AN14105

Understanding SECO Secure Storage and Non-Volatile Memory Management

Rev. 1 — 23 November 2023 Application note

Document information

Information	Content
Keywords	AN14105, i.MX 8, SECO, SCU, NVM, keystore, monotonic counter
Abstract	This document describes some of the key concepts related to the Security Controller (SECO) secure storage and non-volatile memory management in the NXP i.MX 8, i.MX 8X, and i.MX 8XLite device families.

1 Introduction

This document describes some of the key concepts related to the Security Controller (SECO) secure storage and non-volatile memory management in the following NXP device families:

- i.MX 8
- i.MX 8X
- i.MX 8XLite

2 SECO overview

The SECO is a crypto processor running a dedicated firmware. The SECO offers cryptographic services to other processors in the i.MX 8 system-on-chip (SoC), including:

- System Controller Unit (SCU)
- Any of the other application processors: Cortex-A and Cortex-M

When started, the SECO waits for requests from its clients. The communication between the SECO and a client happens through a hardware messaging unit (MU).

Four MUs are available for communication between the SECO and its clients. One of the MUs (MU0) is reserved for communication between the SECO and the SCU. The remaining three MUs can be used by one of the other clients (Cortex-A or Cortex-M).

2.1 Hardware

Figure 1 provides an overview of the SECO connections in an i.MX 8 SoC. The details of individual SECO sub-blocks are out of scope for this document. You can find those details in the security reference manual of the SoC in use.

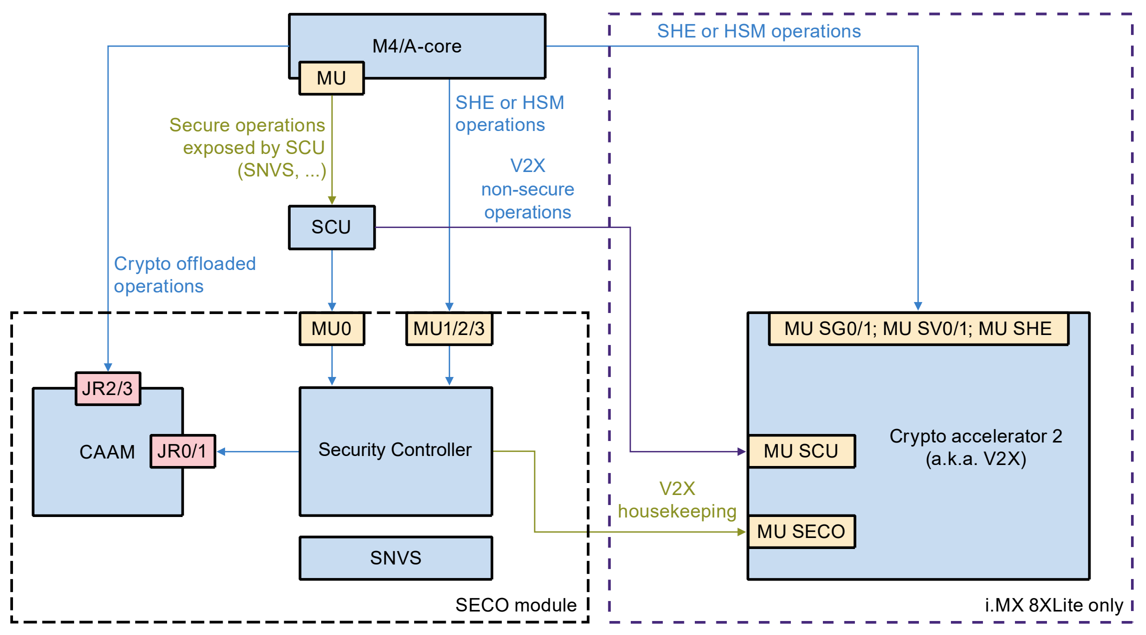

Figure 1. SECO connections in i.MX 8

2.2 Software

The communication protocol between the SECO firmware and its clients over messaging units is abstracted by a software library provided by NXP: seco_libs.

The seco_libs library provides some examples explaining the implementation of several services, including:

- Building blocks needed to implement a Security Hardware Extension (SHE) or a Hardware Security Module (HSM) on SECO
- An abstraction of the communication between the SECO and its clients, running on the Linux operating system

In this document, all references to software APIs should be considered as references to the seco_libs functions. You can find more details on seco_libs at SECO libs repository.

Some of the functions exposed by the SECO can only be accessed through the SoC System Controller Unit (SCU). Most of these functions are related to secure non-volatile storage (SNVS). These functions do not include cryptographic functions related to a SHE or an HSM implementation. The functions gated by the SCU are listed in *SECO API Reference Guide*.

3 Keystores

The basic concepts related to keystores are explained in HSM API document. They are briefly restated here for reference.

3.1 Key management

Keys used by the SECO HSM are securely stored in a structure, known as *keystore*. The SECO operates on keystores located in its internal memory.

A keystore is identified by a 32-bit ID and a 32-bit nonce (password), which is set at the time of keystore creation. To open a keystore, a SECO client must provide both the ID and nonce of the keystore.

The key management is shown in Figure 2 and is explained in the sections that follow.

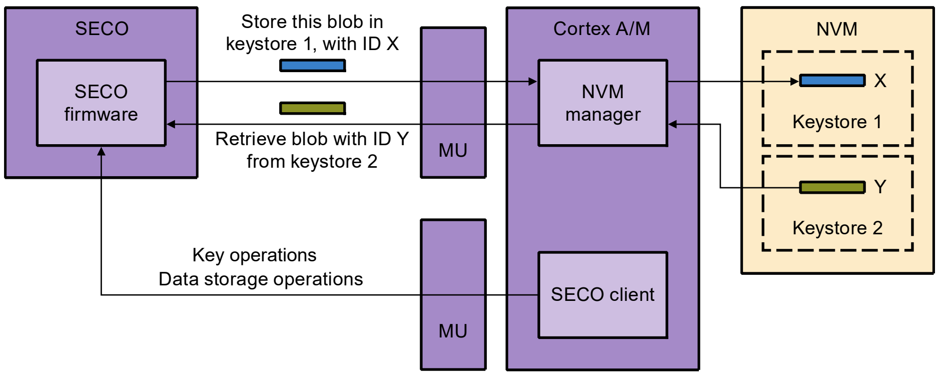

Figure 2. Key management diagram

Note: *Non-Volatile Memory (NVM) manager and other SECO clients can share the same MU.*

3.2 Key groups

Keys are categorized into groups:

- The SECO local memory can handle up to three key groups. These key groups are immediately available to perform crypto operations.
- The external non-volatile memory (NVM) can handle up to 1000 key groups. These key groups can be imported into the local memory, as needed.

The maximum number of keys in a key group depends on the key size and the amount of SECO local memory. For more details, refer to the sections describing chip specificities in HSM API document.

Keys belonging to the same group are written/read from the NVM as a monolithic block. When the local memory is full (three key groups already reside in the SECO local memory) and a new key group is needed to handle an incoming request, the SECO swaps one of the local key groups with the one needed by the user request.

The user can control which key group must be kept in the local memory (cached) through the `manage_key_group` application programming interface (API) lock/unlock mechanism. As a general concept, frequently used keys should be kept (if possible) in the same key group and should be locked in the local memory for performance optimization.

3.3 NVM manager

As the SECO has no means to access external storage, the key swapping operations are delegated to an external component, known as *NVM manager*.

The NVM manager is a SECO firmware client. Typically, it is implemented as a service or daemon that performs one of the following functions:

- Receives requests from the SECO to store encrypted blobs to external storage (data swap out from SECO)
- Retrieve encrypted blobs from external storage and send them back to SECO (data swap into SECO).

The NVM manager stores and retrieves data to and from external storage (NVM) on behalf of the SECO. It has no knowledge of the actual contents of the data that it manipulates. The NVM manager can store data in NVM in any format.

As with all SECO firmware clients, the communication between the SECO firmware and the NVM manager is done over a SECO messaging unit.

Because the SECO supports only one SHE session and two HSM sessions, the SECO can only handle a limited number of keystores in parallel:

- Only one keystore for SHE, managed by a dedicated NVM manager
- Only two keystores for HSM (one per session), managed by a single dedicated NVM manager

On the i.MX 8XLite devices featuring a Vehicle-to-Everything (V2X) crypto-accelerator, a third NVM manager handling NVM requests for the V2X accelerator can also run in the system.

3.4 Data storage

A keystore can also be used by a SECO firmware client to store secure data. The data is sent to the SECO as plain text, and the SECO encrypts it before sending it to a keystore.

In the case of secure data storage, the data is not kept in the SECO internal memory; rather, it is sent to external storage immediately, through the NVM manager. The data storage service is only available for an HSM session. For more details, see Monotonic counter and data storage.

3.5 Keystore manipulation

The correct method of manipulating keystores is explained below. Any deviation from the flow mentioned here may result in unpredictable behavior.

1. Launch an NVM manager client. Typically, an NVM manager is started as a background service at platform launch, and then it is never stopped. Several NVM managers can coexist in the system:
   - One for a SECO SHE session
   - Another for both SECO HSM sessions
   - Possibly a third NVM manager for V2X sessions, if the SoC supports V2X

   The type of NVM manager (SHE or HSM) is set by a flag passed to the SECO, when opening the NVM manager session.
2. Launch a SHE or an HSM session.
3. [Optional] If no keystore exists in the NVM or in the SECO memory, create a keystore, in the context of the SHE or HSM session previously opened.
4. Open an existing keystore in the context of the SHE or HSM session previously opened.
5. Perform key or data storage operations on the keystore — possibly writing them to NVM (STRICT operations).
6. Close the keystore when it is not needed anymore:
   - Closing a SHE keystore does not erase it from the SECO internal memory.
   - Closing an HSM keystore erases it from the SECO internal memory. All data in the keystore not committed to the NVM is lost.

4 Monotonic counter and secure storage

The i.MX 8 devices implement a monotonic counter. This counter is stored in efuses, and it is initially blank (no fuse blown).

The monotonic counter is used only on devices with the Original Equipment Manufacturer (OEM) - closed lifecycle, to mark the irreversible passage of time. It prevents the rollback attempts on operations involving non-volatile memory.

Only the SECO is able to blow fuses in the monotonic counter. It does so as a side effect of calling some SECO firmware APIs. The monotonic counter cannot be manipulated directly by a SECO client; rather, the SECO firmware provides APIs to query the global value of the monotonic counter.

On devices with the OEM-open lifecycle, the SECO never increases the monotonic counter.

The monotonic counter uses a cumulative scale — each time the counter has to be updated, a new fuse is blown. The value of the counter is the number of fuses blown.

The monotonic counter is divided into several partitions, with each partition acting as a separate counter. Each partition gets incremented for a certain type of NVM operations:

- SHE operations
- HSM operations
- All NVM operations performed by the V2X crypto accelerator (applicable for devices featuring such an accelerator)

For further details on monotonic counter partitions, see Monotonic counter partitions.

Table 1 shows the total number of fuses (monotonic counter size) and number of possible partitions in the counter.

Table 1. Monotonic counter characteristics

Device family	Monotonic counter size (bits)	Number of partitions	Partition type
i.MX 8	1620	2	SECO SHE, SECO HSM
i.MX 8X	1620	2	SECO SHE, SECO HSM

Table 1. Monotonic counter characteristics*...continued*

Device family	Monotonic counter size (bits)	Number of partitions	Partition type
i.MX 8XLite	1920	3	SECO SHE, SECO HSM, V2X

When all the fuses for the monotonic counter in a given partition are exhausted, the operations causing a monotonic counter increase in the partition still succeeds. However, the seco_libs return code for the API causing the counter increase is HSM_KEY_STORE_COUNTER. In general, the SECO firmware client must not treat this return code as an error condition.

The main purpose of the monotonic counter is to prevent rollback/replay operations on SECO encrypted storage. Specifically, the monotonic counter is incremented when an operation commits data in the NVM. Such an operation is called a STRICT operation, and it is usually triggered when the STRICT bit is set in the relevant SECO API flags. There are a few exceptions to this rule. Those exceptions are described in Monotonic counter and data storage.

4.1 SECO firmware APIs for monotonic counter

Table 2 lists the SECO and System Controller Unit Firmware (SCFW) APIs that modify and query the monotonic counter. If an API modifies the monotonic counter, the counter update occurs before the API returns.

Table 2. SECO firmware APIs related to monotonic counter

Access method	APIs	Notes
SECO client (seco_libs)	`hsm_open_key_store_service`, `she_storage_create`, `she_storage_create_ext`	When using one of these APIs to create a keystore with the STRICT flag set; the (empty) keystore is committed immediately to the NVM, and the monotonic counter is incremented.
SECO client (seco_libs)	`hsm_generate_key`, `hsm_generate_key_ext`, `she_load_key`, `she_load_key_ext`, `hsm_manage_key`, `hsm_manage_key_ext`	When using one of these APIs to create/derive a key with the STRICT flag set; the key is committed immediately to the keystore in the NVM, and the monotonic counter is incremented.
SECO client SCFW client	`hsm_get_info`, `she_get_info`, `sc_seco_chip_info`	The total number of fuses blown in the monotonic counter is part of the structure returned by these APIs.
SCFW client	`sc_seco_set_mono_cntr_partition`, `sc_seco_set_mono_cntr_partition_hsm`	Use the _hsm variant for devices featuring a V2X accelerator. No fuse is burnt as a consequence of this API — partition boundary fuses are committed to the monotonic counter at the next STRICT operation.

4.2 Monotonic counter partitions

The monotonic counter is divided into partitions, with one partition for counting each of the following types of operations:

- SECO SHE operations (for SECO firmware supporting SHE)
- SECO HSM operations (for SECO firmware supporting HSM)
- V2X accelerator operations (applicable for devices featuring a V2X accelerator)

Table 3 describes monotonic counter partition configurations.

Table 3. Monotonic counter partition configurations

Device	Monotonic counter size (bits)	Useful counter size (bits)	Number of partitions	Default partition sizes
i.MX 8	1620	1618	2	300 (SHE), 1318 (HSM)
i.MX 8X	1620	1618	2	300 (SHE), 1318 (HSM)
i.MX 8XLite	1920	1916	3	300 (SHE), 808 (SECO HSM), 808 (V2X)

As shown in Table 3, each partition has a default size. The size of a partition can be set *only once* before the first STRICT operation in the product lifetime, using one of the following SCFW APIs:

- `sc_seco_set_mono_cntr_partition` (for devices without a V2X accelerator)
- `sc_seco_set_mono_cntr_partition_hsm` (for devices with a V2X accelerator)

The boundary of each partition is marked by bits inside the monotonic counter fuse array itself. It has two consequences:

- After the “partition boundary” bits are committed to the monotonic counter, the partitions can no longer be resized. In other words, the partition size can be set only once in the whole lifetime of a given SoC.
- The number of useful bits in the monotonic counter fuse array is slightly less than the total number of fuses — each partition boundary uses two bits in the array. Those two bits are not available for performing count operations.

The SCFW or SECO API returning the value of the monotonic counter returns the total number of fuses blown in the monotonic counter fuse array, including the "partition boundary" fuses.

The SECO firmware blows the "partition boundary" bits with the first operation that requires querying the monotonic counter. The process to use a non-default monotonic counter partitioning scheme is summarized below:

1. Move the device to the OEM-closed lifecycle. No fuse is blown in the monotonic counter.
2. If needed, call the appropriate SCFW API to change the partition size. No fuse is blown in the monotonic counter.
3. Perform any keystore-related operation. The "partition boundary" fuses are blown at this stage, irrespective of whether the operation is STRICT or not. Remember that if the operation is not STRICT, only the partitions are blown; the value of each monotonic counter remains 0.

Example of setting the SHE partition size and the HSM partition size in i.MX 8DualXLite

To set the SHE partition size to 512 bits and the HSM partition size to 512 bits on the i.MX 8DualXLite device, follow these steps:

1. Move the device to the OEM-closed lifecycle. The monotonic counter value is 0.
2. Call the `sc_set_mono_cnt_partition_hsm(&she, &hsm)` API, where `she` and `hsm` are `int` variables, each having a value 512. When this API returns, the monotonic counter value is still 0.
3. Perform a single STRICT operation (usually committing a keystore containing one key group to NVM, or creating a keystore with the STRICT flag set). Now, the monotonic counter has three bits blown:
   - Two bits for the partition boundaries
   - One bit in the SHE or HSM partition, depending on whether the STRICT operation was for a SHE or an HSM keystore

   Now, the SHE and HSM partitions are 512-bit large. Therefore, the V2X partition size = 1916 - 1024 = 892 bits. These sizes can no longer be changed. At this stage, calling the `sc_seco_info()` API returns 3, that is, the total number of bits blown in the monotonic counter array.

4.3 Monotonic counter and keystore provisioning

The SECO uses the monotonic counter to detect and prevent rollback attempts when it tries to open a keystore. It also prevents overwriting a keystore when the monotonic counter value is non-zero, unless the overwrite operation is validated by a signed message.

The SECO enforces the following rules:

- Whenever an update is made to a keystore in the Non-Volatile Memory (NVM) using one of the APIs listed in Table 2 (that is, a STRICT update occurs), the SECO makes the following changes related to the monotonic counter:
  1. Increments the monotonic counter.
  2. Updates the keystore metadata in NVM to contain the new monotonic counter value.
- When the SECO is requested to open a keystore from NVM, it takes the following actions:
  1. Reads and decrypts the keystore metadata.
  2. Compares the device monotonic counter value with the counter value stored in the keystore metadata:
     - If the two values match, the SECO tries to open the keystore.
     - If the counter value stored in the keystore is lower than the value on the current SoC, the SECO considers the operation to be a rollback attempt (reuse an old keystore), and refuses to open the keystore.
     - If the counter value stored in the keystore is higher than the value on the current SoC, the SECO assumes that:
       - The keystore is valid.
       - During a previous attempt, the SECO was able to decrypt keystore metadata but failed to update the fused monotonic counter.

       The SECO updates the monotonic counter fused value to match the keystore counter value, and then loads the keystore.
- When the SECO is requested to create a keystore in the STRICT mode:
  - If the monotonic counter value is 0, the SECO always accepts a request for keystore creation. It always happens when the SoC is in the OEM-open lifecycle, as the monotonic counter is never incremented in this lifecycle. In this case, the SECO always tries to create a keystore, though some events may prevent it from creating a keystore.
  - If the monotonic counter value is non-zero and a set of metadata exists in the NVM but the keystore ID is not present in the metadata, the SECO accepts a request for keystore creation.
  - If the monotonic counter value is non-zero, and the keystore ID corresponds to an existing keystore in the NVM or the NVM has no keystore at all, the SECO assumes that the keystore creation request intends to overwrite an existing keystore. It rejects the request unless it received a valid `SAB_KEY_STORE_SEC_PROV` signed message previously. For more details on signed message generation, see *Using i.MX 8 Security Controller Signed Messages* (AN13770).

On an OEM-closed device, if the monotonic counter value is non-zero, do not erase the NVM; otherwise, all keystore metadata gets removed. If the keystore metadata has been removed, then, to create a keystore, you must send a signed message to the SECO. The process of generating and signing a message is complicated, and it requires accessing some device-specific information and signing with an OEM private key. Implementing this process in the field after production is impractical.

NVM erasure is common during development phases. Therefore, most development should be done on OEM-open devices, where keystore reprovisioning does not require a signed message.

4.3.1 Reprovisioning and partitions

The monotonic counter partitions are independent in terms of reprovisioning — if the SHE monotonic counter value is 0, and the HSM monotonic counter value is non-zero, then, a signed message is needed only for reprovisioning an HSM keystore.

Because there is no way to retrieve the monotonic counter value of a specific partition, the client software must keep track of the STRICT operations for both SHE and HSM (and possibly V2X), if it needs to know the monotonic counter value for each partition.

Because there is a single HSM monotonic counter and the SECO can maintain only two HSM keystores at most, reprovisioning an HSM keystore also invalidates the second HSM keystore.

4.3.2 NVM backups

The NVM data can be backed up so that a recovery operation can be performed in future when the NVM gets corrupted. Because the NVM data is always checked against the monotonic counter to prevent rollbacks, the backed-up data must be updated each time a STRICT operation is performed on the source data.

4.4 Monotonic counter and data storage

The SECO allows you to store your data blobs in a keystore using the `hsm_data_storage` API. Storing data is based on the following rules:

- All data storage operations are STRICT — they all commit the data blob to the NVM.
- The data storage operations must operate on a keystore already committed to the NVM, that is:
  - Either a keystore created with the STRICT flag
  - Or a keystore containing at least one key that was created with the STRICT flag
- The SECO does not prevent adding a data storage to a keystore only present in the local SECO memory. However, it results in an inconsistent NVM state, and the data storage is not usable on subsequent power cycles. To avoid this condition, perform data storage operations only on keystores already committed to NVM.
- The data storage units are referenced through their 16-bit IDs. The SECO does not provide any protection mechanism on the data unit — writing a data storage unit with the same ID multiple times overwrites the data storage unit.
- The data storage operations do not increment the monotonic counter. There is no rollback protection on data storage.
- Data storage service is only available on the SECO; V2X does not support this service.

Example of creating and using a keystore on i.MX 8QuadXPlus

To create a keystore on an i.MX 8QuadXPlus device and use it for data storage, follow these steps:

1. Move the device to the OEM-closed lifecycle. The monotonic counter value is 0.
2. Create a keystore with the STRICT flag. The default partitioning scheme is applied, and an empty keystore is committed to the NVM. The monotonic counter is set to value 2 (one unit for the "partition boundary" bits, and one unit for keystore creation).
3. Create a data storage with the ID 0x0042 in the newly created keystore, and populate the data storage with some data. The data is immediately committed to the NVM, and the monotonic counter value remains 2.

4.5 Monotonic counter and V2X

On devices featuring a V2X crypto accelerator (for example, i.MX 8XLite), the V2X accelerator has its own monotonic counter partition. This partition is applicable for both the SHE and HSM STRICT operations performed on the V2X accelerators.

5 References

Table 4 lists additional documents/resources related to the SECO secure storage and NVM management.

Table 4. Reference documentation/resources

Document/resource	Link / how to obtain
SHE API document	she_api_manual.pdf
HSM API document	hsm_api_document.pdf
HSM and SHE on i.MX 8QXP and i.MX 8DXL (AN12906)	AN12906.pdf
Using i.MX 8 Security Controller Signed Messages (AN13770)	AN13770.pdf
i.MX 8 security reference manuals	Visit i.MX 8 series page
SECO API Reference Guide	Contact an NXP FAE or sales representative
SECO HSM and SHE example code	hsm_she_examples
SECO libs repository	imx-seco-libs

6 Acronyms

Table 5 lists the acronyms used in this document.

Table 5. Acronyms

Acronym	Description
API	Application programming interface
CAAM	Cryptographic Acceleration and Assurance Module
HSM	Hardware Security Module
JR	Job ring
MU	Messaging unit
NVM	Non-volatile memory
OEM	Original Equipment Manufacturer
SCFW	System Controller Unit Firmware
SCU	System Controller Unit
SECO	Security Controller
SHE	Security Hardware Extension
SNVS	Secure non-volatile storage
SoC	System-on-chip
V2X	Vehicle-to-Everything. In the context of the current document, V2X refers to the V2X crypto accelerator embedded in some i.MX 8X devices.

7 Note about the source code in the document

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

8 Revision history

Table 6 summarizes the revisions to this document.

Table 6. Revision history

Revision number	Release date	Description
1	23 November 2023	Initial public release

Legal information

Definitions

Disclaimers

Trademarks

Contents

Date of release: 23 November 2023
Document identifier: AN14105