AN14741

EdgeLock 2GO Provisioning via Proxy Flow for MPUs

Rev. 1.1 — 21 October 2025 Application note

Document information

Information	Content
Keywords	AN14741, EdgeLock 2GO, EdgeLock, EdgeLock Enclave, EdgeLock Enclave (Advanced-Profile), security, trust provisioning, secure provisioning, i.MX, Security Middleware, EdgeLock 2GO agent, SPSDK, Secure Provisioning SDK
Abstract	This document offers an outline of the EdgeLock 2GO platform and discusses the Device provisioning via proxy flow to interface with host tools and securely provision assets from the EdgeLock 2GO server to the EdgeLock Enclave (ELE) based NXP devices.

1 Introduction

EdgeLock 2GO is a fully managed cloud platform operated by NXP that provides secure provisioning services for easy deployment and maintenance of IoT devices that integrate NXP MCU, MPU, and EdgeLock SE05x secure elements.

EdgeLock 2GO – Managed is a service that allows you to create and manage secure objects, such as symmetric roots of trusts, key-pairs, and certificates, which are then securely provisioned by EdgeLock 2GO – Managed into the i.MX based IoT devices.

Secure objects and certificates created through EdgeLock 2GO – Managed can be used for a wide variety of use cases, including for secure cloud onboarding of IoT devices to your preferred cloud service (for example, AWS IoT Core or Azure IoT Hub), data encryption or decryption, access control, and so on.

EdgeLock 2GO – Managed can be easily used and configured through the EdgeLock 2GO web portal or the EdgeLock 2GO REST API.

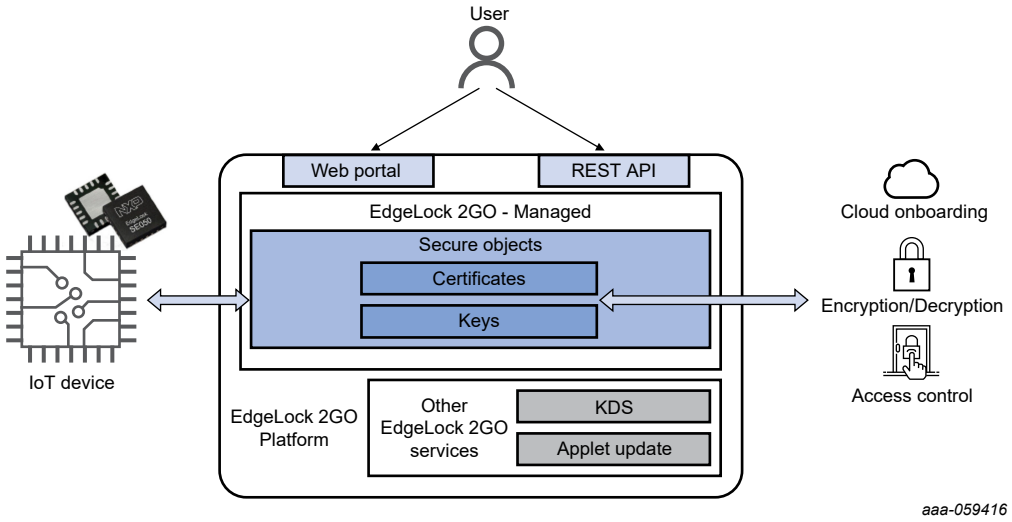

Figure 1. EdgeLock 2GO platform overview

For more details about EdgeLock 2GO, see EdgeLock 2GO homepage and the documentation available at EdgeLock 2GO Portal (account/sign-in required).

2 Scope and objective

This document offers an outline of the EdgeLock 2GO platform and discusses the **Device provisioning via proxy** flow to connect the EdgeLock Enclave (ELE) based NXP devices to the EdgeLock 2GO cloud server.

The document focuses on the registration and management of the devices and provisioning of secure objects in the EdgeLock 2GO cloud server along with its connectivity with the NXP devices. It also gives an overview of the hardware and software setup required to run an EdgeLock 2GO agent application on the device.

3 Prerequisites

3.1 Hardware procurement

During manufacturing, the NXP SoCs are injected with EdgeLock 2GO credential and registered in the EdgeLock 2GO pool of devices. Such NXP parts can be procured at nxp.com.

Procure the device among the ones listed under **Supported Products** at EdgeLock 2GO homepage at nxp.com.

3.2 EdgeLock 2GO account creation

To access the EdgeLock 2GO cloud server, create an account by requesting access.

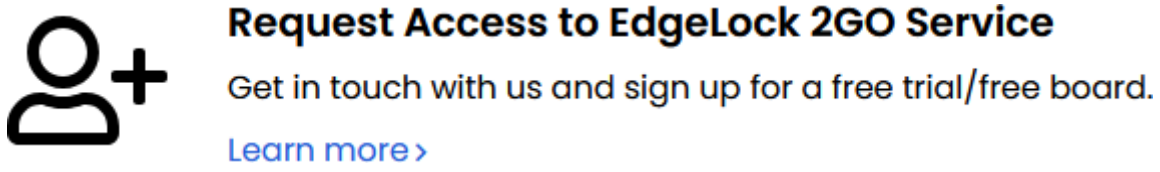

Figure 2. Requesting access to the EdgeLock 2GO cloud server

For more details, see the Logging in the **EdgeLock 2GO – account** section in the *EdgeLock 2GO – Managed* (document AN12691) mentioned in Section 15.

3.3 EdgeLock 2GO account login

Once an account is created in the EdgeLock 2GO server using the email address you provided, you can use an NXP registered account associated with that email to log in to EdgeLock 2GO Portal.

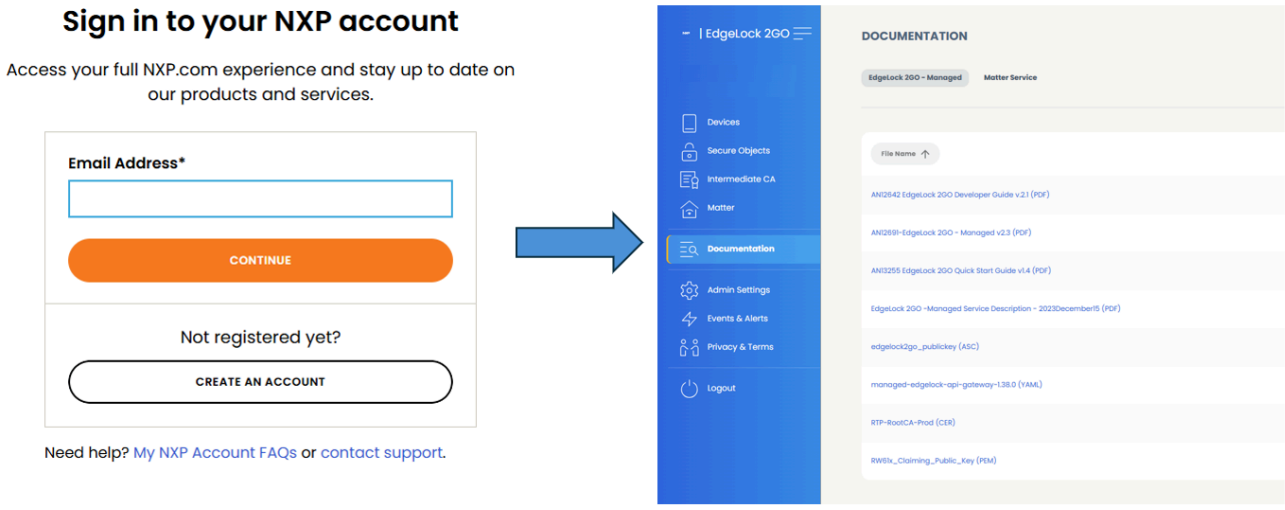

Figure 3. Account login

3.4 Detect EdgeLock 2GO enabled hardware

All i.MX 8ULP and 9 family's production parts (with markings **M** or **MC** on the silicon) are EdgeLock 2GO enabled.

After procuring the Evaluation Kit/Board or Part, the device can also be verified to be EdgeLock 2GO enabled by performing the following steps:

1. Determine the UUID as described in Section 11.
2. Log in to the EdgeLock 2GO Portal.
3. Create a device group.
   Click **Devices ➔ MCU & MPU ➔ New Device Group**.

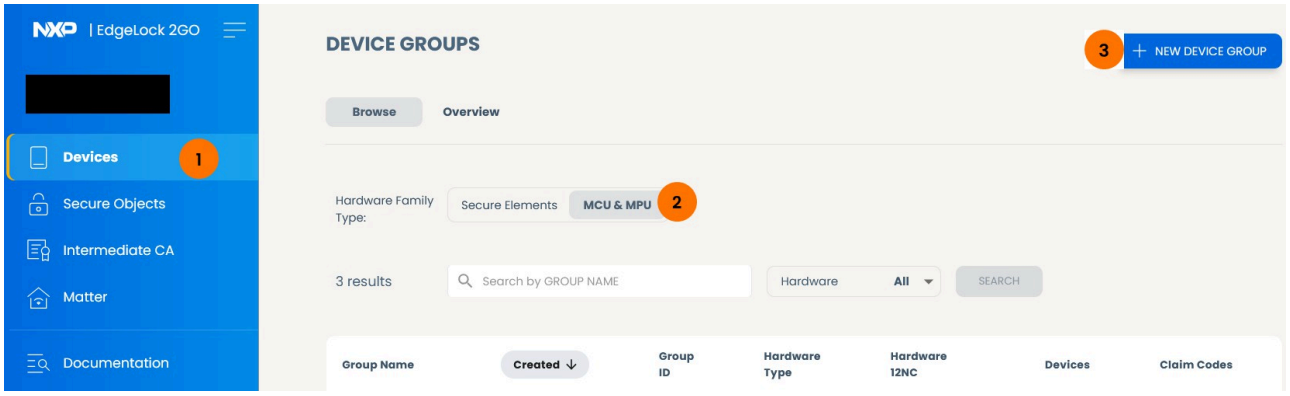

Figure 4. Create a device group

4. Verify the device.

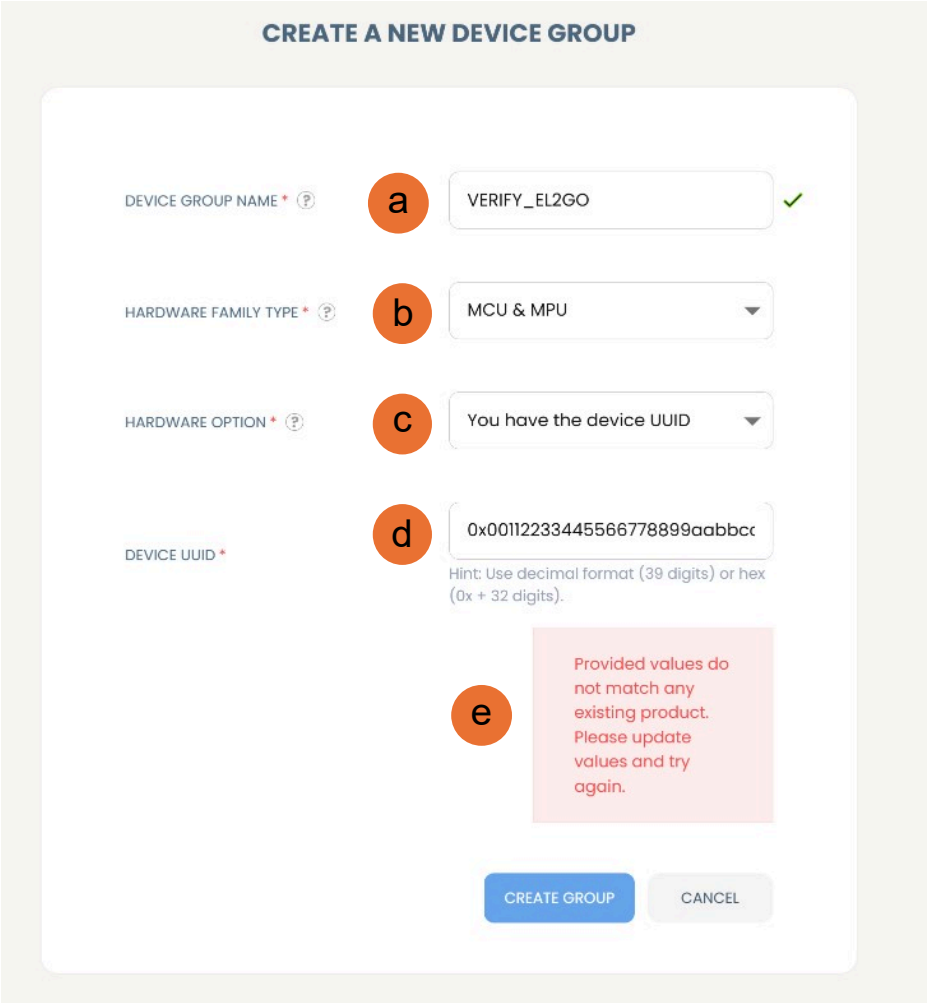

Figure 5. Verify the device

a. Give a sample name to the **Device group name** field.
b. Choose **MCU & MPU** in the **Hardware Family Type** field.
c. Choose **You have the device UUID** in the **Hardware Option** field.
d. Paste the device UUID in the **Device UUID** field and prepend with `0x`.
e. If the device is registered in the EdgeLock 2GO - Managed server, then the Error in red is not shown.

4 (OEM) Create a device group in the EdgeLock 2GO server

Note: *This step is typically carried out by the OEM overseeing the EdgeLock 2GO admin account, to manage registered devices and secure provisioning assets.*

1. Click **Devices ➔ MCU & MPU ➔ New Device Group**.

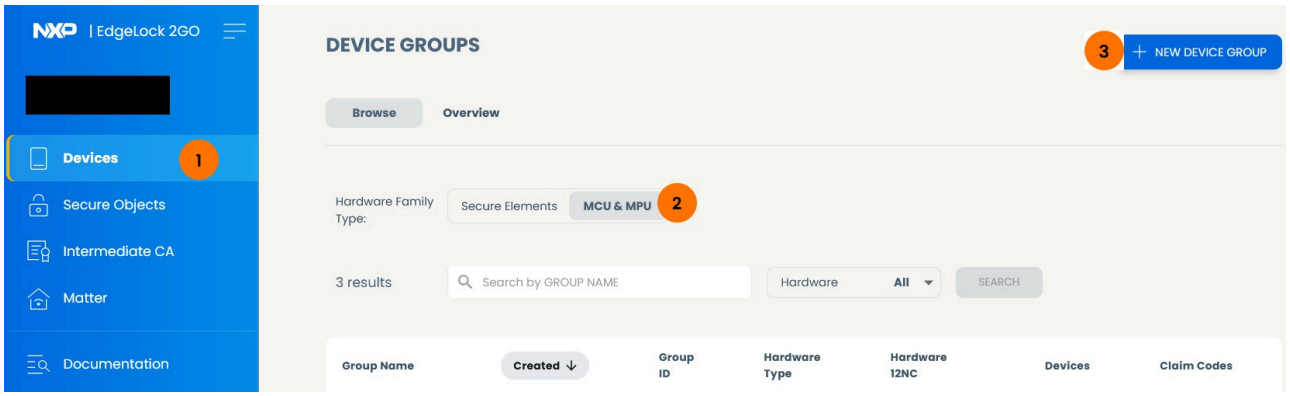

Figure 6. Device Groups window

2. An interface pops up, as shown in Figure 7.

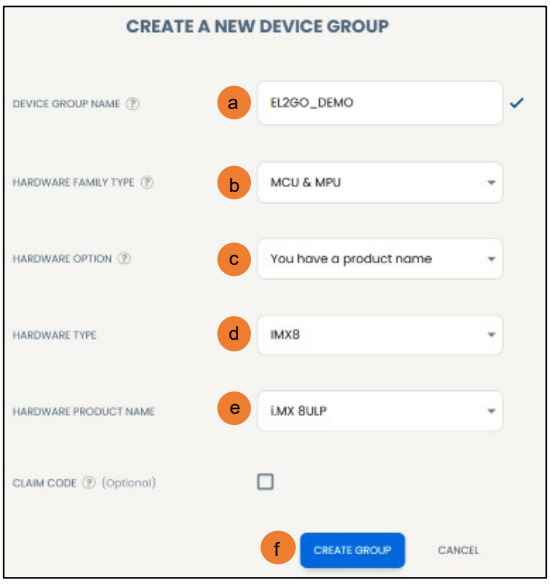

Figure 7. Create A New Device Group interface

a. Input a custom **Device Group Name**.
b. Select **Hardware Family Type** as **MCU & MPU**.
c. Expand **Hardware Option** and select **You have a product name**.
d. Select the appropriate **Hardware Type**.
e. Select the appropriate **Hardware Product Name**.
f. Click **Create Group** to create the device group, in which new devices are registered.

5 Setting up EdgeLock 2GO account with provisioning assets

The EdgeLock 2GO remote trust provisioning service can securely configure, provision, and revoke secure objects on IoT devices.

During the initial provisioning, the service requires these secure objects to be linked to the OEM firmware authentication key hash (`OEM_ROTKH` or `OEM_SRKH`). Therefore, the corresponding `OEM_ROTKH` or `OEM_SRKH` must be recorded in the device group to which the devices are registered.

5.1 Generate OEM firmware authentication key hash

ELE-based NXP MCU and MPU devices require four[1] root keys to be used for performing secure boot by the OEM. The public keys out of these OEM generated key pairs must be hashed to create the **OEM FW Authentication Key Hash**.

Root keys can be generated using the SPSDK tool, as shown in Section 12.1.

5.2 Install root key hash to device group (mandatory)

The following steps describe recording of the **OEM FW Authentication Key Hash** to the device group.

1 Some devices require up to four root keys. For details, see the device specific Security Reference Manual.

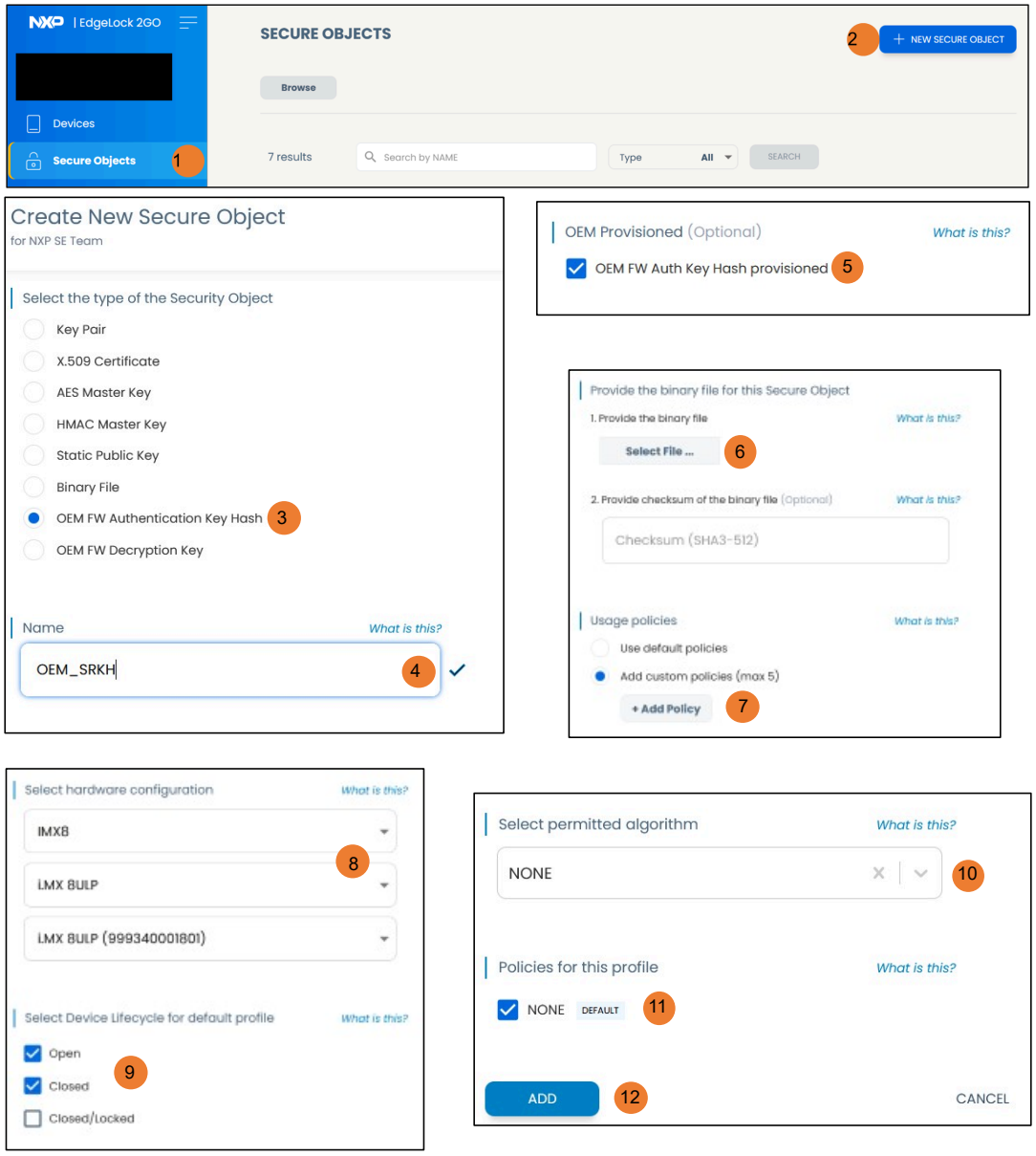

Figure 8. Install root key hash to device group

1. Click **Secure Objects**.
2. Click **New Secure Object**.
3. Select **OEM FW Authentication Key Hash**.
4. Enter the custom name in the **Name** field.
5. Check the **OEM FW Auth Key Hash provisioned** box to indicate that the `OEM_ROTKH/OEM_SRKH` has already been provisioned in the device.
   Note:
   *If the **OEM FW Authentication Key Hash provisioned** box is not checked, the `oem-prov-app` application, at runtime, uses the ELE firmware to provision the `OEM_ROTKH/OEM_SRKH` to the device. If it has already been provisioned, the ELE firmware may throw an error.*
6. In the **Select File** field, choose the `OEM_ROTKH/OEM_SRKH` binary file generated from Step 5.

7. Select **Add custom policy** and click **+Add Policy**. This policy is to attach this secure object to a specific hardware.
8. Choose the appropriate hardware configurations.
9. Select the device lifecycle as **open** and/or **closed**, where this secure object can be imported.
10. Select **NONE** for the permitted algorithm.
11. Select **NONE** for the profile policy.
12. Click **Add** to create the secure object.

5.3 Create and configure secure objects

Using EdgeLock 2GO service, it is possible to dynamically provision secure objects including RSA and ECC key-pairs, AES keys, HMAC keys, X.509 certificates, binary files, and static public keys in the flash of the NXP MCU and MPU device.

First, it is necessary to install the OEM FW Authentication Key Hash to the device group. The secure objects generated, thereafter, are encapsulated using the OEM FW Authentication Key Hash and thus can only be accessed in the devices with the same OEM FW Authentication Key Hash programmed in the eFuses.

For steps to set up different secure objects as needed, see the **Create a new secure object** section in the *EdgeLock 2GO – Managed* (document AN12691) mentioned in Section 15.

5.4 Assign secure objects to the device group

Once the secure objects are created, they can be assigned to a device group. In this way, all devices belonging to the device group are provisioned with the secure objects when they connect to the EdgeLock 2GO service. If a device is added to the device group later, it is provisioned with the secure objects once it connects to the EdgeLock 2GO service.

For further details, see the **Assign a device group to a secure object** section in the *EdgeLock 2GO – Managed* (document AN12691) mentioned in Section 15.

6 EdgeLock 2GO provisioning flows

For an overview on different EdgeLock 2GO services supported on MCUs and MPUs, see the *EdgeLock 2GO Services for MPU and MCU* (document AN14544) mentioned in Section 15.

7 Device provisioning via Proxy

When using Device provisioning via proxy, a host running SPSDK tool is used to facilitate device provisioning. The secure objects to be provisioned are downloaded from the EdgeLock 2GO server to the host. These secure objects are stored on the target device, where they are provisioned by the Provisioning Application running on it. This enables the target to provision secure objects without requiring a direct Internet connection (for ex. Manufacturing environment).

There are two ways to register the EdgeLock 2GO enabled NXP MCU and MPU devices in Device provisioning via Proxy: Device provisioning via Proxy with device UUID and Device provisioning via Proxy per product type.

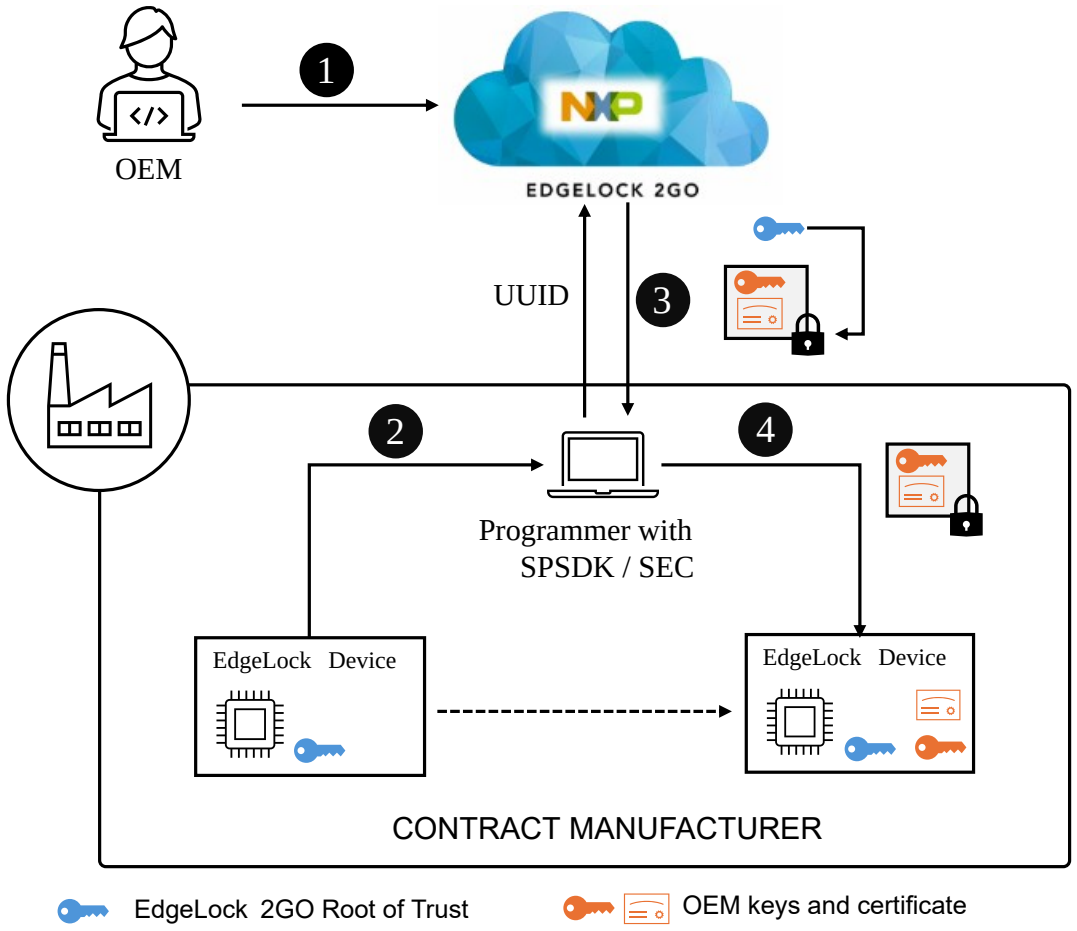

Figure 9. Device provisioning via Proxy

Typical flow in Device Provisioning via Proxy:

1. OEM configures secure objects via the EdgeLock 2GO portal or API.
2. At device manufacturing, readout device UUIDs (one by one or by batch) and send the UUID or list of UUIDs to EdgeLock 2GO over API.
3. EdgeLock 2GO generates the secure objects configured in step 1 and encrypts with EdgeLock 2GO root of trust. The encrypted secure objects are downloaded to the host.
4. The host loads the encrypted secure objects to the device where the Provisioning Application passes the encrypted credentials to the ELE to unwrap them. Internal secure objects are installed into fuses. External secure objects are rewrapped and written to flash.

Note: *Users can interact with EdgeLock 2GO via REST API as well. For how to use the REST API, see the EdgeLock 2GO Developer Guide (document AN12642) mentioned in Section 15.*

8 Device registration and assets deployment

8.1 Device registration and assets deployment with device UUID

Individual devices are registered, based on their UUIDs, to a device group in the EdgeLock 2GO server via the host. Once registered, the secure objects are assigned to these devices in the device group. This provisions all the registered devices in a device group with the secure objects during the device provisioning.

Device registration is handled by the SPSDK tool running on the host. The SPSDK tool connects with the device to retrieve the UUID and registers it to the EdgeLock 2GO server to a specific device group. During this operation, the secure objects assigned to the device group are also downloaded to the host to be subsequently provisioned into the target device.

Flow steps:

1. (OEM) Create API key for device provisioning purposes.
   Note: *This step is typically performed by the OEM to allow the Contract Manufacturer to securely connect with the pre-configured EdgeLock 2GO account and perform different actions.*
   For steps to create the API key, see the **API security model** section in the *EdgeLock 2GO Developer Guide* (document AN12642) mentioned in Section 15.
2. (CM) Register devices and gather secure provisioning assets during manufacturing.
   Note: *This step is typically carried out by the Contract Manufacturer (CM) responsible to register the devices in the OEM configured device group in the EdgeLock 2GO server.*
   For how to install SPSDK, see the SPSDK documentation.
   SPSDK package consists of a dedicated tool called el2go-host responsible to perform retrieval of secure provisioning assets and programming these assets in the device for secure provisioning. The YAML configuration file needed to perform this task. It consists of several options that must be populated.
   Note: *The configuration file can be prepared by the OEM and securely transferred to CM to be used for this process.*
   The configuration file template is generated using the get-template command:

```
$ el2go-host get-template -f mimx9352 -o el2go_template.yaml
```

   Note: *The family (`-f`) parameter must be chosen accordingly.*
   For more details about the parameters in the YAML configuration file template, see the template itself. The important fields are discussed here briefly, and the rest of the configuration fields can be at default:
   - `api_key`: Enter the API key received from the OEM, as described in step 1.
   - `device_group_id/nc12`: Enter the device group ID and NC12 dedicated to the device group. This information is received from the OEM, as shown in Figure 10.

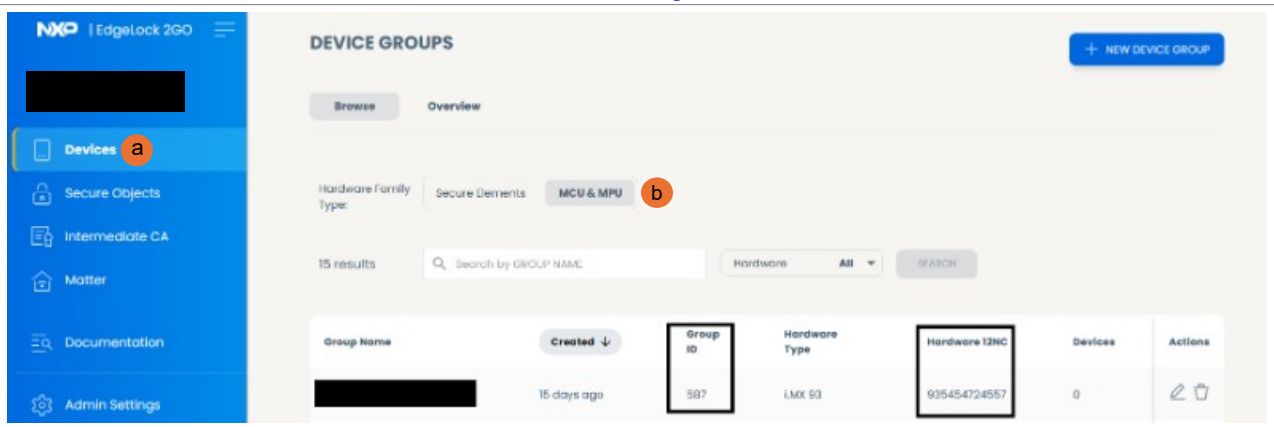

Figure 10. Device Groups window

   a. Click **Devices**.

b. Select **MCU & MPU** in the **Hardware Family Type** tab
c. Choose the **Group Name**, in which the devices must be registered. Capture the **Group ID** and **Hardware 12NC** number needed for the configuration file.

- `secure_objects_address`: Choose the unused memory address in DDR where the secure objects can be stored.
- `uboot_path`: Provide path of a bootable bootloader (*imx-boot/flash.bin*) image for the device being provisioned. Typically, OEM provides this bootloader image as part of the provisioning boot package.
- `oem_provisioning_config_filename`: Provide path to the configuration file to be used by `oem-prov-app`. Typically, this file is provided by the OEM to the contract manufacturer to include in the provisioning process. More on this is covered in Section 9.1.

As the SPSDK configuration file is ready, the device can now be registered to the EdgeLock 2GO server and the corresponding secure objects assigned to the device group, in which this device is registered, can be securely downloaded on to the host machine to be provisioned to the device.
The `el2go-host` tool can achieve the following steps in one operation:

a. Retrieve the UUID from the device.
b. Connect to the EdgeLock 2GO server and register the device UUID to the specified device group.
c. Retrieve the secure objects dedicated to the device group in which the device is registered and store it in the host machine.

Boot the device in serial download mode, connect the USB serial download port to the host and run the following command in the host machine.

```
$ el2go-host -v get-secure-objects -f mimx9352 --force --config
 el2go_template.yaml --output secure_objects_open.bin
```

3. (CM) Install Secure Objects in the device.
Note: *This step is typically carried out by the Contract Manufacturer (CM) responsible to deploy the secure provisioning assets to the device.*
Once the secure objects have been downloaded from the EdgeLock 2GO server, they are ready to be installed in the device for provisioning.
Boot the device to u-boot prompt, connect the USB serial download port to the host and run the following command in the host machine. Prepare the same configuration file for retrieving the secure provisioning assets, which can be used here.

```
$ el2go-host -v provision-objects -f mimx9352 --force --config
 el2go_template.yaml --secure-objects-file secure_objects_open.bin
```

This step deploys the secure objects file to the filesystem partition provided in the configuration file (typically the boot partition containing the Linux kernel image). Provide the same location to the provisioning tool used on the target to securely provision the assets to the device.
After this step, unwrap the secure objects by the EdgeLock Enclave and install them on the device by using the OEM Provisioning Application, as described in Section 9.

9 Secure provisioning with OEM provisioning application

The OEM provisioning application (`oem-prov-app`) is a reference user-space tool provided by NXP that facilitates the secure provisioning of assets (secure objects) into the EdgeLock Enclave. These assets are prepared on the EdgeLock 2GO server and can be provisioned either by connecting directly to the server or by using pre-deployed assets stored in the local filesystem.

For more details regarding the `oem-prov-app`, see here.

9.1 Prepare the YAML configuration file

Note: *This configuration file is typically provided by OEM to the CM to be included in the boot partition as mentioned in step 2 in Section 8.1. If already available, this step can be skipped.*

Boot the device with the `oem-prov-app` user space application installed. To install the `oem-prov-app`, if not already installed, follow the steps described in Steps to Install OEM Provisioning Application. The `oem-prov-app` requires a configuration file, which includes all the necessary information to be able to securely provision the assets in the device.

The sample configuration template file can be found at */etc/opt/oem-prov-app* folder on the device. Create a copy of this configuration file and modify it.

For more details about the parameters in the YAML configuration file template, see the template itself. The important fields are discussed here briefly:

- `partition`: This is the same partition where the secure objects were installed in step 2 in Section 8.1. For example: */dev/mmcblk0p1*.
- `mount_point`: This is the location where the boot partition is mounted. For example: */run/media/boot-mmcblk0p1*.
- `filename`: The filename of the secure objects binary provided in step 3 in Section 8.1 is provided here. For example: *secure_objects_open.bin*.

Note: *Leave the remaining configuration fields at their default values.*

9.2 Secure provisioning of assets (secure objects)

Note: *(Skip this step if the daemon is already running). The application provisions the secure objects in the NVM Secure Enclave storage. Therefore, the `nvm_daemon` daemon must be run prior to running the `oem-prov-app` user space application.*

```
$ root@imx8ulpevk:~# systemctl start nvm_daemon
$ root@imx8ulpevk:~# systemctl status nvm_daemon
* nvm_daemon.service - ELE Blob Process
Loaded: loaded (/etc/systemd/system/nvm_daemon.service; disabled; preset: disab>
Active: active (running) since Thu 2024-12-29 05:09:03 UTC; 3s ago Main PID: 564
 (nvm_daemon)
Tasks: 1 (limit: 1460) Memory: 172.0K (peak: 816.0K)
CPU: 28ms
CGroup: /system.slice/nvm_daemon.service
`-564 /usr/bin/nvm_daemon /etc/ele/ele_nvm_master /etc/ele/ 0 Dec 29 05:09:03
 imx8ulpevk systemd[1]: Started ELE Blob Process.
```

As the configuration file is ready, run the `oem-prov-app` to securely provision the assets (secure objects), which were previously installed in boot partition, using following command:

```
$ oem-prov-app -i /path/to/config.yaml
```

Once the `oem-prov-app` application finishes successfully, the secure objects created during EdgeLock 2GO server setup are securely provisioned and wrapped using ELE and stored in */etc/ele* folder.

9.3 Usage of provisioned assets (secure objects)

The assets provisioned in the device, using the methods described above, are ready to be used for cryptographic operations as necessary by the user. The Security Middleware (SMW), provided by NXP, can be used for such purposes. For further guidance, see the documentation folder present in the SMW repository.

9.4 Using oem-prov systemd service

The `oem-prov-app` meta-layer distributed within the i.MX Linux BSP contains multiple recipes to assist building the OEM provisioning application, deploying a sample config file and starting a daemon service which automates device provisioning. The daemon service starts the `nvm_daemon` (requirement to start `oem-prov-app`) and invokes the cloud/proxy provisioning method during bootup. For more details, see the `meta-oem-prov-app` directory within the oem-prov-app online repository (refer to the latest revision of this repository).

To automate the device provisioning via cloud using UUID, modify the `oem-prov.service` as indicated in Section 9.4.1.

To check the `oem-prov` status:

```
$ systemctl status oem-prov
```

If disabled, it can be enabled as follows:

```
$ systemctl enable oem-prov
```

The `oem-prov` system service must be customized depending on the device provisioning flow required. Once provisioning is complete and no longer needed, it is recommended to disable the service as follows:

```
$ systemctl disable oem-prov
```

To set the `oem-prov` systemd unit file within a Yocto build, see the Section 10.2.

9.4.1 Example of oem-prov unit file for device provisioning via proxy

The sample `oem-prov` unit file is provided as part of the `meta-oem-prov-app` meta layer to automate secure objects provisioning via proxy.

10 Build OEM provisioning application

NXP provides a Yocto meta-layer, which allows to integrate the OEM provisioning application into the bootable image (filesystem). The Yocto framework is used to build this application and deploy the final bootable image.

This chapter explains how to integrate the OEM provisioning application into the i.MX Linux BSP and how to customize its configuration file and its daemon service before deployment. If the `oem-prov-app` is included in the BSP, see Step 3 in Section 10.1 for editing its YAML file and Section 10.2 for editing its systemd's unit file. Two scenarios are covered:

- Building the entire i.MX Linux image with the OEM provisioning application included.
- Building and deploying the OEM provisioning application separately using Yocto's devtool.

10.1 Install the OEM provisioning application Yocto layer into i.MX Yocto environment

To install the OEM provisioning application Yocto layer into the i.MX Yocto environment, perform the following steps:

1. Set up the Yocto build system.
   To set up the Yocto project for a specific release version and machine, see the **Yocto Project Setup** and **Image Build** sections in the *i.MX Yocto Project User's Guide* (document UG10164).
   For example:

```
$ mkdir <yocto-directory>

$ cd <yocto-directory>
repo init -u https://github.com/nxp-imx/imx-manifest \
-b imx-linux-styhead -m imx-6.12.3-1.0.0.xml
$ repo sync
$ DISTRO=fsl-imx-xwayland MACHINE=imx93-11x11-lpddr4x-evk \
source imx-setup-release.sh -b <yocto-build-directory>
```

   Note: *To know which i.MX Yocto project images are supported, review the oem-prov-app online documentation.*
2. Clone the `oem-prov-app` repository.

```
$ git clone https://github.com/nxp-imx/oem-prov-app.git
```

   There is a release tag for each i.MX Linux BSP. The `oem-prov-app` release tag must match the release tag of the i.MX Linux BSP to avoid build issues. An example is provided:

```
$ cd ./oem-prov-app
$ git checkout <oem-provision-app-rel-tag> [ -b <branch-name> ]
```

3. Edit the `oem-prov-app` YAML configuration file.
   The `oem-prov-app` requires to parse a YAML file at runtime. Such a YAML file contains different configuration parameters used according to the device provisioning flow needed. Edit the YAML file using your preferred text editor. To edit the configuration file, see Section 9.1. The following example shows a command to open and edit the YAML file from its location within the `oem-prov-app` repository.

```
$ vim ./meta-oem-prov-app/recipes-oem-prov-app/oem-prov-config/config.yaml
```

Note: *If this step is not done, then the config.yaml file has to be modified after deploying the* `meta-oem-app` *and* `oem-prov-config` *Yocto layers into the image root file system.*

4. Add the `oem-prov-app` meta layer to the Yocto build directory.
   Copy the `oem-prov-app` meta layer to the Yocto sources directory.

```
$ cp -r ./meta-oem-prov-app/ <yocto-directory>/sources/
```

A suggested method to install the `oem-prov-app` meta layer into the Yocto build is to run the `oem-prov-app-setup.sh` script as follows:

```
$ cd ./<yocto-build-directory>
$ ../sources/meta-oem-prov-app/tools/oem-prov-app-setup.sh
```

Note: *For more information about the meta layer configuration, dependencies, and installation steps, see the README.md file provided within the* `meta-oem-prov-app` *directory.*

10.2 Modify the oem-prov-app systemd service

Modify the `oem-prov` systemd service, according to device provisioning flow requirements before deployment. An example of a unit configuration file is available at Section 9.4.

```
$ vi ./sources/meta-oem-prov-app/recipes-oem-prov-app/oem-prov-config/oem-prov.service
```

10.3 Build and deploy within the i.MX Linux BSP image

To deploy the `oem-prov-app` and all of its dependencies as part of the i.MX Linux BSP image, run the Yocto bitbake command using a supported imx-image target. An example command is shown below.

```
$ bitbake imx-image-core
```

Note: *To know which i.MX Yocto project images are supported, see the oem-prov-app online documentation.*

10.4 Build standalone from i.MX Linux BSP image and deploy

The `oem-prov-app` can also be built separately from the i.MX Linux BSP image.

10.4.1 Using Yocto devtool

The following steps describe how to use Yocto's devtool for building and deploying the `oem-prov-app` to the target board.

1. Add the `oem-prov-app` recipe to the devtool environment.

```
$ devtool modify oem-prov-app
```

2. Build the `oem-prov-app` recipe using the devtool.

```
$ devtool build oem-prov-app
```

3. Deploy the binaries and dependencies previously built to the target board.

```
$ devtool deploy-target oem-prov-app <login-id>@<target-ip>
```

4. When encountering any dependency issue while building or testing the deployed `oem-prov-app`, attempt to build the `el2go-agent` and `cyaml` recipes, if available. The steps are as follows:

```
$ devtool modify el2go-agent
$ devtool modify cyaml

$ devtool build el2go-agent
$ devtool build cyaml

$ devtool deploy-target cyaml <login-id>@<target-ip>
$ devtool deploy-target el2go-agent <login-id>@<target-ip>
```

5. The following commands build and deploy the `oem-prov-config` recipe, which contains the `oem-prov-app` YAML configuration file and the `oem-prov` unit file. Build and deploy this recipe as follows.

```
$ devtool modify oem-prov-config
$ devtool build oem-prov-config
$ devtool deploy-target oem-prov-config <login-id>@<target-ip>
```

Modify the YAML configuration file and the systemd unit file (if secure provisioning during boot is required) according to the desired provisioning flow. For more details, see Section 9.1 and Section 9.4 respectively.

11 UUID retrieval

The UUID is a unique identifier fused in the NXP SoCs, used to validate against the registered devices in the EdgeLock 2GO backend server and to generate the secure objects unique to a device.

11.1 Read UUID from U-Boot

Boot the i.MX EVK with a pre-built BSP image available at Embedded Linux for i.MX Applications Processors. U-Boot boot log prints the UUID of the device as below.

```
…
mmc0(part 0) is current device
UID: 00112233445566778899aabbccddeeff flash target is MMC:0
…
```

11.2 Read UUID from the oem-prov-app

Execute the `oem-prov-app` as follows:

```
$ oem-prov-app --uuid
…
UUID read option
UUID is 0x00112233445566778899aabbccddeeff
Status(oem-prov-app):SUCCESS
…
```

11.3 Read using SPSDK tool

To use the SPSDK tool, generate the boot image with secure boot configuration enabled in U-boot. Yocto framework can be used to build such image.

To install the SPSDK, follow the instructions here.

1. Set up the Yocto build system.
   To set up the yocto project for a specific release version and machine, see the **Yocto Project Setup** and **Image Build** sections in the *i.MX Yocto Project User's Guide* (document UG10164).
   For example:

```
$ repo init -u https://github.com/nxp-imx/imx-manifest -b imx-linux-scarthgap
 - m imx-6.6.52-2.2.0.xml
$ repo sync
$ DISTRO=fsl-imx-xwayland MACHINE=imx8ulpevk source imx-setup-release.sh -b
 build-xwayland
```

2. Enable Secure Boot in the u-boot package using devtool.

```
$ devtool modify u-boot-imx
$ cd workspace/sources/u-boot-imx
$ echo CONFIG_AHAB_BOOT=y>>configs/imx8ulp_evk_defconfig
$ git add .
$ git commit -m “Enable secure boot config”
$ cd -
```

```
$ devtool update-recipe u-boot-imx //Modifies corresponding recipe
$ devtool reset -n -r u-boot-imx //Resets the devtool workspace
```

Note: *Known issue:* `devtool` *automatically creates a folder called* `u-boot-imx`*, in which patches are stored. Bitbake reports an error while building the recipe since it cannot find the patch.*
Modify this folder name in which the patch is installed:

```
$ mv ../sources/meta-imx/meta-imx-bsp/recipes-bsp/u-boot/u-boot-imx/ ../
 sources/meta-imx/meta-imx-bsp/recipes-bsp/u-boot/files
```

Note: *Choose the appropriate board configuration file instead of imx8ulp_evk_defconfig above.*

3. Build the flash image.

```
$ bitbake imx-boot
```

The built flash image `imx-boot` is available at the *tmp/deploy/images/imx8ulpevk* folder.
Note: *Choose the appropriate MACHINE folder instead of* `imx8ulpevk` *above.*

4. Program the board using the UUU.
   For instructions, see UUU documentation.
   a. Switch boot pins to Serial Download.
   b. Run UUU to boot an updated bootloader.

```
$ uuu imx-boot
```

5. Get UUID via the `NXPELE` app in the SPSDK tool.
   Stop at the u-boot countdown (console) and disconnect the serial connection.
   Use the `nxpele` app to retrieve the UUID.
   For example:

```
$ nxpele -f mx8ulp -p /dev/ttyUSB2 get-info
ELE get info ends successfully:
…
UUID: 0123456789abcdef0123456789abcdef
…
```

Note: *Choose the appropriate -f family and -p port value.*

12 Steps to generate OEM firmware authentication key hash (OEM_RTKH/OEM_SRKH)

12.1 Steps to generate OEM_RTKH/OEM_SRKH using SPSDK tool

To install the SPSDK, follow the instructions here.

1. Generate root key pairs.
   Note: *This step is optional if key pairs are already generated.*
   EdgeLock 2GO enabled MCU and MPU devices require four[2] sets of root key pairs used for the secure boot process. The same keys are used by EdgeLock 2GO for the creation of a secure object.
   Generate four root keys:

```
$ nxpcrypto pki-tree ahab -k <key-type> -o <output-directory>
```

2. Generate `OEM_RTKH/OEM_SRKH`.

```
$ cd <output-directory>/crts
$ nxpcrypto rot calculate-hash -f <SoC family> -k <SRK0>.pem -k <SRK1>.pem -k
 <SRK2>.pem -k <SRK3>.pem -o OEM_SRKH.bin
```

2 Some devices require up to four root keys. For details, see the device specific Security Reference Manual.

13 Steps to connect the board to the Internet

13.1 Connect Ethernet

To connect to the Ethernet, perform the following steps:

1. Connect the Ethernet cable to the board.
2. An IP address is assigned automatically. If not, run the `udhcpc -i eth0` command.

```
root@imx8ulpevk:~# [22114.692597] fec 29950000.ethernet eth0: Link is Up -
 100Mbps/Full - flow control rx/tx
root@imx8ulpevk:~#ifconfig
eth0 Link encap:Ethernet HWaddr 8E:C4:62:DB:E6:67
            inet addr:192.168.1.33 Bcast:192.168.1.255 Mask:255.255.255.0
            inet6 addr: fe80::8cc4:62ff:fedb:e667/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
            RX packets:12 errors:0 dropped:0 overruns:0 frame:0
            TX packets:19 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:1396 (1.3 KiB) TX bytes:2690 (2.6 KiB)
```

13.2 Connect Wi-Fi

To connect to the WI-Fi, perform the following steps:

1. If the target image supports `connman`, follow the steps mentioned in the **Connectivity** section of the corresponding i.MX Linux User's Guide available here.
2. If the moal kernel module is not present (check using `lsmod`), then add the kernel module using the following command:

```
$ modprobe moal mod_para=nxp/wifi_mod_para.conf
```

3. Open the configuration file */etc/wpa_supplicant.conf*, which includes the network details of the local Access Point (AP) that you want to connect to.
   For example, the below configuration file shows the network details of the AP with SSID **NXP_Demo** along with WPA2 security. Replace the `ssid` and `psk` fields with known AP credentials.

```
$ cat /etc/wpa_supplicant.conf
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
update_config=1
network={
    ssid="NXP_EL2GO_Demo"
    key_mgmt=WPA-PSK
    psk="123456789"
}
```

   Note: *Modify `ssid`, `key_mgmt`, and `psk` fields accordingly.*
4. Verify the STA mode interface using the following command:

```
$ ifconfig mlan0 up
$ ifconfig mlan0
mlan0     Link encap:Ethernet HWaddr c0:e4:34:93:a4:63
          UP BROADCAST MULTICAST MTU:1500 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

5. Make sure to stop any running instance of the `wpa_supplicant` to avoid failures. Use the following command to kill the already-running `wpa_supplicant` process. Ignore any error output.

```
$ killall wpa_supplicant
```

6. Connect STA device to AP
   This section describes how to connect the STA device to the external AP using the `wpa_supplicant` utility. The `wpa_supplicant` is a `daemon` program that runs in the background to control the wireless connection.
   Note: *For executing `wpa_supplicant`, the network interface must be available. That means that the physical device must be present and enabled, and the driver for the device must be loaded. The process exits immediately if the device is not available.*

```
$ wpa_supplicant -imlan0 -Dnl80211 -c /etc/wpa_supplicant.conf -B
Successfully initialized wpa_supplicant rfkill: Cannot open RFKILL control
 device
root@imx8ulpevk:~# [28883.311410] wlan: mlan0 START SCAN [28888.348936] wlan:
 SCAN COMPLETED: scanned AP count=3 [28893.361275] wlan: mlan0 START SCAN
[28898.398365] wlan: SCAN COMPLETED: scanned AP count=7
[28898.409171] wlan: HostMlme mlan0 send auth to bssid 9a:XX:XX:XX:93:6c
 [28898.421290] mlan0:
[28898.421310] wlan: HostMlme Auth received from 9a:XX:XX:XX:93:6c
 [28898.433598] CMD_RESP: cmd 0x121 error, result=0x2
[28898.438411] IOCTL failed: 00000000ab8b7cfe id=0x200000, sub_id=0x200024
 action=2, status_code=0x3
[28898.447405] Get multi-channel policy failed
[28898.512561] wlan: HostMlme mlan0 Connected to bssid 9a:XX:XX:XX:93:6c
 successfully
[28898.523859] mlan0:
[28898.523892] wlan: Send EAPOL pkt to 9a:XX:XX:XX:93:6c [28898.558738]
 mlan0:
[28898.558761] wlan: Send EAPOL pkt to 9a:XX:XX:XX:93:6c
[28898.618226] woal_cfg80211_set_rekey_data return: gtk_rekey_offload is
 DISABLE
```

7. Once the connection is established successfully, start the `udhcp` client by using the following command to get the dynamic IP address from the AP and it could be any IP address of the format `xxx.xxx.xxx.xxx`.
   Note: *Wait for some time after executing the following command. On successful execution of the command, it returns to the root prompt after pressing the **Enter** key. If it does not return to root prompt, there is an unexpected issue.*

```
$ udhcpc -i mlan0
udhcpc: started, v1.36.1
Dropped protocol specifier '.udhcpc' from 'mlan0.udhcpc'. Using
 'mlan0' (ifindex=3).
udhcpc: broadcasting discover
udhcpc: broadcasting select for 192.168.1.25, server 192.168.1.1
udhcpc: lease of 192.168.1.25 obtained from 192.168.1.1, lease time 86400
/etc/udhcpc.d/50default: Adding DNS 192.168.1.1
Dropped protocol specifier '.udhcpc' from 'mlan0.udhcpc'. Using
 'mlan0' (ifindex=3).
```

8. Verify the IP address assigned to the STA device interface using the following command.

```
$ ifconfig mlan0
mlan0      Link encap:Ethernet HWaddr 20:4E:F6:D7:B6:53
           inet addr:192.168.1.25 Bcast:192.168.1.255 Mask:255.255.255.0
           inet6 addr: fe80::224e:f6ff:fed7:b653/64 Scope:Link
           UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
           RX packets:28820 errors:0 dropped:0 overruns:0 frame:0
           TX packets:1447 errors:0 dropped:0 overruns:0 carrier:0
           collisions:0 txqueuelen:1000
           RX bytes:3041889 (2.9 MiB) TX bytes:123619 (120.7 KiB)
```

14 Acronyms/Definitions

Table 1 lists the Acronyms and Definitions in this document.

Table 1. Acronyms/Definitions

Acronym	Meaning
CM	Contract Manufacturer/Customer
ELE	EdgeLock Enclave
FW	Firmware
OEM	Original Equipment Manufacturer
SMW	Security MiddleWare
UUID	Universally Unique Identifier

15 References

Table 2. References

Document	Description
AN12642	EdgeLock 2GO Developer Guide
AN12691	EdgeLock 2GO – Managed
AN13255	EdgeLock 2GO Quick Start Guide
EdgeLock 2GO -Managed Service Description	EdgeLock 2GO - Managed Service Description
AN14544	EdgeLock 2GO Services for MPU and MCU
SPSDK	Secure Provisioning SDK
SMW	i.MX Security MiddleWare Project
ELE Library	EdgeLock Enclave User space Library
UUU	Universal Update Utility

16 Note about the source code in the document

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

Copyright 2025 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

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

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

17 Revision history

Table 3 summarizes the revisions to this document.

Table 3. Revision history

Document ID	Release date	Description
AN14741 v1.1	21 October 2025	Update the code in Section 8.1
AN14741 v1.0	06 August 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

Trademarks

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

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

Tables

Figures

Contents