keys enable encryption, decryption, and secure communication between the MCU and external systems such as the cloud or other devices. Protecting cryptographic keys during manufacturing and provisioning is critically important.

EdgeLock Secure Enclave (ELE) of NXP integrates a dedicated security subsystem that protects and isolates security functions critical to device operation. It provides tamper-resistant key storage, cryptographic services, and lifecycle management features that enable secure key provisioning from manufacturing to deployment.

This application note outlines a secure approach for provisioning an OEM symmetric key on the RT1180 platform using ELE Hardware Security Module (HSM) services and the Secure Provisioning SDK (SPSDK) tool during the production phase. The same method also supports importing a private key of an asymmetric keypair. By using the hardware-based security features of ELE, OEMs ensure that sensitive key material remains protected during import to storage and uses it safely.

2 OEM key provisioning flow

The OEM key provisioning process follows the below steps:

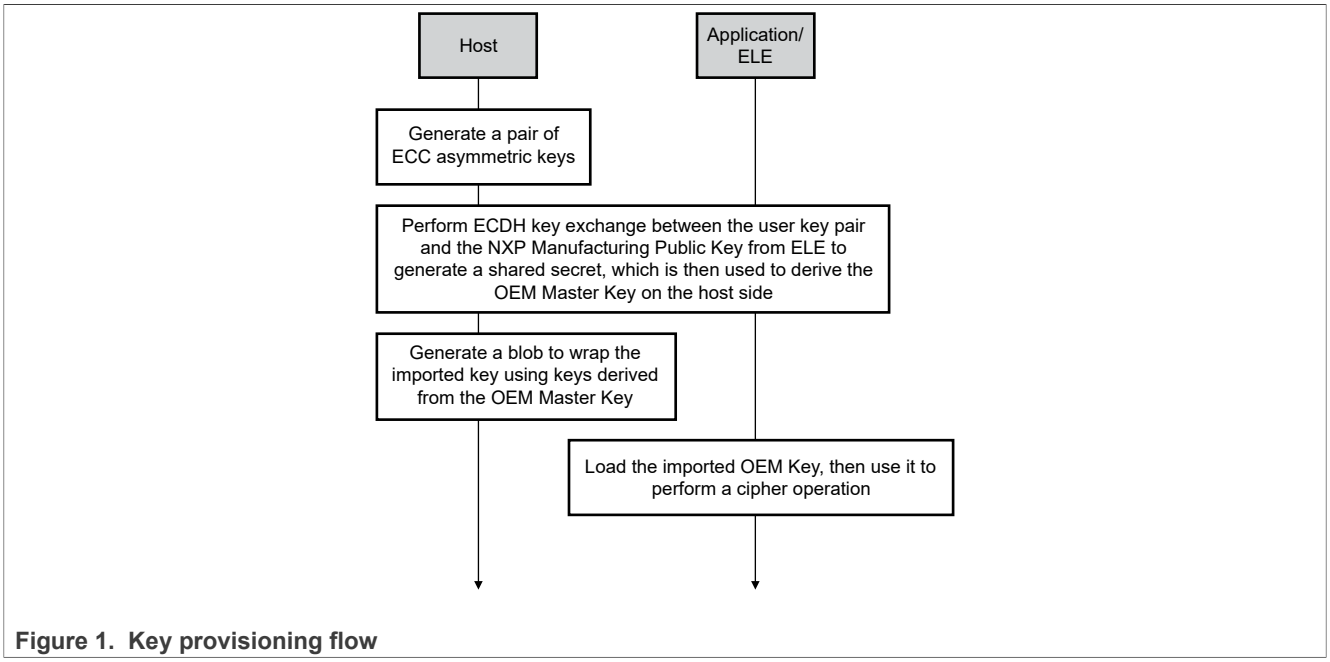

Figure 1. Key provisioning flow

1. Generate the OEM master key (OEM_IMPORT_MK_SK). To import a key into ELE key storage, execute the ELE *key exchange* command first. This command generates a shared key, which is then used to derive OEM_IMPORT_MK_SK.
2. Wrap the user-imported key and sign its key attributes. The keys OEM_IMPORT_WRAP_SK and OEM_IMPORT_CMAC_SK, both derived from OEM_IMPORT_MK_SK, are required for OEM key encryption and signature generation.
3. After the wrapped OEM key (key blob) is generated, import it into ELE using the *key import* command.
4. Use the imported key with an ELE cryptographic API for operations such as encryption in the user application.

3 EdgeLock Secure Enclave (ELE)

To protect and isolate security functions critical to device operation, NXP equips many of its latest MCUs, MPUs, and crossover processors with a dedicated security unit, called ELE.

Physically isolated from the rest of the SoC, the ELE has its own CPU core and memory. The ELE is not customer programmable. To use the ELE provided services, the application must write a command sequence to Messaging Units (MUs) to communicate with ELE.

The ELE cannot implement all APIs described in a single firmware because of the RAM constraint.

Following are the two different firmware binaries used in this document:

- ELE OEM provisioning firmware: Designed to exchange a shared secret with the OEM and import assets using this shared key.
- ELE runtime firmware: Designed for use at runtime after all assets are injected into ELE using the other ELE firmware. This firmware uses the injected keys or generates new keys.

The ELE firmware is located in the *<SDK_FOLDER>\firmware\edgelock\S400* directory.

The subsections that follow provide essential information about the *key exchange* and *key import* commands.

For more detail, refer to the *EdgeLock Secure Enclave i.MX RT1180 B0/C0 User Guide* (document UG10192).

4 Key attributes

Key attributes are a set of information and policies that describes a key. These attributes are set when a key is created through generation, derivation, or import. Key attributes are fixed and cannot be changed during the key lifespan. Globally, key attributes restrict the key to a specific cryptographic operation.

4.1 Key identifier

The key identifier is a 32-bit integer value that acts as a name for keys in ELE.

Table 1. Key identifier ranges

Range	Description
0x00000001 – 0x3FFFFFFF	Persistent and permanent keys that are stored in external memory
0x40000000 – 0x6FFFFFFF	Volatile keys that are kept in RAM
0x70000000 – 0x7FFFFFFF	OEM and NXP Fab injected and provisioned keys

4.2 Key type

This attribute defines the key type, referring to symmetric keys or asymmetric key pairs. It is an unsigned 16-bit integer.

4.3 Key size

This attribute defines the key security size in bits. It is an unsigned 16-bit integer.

4.4 Key lifetime

This attribute consists of two indicators:

- A persistence level indicator (8 bits), which indicates the actions that can cause the deletion of the key.

- A location indicator (24 bits).

Persistence level indicator values can be:

- Volatile (0x00)
- Persistent (0x01)
- Permanent (0xFF)

To create a key using the import key command with the ELE option, the key location must be set to 0xC00200.

4.5 Key lifecycle

This attribute defines the device lifecycle in which the key is usable. It is an unsigned 32-bit integer.

Table 2. Key lifecycle

Lifecycle	Value	Description
OPEN	0x01	Object is usable in open device lifecycle
CLOSED	0x02	Object is usable in closed device lifecycle
CLOSED and LOCKED	0x04	Object is usable in closed and locked lifecycle

4.6 Key usage

This attribute defines the key usage as described in this document. This document demonstrates AES ECB operation. For a complete list of supported key usages, refer to the *EdgeLock Secure Enclave i.MX RT1180 B0/C0 User Guide* (document UG10192).

Table 3. Key usage

Usage	Value	Description
Encrypt	0x00000100	Permission to encrypt messages (cipher, AEAD, or asymmetric encryption) with the key.
Decrypt	0x00000200	Permission to decrypt messages (cipher, AEAD, or asymmetric decryption) with the key.
Derive	0x00004000	Permission to derive other keys from this key.

4.7 Key permitted algorithm

This attribute defines a policy that restricts the key to a single algorithm or all algorithms supported by a cryptographic operation. It is an unsigned 32-bit integer.

This document demonstrates the AES ECB cipher algorithm. For other supported cryptographic algorithms, refer to the *EdgeLock Secure Enclave i.MX RT1180 B0/C0 User Guide* (document UG10192).

Table 4. Key permitted algorithm

Algorithm	Value
ECB NO PADDING	0x04404400
CBC NO PADDING	0x04404000
CTR	0x04C01000
CFB	0x04C01100
OFB	0x04C01200
All cipher	0x84C0FF00

5 ELE Hardware Security Module (HSM) usages

ELE HSM services provide secure key storage and cryptographic operations. The ELE firmware implements these services and provides access through dedicated commands.

This section focuses on the *key exchange* and *key import* commands used during the key import procedure.

For more details on ELE HSM services, refer to the *EdgeLock Secure Enclave i.MX RT1180 B0/C0 User Guide* (document UG10192).

5.1 Service flow

To use ELE HSM key import service, the requester initiates a service flow by invoking the corresponding API. This section describes the required functions and commands to import a key into the key storage and use it in the cryptographic operations.

To ensure proper initialization, key exchange, and secure key import, follow the steps below:

1. Initialize ELE HSM services using the function *ELE_InitServices*.
2. Register read/write NVM functions with *ELE_Register_NVM_Manager*.
3. Open a session for ELE services using *ELE_OpenSession*.
4. Create a keystore using *ELE_CreateKeystore* before creating a key.
5. To store keys in NVM using *ELE_OpenNvmStorageService*, open the storage service.
6. Perform a combined key agreement and key derivation operation using ELE_ExchangeKey. This process derives shared secrets required for key operations.
7. Import the wrapped and signed user key (in TLV format) by calling *ELE_ImportKey_HSM*. In this demo, ELE triggers the application to store the encrypted keys and load them when needed.
8. Close the session using *ELE_CloseSession* after completing all operations. This also closes all opened services.

5.2 Key exchange command

The *key exchange* command performs a combined key agreement and derives a shared OEM master key (OEM_IMPORT_MK_SK).

To ensure trust in some input parameters, such as the peer public key and derived key attributes, the input content must be OEM-signed. The signing key is the root key, which is used to sign the image.

In the attached demo project, the following is the reference API for the *key exchange* command:

```
status_t ELE_ExchangeKey(S3MU_Type *mu,
uint32_t keyHandleID,
uint16_t flags,
const uint8_t *signed_content,
uint32_t signed_content_size,
const uint8_t *user_fixed_info,
uint32_t user_fixed_info_size,
const uint8_t *peer_pub_key,
uint32_t key_size,
uint32_t *derived_key_ID);
```

The following Table 5 shows the parameters for *key exchange* API:

Table 5. ELE_ExchangeKey function Parameters

Parameters	Description
mu	Peripheral ELE MU_RT_S3MUA base address
keyHandleID	Pointer to the unique key management handle ID returned by the ELE_OpenKeyService function
flags	Bit 0: 0 indicates that the input is a signed content. Bit 5: Monotonic counter increment used for storing rollback protection data. Bit 7: SYNC operation (the request is completed only when the new key has been written in NVM, external memory).
signed_content	The least significant byte (LSB) of the address where the signed input content is located
signed_content_size	Input content size in bytes
user_fixed_info	LSB of the address in the requester space where the input user-fixed information can be found; set to NULL if no fixed information is used in the key derivation process Fixed information can be used in the key derivation process
user_fixed_info_size	Input user fixed info size in bytes. Set to 0 if the input user fixed information LSB address field is set to NULL
peer_pub_key	ECC public key on the user side
key_size	ECC public key size
derived_key_ID	The desired key ID for the derived key; set to 0x0 to allow the ELE firmware to select the key identifier

The following Table 6 shows the fields in the signed content payload for *key exchange* command:

Table 6. Key exchange signed content payload

Field	Size (in bits)	Description
TAG	8	Must set to 0x47
Version	8	Must set to 0x07
Reserved	16	Reserved bits.
Key store ID	32	Key store ID where to store the derived key; it must be the key store ID related to the key management handle set in the command API
Key exchange algorithm	32	Key exchange algorithm; in this demo, use ECDH HKDF SHA256 KEY IMPORT (0x09020109)
Derived key group	16	Indicates the derived key group ID. 100 groups are available per key store. It must be a value in the range [0; 99]
Salt flags	16	Bit 0: Salt in key derivation process • 0: Use zeros salt • 1: Use peer public key digest as salt Bit 1: If there is an OEM master key generation for *key import* purpose, salt is used to derive OEM_IMPORT_WRAP_SK and OEM_IMPORT_CMAC_SK: • 0: Zeros string • 1: Device SRKH
Derived key type	16	Set to OEM_IMPORT_MK_SK (0x9200) in this document
Derived key security size bits	16	Must be 256 bits for OEM_IMPORT_MK_SK
Derived key lifetime	32	Derived key lifetime attribute
Derived key usage	32	Derived key usage attribute

Table 6. Key exchange signed content payload*...continued*

Field	Size (in bits)	Description
Derived key permitted algorithm	32	Derived key permitted algorithm attribute
Derived key lifecycle	32	Derived key lifecycle attribute
Derived key ID	32	The desired key identifier for the derived key; set to 0x0 to allow the ELE firmware to select the key identifier
Private key ID	32	ECC private key ID on the ELE side
Input peer public key digest word #0	32	First 32-bit word (in little-endian format) of the digest buffer containing the ECC public key from the user side; the digest algorithm must be SHA256
…	32	Intermediate 32-bit words (little-endian) of the input peer public key digest buffer; the digest algorithm must be SHA256
Input peer public key digest word #7	32	Last 32-bit word (little-endian) of the input peer public key digest buffer; the digest algorithm must be SHA256
Input user fixed info digest word #0	32	• First 32-bit word (little-endian) of the input user fixed information digest buffer; the digest algorithm must be SHA256 • User fixed information is used to derive OEM_IMPORT_MK_SK; set to NULL if the input user-fixed information is not used
…	32	Intermediate 32-bit words (little-endian) of the input user-fixed information digest buffer; set to NULL if the input user-fixed information is not used
Input user fixed info digest word #7	32	Last 32-bit word (little-endian) of the input user-fixed information digest buffer; the digest algorithm must be SHA256 Set to NULL if the input user-fixed information is not used

5.3 Key import command

The *key import* command imports either a symmetric key or the private key of an asymmetric key pair into the ELE key storage. When importing an OEM key, the input must be a Type-Length-Value (TLV) blob, formatted as shown in Table 8.

The keys used to encrypt the OEM key and sign the TLV blob must be OEM-derived keys, derived from OEM_IMPORT_MK_SK. This master key is generated through the *key exchange* command during the OEM provisioning process.

In the attached demo project, the following is the reference API for the Key Import command:

```
status_t ELE_ImportKey_HSM(S3MU_Type *mu,
uint32_t keyHandleID,
const uint8_t *input,
uint32_t inputSize,
bool keyGroupAuto,
uint16_t keyGroupID,
ele_import_key_option_t option,
bool sync,
bool monotonic,
uint32_t *keyID);
```

The following Table 7 shows the parameters for *key import* API:

Table 7. ELE_KeyImport_HSM function parameters

Parameters	Description
mu	Peripheral ELE MU_RT_S3MUA base address
keyHandleID	A pointer to a unique key management handle ID returned by the ELE_OpenKeyService function
input	Pointer to the TLV blob containing the key to be imported
inputSize	Size of the TLV blob
keyGroupAuto	True: ELE automatically chooses a key group False: The user assigns a value to the key group field
option	0: ELE option
sync	Flag for SYNC operation; when a SYNC operation is invoked, the keys contained in the specified key group are written to NVM
monotonic	Flag for monotonic counter increment and set to protect against storage rollback
key_ID	Output parameter that returns the key ID of the imported key

The following Table 8 shows the fields in the Key Import TLV blob with their tags, lengths, and descriptions:

Table 8. Key import ELE option TLV blob

Field	TAG	Length (bytes)	Description/Value
Magic	0x40	11	For OEM imported key, must be the hexadecimal string 'edgelockenclaveimport' [65 64 67 65 6c 6f 63 6b 65 6e 63 6c 61 76 65 69 6d 70 6f 72 74]
Key ID	0x41	4	Key identifier
Key properties			**Description/Value**
Algorithm	0x42	4	Permitted algorithm for imported key
Usage	0x43	4	Key usage
Type	0x44	2	Type of key
Bits	0x45	4	Key security size in bits
Lifetime	0x46	4	Lifetime
Key lifecycle	0x47	4	Key lifecycle flags
OEM_IMPORT_MK_SK key ID	0x50	4	OEM_IMPORT_MK_SK key identifier
Wrapping algorithm	0x51	4	Wrapping algorithm of the key blob. Possible values are: • 0x01: RFC 3394 wrapping • 0x02: AES-CBC wrapping
IV	0x52	16	IV for CBC wrapping; not used if the wrapping algorithm is not equal 0x02
Signing algorithm	0x54	4	The algorithm is used to sign the blob itself; the result is set to the field Signature of this blob. It must be 0x01 for the CMAC algorithm
Wrapped private key	0x55	Variable	Private key data in encrypted format as defined by the Wrapping Algorithm
Signature	0x5E	16	Signature of all previous fields of this blob, including the signature tag (0x5E) and signature length fields

6 Secure Provisioning SDK (SPSDK)

The SPSDK is a unified, reliable, and easy to use Python SDK library designed to work across the NXP MCU portfolio. It provides a strong foundation for everything from quick customer prototyping up to production deployment. The SPSDK enables you to connect and communicate with the device, configure it, prepare, download, and upload data, and perform security operations.

Following are the SPSDK command-line tools used in this document:

- nxpimage: Generates signed content for the *key exchange* command and creates a TLV blob for the *key import* command. It can also convert a binary file into C array string.
- nxpcrypto: Generates ECC key pairs on user side.

The complete SPSDK installation guide is available here. For more information on installing SPSDK directly from the GitHub repository or on other operational systems, refer to the SPSDK Installation Guide page.

The following steps describe how to install SPSDK on Windows systems. Make sure Python 3.9 or later is installed, then open the Windows command prompt (`cmd.exe`) and run the commands mentioned below:

1. Create a virtual environment by running python -m venv venv.
2. Activate the virtual environment using venv/Scripts/activate.
3. Upgrade pip by running python -m pip install --upgrade pip.
4. Install SPSDK with pip install spsdk.

7 How to import an OEM key with ELE HSM

This section describes how an OEM can import a key into the ELE key storage.

In this document, the key used is 'NXP PROD MANUFACT KEY AGREEMENT'. This key acts as the private key on the ELE side specifically for key import purposes. The key ID *0x70000000* identifies this key.

This key allows OEMs to derive the same OEM_IMPORT_MK_SK and its associated derived keys for all products that have the same Super Root Key Hash (SRKH) fused into the device.

The attached example project, developed using MCUXpresso IDE, demonstrates the implementation of the *key exchange* command, *key import* command, and the complete *OEM key* import procedure.

To adapt the project for the specific use case, you must update the parameters used in both the *key exchange* and *key import* commands accordingly.

7.1 Prerequisites for OEM key import using ELE HSM

The following are the prerequisites for importing the OEM key using the ELE HSM demo project:

- i.MX RT1180 board with the SRKH already fused
- NXP SPSDK version 3.3.0 or later installed
- SD card (optional)

  Note:

  – *For more details on generating and burning the SRKH, refer to the Secure Boot on i.MX RT118x (document AN14227).*

  – *When testing the ELE HSM service, it is not necessary to advance the lifecycle of the i.MX RT1180.*

7.2 Import demo project

To import the SDK example project (archived folder) into MCUXpresso IDE, follow the steps below:

1. Open **MCUXpresso IDE**.

2. In the **Quickstart** panel (at the bottom-left corner of the IDE), click **Import project(s) from file system**.

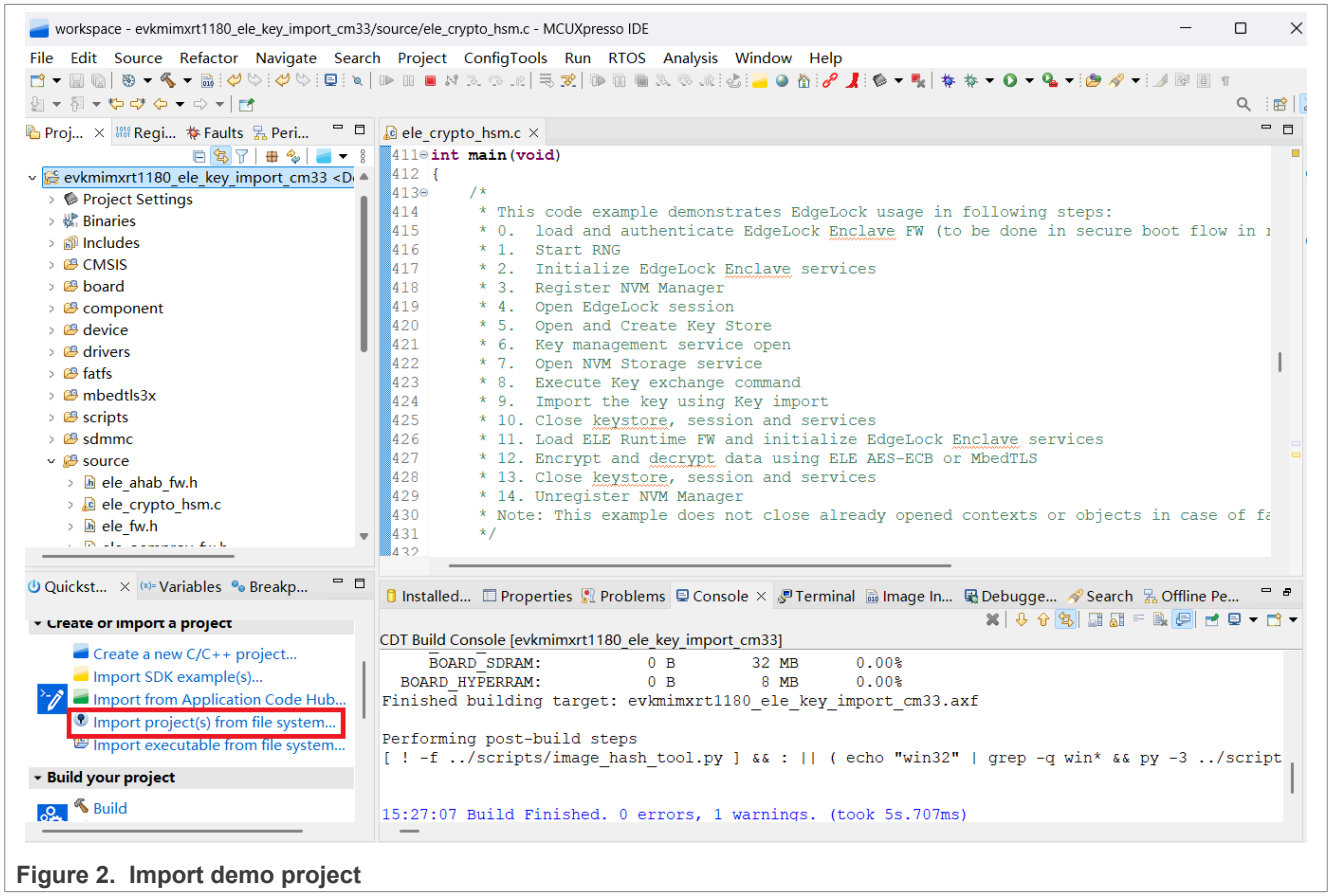

Figure 2. Import demo project

3. Click **Browse** next to the **Archive** option.

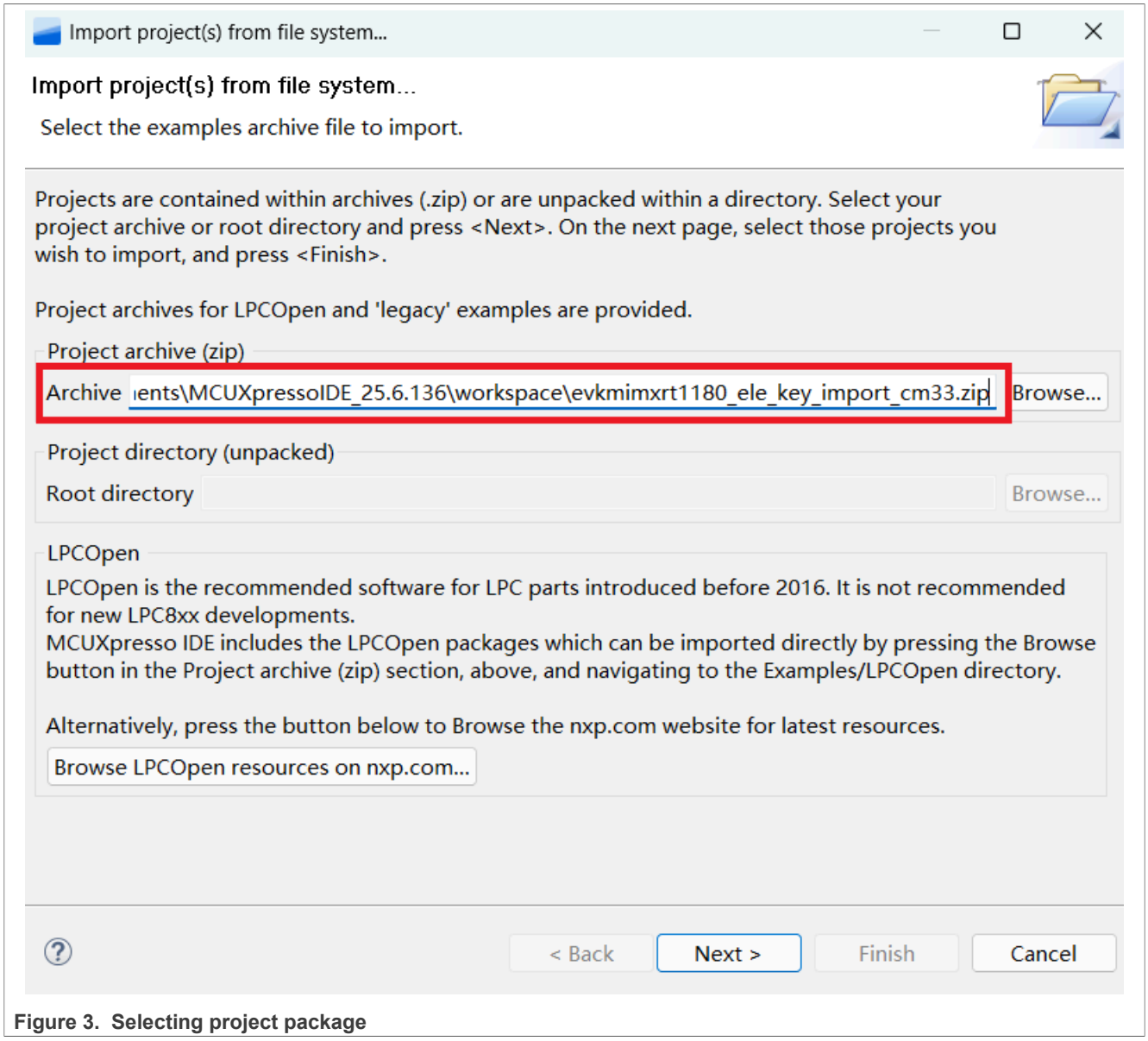

Figure 3. Selecting project package

4. Select the ZIP folder containing the SDK example project.
5. Click **Finish** to complete the import process.

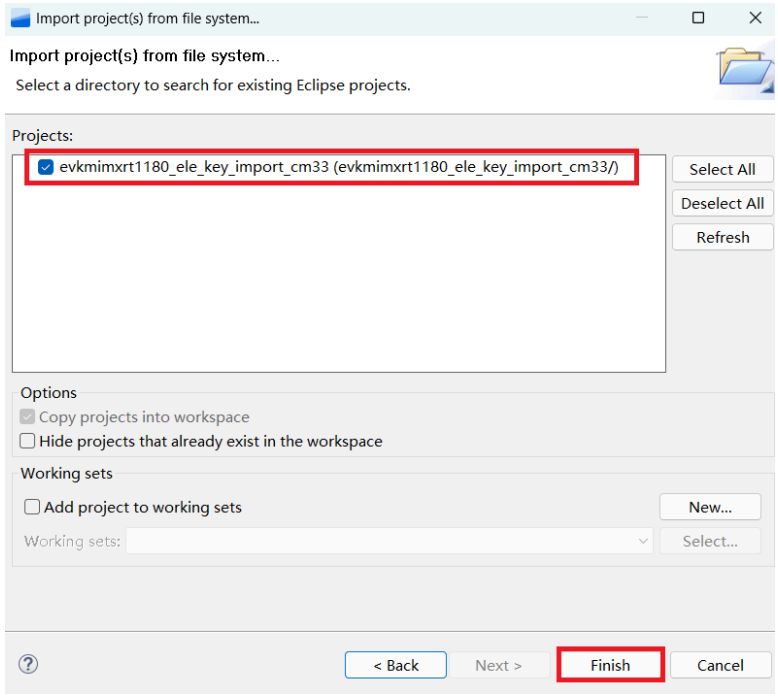

Figure 4. Selecting application project

7.3 Get NXP Manufacturing Public Key from ELE

An Elliptic Curve Cryptography (ECC) key pair (SECP R1/NIST P-256) is embedded in the ELE for key import operations. The NXP Manufacturing Public Key is used as a parameter in the Elliptic Curve Diffie-Hellman (ECDH) process to derive the OEM_IMPORT_MK_SK.

To retrieve the NXP Manufacturing Public Key, follow the steps below:

1. Call the ELE_GeneratePubKey function in the application. You can do this by either:
   - Running the attached demo to trace the public key, or
   - Invoking the ele_crypto_hsm demo at <SDK_PATH>\boards\evkmimxrt1180\ele_crypto
2. After retrieving the public key, save it as a binary file.
3. Convert the binary file to PEM format using the following SPSDK command:

```
nxpcrypto key convert -e PEM -i nxp_prod.bin -o nxp_prod.pub
```

7.4 Generate signed content and TLV blob

To generate signed content for the *key exchange* command and the wrapped OEM key (TLV blob) for the *key import* command on the host PC, follow the steps below:

1. To generate a local ECC key pair for ECDH key agreement with 'NXP PROD MANUFACT KEY' embedded in ELE, use SPSDK and run the following commands.

```
nxpcrypto key generate -k secp256r1 --force -o app_ecc256.pem
```

2. Convert the ECC public key generated in step 1 into C array format using the following commands:

```
nxpcrypto key convert -e RAW -i app_ecc256.pub -o ecc256_pub_key.bin
nxpimage utils convert bin2carr -i ecc256_pub_key.bin -e little -c 8 -n
 ecc256_pub_key -o app_ecc256_pub.c
```

3. Compute the SHA-256 digest of the raw data of public key generated in step 2.

```
nxpcrypto digest -h sha256 -i ecc256_pub_key.bin
```

4. Retrieve the template configuration file for the *key exchange* signed content using the following command:

```
nxpimage signed-msg get-template -f rt118x -m KEY_EXCHANGE_REQ -o
 signed_msg_config.yaml --force
```

5. Populate the generated template with the required configuration. For *key import* purpose, set the following fields as shown below:
   - derived_key_type: OEM_IMPORT_MK_SK
   - derived_key_size_bits: 256
   - derived_key_usage: Derive
   - derived_key_permit_algo: HKDF SHA256

   The user must set the user_fixed_info field only if *key exchange* API explicitly requires it. All the other fields must be configured based on your test environment. Below is an example configuration:

```
# = Signed message Configuration template for mimxrt1189, Revision: latest. =
# ====================
# == General Options ==
# ====================
# ===== The chip family name [Required] =====
family: mimxrt1189
# ===== MCU revision [Optional] =====
revision: latest
# ===== Output file name [Required] =====
output: signed_message.bin

# ================================
# == Settings of Signed Message ==
# ================================
# ===== Super Root Key (SRK) set [Required] =====
# Description: Defines which set is used to authenticate the signed message.
srk_set: oem
# ===== Used SRK [Required] =====
# Description: Which key from SRK set is being used.
used_srk_id: 0
# ===== SRK revoke mask [Required] =====
srk_revoke_mask: 0
# ===== Fuse version [Required] =====
fuse_version: 0
# ===== Software version [Required] =====
sw_version: 0
# ===== Signed Message container signer [Required] =====
signer: type=file;file_path=..\keys\SRK1_p256.pem

# ================================================
# == Configuration of Signed Message SRK table ==
# ================================================
# ===== SRK Table [Required] =====
# Description: SRK (Super Root key) table definition.
srk_table:
 # ===== CA Flag [Optional] =====
 flag_ca: false
 # ===== Hash Algorithm [Optional] =====
 hash_algorithm: default
 # ===== Super Root Key (SRK) table [Required] =====
 srk_array:
 - ..\keys\SRK1_p256.pub
```

```
 - ..\keys\SRK2_p256.pub
 - ..\keys\SRK3_p256.pub
 - ..\keys\SRK4_p256.pub

# ========================
# == Settings of Message ==
# ========================
message:
 # ===== Certificate version [Optional] =====
 cert_version: 0
 # ------------------------------------===== Certificate permission [Optional]
 ======================================
 # Description: Certificate permission, to be used in future. The stated
 permission must allow the operation requested
 # by the signed message.
 cert_permission: 0
 # ===== Issue date [Optional] =====
 issue_date: 2024-09
 # ===== Device UUID [Optional] =====
 uuid: '0000000000000000'
 command:
 # ===== Key exchange [Required] =====
 KEY_EXCHANGE_REQ:
 # ===== Key store ID [Required] =====
 # Description: Key store ID where to store the derived key. It must be the
 key store ID related to the key
 # management handle set in the command API
 key_store_id: 0x12345678
 # ===== Key exchange algorithm [Required] =====
 key_exchange_algorithm: HKDF SHA256
 # ===== Salt Flags [Required] =====
 salt_flags: 0
 # ===== Derived key group [Required] =====
 # Description: Derived key group. 100 groups are available per key store.
 derived_key_grp: 45
 # ===== Derived key size bits [Required] =====
 derived_key_size_bits: 256
 # ===== Derived key type [Required] =====
 derived_key_type: OEM_IMPORT_MK_SK
 # ===== Derived key lifetime [Required] =====
 derived_key_lifetime: PERSISTENT
 # ===== Derived key usage [Required] =====
 derived_key_usage:
 - Derive
 # ===== Derived key permitted algorithm [Required] =====
 derived_key_permitted_algorithm: HKDF SHA256
 # ===== Derived key lifecycle [Required] ====
 derived_key_lifecycle: CURRENT
 # ===== Derived key ID [Required] =====
 derived_key_id: 0
 # ===== Private key ID [Required] =====
 # Description: Identifier in the ELE key storage of the private key
 private_key_id: 0x70000000
 # ===== Input peer public key digest [Optional] =====
 input_peer_public_key_digest:
 '0xf3dfec59defa67f3370dff1273fa1266fd6ac19c06c80d54986bf67100d1e67e'
 # ===== Input user fixed info digest [Optional] =====
 #input_user_fixed_info_digest:
 '0x8f21e3973d62fe7f9ea799a9541cc11cbe629f7be4125ff4e58312d2326ec61f'
 # ===== OEM Private Key for ECDH [Optional] =====
```

```
 oem_private_key: app_ecc256.pem
 # ===== NXP Production Key Agreement Public Key [Optional] =====
 nxp_prod_ka_pub: nxp_prod.pub
```

6. Build the signed content binary for the *key exchange* process. The output file, named as signed_message.bin, is generated based on the configuration specified in the previous configuration file. The OEM_IMPORT_MK_SK key is saved in the file *oem_import_mk_sk.bin* located in the *ecdh_derived_key* folder.

```
nxpimage signed-msg export -c signed_msg_config.yaml -w ecdh_derived_key
```

7. Convert the binary file generated in step 6 into C array named as signed_msg_bin in the file, signed_message.c, using the following command.

```
nxpimage utils convert bin2carr -i signed_message.bin -e little -c 8 -n
 signed_msg_bin -o signed_message.c
```

8. Retrieve the template configuration file for the *key import* TLV blob using the following command:

```
nxpimage signed-msg tlv get-template -f rt118x -o oem_import_key.yaml --
force
```

9. Populate the generated TLV blob template file with the required configuration. Specify the imported OEM key, its key attributes, and OEM_IMPORT_MK_SK in this configuration file. Below is an example configuration:

```
# =  TLV Configuration template for mimxrt1189, Revision: latest.  =
# ====================
# == Basic Settings ==
# ====================
# ===== Output Image name [Required] =====
output: tlv.bin
# =====================
# == General Options ==
# =====================
# ===== The chip family name [Required] =====
family: mimxrt1189
# ===== MCU revision [Optional] =====
revision: latest
command:
  # ===== Key import TLV [Required] =====
  KEY_IMPORT:
    # ===== Key ID [Conditionally required] =====
    # Description: Key store ID where to store the imported key. If set to 0,
 the actual ID is determined by ELE.
    key_id: 0x3FFFFFFE
    # ===== Permitted algorithm [Conditionally required] =====
    permitted_algorithm: 'ALL CIPHER'
    # ===== Derived key usage [Conditionally required] =====
    key_usage:
      - Encrypt
      - Decrypt
    # ===== Import key type [Conditionally required] =====
    key_type: AES
    # ===== Import key size bits [Conditionally required] =====
    key_size_bits: 256
    # ===== Imported key lifetime [Conditionally required] =====
    key_lifetime: ELE_KEY_IMPORT_PERSISTENT
    # ===== Imported key lifecycle [Conditionally required] =====
    key_lifecycle: CURRENT
    # ===== OEM_IMPORT_MK_SK key ID [Conditionally required] =====
```

```
    oem_mk_sk_key_id: 0x3FFFFFFF
    # ===== Imported key wrapping algorithm [Conditionally required] =====
    key_wrapping_algorithm: RFC3394
    # ===== Wrapped key signing algorithm [Optional] =====
    signing_algorithm: CMAC
    # ===== Raw import key [Conditionally required] =====
    import_key: oem_key.bin
    # ===== OEM_IMPORT_MK_SK Key [Conditionally required] =====
    oem_import_mk_sk_key:
'0x1292bbcbf18adbcf2a5582df5e45dbbcd4d154adcc10a35ff1f00cfafac54aaa'
```

10. Build a TLV blob for the *key import* process. The output file tlv.bin is generated based on the configuration specified in the previous configuration file.

```
nxpimage signed-msg tlv export -c oem_import_key.yaml
```

11. Convert TLV blob binary generated in the step 10 into C array named as oem_tlv_blob in the file, oem_tlv_blob.c, using the following command:

```
nxpimage utils convert bin2carr -i tlv.bin -e little -c 8 -n oem_tlv_blob -o
 oem_tlv_blob.c
```

7.5 Key import procedure

The following steps describe the procedure for importing the OEM key using the example project:

1. Replace the contents of the following local ECC public key arrays in the demo project with the C arrays generated in section 7.4.
   - ecc256_pub_key
   - signed_msg_bin array
   - oem_tlv_blob array

   These arrays are located in the ele_crypto_hsm.c file within the project.
2. Build the demo project and run it on the RT1180 board. The program performs the following operations:
   - To generate a shared OEM master key (OEM_IMPORT_MK_SK) by calling the ELE_ExchangeKey function within the ELE, perform an ECDH key agreement.
   - Import the user key wrapped in the TLV blob by calling the ELE_ImportKey_HSM function.

Note: *When the sync flag is enabled in the ELE_ImportKey_HSM function, the ELE exports the encrypted OEM key and instructs the application to store it in Non-Volatile Memory (NVM) for future use.*

- *In the example project, the exported OEM key is stored in RAM instead of external memory.*
- *To store the exported OEM key in the FAT file system (FATFS), set the macro USE_RAMFS to 0.*
- *An SD card is required for this configuration.*

7.6 Use the imported key with an ELE crypto API

The user can reference the key by its key ID when using it with the ELE. The cipher service is provided by the ELE runtime firmware.

After the key import procedure, the demo automatically performs the following operations:

- The firmware is loaded after the key import procedure.
- The cipher encrypt command is invoked to perform an AES-ECB encryption operation.
- To verify the result, the demo also uses the MbedTLS AES API for comparison.

8 Acronyms

Table 9 lists the acronyms used in this document along with their description.

Table 9. Acronyms

Acronym	Description
ELE	EdgeLock Secure Enclave
HSM	Hardware Security Module
LSB	Least Significant Byte
MUs	Messaging Units
NVM	Non-Volatile Memory
SPSDK	Secure Provisioning SDK
RKH	Root Key Hash
TLV	Type-Length-Value

9 Related documentation

Table 10 lists the references used to supplement this document.

Table 10. Related documentation/resources

Document	Link/how to access
Secure Boot on i.MX RT118x (document AN14227)	AN14227
EdgeLock Secure Enclave i.MX RT1180 User Guide (UG10192)	Contact your local NXP FAE or sales representative.
SPSDK	SPSDK

10 Note about the source code in the document

11 Revision history

summarizes the revisions to this document.

Table 11. Revision history

Document ID	Release date	Description
AN14861 v.1.0	13 November 2025	Initial public release

Legal information

Definitions

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

Disclaimers

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

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

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

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

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

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

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

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

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