TECHNICAL DOCUMENT

OnBoard™ IoT Security User Manual

Building the trusted foundation for a connected world

Version	V3.2.1
Date	2026-01-15



Version: v3.2.1

Welcome to OnBoard™ IoT Security

Overview

OnBoard™ IoT Security (OBIS) is an authorization-based security platform designed to address the fundamental challenge of scaling secure device provisioning globally while retaining OEM control over cryptographic trust. It is built to protect the integrity, confidentiality, and authorized use of product digital assets. OBIS is delivered as a cloud platform with flexible deployment options to meet different data residency and operational requirements. It is not a PKI service or a factory security system—it is a security control plane that governs how product digital assets may be used across the device manufacturing lifecycle.

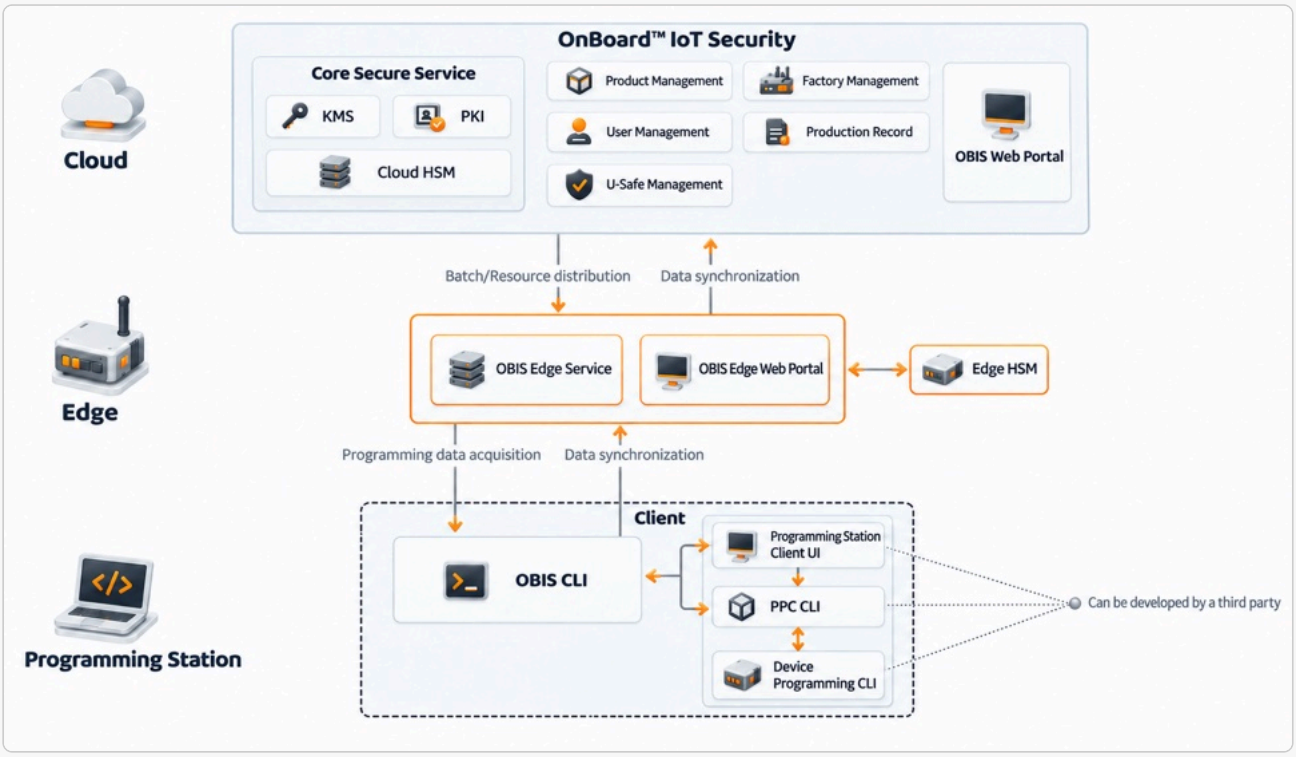
Built on an **authorization-based model** with **centralized control and distributed execution**, OBIS enables OEMs to govern all product digital assets and every security-critical operation across any number of factories and partners, without ever exposing sensitive material.

This architecture enables OEMs to:

- onboard new factories
- switch production partners or silicon suppliers
- scale across regions

All while maintaining centralized control and monitoring. Rather than requiring factories to be trustworthy, OBIS makes trust unnecessary by design.

Product architecture



OBIS consists of three main components: Cloud, Edge, and Programming Station:

Components	Description

Cloud	OnBoard™ IoT Security includes:
	• Core Secure Service, which is responsible for providing fundamental security services and management functions, including:
	  ◦ Key Management System (KMS)
	  ◦ Public Key Infrastructure (PKI)
	  ◦ Cloud Hardware Security Module (HSM)
	• Product Management
	• Factory Management
	• User Management
	• Production Record
	• U-Safe Management
	• OBIS Web Portal: Users can perform product-related operations and management through the website, such as product management, factory management, user management, key management, and certificate management.

Edge	• OBIS Edge Service: Provides on-site production management capabilities for factories. • OBIS Edge Web Portal: Factory managers can manage on-site operations for factories visually through the website. • Edge HSM: A hardware security device used on production line servers.
Programming Station	OBIS CLI (Command Line Interface), Programming Station Client UI, PPC (Programming Process Control) CLI, and Device Programming CLI are provided by the Client, which are used for executing device programming.

Core features

- **Collaborative Digital Asset Management**

Configures and manages digital assets with the "Product" as the central unit, OBIS supports multi-role collaboration such as OEM and ODM. Role-based access control ensures the security of digital assets. Through product configuration and task management, OBIS enables full-process collaboration from modeling to pre-mass production. The main functionalities include:

  - Product modeling: Define and maintain basic product information and parameters.
  - Team collaboration: Invite multiple participants, including cross-tenant members, to collaboratively manage products.
  - Resource management: Unified management of resources such as chip parameters, matter configurations, firmware, certificates, and keys.
  - Production programming management: Standardize each production programming task through version control and batch management.

- **Secure Provisioning**

  Provides an end-to-end secure programming process from Cloud configuration to Edge execution, covering key writing, certificate generation, device programming, secure boot, secure debugging, and other critical steps, ensuring that each device's parameters and data are trustworthy and secure during the production phase.

- **Device Lifecycle Management**

  Supports continuous management of programmed devices, offering capabilities such as device status monitoring and secure firmware updates, enabling full lifecycle visibility, controllability, and traceability from production to deployment.

- **Security Infrastructure**

  Provides standard KMS and PKI services, and combines them with HSM to implement distributed KMS and PKI services at the Edge (factory side), while ensuring end-to-end security across OBIS.

OBIS Positioning and System Boundary

OBIS provides OEMs with a centralized control plane for product digital assets and authorization management—positioned between upstream silicon manufacturing and downstream device lifecycle management.

- **Upstream: Trust Artifact Ingestion**

  OBIS can receive trust artifacts generated during semiconductor manufacturing, such as hardware Root of Trust materials or initial device secrets injected by chip vendors. These artifacts are incorporated into the OEM's product security model and managed consistently alongside other product digital assets within OBIS.

- **Manufacturing: Production Batch Execution**

  OBIS issues explicit Production Batches to OEM-owned or partner-operated factories, defining what may be produced, where, when, and at what scale.

- **Downstream: Trust Artifact Handoff**

  OBIS collects all production outcomes—including per-device security artifacts and immutable production records—and hands them off to OEM-managed or OEM-authorized device management platforms, enabling secure device onboarding and supporting downstream lifecycle operations.

  OBIS is not a factory execution system, a device management platform, or a replacement for MES—it is a centralized product digital asset management and authorization-driven security governance layer—enabling OEMs to coordinate security across all manufacturing partners through a single control plane.

Role classification and permission description

Supports role-based permission management. The current roles are divided into three categories: OBIS roles, product roles, and factory roles. The following six tables describe each role and the corresponding operational permissions.

OBIS roles

OBIS roles are predefined by the system to meet common permission requirements. The following roles can accommodate most usage scenarios, and users can configure one or more roles as needed. For special permission control requirements, please contact Snowball to create corresponding custom roles.

Roles	Description
System Manager	This role has permissions for all roles.
Product Manager	This role has permissions to view and operate the Product Management functional pages.
Resource Manager	This role has permissions to view and operate the KMS and PKI functional pages.
Factory Manager	This role has permissions to view and operate the Factory Management functional pages.
Normal User	This role has permissions to view all functional pages.

The following table summarizes the access and operational permissions of each OBIS role across different functions.

Functions	Permissions	System Manager	Product Manager	Factory Manager	Resource Manager	Normal User
Product	Page Access	✓	✓			✓
	Page Operations	✓	✓			
Factory	Page Access	✓		✓		✓
	Page Operations	✓		✓		
Records	Page Access	✓	✓	✓	✓	✓

Functions	Permissions	System Manager	Product Manager	Factory Manager	Resource Manager	Normal User
KMS	Page Access	✓			✓	✓
	Page Operations	✓			✓	
PKI	Page Access	✓			✓	✓
	Page Operations	✓			✓	
Account	Page Access	✓				✓
	Page Operations	✓				
Group	Page Access	✓				✓
	Page Operations	✓				

Product roles

Product roles are designed to control access and operational permissions of product resources. Users can configure one or more roles as needed.

Roles	Description
Product Owner	This role defaults to the product creator, with full operational permissions for the product.

Roles	Description
Product Manager	This role has full access to all product operations.
Resource Manager	This role has permissions to view the product and manage its resources.
Version Manager	This role has permissions to view the product and manage its versions.
Batch Manager	This role has permissions to view the product and manage its batches.
Member	Default user role, with permission to view the product.

The following table summarizes the operational permissions of each role for the Product function.

Functions	Operations	Product Owner	Product Manager	Resource Manager	Version Manager	Batch Manager	Member
Product	Configure a product	✓	✓	✓			
	Check products	✓	✓	✓	✓	✓	✓
	Manage products	✓	✓				
	Manage members	✓	✓				
	Create a version	✓	✓		✓		
	Check versions	✓	✓	✓	✓	✓	✓

Functions	Operations	Product Owner	Product Manager	Resource Manager	Version Manager	Batch Manager	Member
	Manage versions	✓	✓		✓		
	Create a batch	✓	✓			✓	
	Check batches	✓	✓	✓	✓	✓	✓
	Check programmed devices	✓	✓	✓	✓	✓	✓

Factory role

The factory role is designed to manage and handle factory-related business, and can be configured (local factory & cloud factory) by users as needed.

Role	Description
Factory Manager	This role has full access to all factory operations on both Cloud and Edge.

The following table summarizes the operational permissions of the Factory Manager on both Cloud and Edge.

Components	Functions	Factory Type	Operations	Factory Manager
Cloud	Factory	Local factory & Cloud factory	Create a factory	✓
		Local factory & Cloud factory	Configure and manage factories	✓
		Local factory & Cloud factory	Configure and manage programming stations	✓

Components	Functions	Factory Type	Operations	Factory Manager
		Cloud factory	Manage batches and assign programming stations	✓
		Local factory & Cloud factory	Check factories	✓
Edge	Factory	Local factory	Manage factories	✓
	Station	Local factory	Configure and manage programming stations	✓
	Batch	Local factory	Manage batches and assign programming stations	✓

Version: v3.2.1

Register and sign in to OnBoard™ IoT Security

Welcome to Snowball OnBoard™ IoT Security (OBIS)! This article will guide you to quickly complete the OBIS account registration and sign-in, so that you can start using the OBIS functions immediately. Please follow the steps below:

Step one: Register an OBIS account

Please contact us with the following information, and we will assist you in completing your OBIS account registration:

- Company Name
- Name
- Email

After you submit the information, we will contact you as soon as possible to complete the registration process.

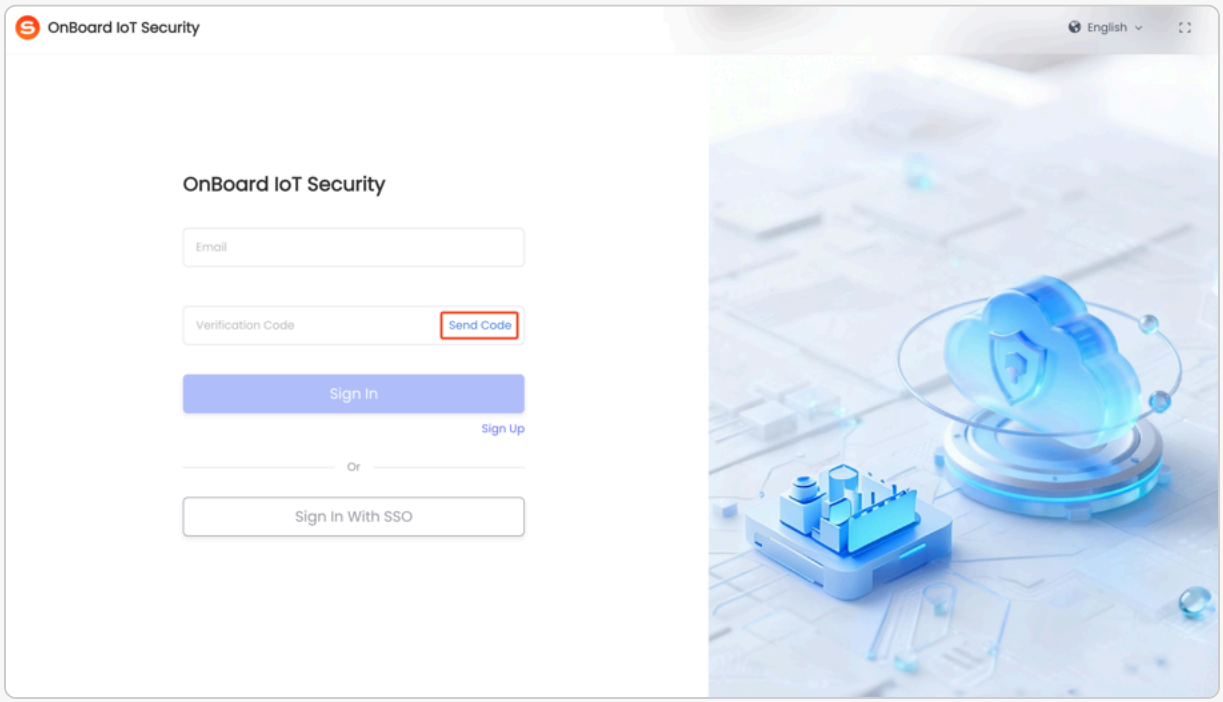
Step two: Sign in to OBIS

After the account is registered, you can sign in using either a verification code or the SSO (Single Sign-On) method:

Method one: Verification code

> **TIP**
>
> You can contact Snowball to enable the enterprise **Self-Registration** feature, allowing users to sign up accounts on the OBIS sign-in page using their company email and sign in with a verification code. OBIS will automatically assign default role permissions to them.

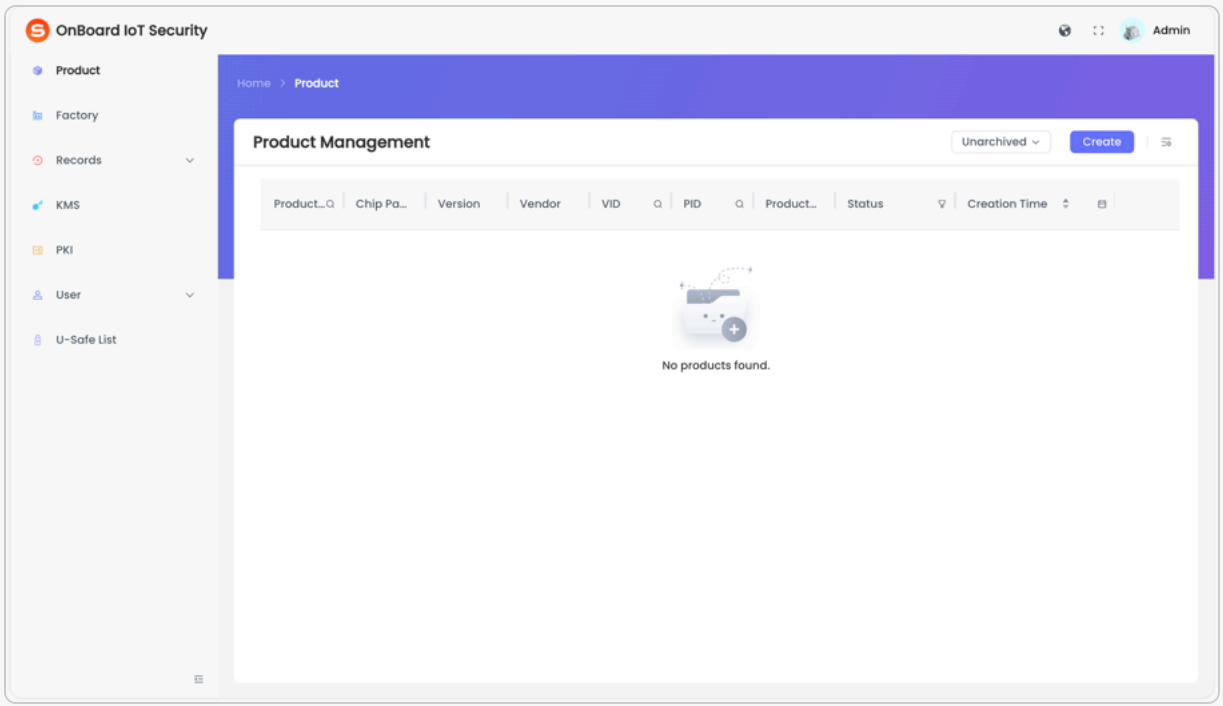
1. On the OBIS sign-in page, enter the email you provided when registering for the OBIS account, and then click **Send Code**.



2. Sign in to your email, check the verification email from OBIS which contains a six-digit verification code.

3. On the OBIS sign-in page, enter the six-digit verification code you received in your email, and then click **Sign In**.

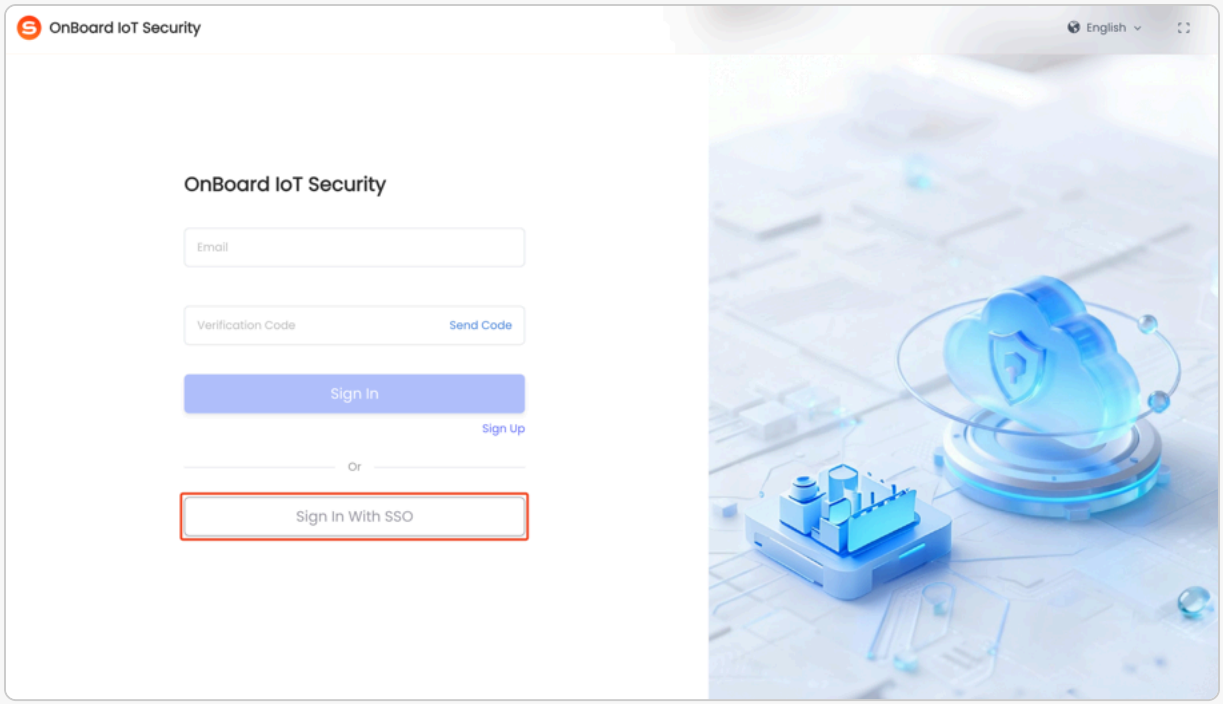
4. After a successful sign-in, the page will navigate to the OBIS home page.



Method two: SSO

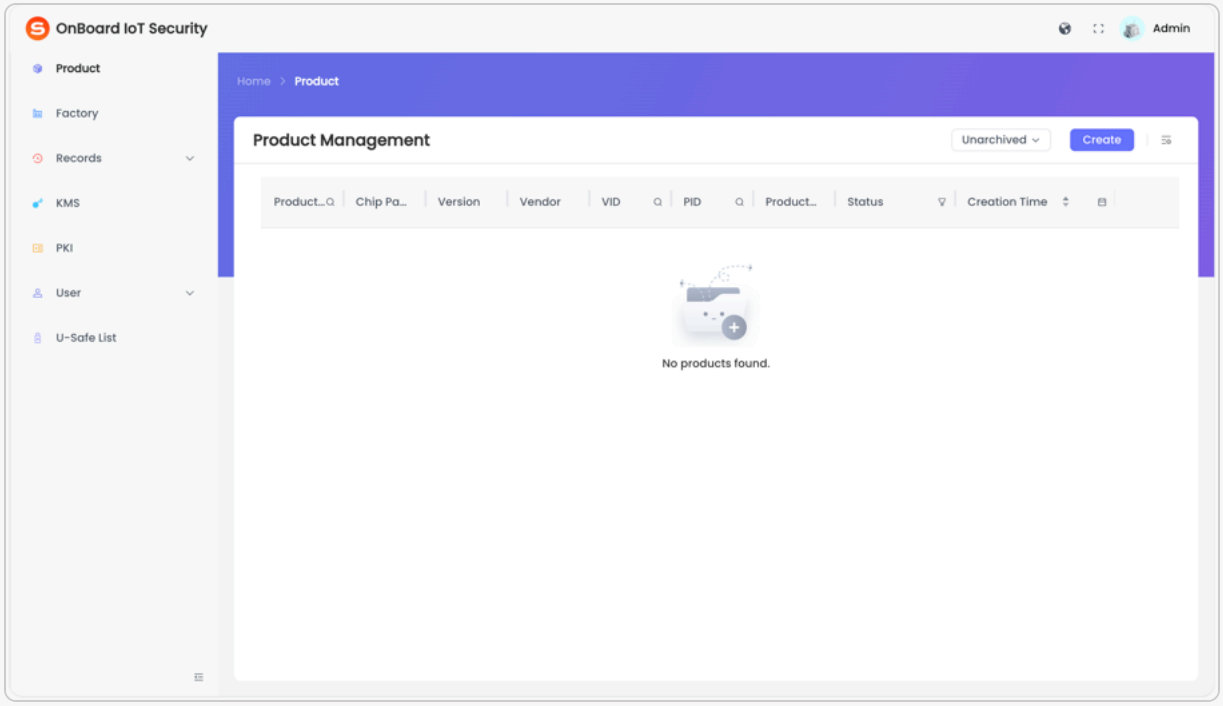
> **TIP**
>
> You can contact Snowball to enable the **Sign in with SSO** feature, allowing users to sign in with their company email through the Enterprise Identity Provider (IdP). OBIS will automatically create an account for them and assign default role permissions. For more information, refer to Configure Sign in with SSO.

1. On the OBIS sign-in page, select **Sign In With SSO**.



2. Enter your company email and then click **Next**, the page will navigate to the enterprise IdP authentication page.

3. After successful authentication, the page will navigate to the OBIS home page.


OnBoard IoT Security
Admin
Product
Factory
Records
KMS
PKI
User
U-Safe List
Home
Product
Product Management
Unarchived
Create
Product...
Chip Pa...
Version
Vendor
VID
PID
Product...
Status
Creation Time
No products found.

Version: v3.2.1

Product creation and device programming

Welcome to OnBoard™ IoT Security (OBIS)! This article provides a quick-start guide for the following steps:

- Creating and configuring a product
- Creating a version and batch
- Assigning a programming station to the batch and completing device programming

By following these steps, you'll quickly get started with OBIS and complete the entire process—from product creation to device programming.

Prerequisites

Before starting product creation and device programming, ensure that the following conditions are met:

- Your account has the permissions to complete the process—from product creation to device programming in OBIS.
- Contact Snowball to obtain the HSM U-Safe and Identity U-Safe.
- You have completed the creation and configuration of the factory. For detailed instructions, refer to the following steps:

- Creating a factory
- Configuring a factory (local factory & cloud factory)
- Configuring a programming station (local factory & cloud factory)

Operational diagram

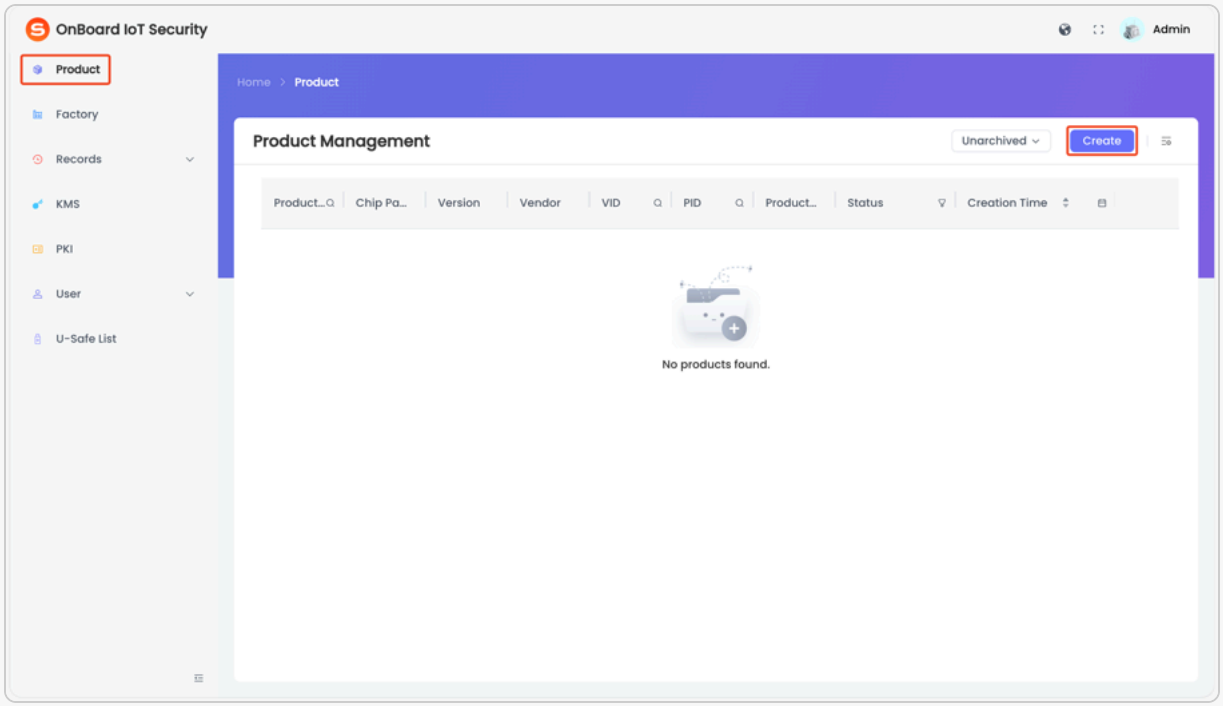


Step one: Create and configure a product

Refer to the steps below to create and configure your product in OBIS:

1. Sign in to OBIS.

2. In the left menu, click **Product**, and then click **Create**.

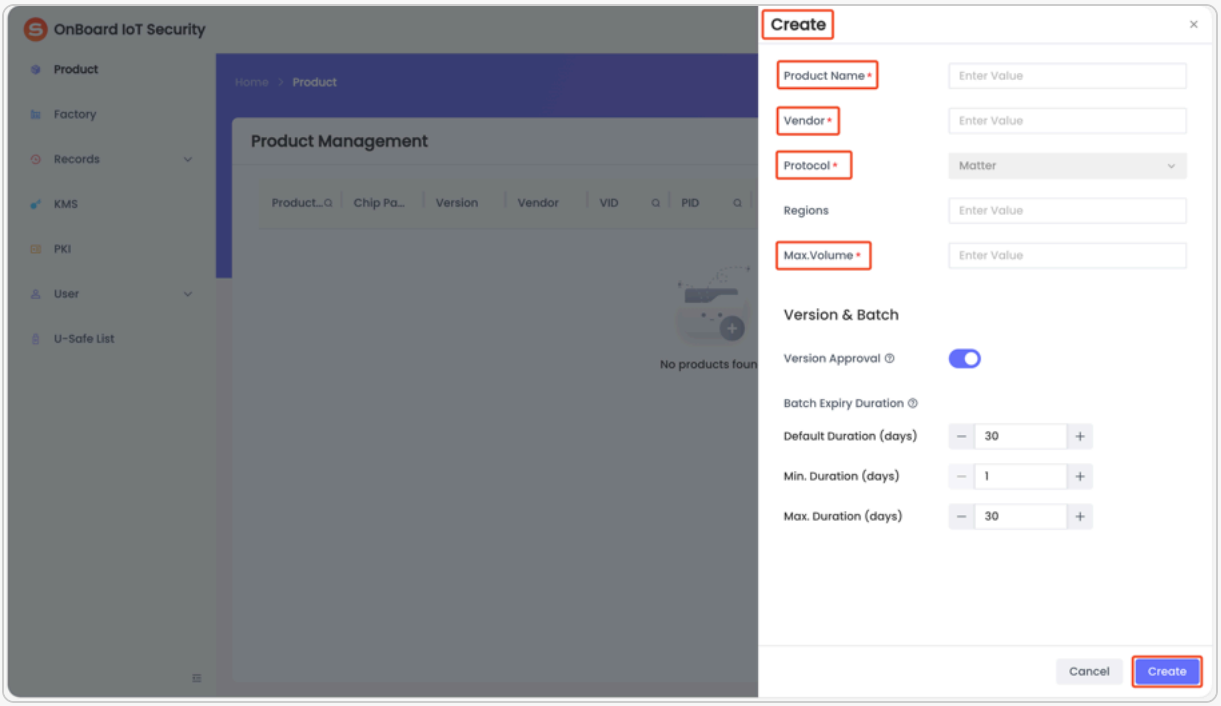


3. On the **Create** panel, enter the required basic information:

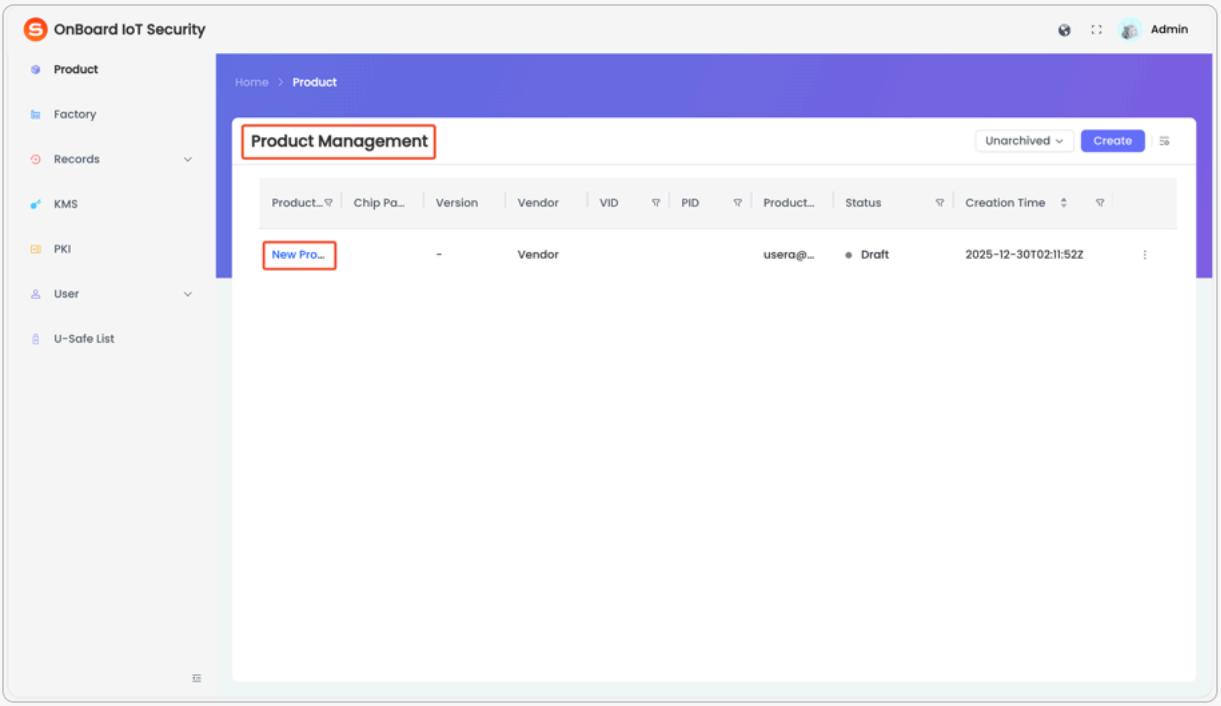
    - **Product Name**
    - **Vendor**
    - **Protocol**
    - **Max.Volume**

    Then, click **Create**. After your product is created, the page will navigate to the **Product Management** page.

> **NOTE**
>
> For more information, refer to the description of product basic information.



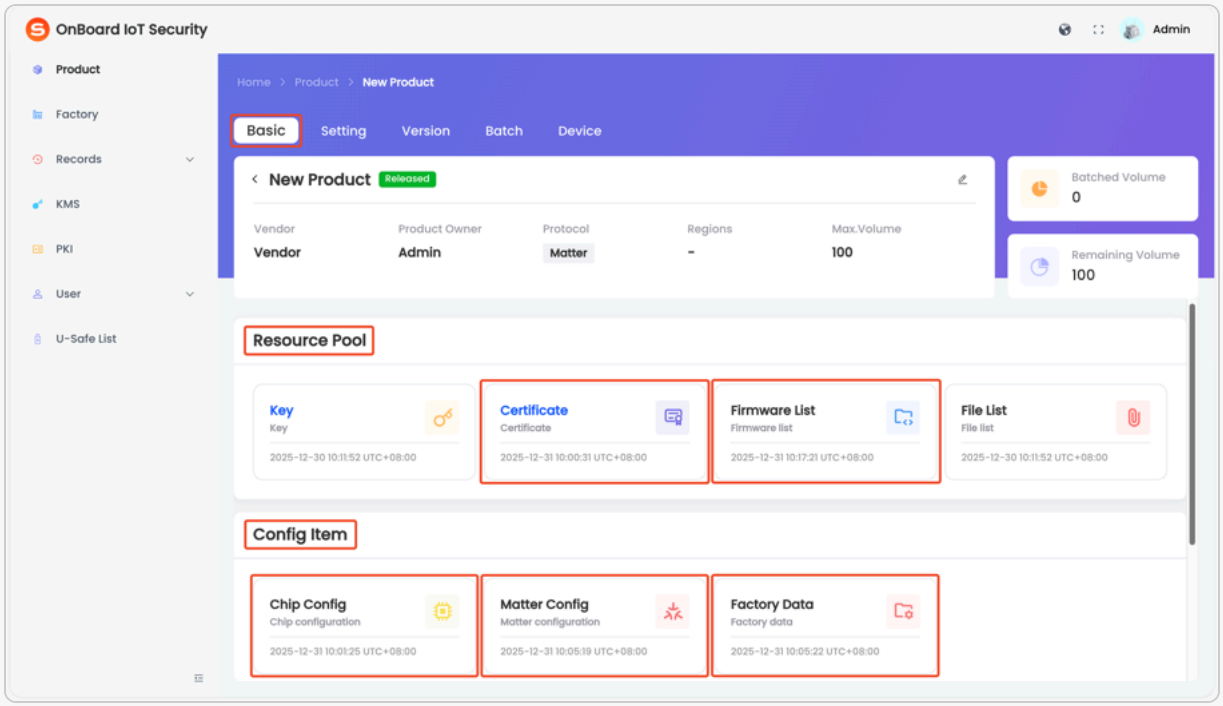
4. On the **Product Management** page, select the created product.



5. On the product page, select the **Basic** tab, locate the **Resource Pool** and **Config Item** areas. Then, select and configure the required items:

    - **Certificate**
    - **Firmware List**
    - **Chip Config**
    - **Matter Config**
    - **Factory Data**

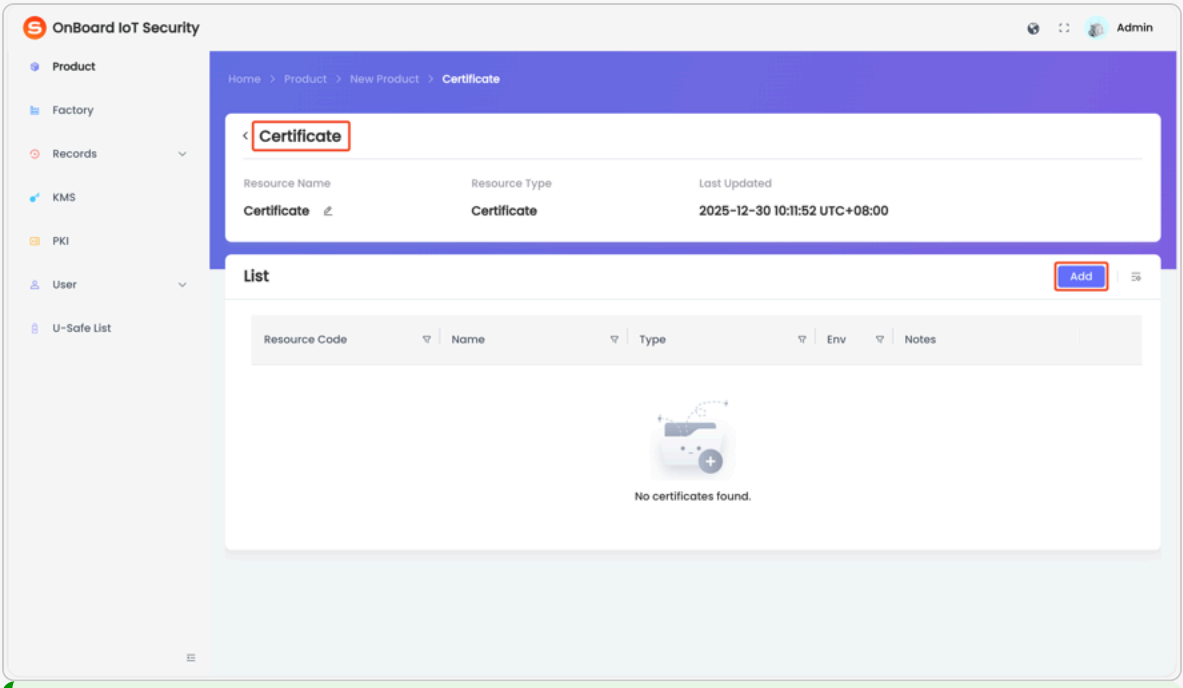
    > **TIP**
    >
    > The configurations of the **Resource Pool** and **Config Item** areas will be used for version creation. For more information, refer to the steps for configuring a product. The files uploaded when configuring Firmware List, Matter Config, and Factory Data can be downloaded at any time from the configuration page of the corresponding resource.



Certificate | Firmware List | Chip Config | Matter Config | Factory Data

> **TIP**
>
> Certificate allows you to add a DAC template from PKI to your product, making it available for selection during the Matter Config.
>
> DAC templates can be created in PKI. For more information, refer to the steps for creating a certificate.

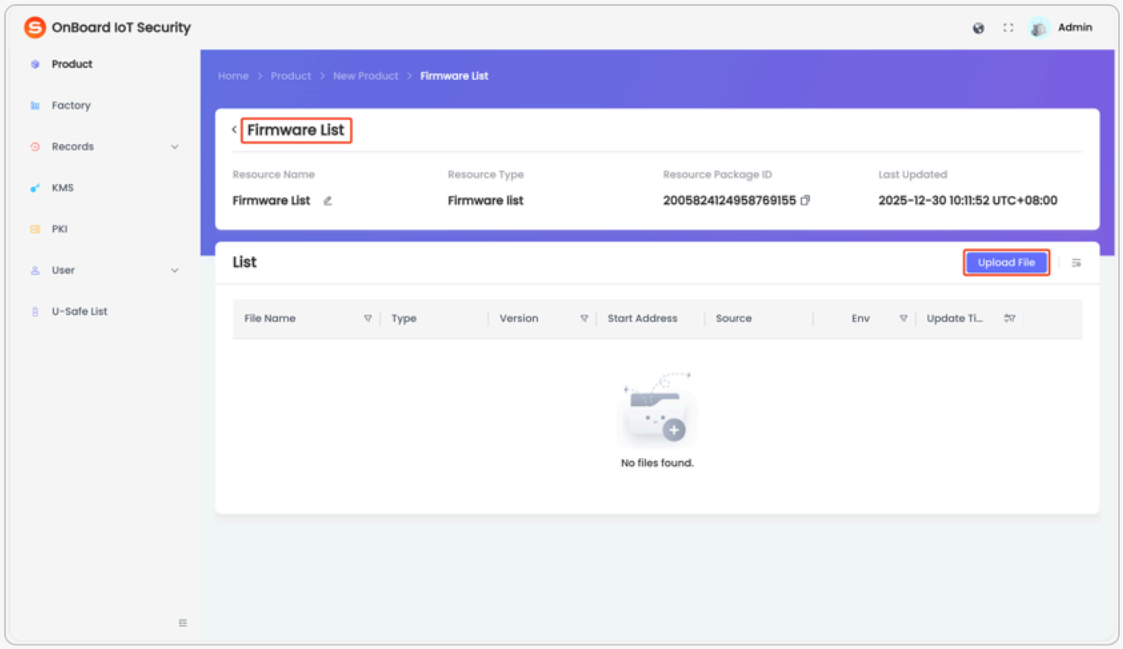
On the **Certificate** page, click **Add**. Then, select and add the **CA Template**.



TIP

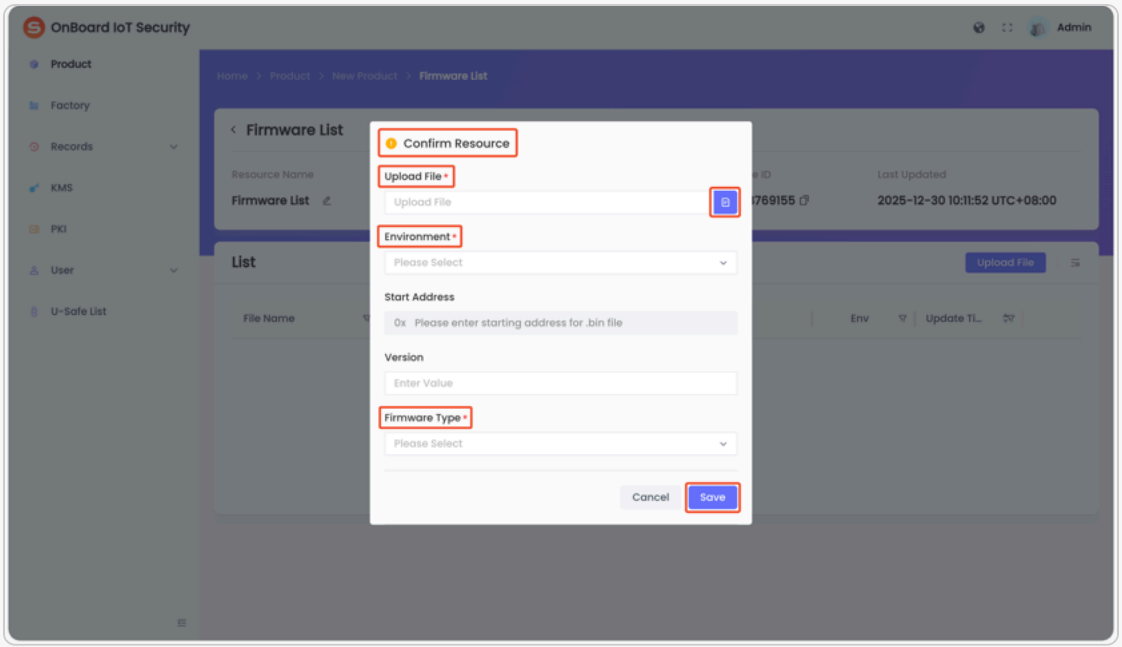
Firmware List allows you to upload the firmware files required for programming the product. Firmware formats supported: **.bin**, **.hex**, and **.s37**. Then, the uploaded firmware can be selected when creating a version.

i. On the **Firmware List** page, click **Upload File**.



ii. In the **Confirm Resource** window, click the File icon to upload the firmware files. Then, select the **Environment** (**TEST** or **PROD**) and **Firmware Type** (**Application** or **Bootloader**), and click **Save**.

> **NOTE**
>
> For more information, refer to the description of Firmware information.



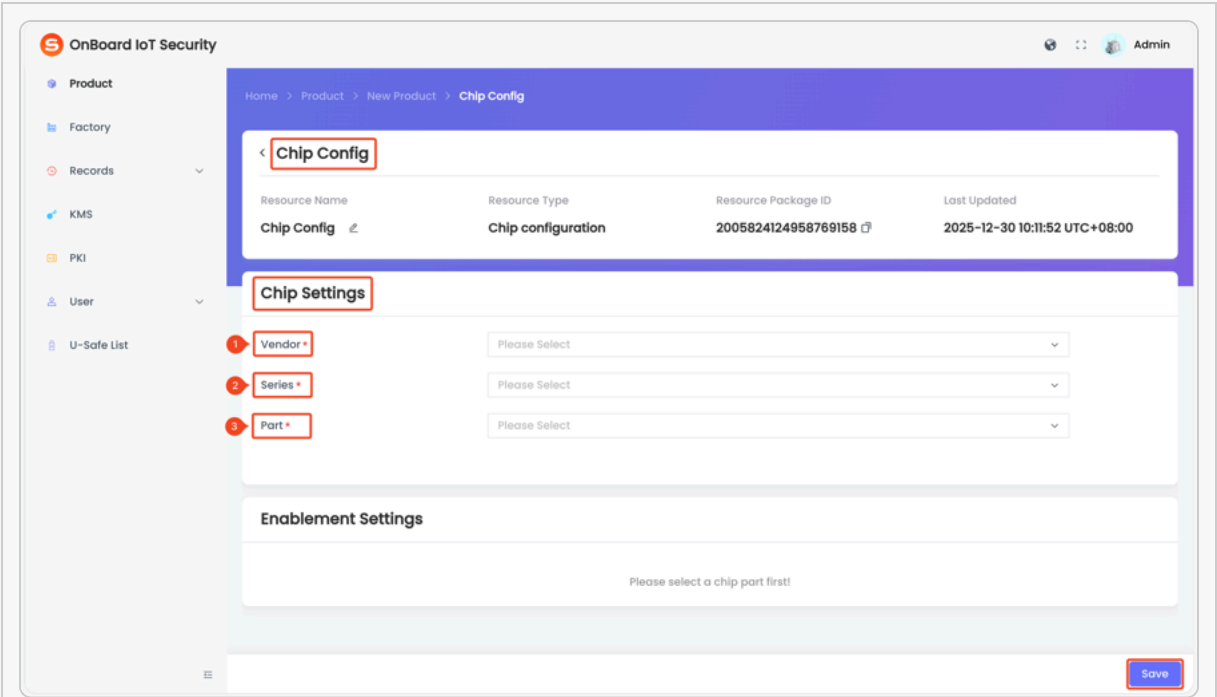
TIP

Chip Config allows you to select the chip part and security options used by your product.

Please refer to the supported chip settings for the current list. Additional chip vendors, series, and parts will be added based on demand.

For more information, refer to the steps for setting Chip Config.

On the **Chip Config** page, locate the **Chip Setting** area, select the **Vendor**, **Series**, and **Part**, and then click **Save**.



TIP

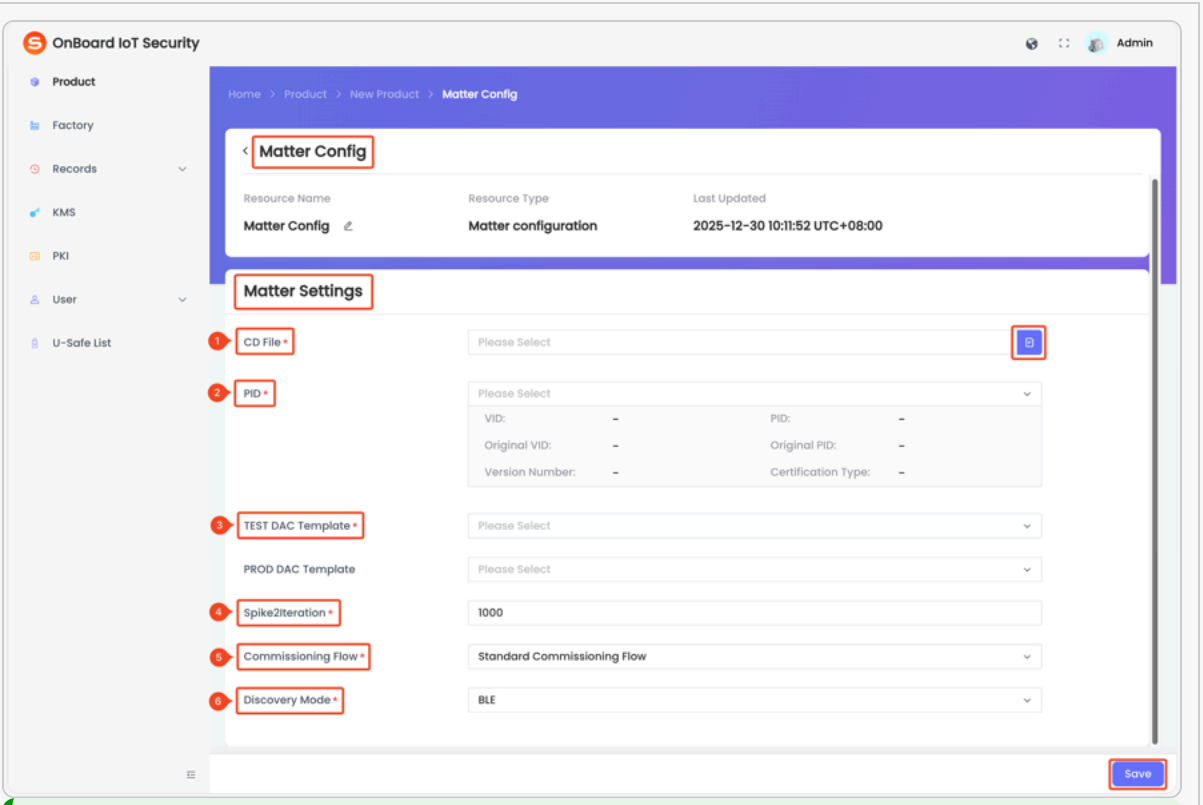
Matter Config allows you to select and configure settings related to the Matter protocol.

In this step, you need to upload the CD file and select the DAC template that was added when configuring Certificate. Please ensure that the VID (Vendor ID) and PID (Product ID) in the CD file match those in the DAC template.

You can replace the CD file as needed when creating a version later.

For more information, refer to the description of Matter Settings information.

On the **Matter Config** page, locate the **Matter Settings** area, click the File icon to select and upload the CD file. Then, select **PID**, **TEST DAC Template**, **Spike2Iteration**, **Commissioning Flow**, and **Discovery Mode**, and click **Save**.

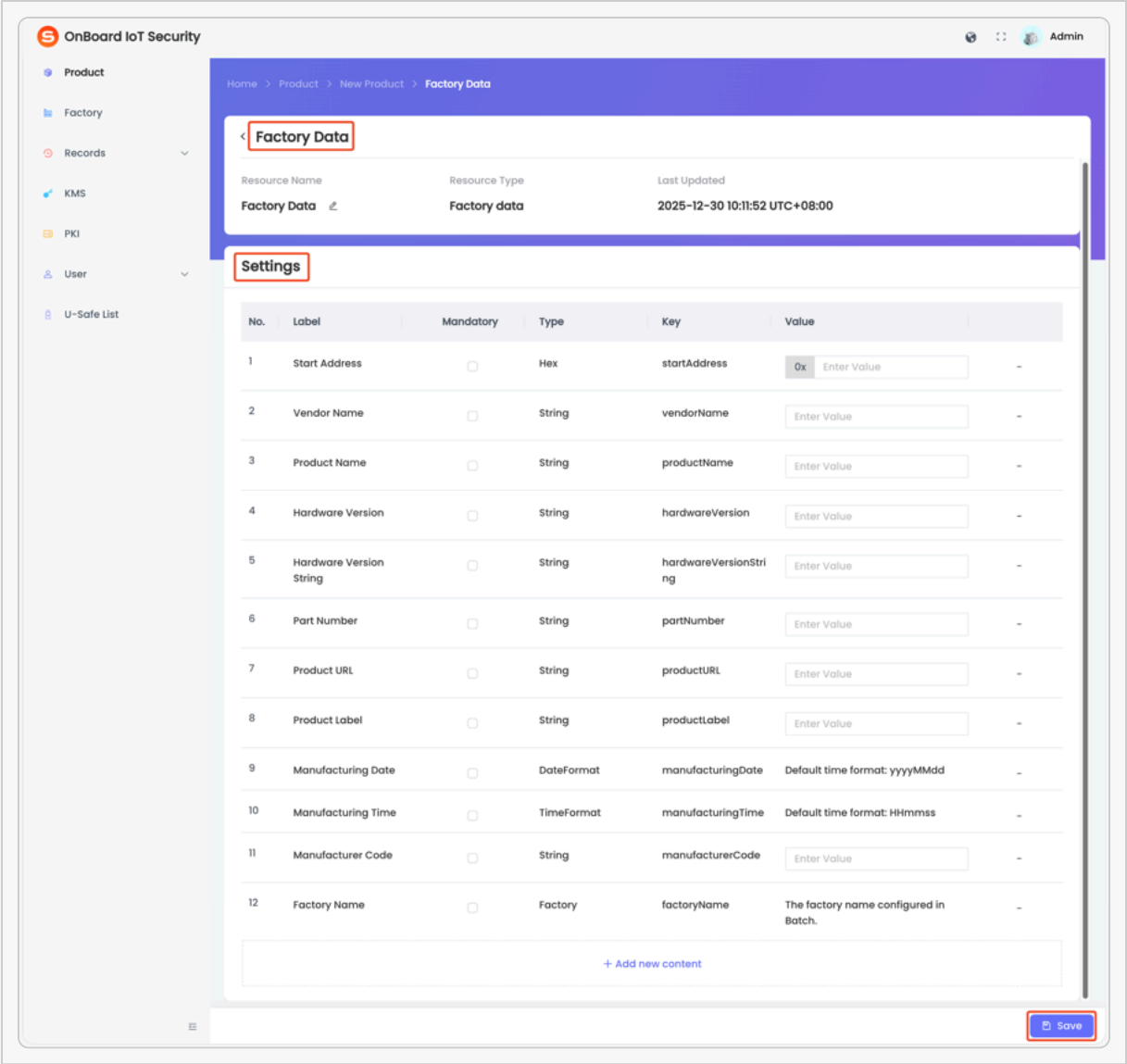


TIP

Factory Data allows you to configure factory-related information and supports all fields required by the Matter specification. You can update the Factory Data as needed when creating a version later.

The Factory Data selected here will be used as a required field when creating a version. For more information, refer to the description of Factory Data information.

On the **Factory Data** page, locate the **Setting** area, enter the Factory Data for your product (values can be left empty), and then click **Save**.

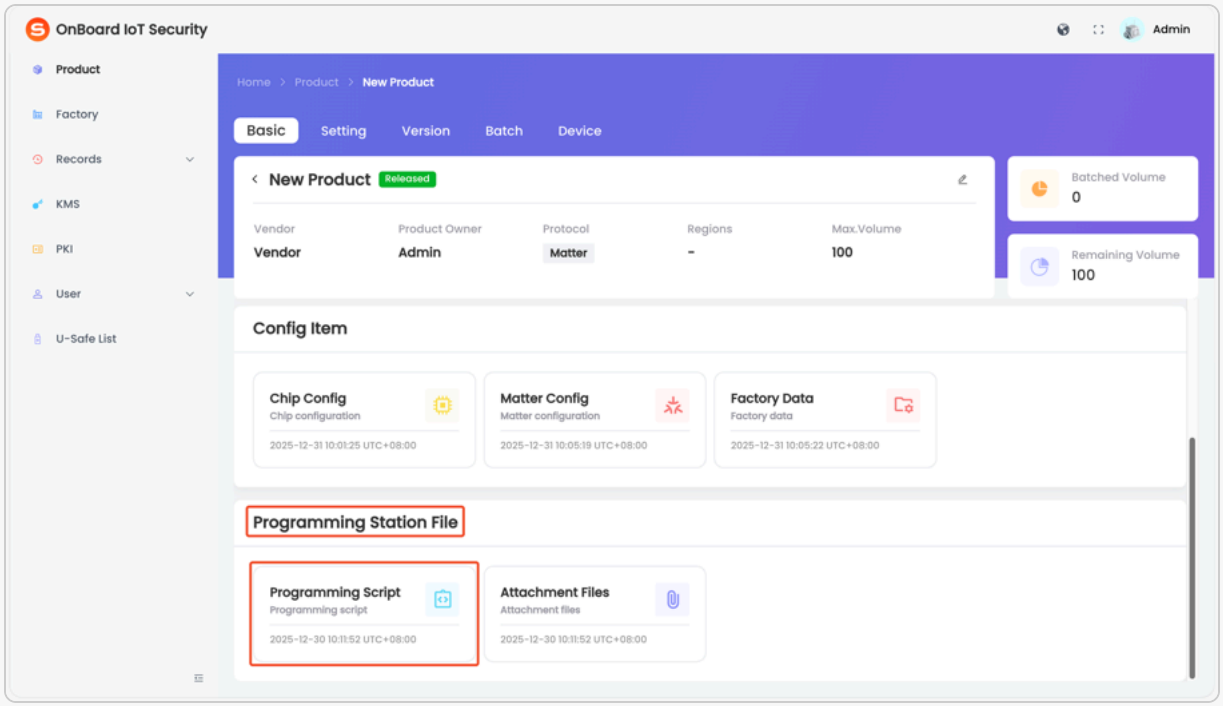


6. On the product page, locate the **Programming Station File** area and upload the corresponding files.

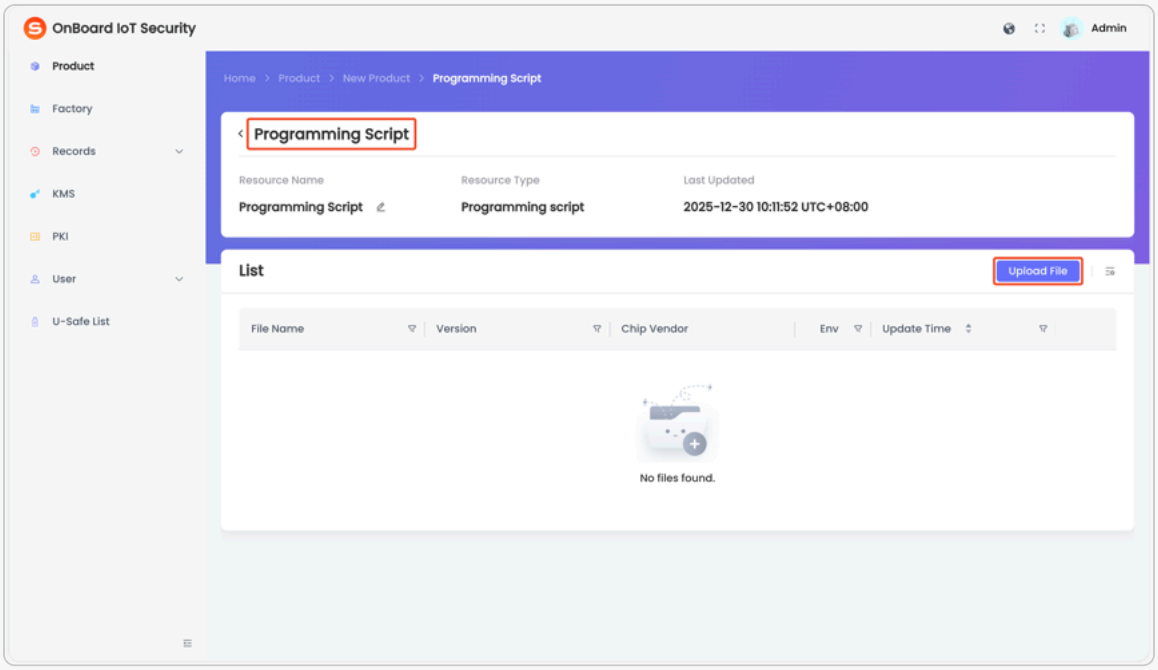
> **NOTE**
>
> Files in the **Programming Station File** area will be used for batch creation. For more information, refer to the steps for uploading Programming Station files. The files uploaded when configuring Programming Station File can be downloaded at any time from the configuration page of the corresponding resource.

For example, refer to the steps below to upload a programming script.

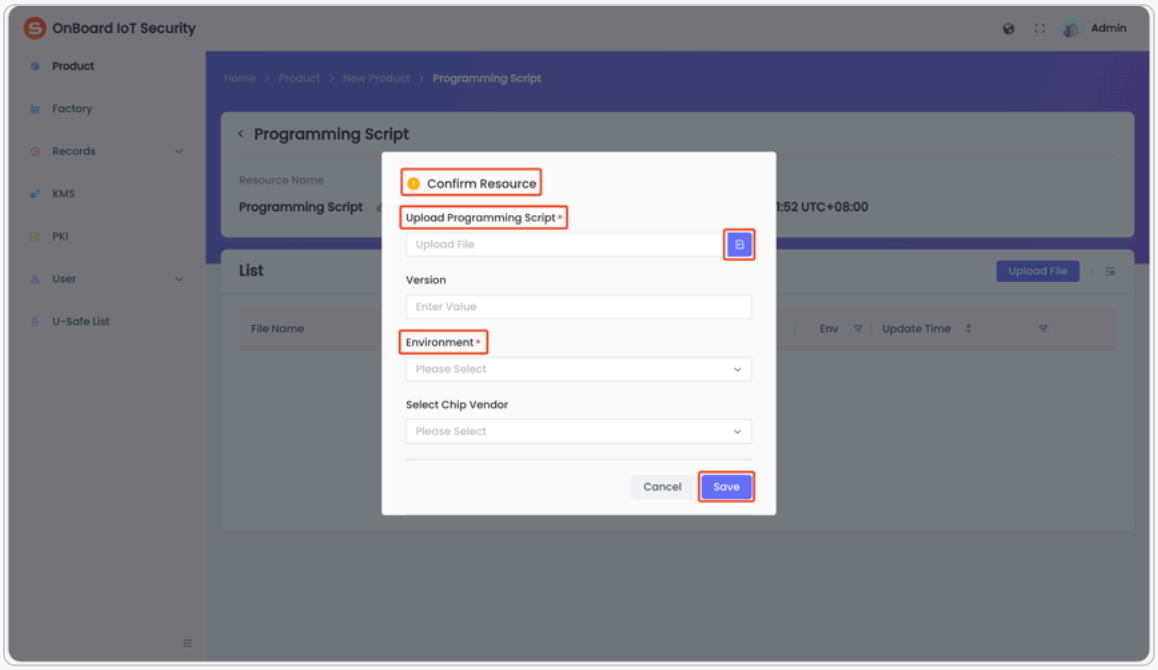
> **TIP**
>
> **Programming Script** allows you to upload script files used for device programming (any script format is supported). The uploaded programming script can be selected when creating a batch.
>
> You can select and upload the programming script as needed. If it is not required at the moment, you can also select it for a specific batch later in the Programming Station client.



i. On the **Programming Script** page, click **Upload File**.



ii. In the **Confirm Resource** window, click the File icon to upload the script file. Then, select **Environment** (**TEST** or **PROD**) and click **Save**.



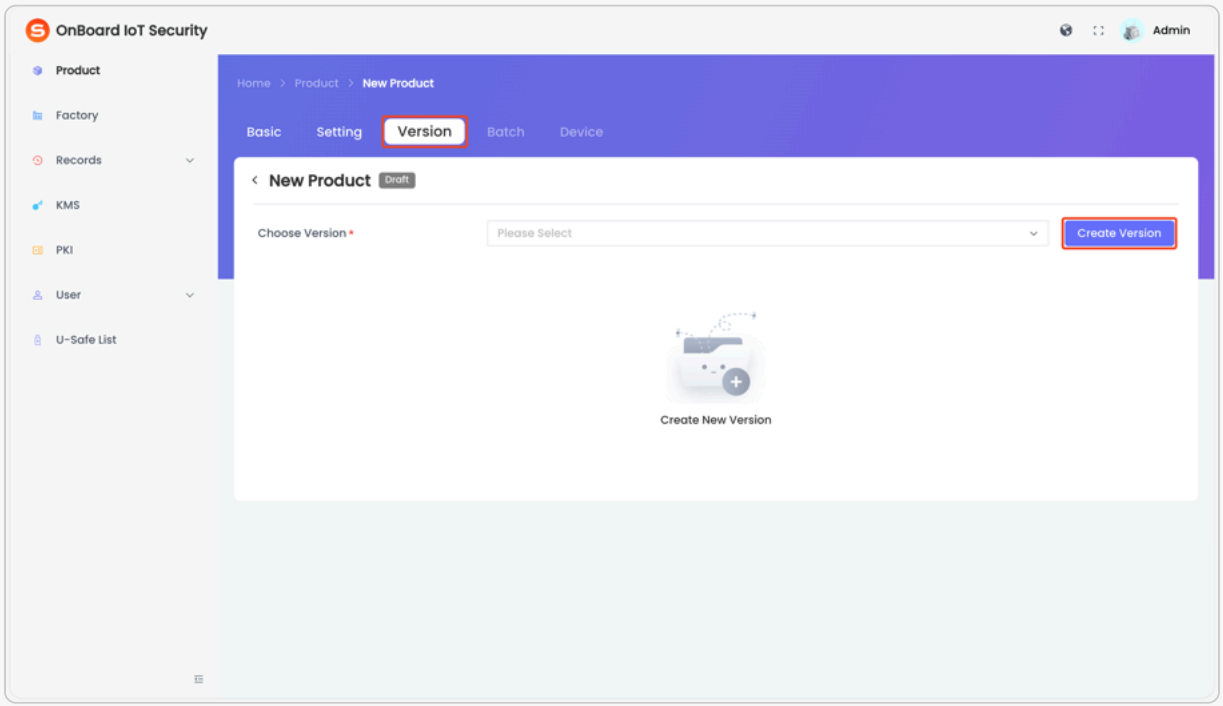
Step two: Create a version

> **TIP**
>
> Each product can create multiple versions.

After your product is created and configured, refer to the steps below to create a version in OBIS:

1. On the product page, select the **Version** tab, and then click **Create Version**. The page will navigate to the **Create Version** tab.

> **TIP**
>
> If the **Version** tab appears grayed out, it is currently unavailable. Please check whether all required resources and configuration items for your product have been configured. Once everything is properly configured, the **Version** tab will be enabled automatically.



2. On the **Create Version** tab:

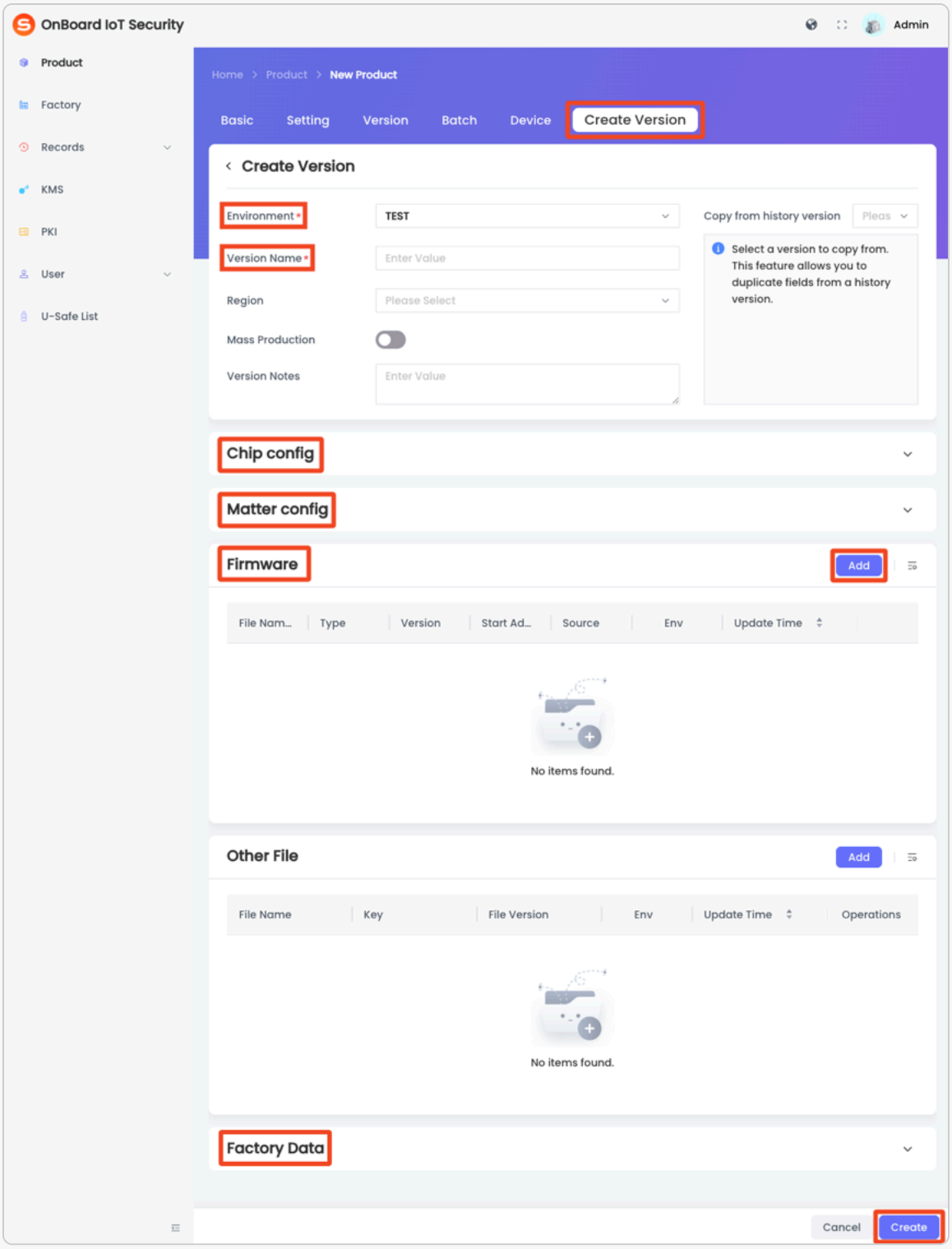
> **NOTE**
>
> For more information, refer to the description of version information.

   i. Select the **Environment**, and then enter the **Version Name**.

   ii. Check the information in the **Chip config**, **Matter config**, and **Factory Data** areas.

iii. In the **Firmware** area, click **Add**, then select and add the firmware uploaded when configuring your product.

iv. Click **Create**. After the version is created, the page will navigate to the **Version** tab.


OnBoard IoT Security
Admin
Product
Factory
Records
KMS
PKI
User
U-Safe List
Home > Product > New Product
Basic
Setting
Version
Batch
Device
Create Version
Create Version
Environment *
TEST
Copy from history version
Pleas
Version Name *
Enter Value
Select a version to copy from. This feature allows you to duplicate fields from a history version.
Region
Please Select
Mass Production
Version Notes
Enter Value
Chip config
Matter config
Firmware
Add
File Nam...
Type
Version
Start Ad...
Source
Env
Update Time
No items found.
Other File
Add
File Name
Key
File Version
Env
Update Time
Operations
No items found.
Factory Data
Cancel
Create

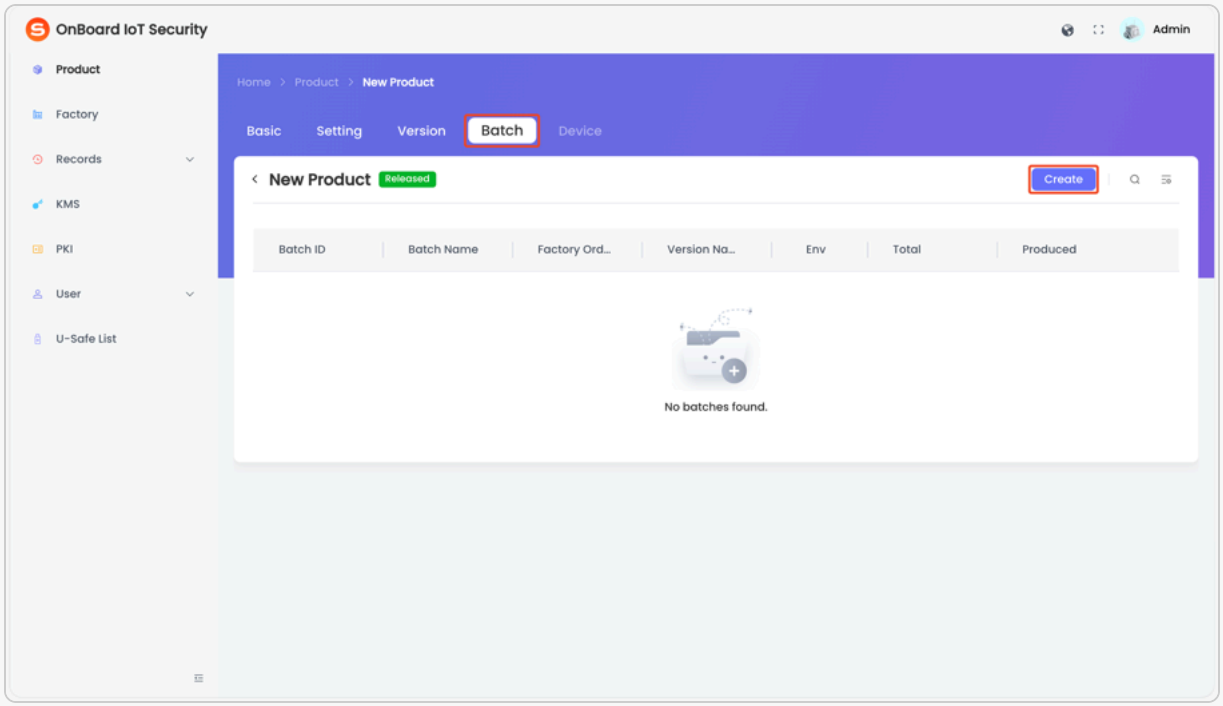
Step three: Create a batch

> **TIP**
>
> Each version can create multiple batches, and a batch represents a production plan based on that version.

After the version is created, refer to the steps below to create a batch in OBIS:

1. On the product page, select the **Batch** tab, and then click **Create**. The page will navigate to the **Create Batch** tab.

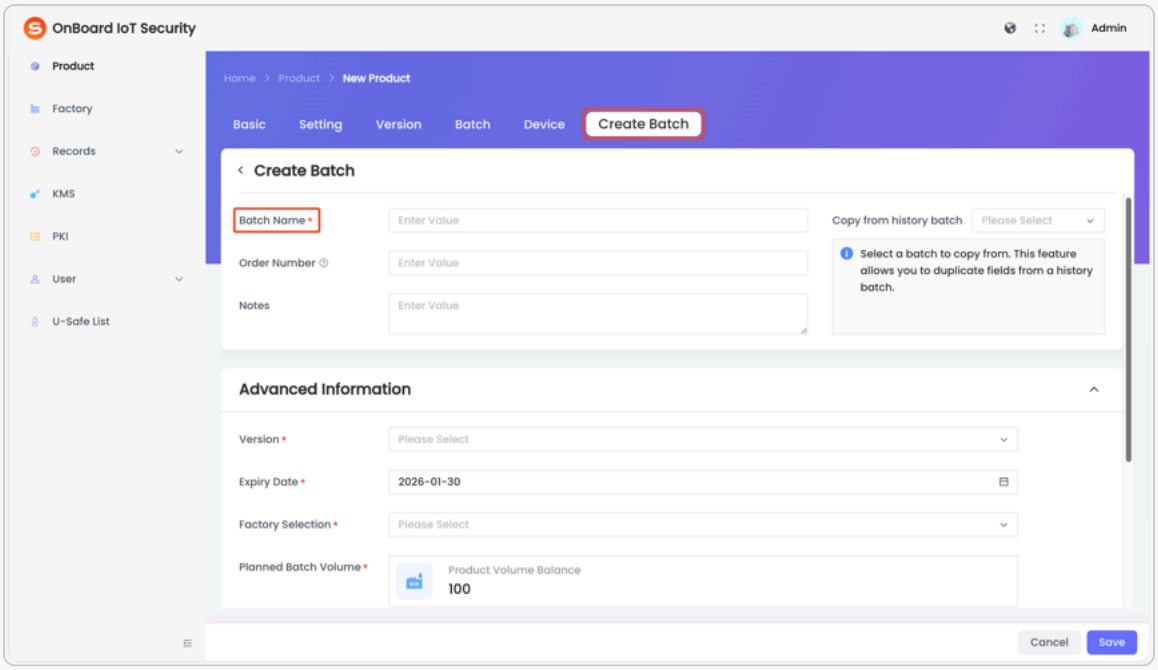
   > **TIP**
   >
   > If the **Batch** tab appears grayed out, it is currently unavailable. Please check whether a version has been created. After a version is created, the **Batch** tab will be enabled automatically.



2. On the **Create Batch** tab:

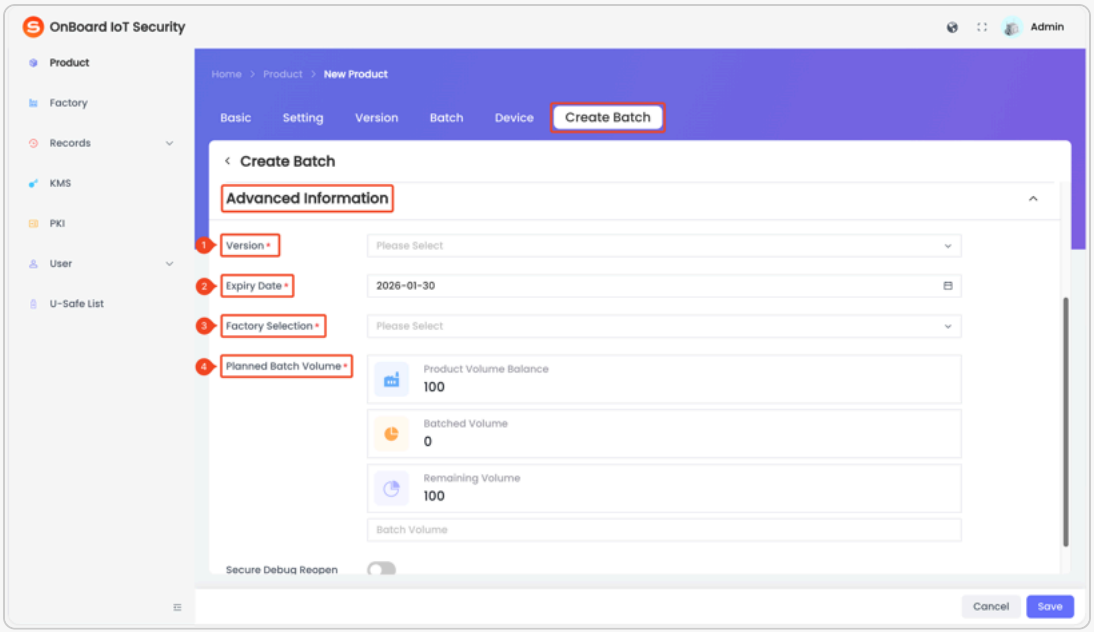
> **NOTE**
>
> For more information, refer to the description of batch information.

   i. Enter the **Batch Name**.

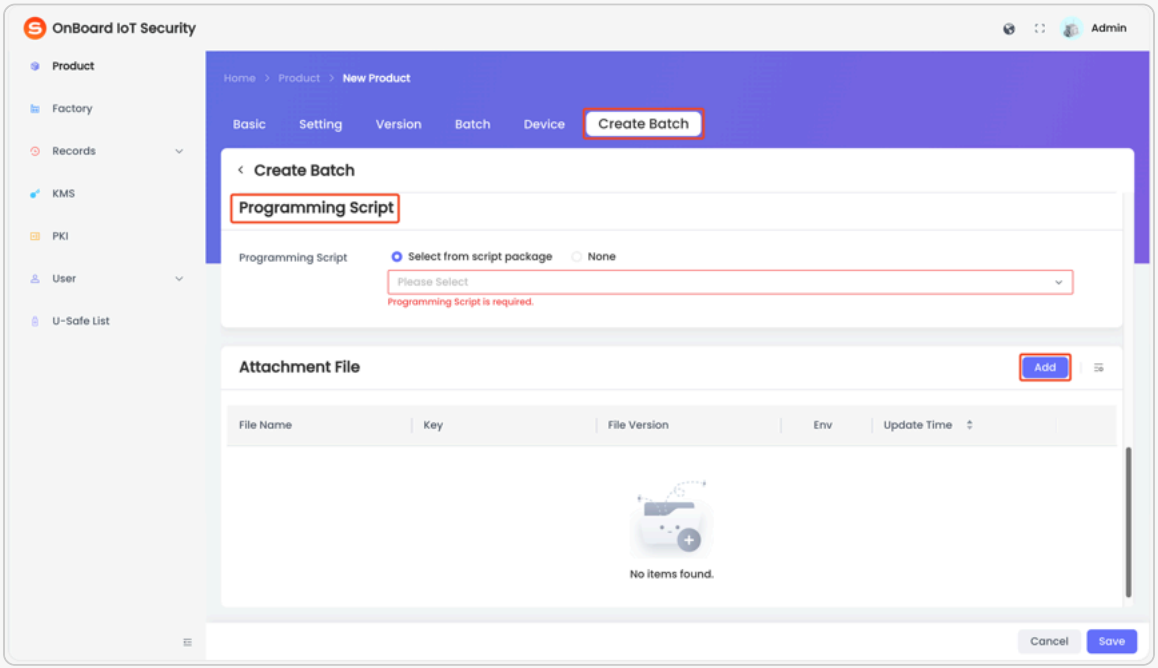


ii. In the **Advanced Information** area:

- Select the version created in step two
- Set the **Expiry Date**
- Select the factory
- Enter the **Planned Batch Volume**



iii. In the **Programming Script** area, select the programming script uploaded when configuring your product as needed. If it is not required at the moment, you can select **None** and select the programming script for a specific batch later in the Programming Station client.



iv. Click **Save**. After the batch is created, the page will navigate to the **Batch** tab.

Step four: Assign a programming station to the batch

> **TIP**
>
> The purpose of assigning a programming station to a batch is to bind the production plan to a programming station.

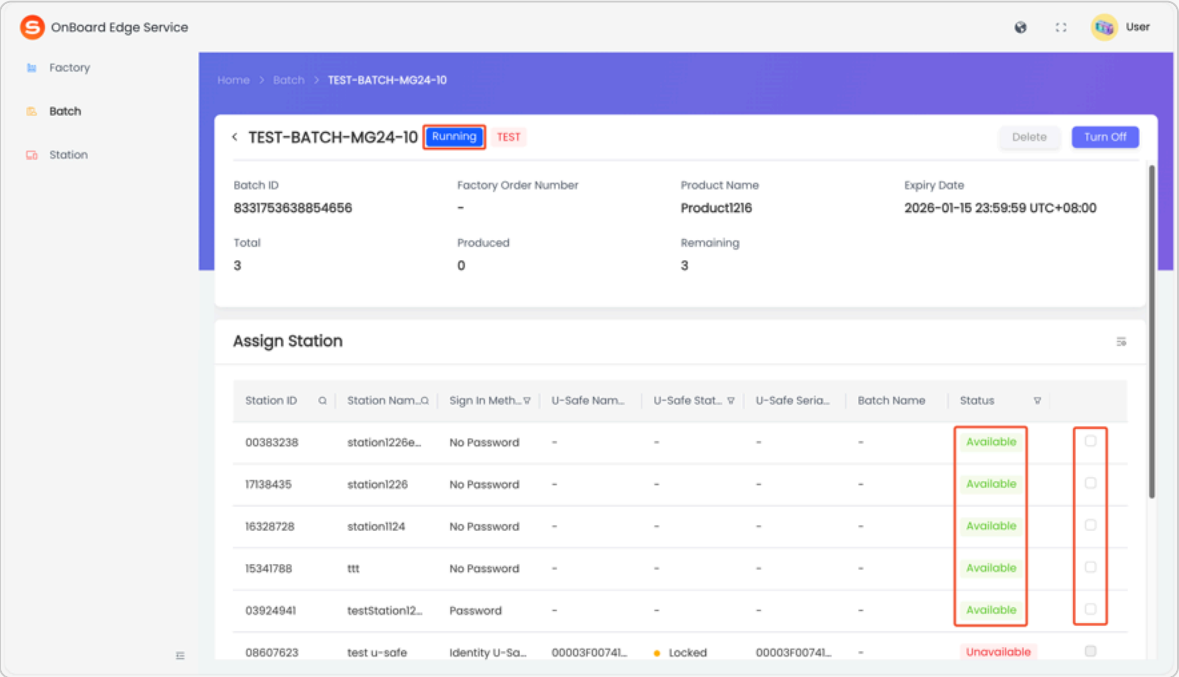
Local Factory | Cloud Factory

Contact Snowball for assistance in deploying the OnBoard Edge Service on the factory side.

After the deployment is complete, you can sign in to the OnBoard Edge Service using the generated service address to manage factories, programming stations, and batches:

1. Sign in to OnBoard Edge Service.
2. In the left menu, click **Batch**, and then select the batch created in step three from the **Batch List**.
3. On the batch page, verify that the batch status is **Running**, and then select a programming station with the **Available** status for assignment.

> **NOTE**
>
> For more information, refer to the steps for adding a programming station.

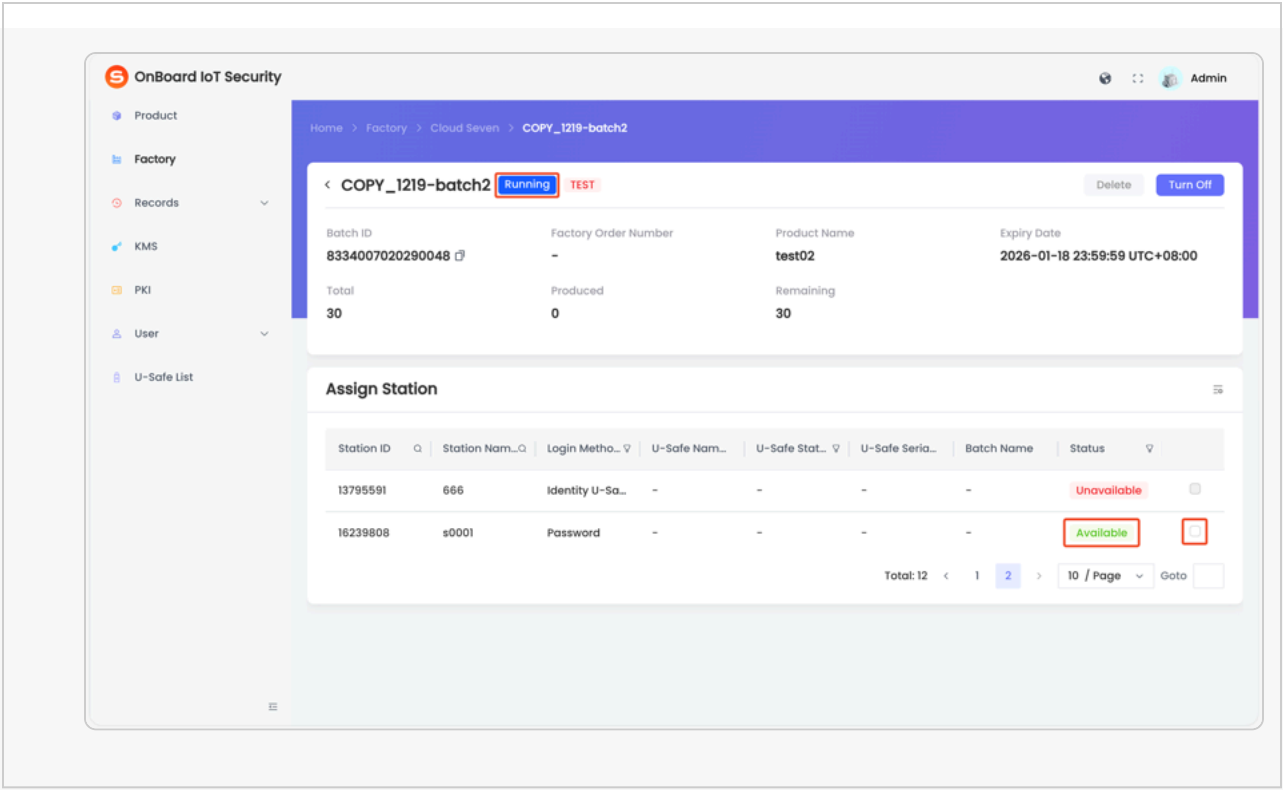


1. Sign in to OBIS.

2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page and select the **Batch** tab.

3. Select the batch created in step three from the **Batch List**, verify that the batch status is **Running**, and then select a programming station with the **Available** status for assignment.

> **NOTE**
>
> For more information, refer to the steps for adding a programming station.

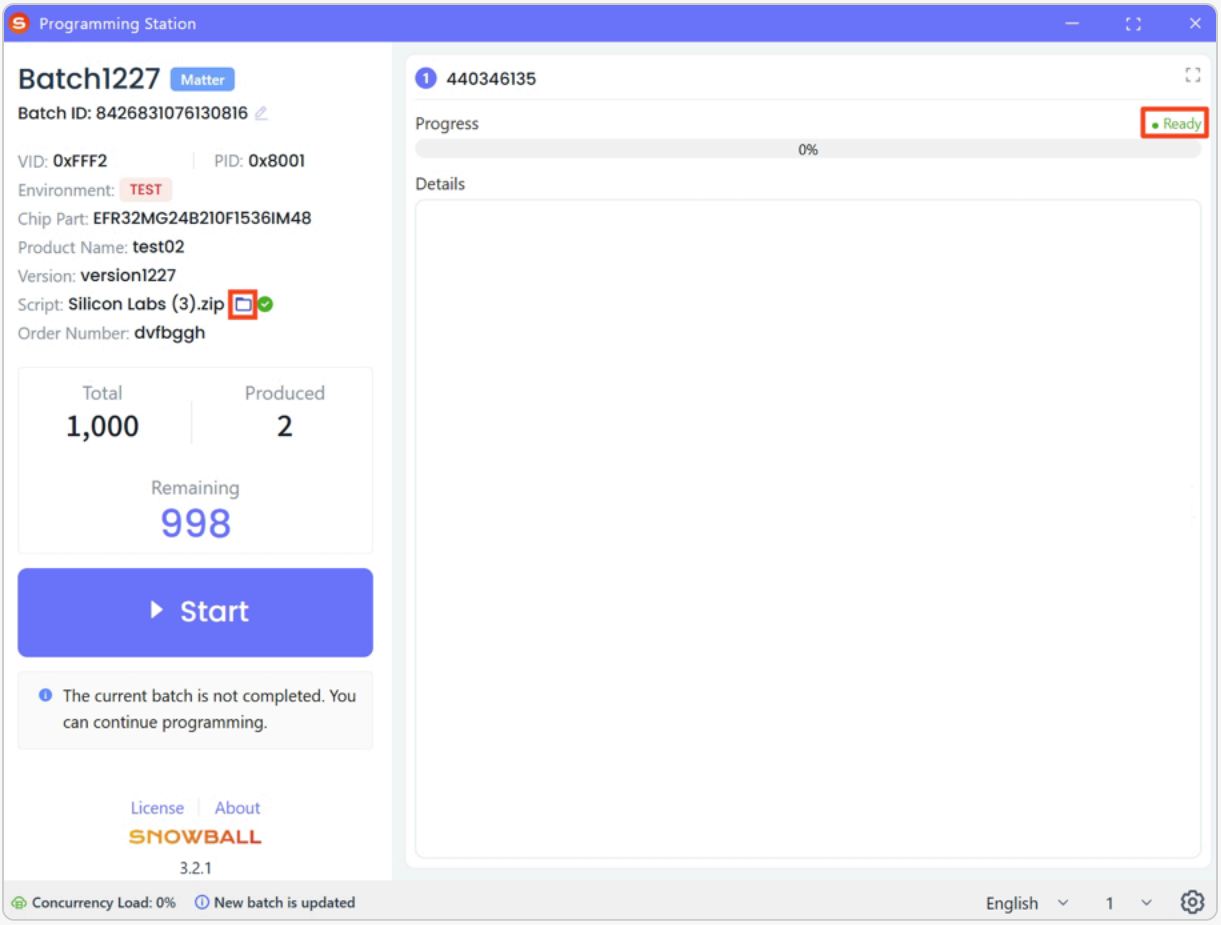


Step five: Device programming

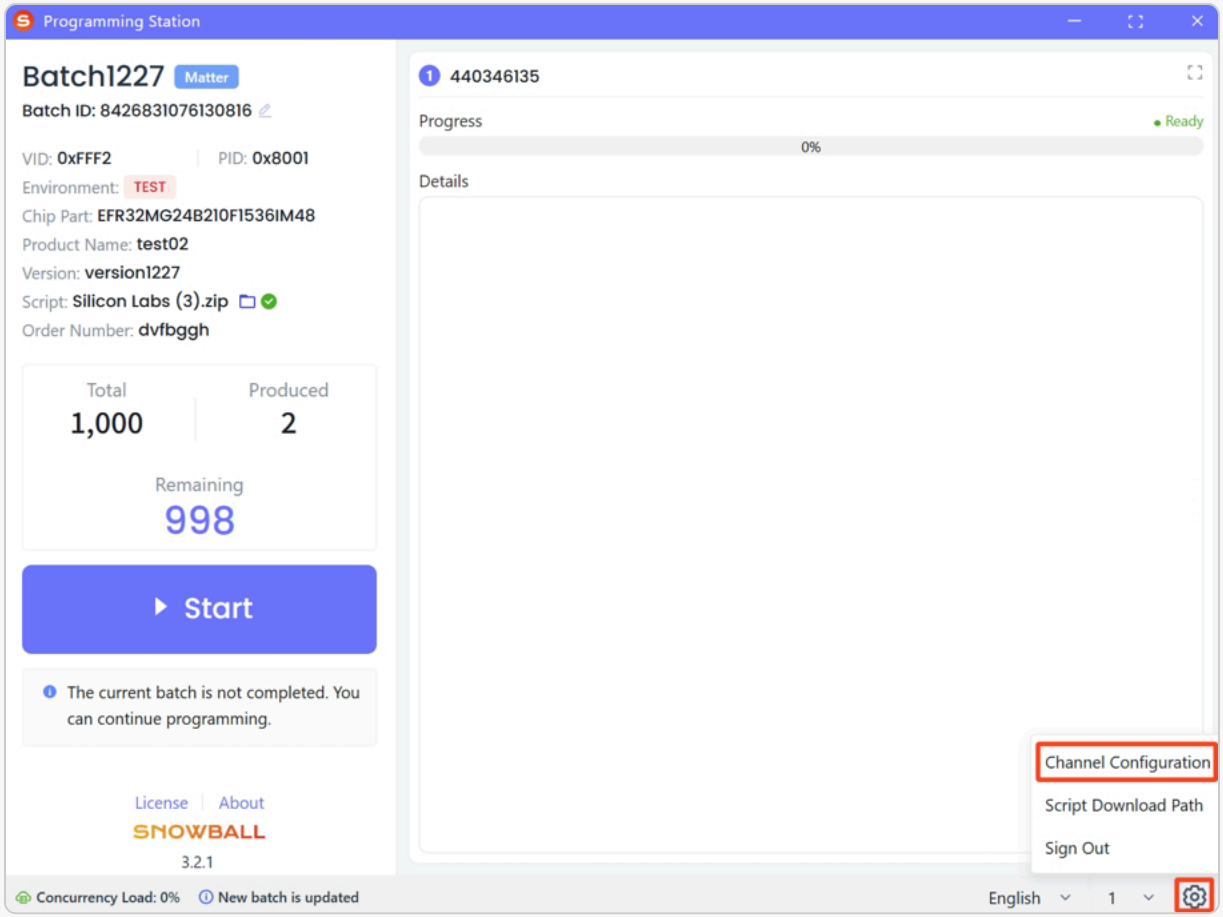
Programming Station is a client for device programming. After assigning a programming station to the batch in the OnBoard Edge Service, refer to the steps below to start programming:

1. Install and sign in to the Programming Station client, verify that the device status is **Ready**.

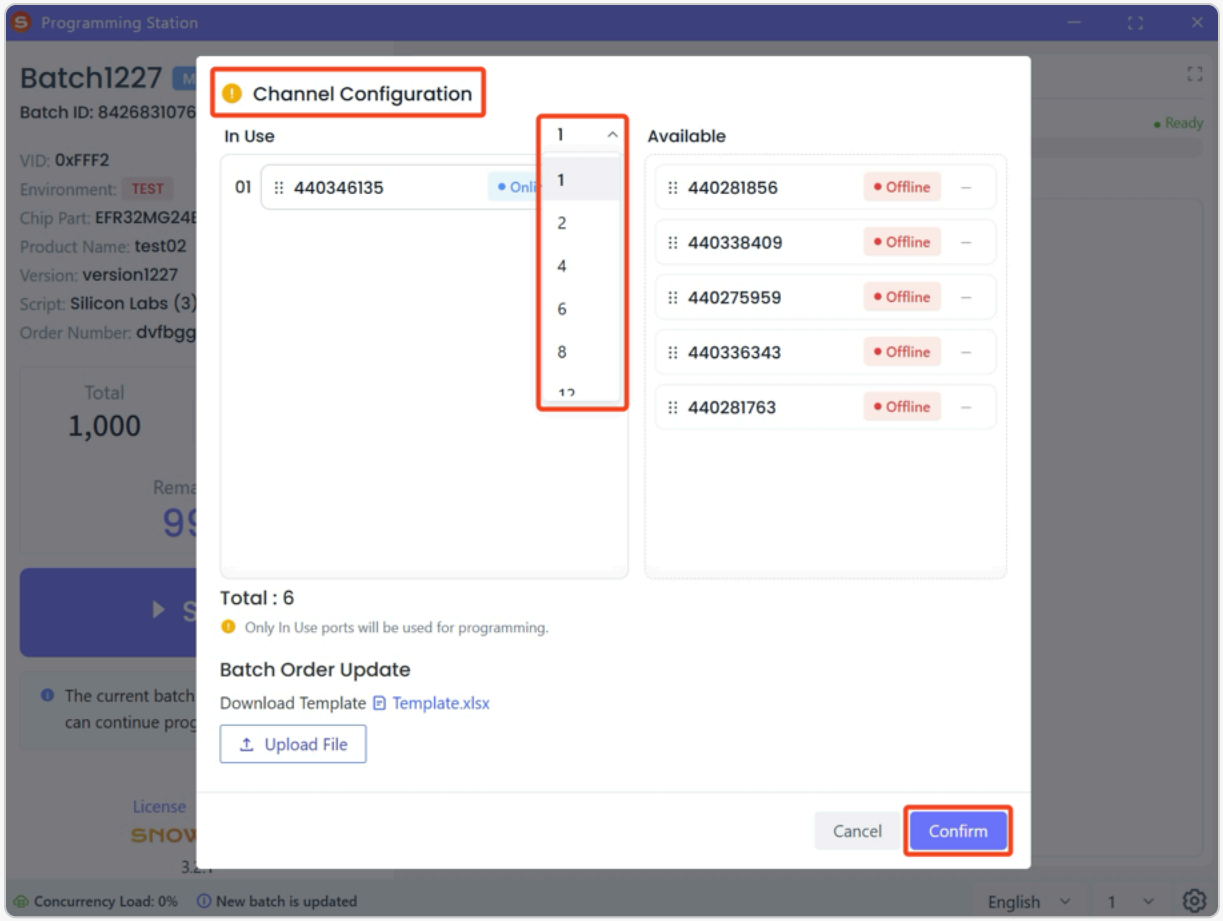
> **TIP**
>
> If the programming script is not selected when creating a batch, or if you want to select a different programming script, you can click the Folder icon to select the required programming script (any script format is supported).



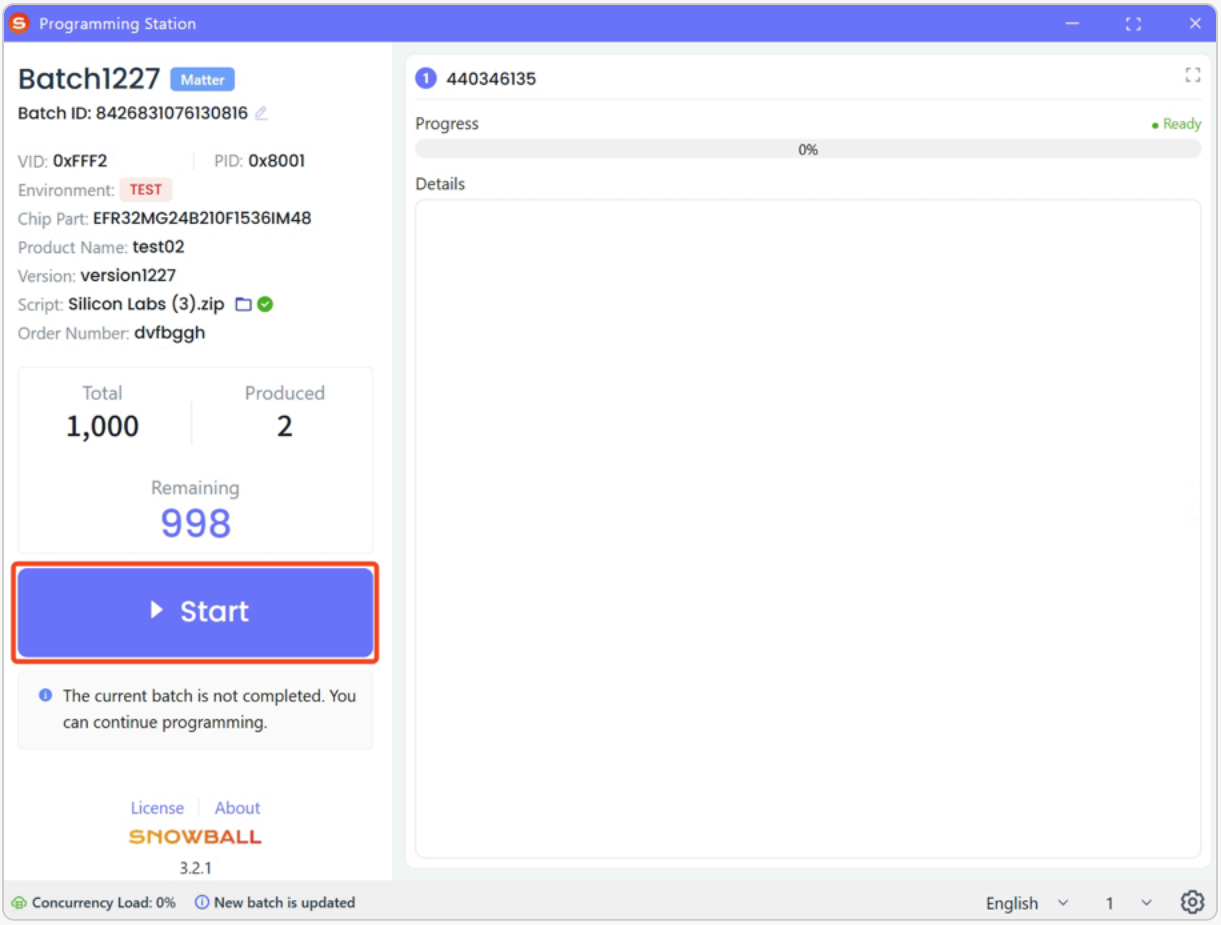
2. In the main window of the Programming Station client, click the Settings icon in the bottom-right corner and then select **Channel Configuration**.



3. In the **Channel Configuration** window, set the number of programmers connected (**1**, **2**, **4**, **6**, **8**, **12**, or **16**).

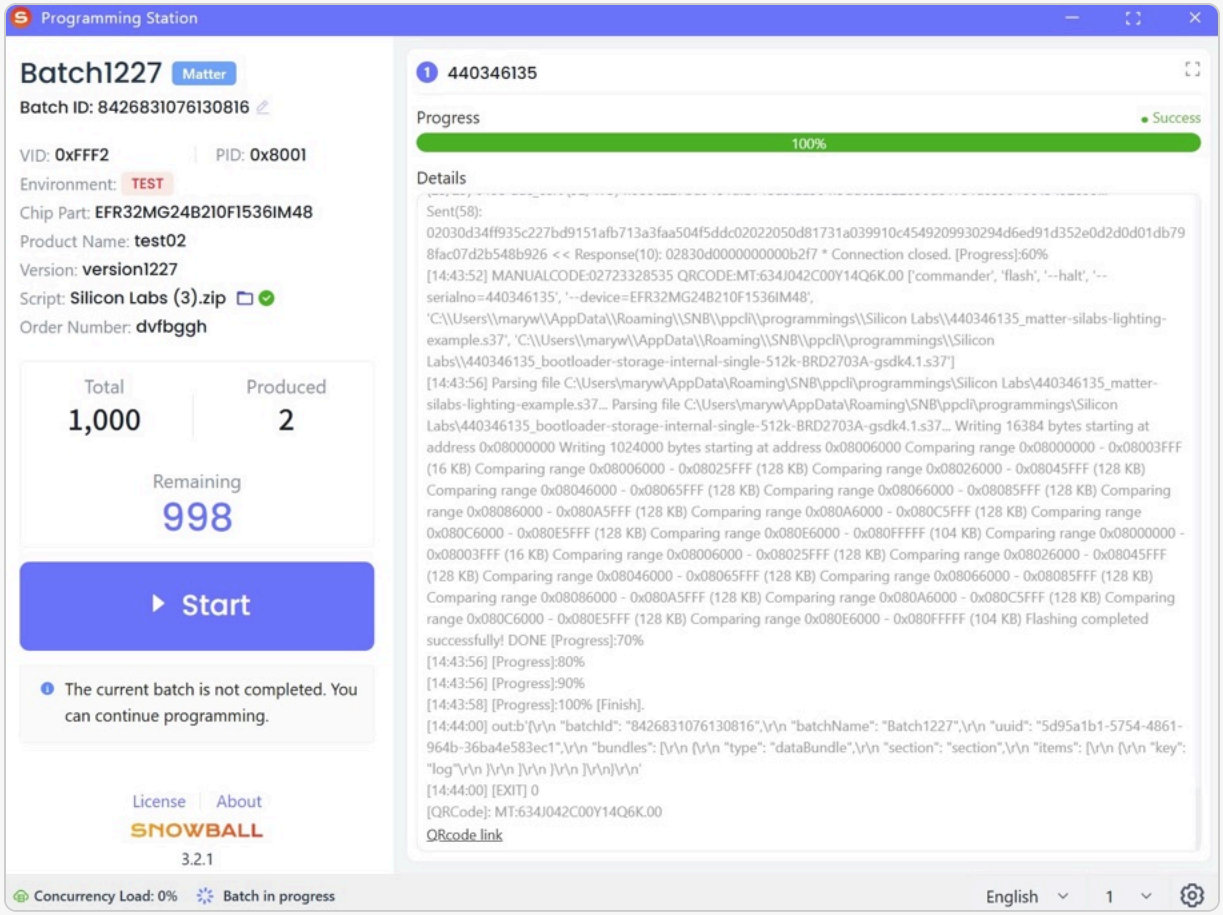


4. In the main window of the Programming Station client, click **Start** to execute device programming.



5. After the programming is complete, the programming details will be displayed.

> **TIP**
>
> You can also check the programming details in OBIS.


Programming Station
Batch1227
Matter
Batch ID: 8426831076130816
VID: 0xFFF2
PID: 0x8001
Environment: TEST
Chip Part: EFR32MG24B210F1536IM48
Product Name: test02
Version: version1227
Script: Silicon Labs (3).zip
Order Number: dvfbggh
Total
1,000
Produced
2
Remaining
998
Start
The current batch is not completed. You can continue programming.
License
About
SNOWBALL
3.2.1
440346135
Progress
Success
100%
Details
Sent(58):
02030d34ff935c227bd9151afb713a3faa504f5ddc02022050d81731a039910c4549209930294d6ed91d352e0d2d0d01db798fac07d2b548b926 << Response(10): 02830d0000000000b2f7 * Connection closed. [Progress]:60%
[14:43:52] MANUALCODE:02723328535 QRCODE:MT:634J042C00Y14Q6K.00 ['commander', 'flash', '--halt', '--serialno=440346135', '--device=EFR32MG24B210F1536IM48', 'C:\\Users\\maryw\\AppData\\Roaming\\SNB\\ppcli\\programmings\\Silicon Labs\\440346135_matter-silabs-lighting-example.s37', 'C:\\Users\\maryw\\AppData\\Roaming\\SNB\\ppcli\\programmings\\Silicon Labs\\440346135_bootloader-storage-internal-single-512k-BRD2703A-gsdk4.1.s37']
[14:43:56] Parsing file C:\Users\maryw\AppData\Roaming\SNB\ppcli\programmings\Silicon Labs\440346135_matter-silabs-lighting-example.s37... Parsing file C:\Users\maryw\AppData\Roaming\SNB\ppcli\programmings\Silicon Labs\440346135_bootloader-storage-internal-single-512k-BRD2703A-gsdk4.1.s37... Writing 16384 bytes starting at address 0x08000000 Writing 1024000 bytes starting at address 0x08006000 Comparing range 0x08000000 - 0x08003FFF (16 KB) Comparing range 0x08006000 - 0x08025FFF (128 KB) Comparing range 0x08026000 - 0x08045FFF (128 KB) Comparing range 0x08046000 - 0x08065FFF (128 KB) Comparing range 0x08066000 - 0x08085FFF (128 KB) Comparing range 0x08086000 - 0x080A5FFF (128 KB) Comparing range 0x080A6000 - 0x080C5FFF (128 KB) Comparing range 0x080C6000 - 0x080E5FFF (128 KB) Comparing range 0x080E6000 - 0x080FFFFF (104 KB) Comparing range 0x08000000 - 0x08003FFF (16 KB) Comparing range 0x08006000 - 0x08025FFF (128 KB) Comparing range 0x08026000 - 0x08045FFF (128 KB) Comparing range 0x08046000 - 0x08065FFF (128 KB) Comparing range 0x08066000 - 0x08085FFF (128 KB) Comparing range 0x08086000 - 0x080A5FFF (128 KB) Comparing range 0x080A6000 - 0x080C5FFF (128 KB) Comparing range 0x080C6000 - 0x080E5FFF (128 KB) Comparing range 0x080E6000 - 0x080FFFFF (104 KB) Flashing completed successfully! DONE [Progress]:70%
[14:43:56] [Progress]:80%
[14:43:56] [Progress]:90%
[14:43:58] [Progress]:100% [Finish].
[14:44:00] out:b'{\r\n "batchId": "8426831076130816",\r\n "batchName": "Batch1227",\r\n "uuid": "5d95a1b1-5754-4861-964b-36ba4e583ec1",\r\n "bundles": [\r\n {\r\n "type": "dataBundle",\r\n "section": "section",\r\n "items": [\r\n {\r\n "key": "log"\r\n }\r\n]\r\n }\r\n]\r\n}\r\n'
[14:44:00] [EXIT] 0
[QRCode]: MT:634J042C00Y14Q6K.00
QRcode link
Concurrency Load: 0%
Batch in progress
English
1

Version: v3.2.1

Feature guide - OnBoard™ IoT Security

OnBoard™ IoT Security provides a comprehensive set of features to manage every stage of the IoT device provisioning lifecycle. This guide covers each core module—including product management, factory management, key and certificate management, user management, U-Safe management, and device record tracking. Select a feature area below to explore its capabilities and related topics.

Product

- Create a product
- Configure a product
- Manage products
- Create a version
- Create a batch

Factory

LOCAL FACTORY

- Configure and manage factories
- Configure and manage programming stations

CLOUD FACTORY

- Configure and manage factories
- Configure and manage programming stations
- Manage batches and assign programming stations

Records

- Check device history

KMS

- Create a key
- Check keys
- Manage keys

PKI

- Create a certificate
- Check certificates
- Manage certificates

User

ACCOUNT

- Create an account
- Check accounts
- Manage accounts

GROUP

- Create a group
- Check groups
- Manage groups

U-Safe List

- Check U-Safes

Version: v3.2.1

How to create a product in OnBoard™ IoT Security

> **INFO**
>
> Currently, products can't be deleted after creation.

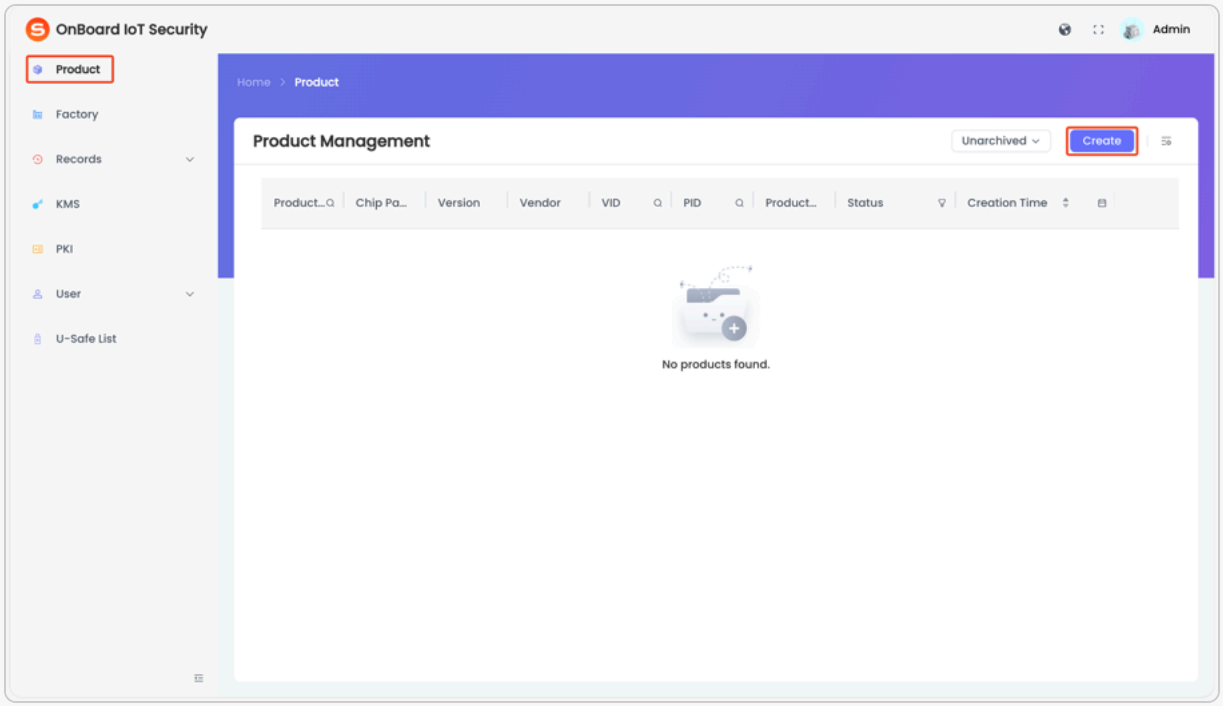
This article will guide you on how to create a product in OnBoard™ IoT Security (OBIS) to configure its basic information and prepare for product materials configuration.

Prerequisites

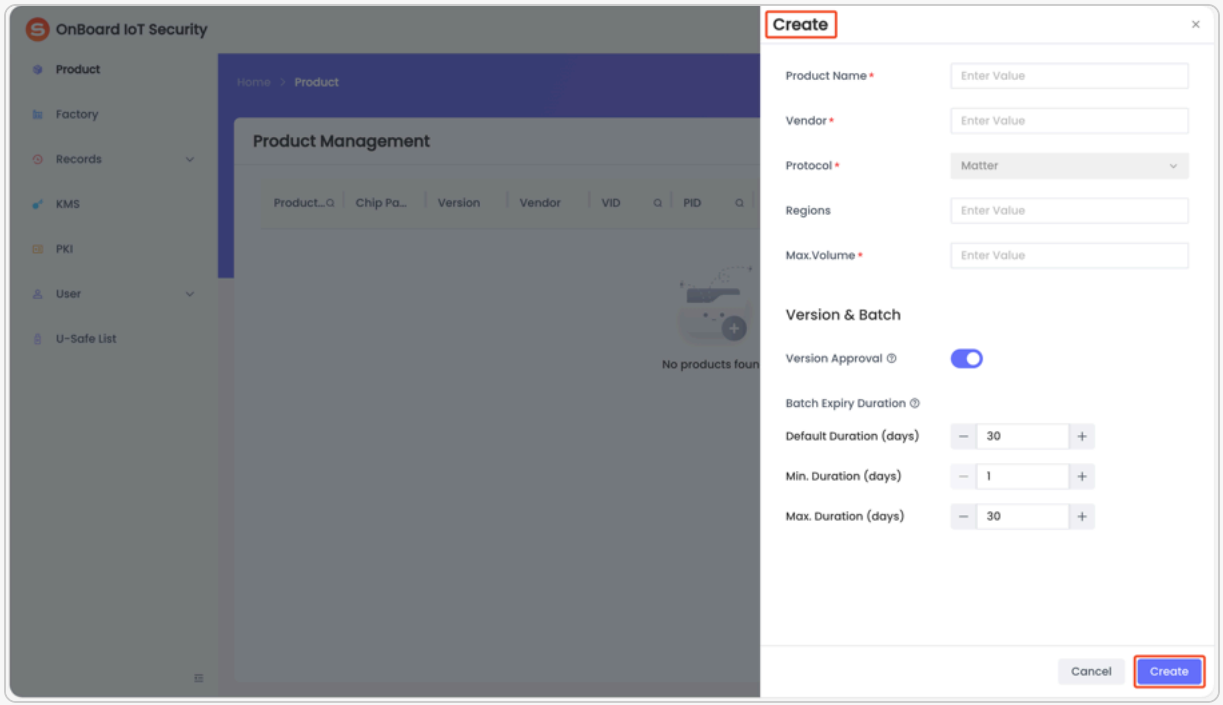
Before getting started, please ensure that you have OBIS role permissions as a System Manager or Product Manager.

Instructions

1. Sign in to OBIS.
2. In the left menu, click **Product**, and then click **Create**.



3. On the **Create** panel, select and enter the required product basic information. After verifying that the information is correct, click **Create** to complete the product creation.



Description of product basic information:

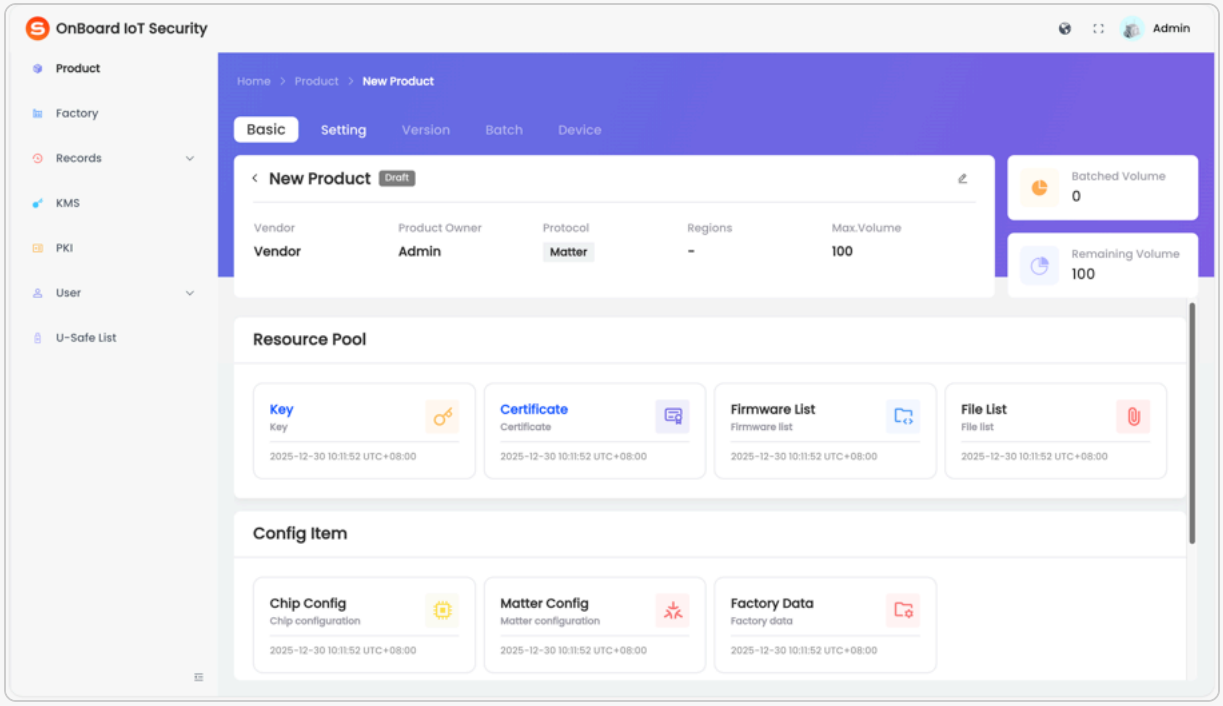
Information	Mandatory	Description
Product Name	Yes	The product name must be unique.
Vendor	Yes	Automatically filled based on the tenant to ensure accuracy and reduce manual errors. You can manually change it if a different vendor name is needed.
Protocol	Yes	Currently, only the Matter protocol is supported.
Regions	No	Specify the regions where the product is supported. After entering a region, press **Enter** to generate a region label. Multiple region labels can be added. **Note**: When creating a version, the selectable regions will be limited to these specified regions.
Max.Volume	Yes	The maximum production volume of a product can't exceed 999,999,999. The total volume of batches can't exceed this limit.

Information	Mandatory	Description
Version & Batch	No	◦ **Version Approval**: This feature is enabled by default to ensure that all version data used for device production is verified. After the feature is enabled, any version created in the PROD environment requires approval from the Product Owner or Product Manager before it can be used to create a batch. ◦ **Batch Expiry Duration**: You can set the **Default Duration**, **Min. Duration** and **Max. Duration** (in days). When creating a batch, the Expiry Date must be within this set range.

4. After the product is created, the page will navigate to the product page.

> **INFO**
>
> After a product is created, its initial status is **Draft**. Once a version is created, the product status will change to **Released**. For more information, refer to the description of product status.

> **TIP**
>
> By default, the product creator is set as the Product Owner. To change the Product Owner, you can click the Edit icon on the product page, select a new Product Owner on the **Edit** panel, and then click **Save**.


OnBoard IoT Security
Admin
Product
Factory
Records
KMS
PKI
User
U-Safe List
Home > Product > New Product
Basic
Setting
Version
Batch
Device
New Product
Draft
Vendor
Vendor
Product Owner
Admin
Protocol
Matter
Regions
-
Max.Volume
100
Batched Volume
0
Remaining Volume
100
Resource Pool
Key
Key
2025-12-30 10:11:52 UTC+08:00
Certificate
Certificate
2025-12-30 10:11:52 UTC+08:00
Firmware List
Firmware list
2025-12-30 10:11:52 UTC+08:00
File List
File list
2025-12-30 10:11:52 UTC+08:00
Config Item
Chip Config
Chip configuration
2025-12-30 10:11:52 UTC+08:00
Matter Config
Matter configuration
2025-12-30 10:11:52 UTC+08:00
Factory Data
Factory data
2025-12-30 10:11:52 UTC+08:00

Version: v3.2.1

How to configure a product in OnBoard™ IoT Security

This article will guide you through the following operations in OnBoard™ IoT Security (OBIS) to configure product materials and prepare for version creation:

- Configure the resource pool
- Set configuration items
- Upload programming station files

Prerequisites

Before getting started, please ensure that:

- You have product role permissions for the corresponding product: Product Owner, Product Manager, or Resource Manager.
- The certificates and keys have already been created.

Product material categories

Each product has its own independent material management system used to centrally manage various types of product materials. Users can upload firmware, configuration files, programming files, associate keys and certificate templates, and pre-configure product parameters. Product materials are divided into three categories, each containing different resource types:

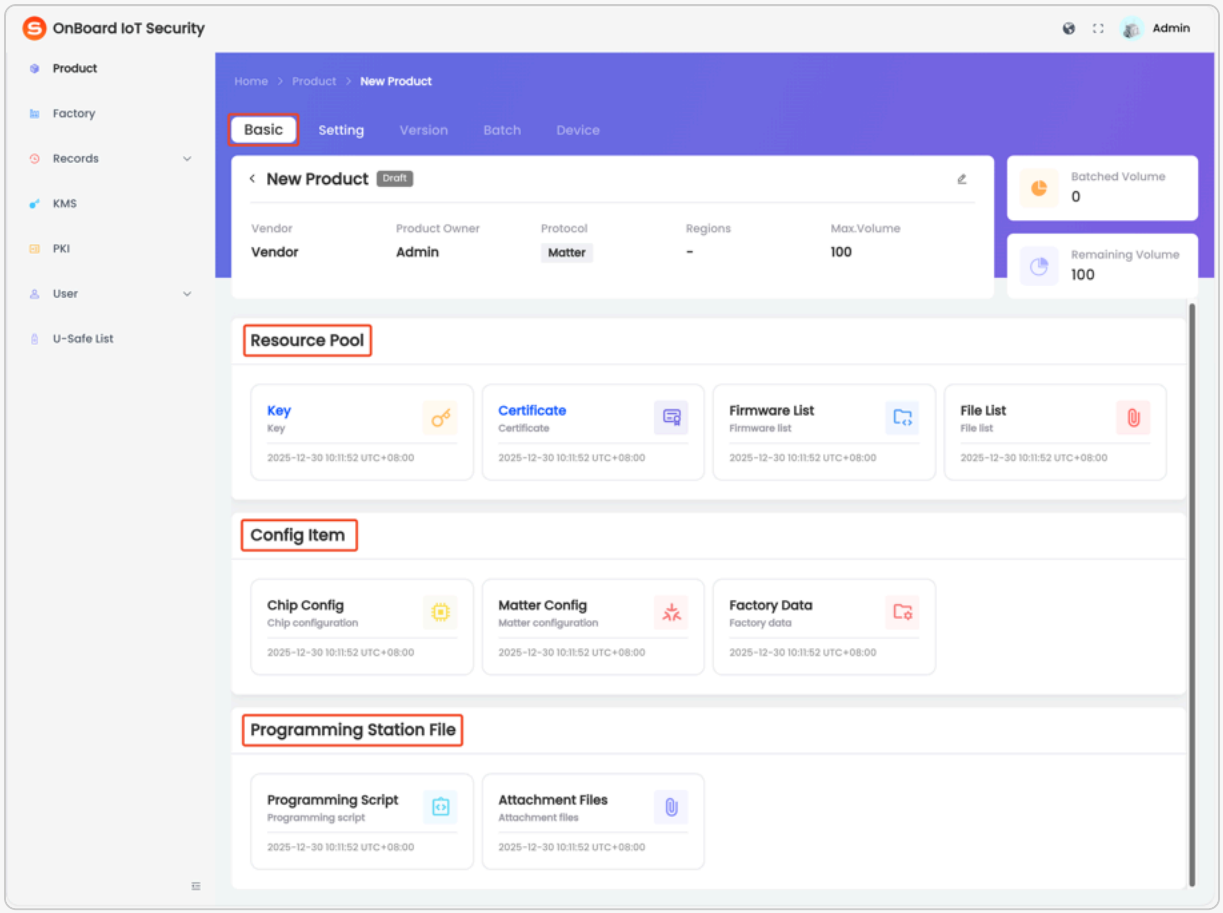
Categories	Material Type	Description
Resource Pool	Key	Add keys from KMS to the product.
	Certificate	Add CA templates from PKI to the product.
	Firmware List	Upload the firmware files required for programming the product. Firmware formats supported: **.bin**, **.hex**, and **.s37**.
	File List	Upload other files required for version creation (excluding firmware). Files in any format are supported.
Config Item	Chip Config	Select and configure the chip part and security options used by the product.
	Matter Config	Select and configure settings related to the Matter protocol.
	Factory Data	Configure factory-related information.
Programming Station File	Programming Script	Upload script files used for device programming (any script format is supported).
	Attachment Files	Upload other files required for batch creation (excluding programming scripts). Files in any format are supported.

Instructions

1. Sign in to OBIS.

2. In the left menu, click **Product** and select the corresponding product. Then, on the product page, configure the items in the **Resource Pool**, **Config Item**, and **Programming Station File** areas.

   > **INFO**
   >
   > - The configurations of the Resource Pool and Config Item areas will be used for version creation. In these areas, configure the necessary resources and configuration items for the product (**Certificate**, **Firmware List**, **Chip Config**, **Matter Config**, and **Factory Data**). If the required configuration is not completed, version creation will be blocked, and the product status will remain in **Draft**.
   > - Files in the Programming Station File area will be used for batch creation. Upload the necessary files for programming (**Programming Script**) in this area. Alternatively, you can select a programming script for a specific batch later in the Programming Station client.



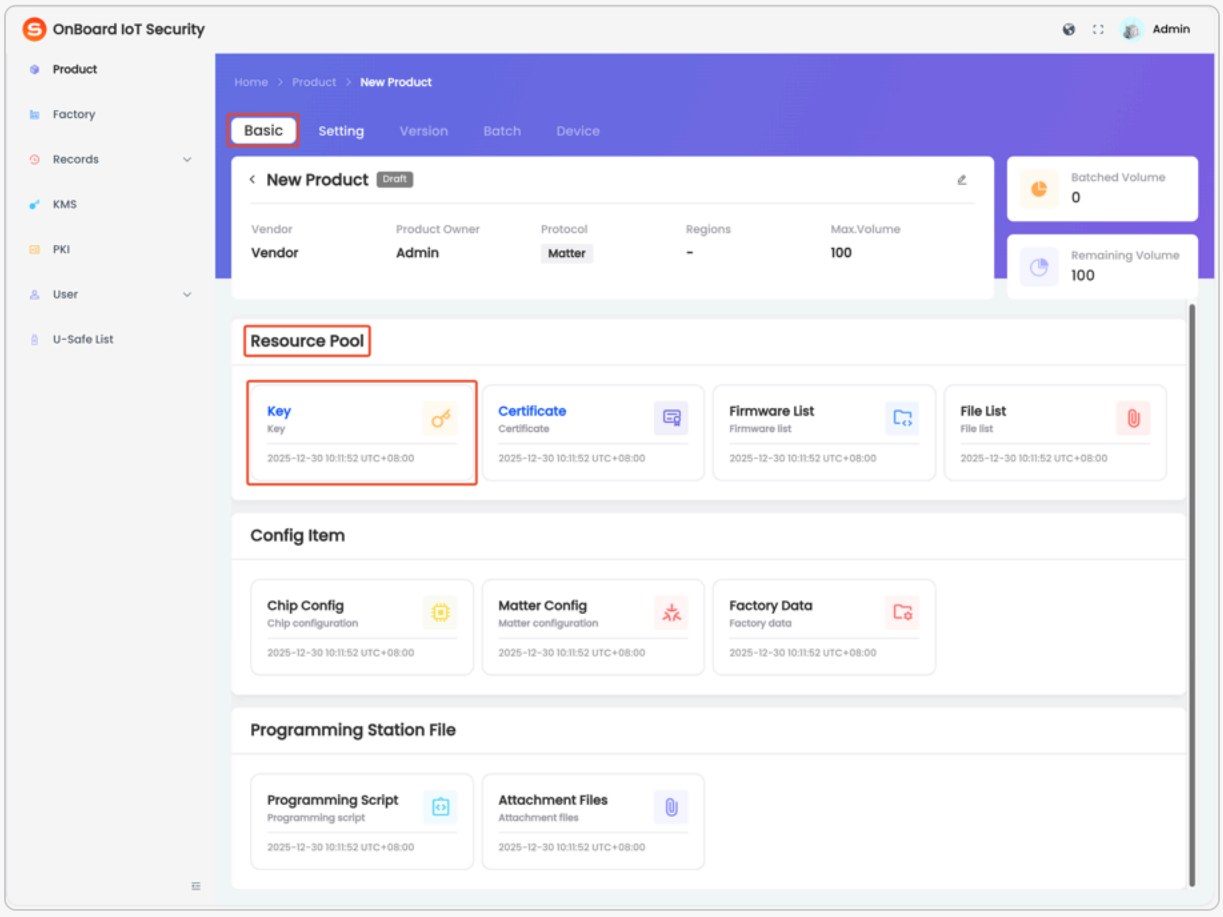
Step one: Configure the resource pool

> **TIP**
>
> The configurations in the **Resource Pool** area will be used for version creation. The files uploaded when configuring Firmware List and File List can be downloaded at any time from the configuration page of the corresponding resource.

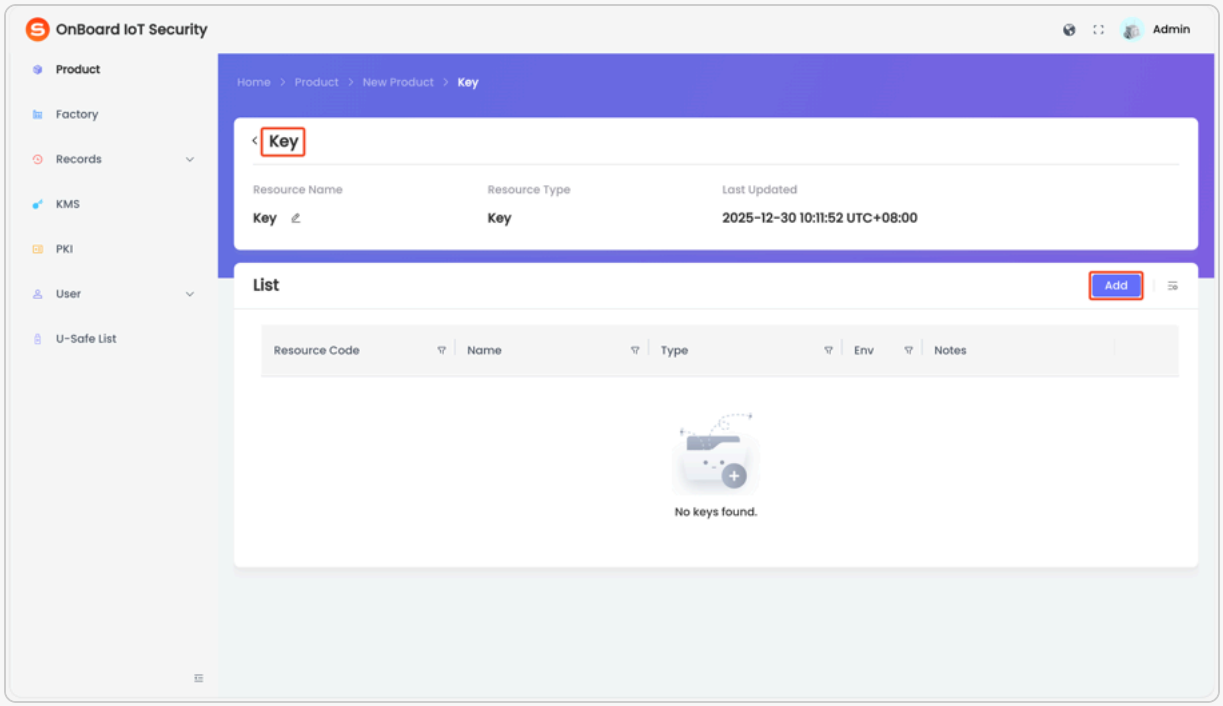
Key (optional)

> **TIP**
>
> **Key** allows you to add keys from KMS to the product. Added keys can be selected during the Chip Config.
>
> Keys can be created in KMS. For more information, refer to the steps for creating a key.

1. On the product page, select **Key** in the **Resource Pool** area to navigate to the **Key** page.



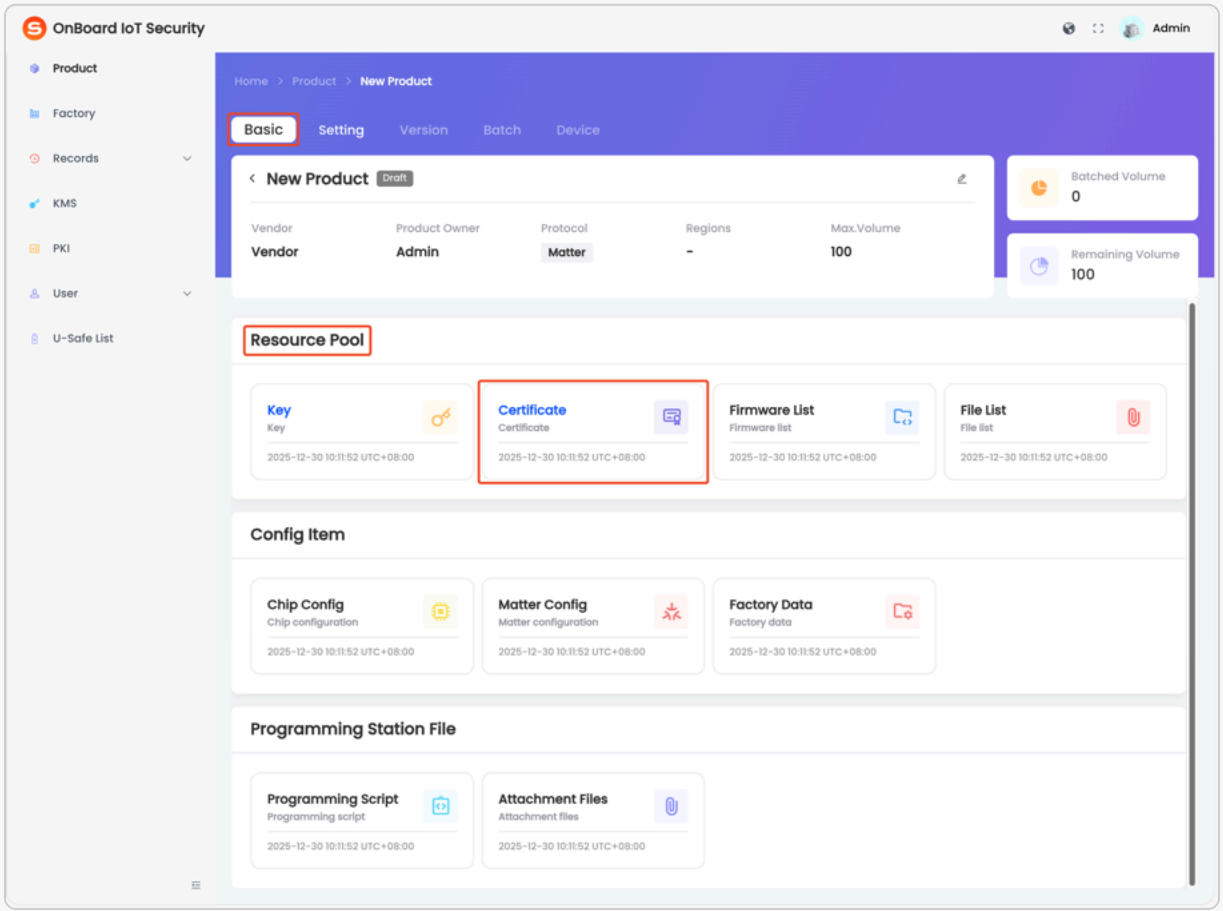
2. On the **Key** page, click **Add**, then select and add keys.


OnBoard IoT Security
Admin
Product
Factory
Records
KMS
PKI
User
U-Safe List
Home > Product > New Product > Key
Key
Resource Name
Key
Resource Type
Key
Last Updated
2025-12-30 10:11:52 UTC+08:00
List
Add
Resource Code
Name
Type
Env
Notes
No keys found.

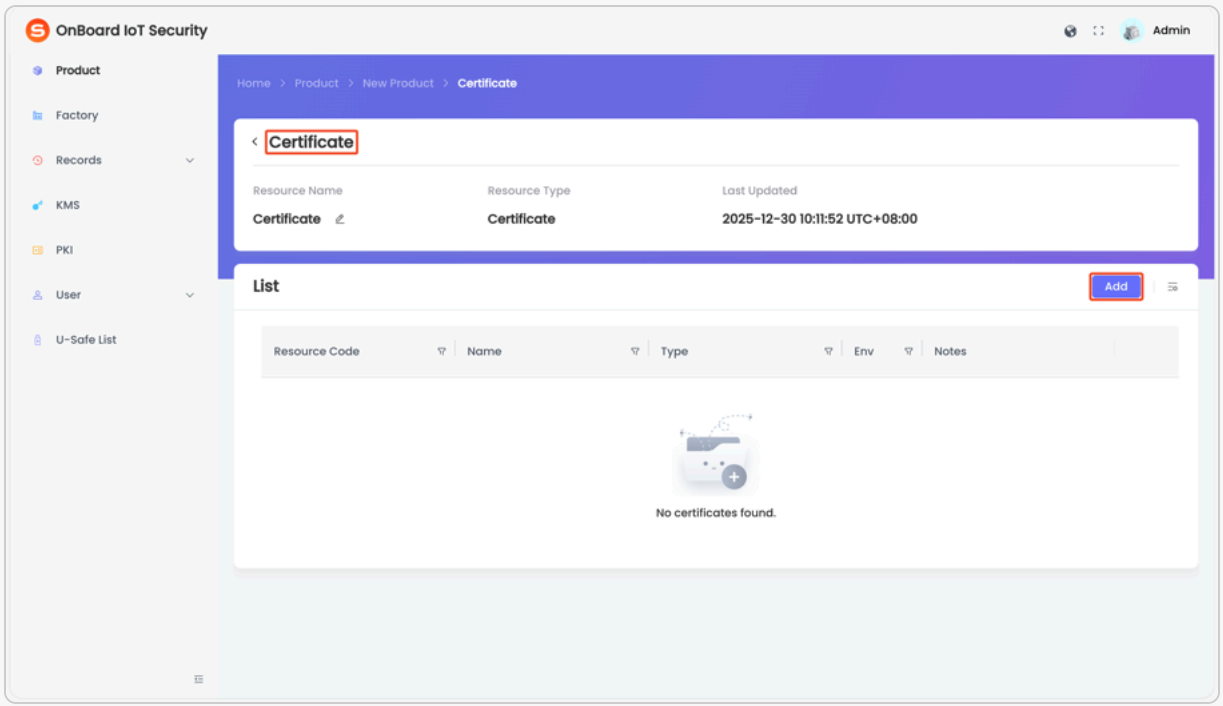
Certificate (required)

> **TIP**
>
> **Certificate** allows you to add CA templates (DAC templates or end entity templates) from PKI to the product. Added CA templates can be selected during the Matter Config.
>
> CA templates can be created in PKI. For more information, refer to the steps to create a certificate.

1. On the product page, select **Certificate** in the **Resource Pool** area to navigate to the **Certificate** page.



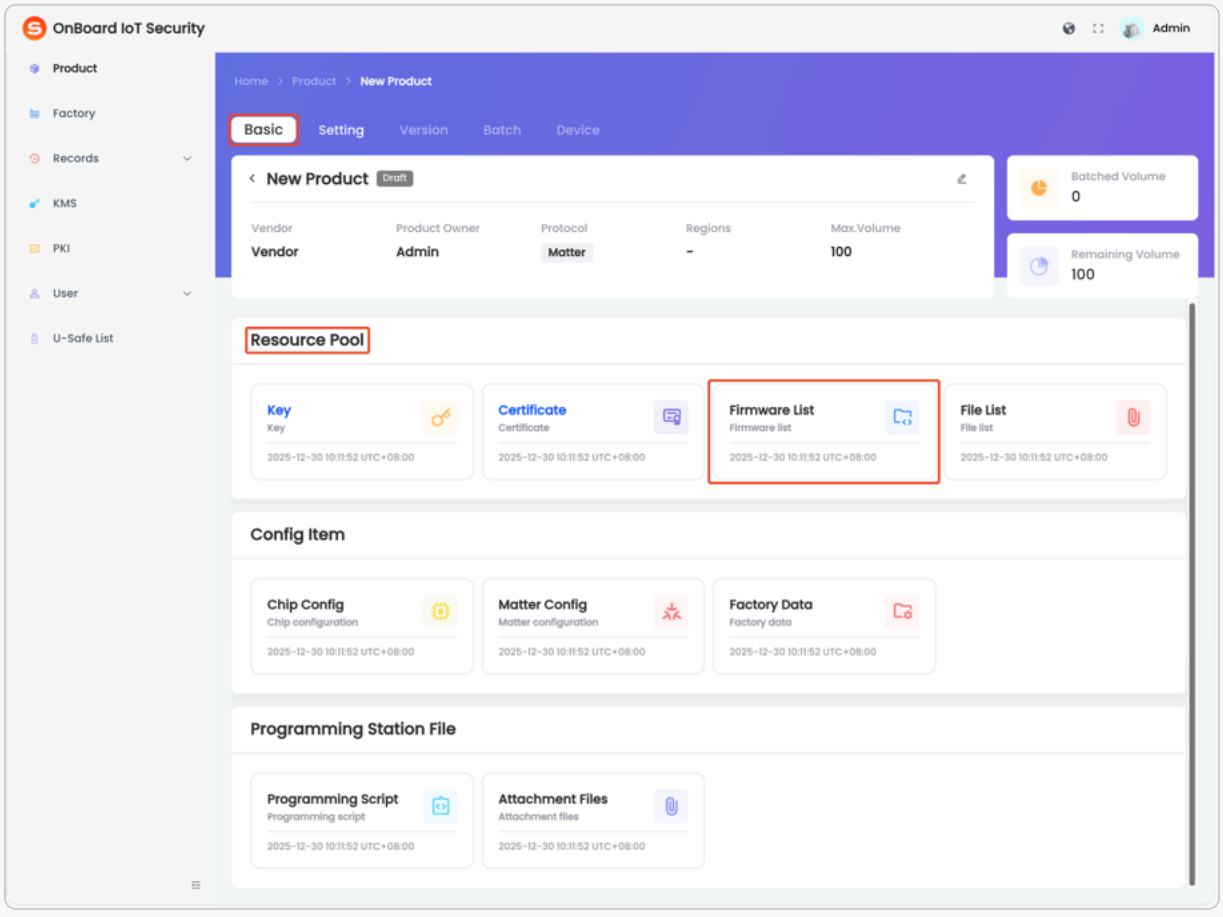
2. On the **Certificate** page, click **Add**, then select and add CA templates.



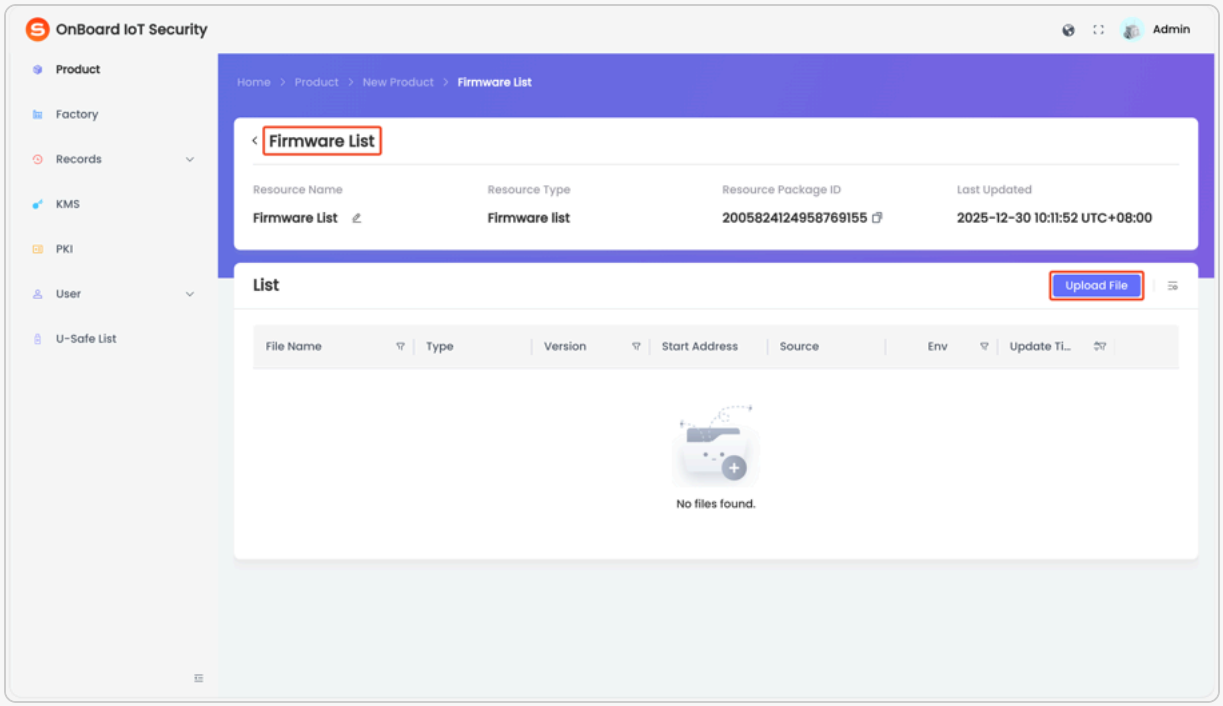
Firmware List (required)

> **TIP**
>
> **Firmware List** allows you to upload the firmware files required for programming the product. Firmware formats supported: **.bin**, **.hex**, and **.s37**. Then, the uploaded firmware files can be selected when creating a version.

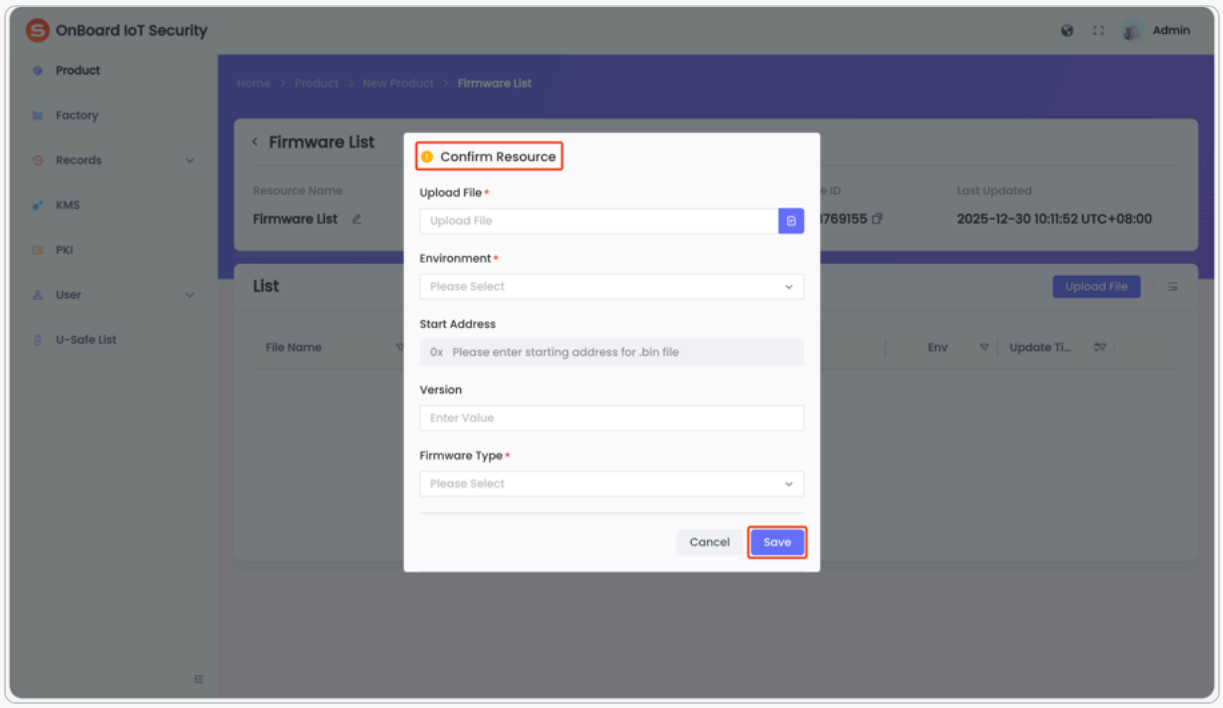
1. On the product page, select **Firmware List** in the **Resource Pool** area to navigate to the **Firmware List** page.



2. On the **Firmware List** page, click **Upload File**.



3. In the **Confirm Resource** window, select and enter the firmware-related information.



Description of Firmware information:

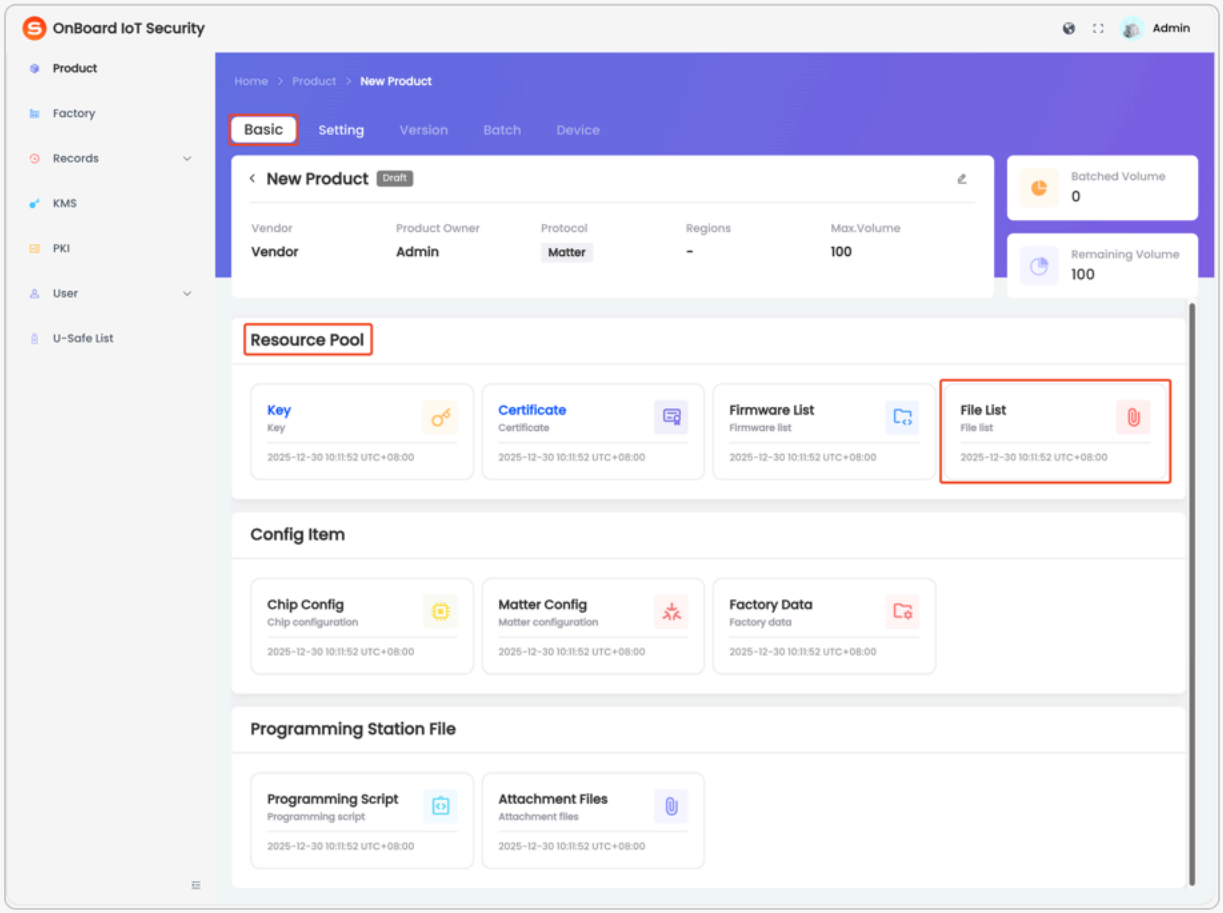
Information	Mandatory	Description
Upload File	Yes	Upload files by selecting the local firmware files.
Environment	Yes	Two environments can be selected: **TEST** and **PROD**.
Start Address	No	Specify the start address of the firmware.
Version	No	Indicate the firmware version.
Firmware Type	Yes	Two types of firmware are supported: ◦ **Application**: Used to implement device functionality, relying on the environment provided by the Bootloader. ◦ **Bootloader**: Used for hardware initialization, firmware loading, and updates.

4. Click **Save** to complete the configuration of the Firmware List.

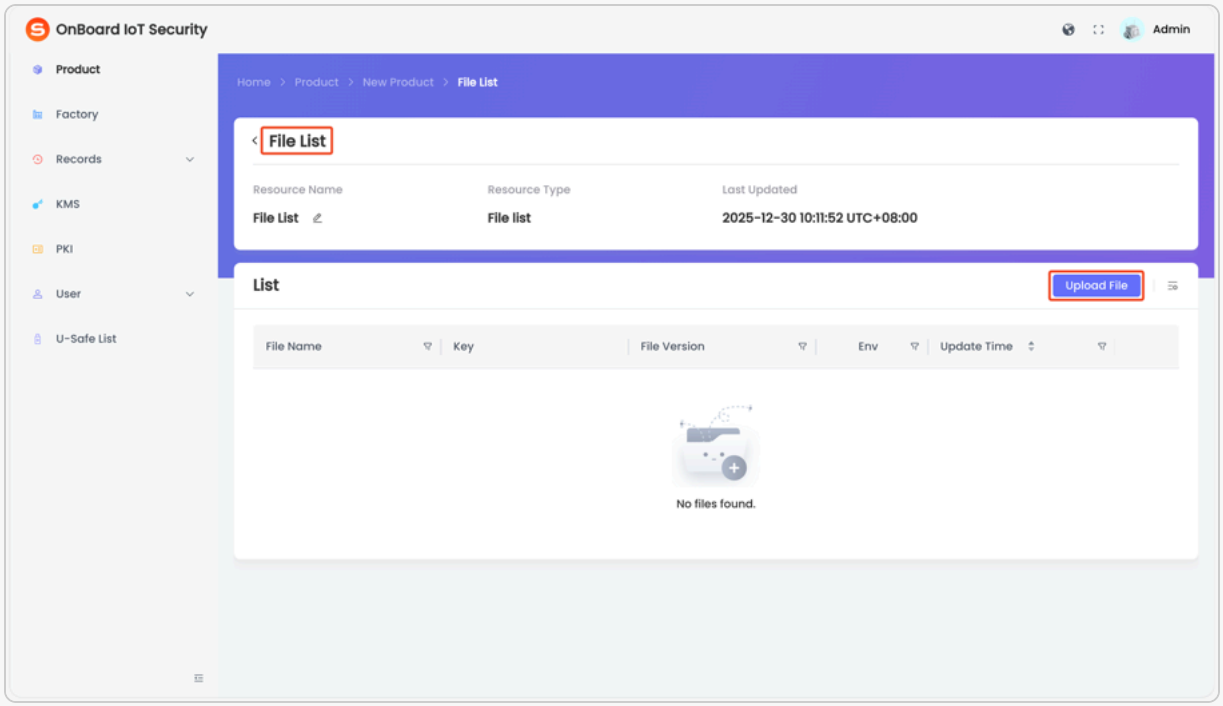
File List (optional)

> **TIP**
>
> **File List** allows you to upload other files required for version creation (excluding firmware). Files in any format are supported.
>
> The uploaded files can be selected when creating a version.

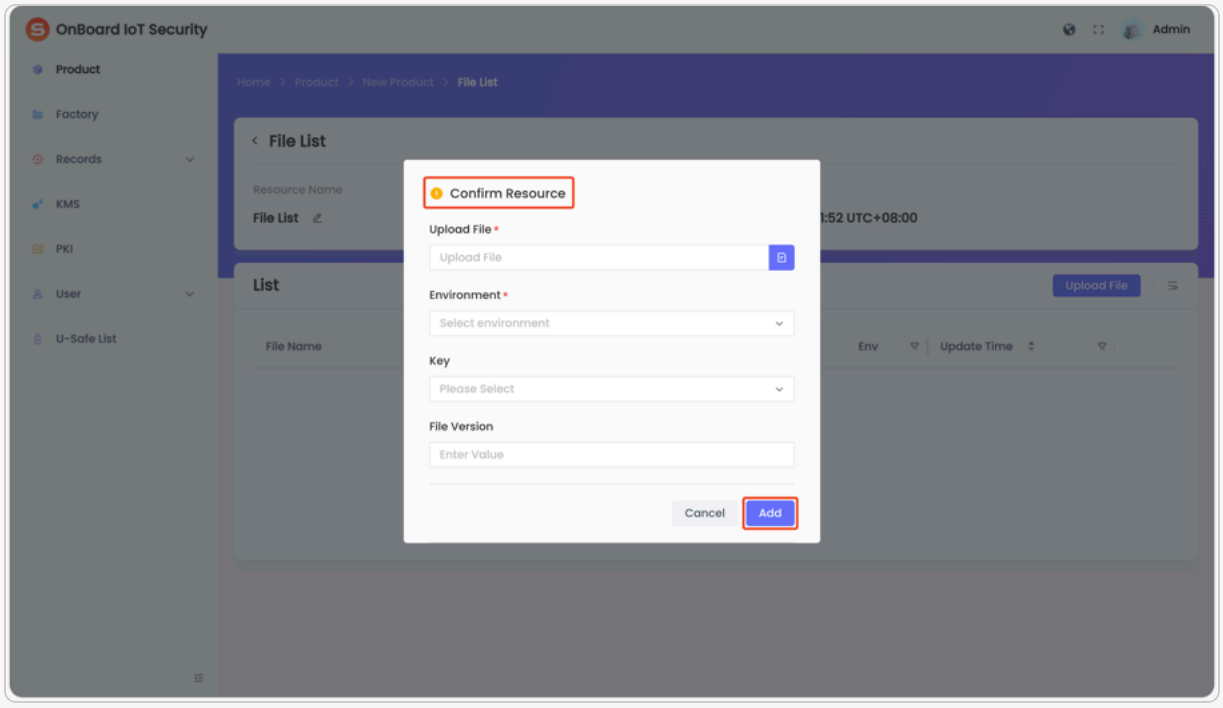
1. On the product page, select **File List** in the **Resource Pool** area to navigate to the **File List** page.



2. On the **File List** page, click **Upload File**.



3. In the **Confirm Resource** window, select and enter the relevant information.



Description of File List information:

Information	Mandatory	Description
Upload File	Yes	Upload files by selecting the local files.
Environment	Yes	Two environments can be selected: **TEST** and **PROD**.
Key	No	Assign keys to files. During later programming, the programming script can retrieve and use the corresponding files through their keys. **Note**: A file can be assigned multiple unique keys. Meanwhile, the same key can also be assigned to multiple files. You can add new keys or select from existing keys.
File Version	No	Identify the file version and use it to track file changes.

4. Click **Add** to complete the configuration of the File List.

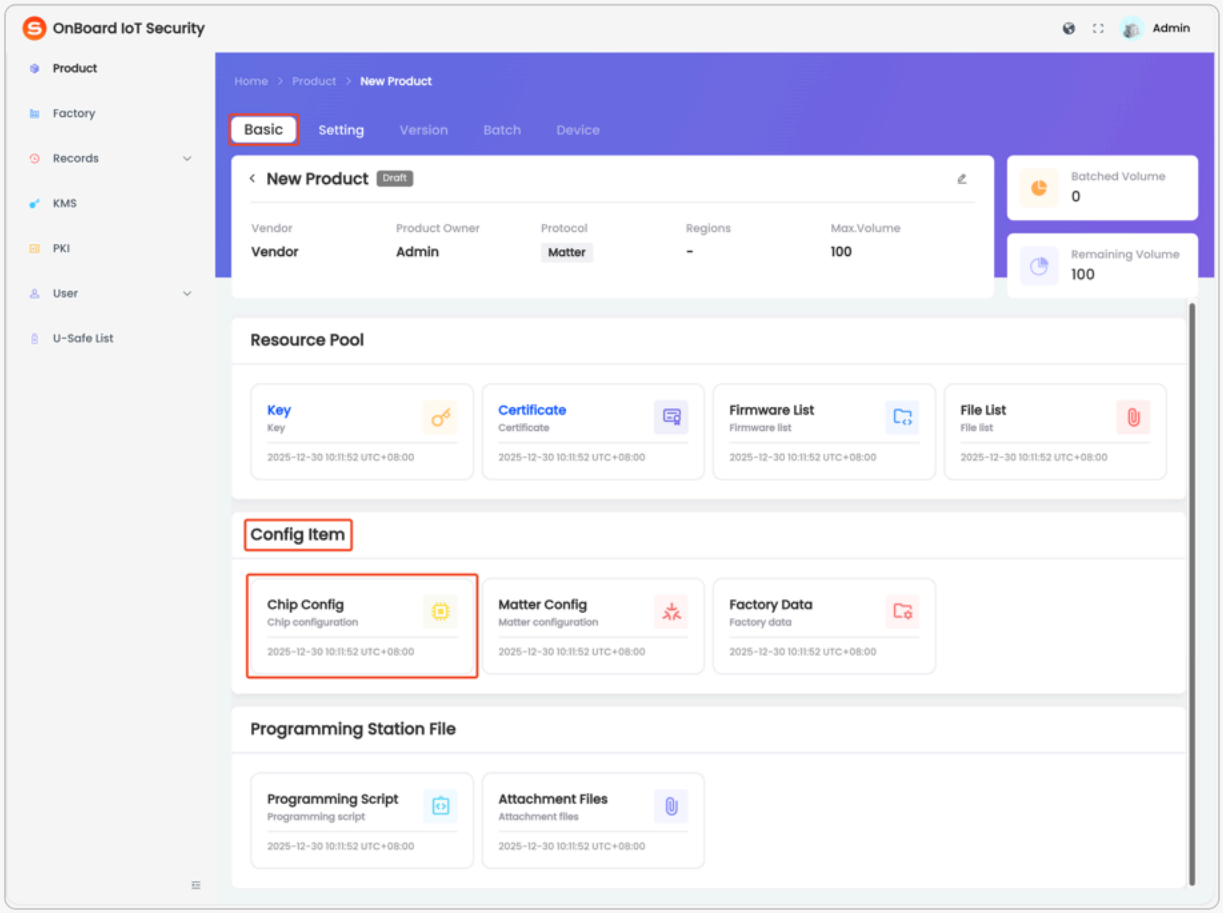
Step two: Set configuration items

> **TIP**
>
> The configurations in the **Config Item** area will be used for version creation. The files uploaded when configuring Matter Config and Factory Data can be downloaded at any time from the configuration page of the corresponding resource.

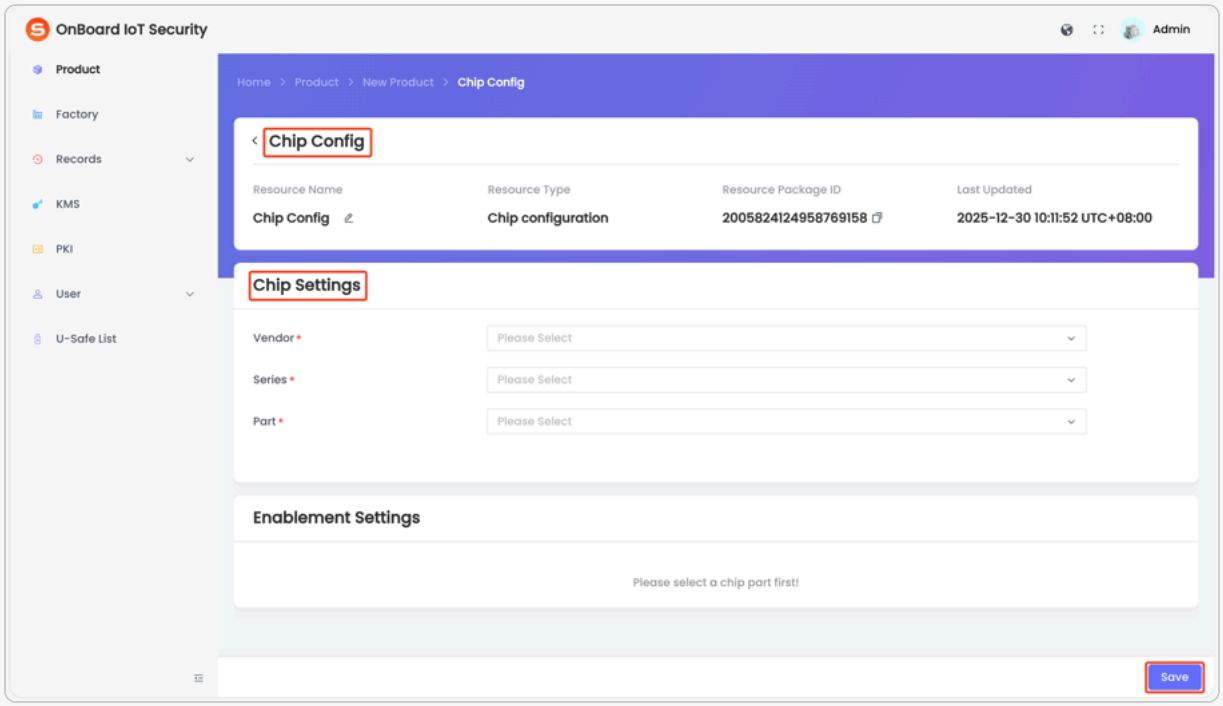
Chip Config (required)

> **TIP**
>
> **Chip Config** allows you to select and set the chip part and security options used by the product. Different chip parts provide corresponding security options in the Enablement Setting, and you can configure them as needed.
>
> Please refer to the supported chip settings for the detailed list. Additional chip vendors, series, and parts will be added based on demand.

1. On the product page, select **Chip Config** in the **Config Item** area to navigate to the **Chip Config** page.



2. On the **Chip Config** page, locate the **Chip Settings** area, select the **Vendor**, **Series**, and **Part**, and then click **Save**.



Currently supported chip settings:

Vendor	Series	Part
Beken	BK7238	BK7238
NXP	s32k1	s32k144
Qorvo	QPG6200	QPG6200

Vendor	Series	Part
Silicon Labs	EFR32MG24	◦ EFR32MG24B210F1536IM48 ◦ EFR32MG24B020F1536IM48 ◦ EFR32MG24A110F1024IM48 ◦ EFR32MG24A020F1536IM48 ◦ EFR32MG24A420F1536IM48 ◦ EFR32MG24B020F1536IM40 ◦ EFR32MG24A010F1024IM40 ◦ EFR32MG24A020F1536IM40 ◦ EFR32MG24A420F1536IM40 ◦ EFR32MG24B010F1024IM48 ◦ EFR32MG24A010F1024IM48 ◦ EFR32MG24B110F1536IM48 ◦ EFR32MG24A021F1024IM40 ◦ EFR32MG24B010F1536IM40 ◦ EFR32MG24B220F1536IM48 ◦ EFR32MG24A020F1024IM40 ◦ EFR32MG24A010F1536IM40 ◦ EFR32MG24B010F1536IM48 ◦ EFR32MG24A410F1536IM48 ◦ EFR32MG24A010F1536IM48 ◦ EFR32MG24B120F1536IM48 ◦ EFR32MG24A410F1536IM40 ◦ EFR32MG24B310F1536IM48 ◦ EFR32MG24B020F1024IM48

Vendor	Series	Part
		◦ EFR32MG24A020F1024IM48
	SIMG301	SiMG301M113WIH
Telink	TLSR92x	TLSR9268J

3. In the **Enablement Setting**, you can select and enable the **Secure Boot** and **Secure Debug** features in the PROD or TEST environment as needed. In the TEST environment, the enablement of the **Secure Boot** and **Secure Debug** features can be changed in the version.

> **TIP**
>
> When creating a batch, if you need to enable the Secure Debug Reopen feature to allow the chip to be reprogrammed, you must enable the **Secure Debug** feature.

Description of Enablement Setting items (different chip settings will display different setting items):

Setting items	Description
Secure Boot	A security protection mechanism that prevents unauthorized software from being loaded.

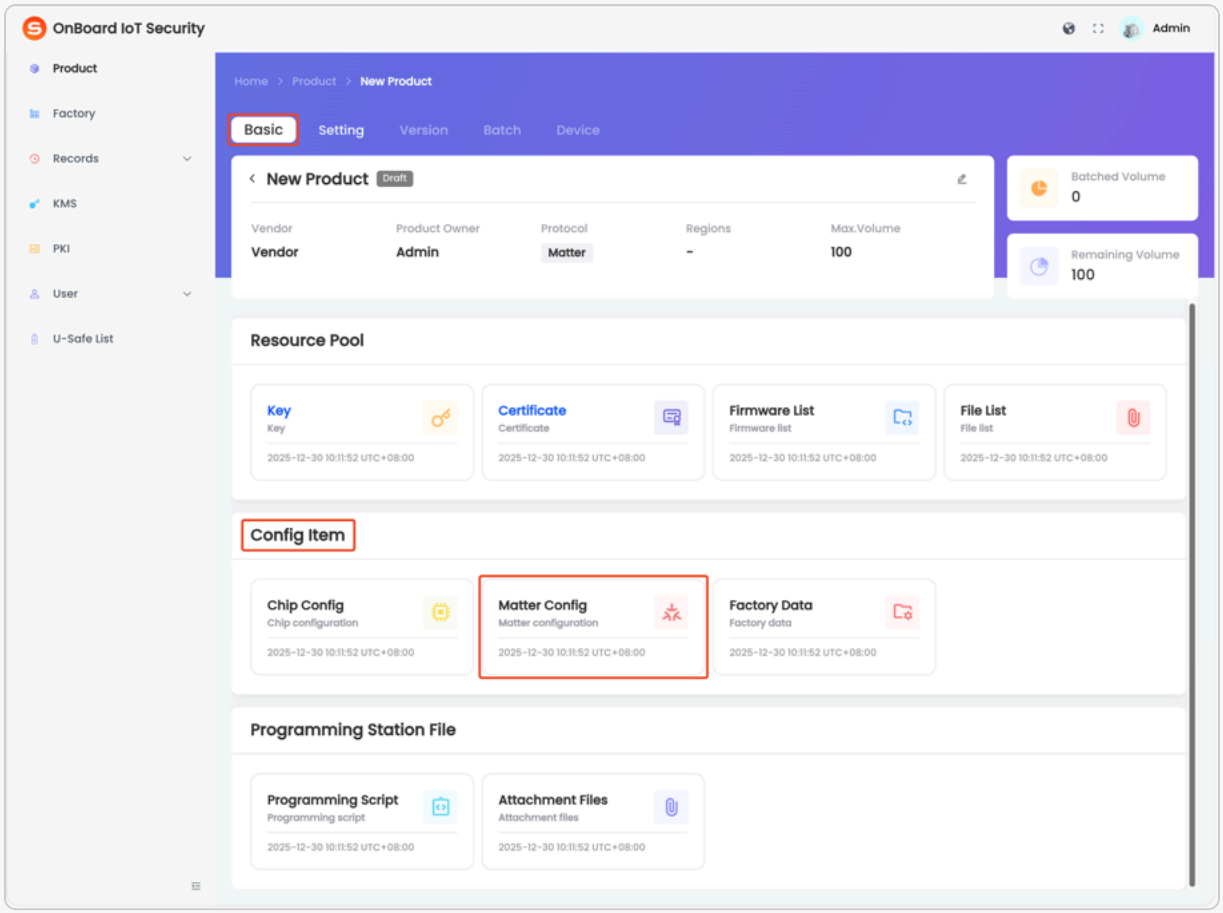
Setting items	Description
Secure Debug	A security protection mechanism to prevent unauthorized debugging.
Secure Boot Key	Select the key added in the **Key** configuration. The key is used to verify the firmware and ensure the secure boot of the device.
Secure Boot Key Hash	The hash value of the Secure Boot Key.
Secure Debug Key	Select the key added in the **Key** configuration. The key is used to authorize debugging access. **Note**: For Telink chips, this key is derived from the MCU Boot Key and Debug Plaintext.
MCU Boot Key	Select the key added in the **Key** configuration. The key is used for device authentication and data encryption.
Debug Plaintext	Unencrypted debug data.
DAC Partition Address	Set the storage address in flash for downloading DAC certificates.

4. Click **Save** to complete the setting of the Chip Config.

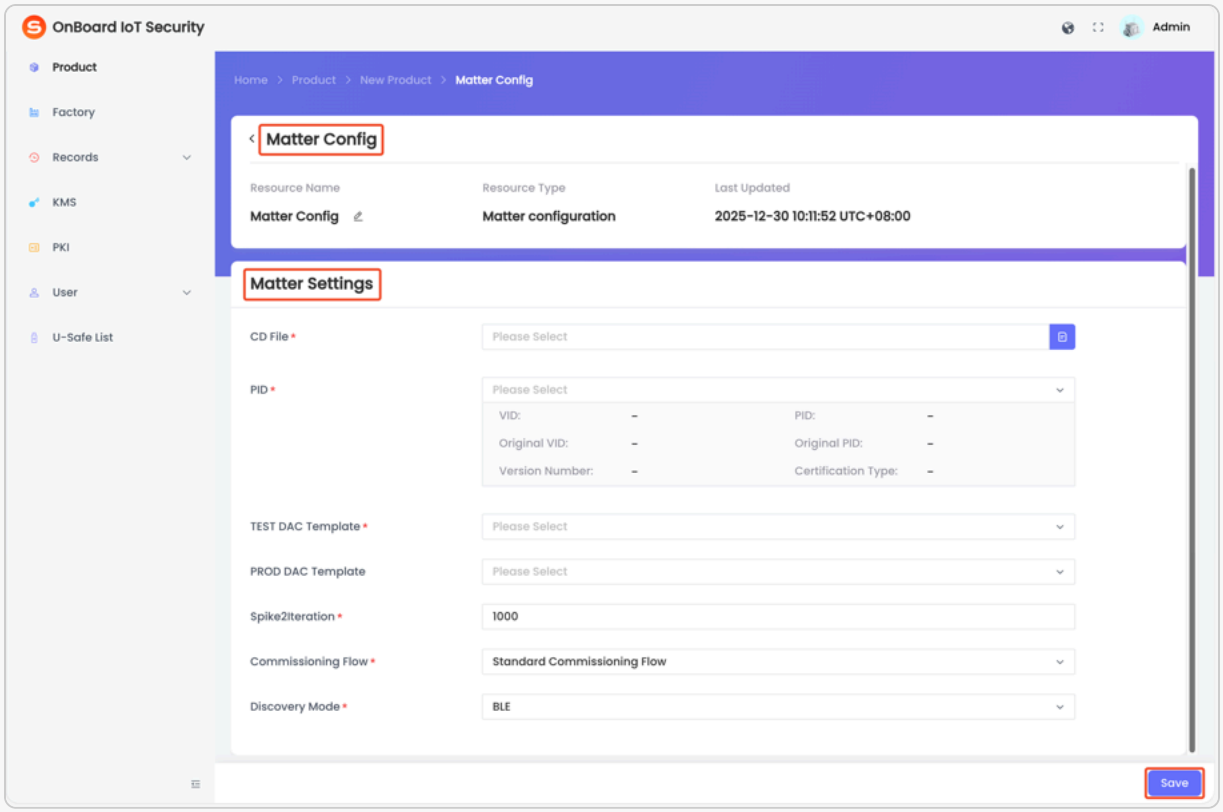
Matter Config (required)

> **TIP**
>
> **Matter Config** allows you to select and configure settings related to the Matter protocol.
>
> In this step, you need to select and upload the CD file and select the DAC template added in the Certificate configuration. Please ensure that the VID (Vendor ID) and PID (Product ID) in the CD file match those in the DAC template.
>
> You can replace the CD file with another one as needed when creating a version later.

1. On the product page, select **Matter Config** in the **Config Item** area to navigate to the **Matter Config** page.



2. On the **Matter Config** page, select and enter the relevant information in the **Matter Settings** area.



Description of Matter Settings information:

Information	Mandatory	Description
CD File	Yes	Notes for uploading files: ◦ The VID and PID in the CD file must match those in the DAC template. ◦ If the CD file contains multiple PIDs, at least one must match the PID in the DAC template. **Note**: Only **.der** format CD files are supported.

Information	Mandatory	Description
PID	Yes	After uploading the CD file, the corresponding PID will be displayed.
TEST DAC Template	Yes	Select the DAC template configured under the TEST environment in **Certificate**. The VID and PID in the DAC template must match those in the CD file.
PROD DAC Template	No	Select the DAC template configured under the PROD environment in **Certificate**. The VID and PID in the DAC template must match those in the CD file.
Spike2Iteration	Yes	A parameter of the SPAKE2+ protocol used during Matter commissioning, ensuring the security of the commissioning process.
Commissioning Flow	Yes	The device commissioning flow, adaptable to different scenarios: ◦ Standard Commissioning Flow ◦ User-Intent Commissioning Flow ◦ Custom Commissioning Flow

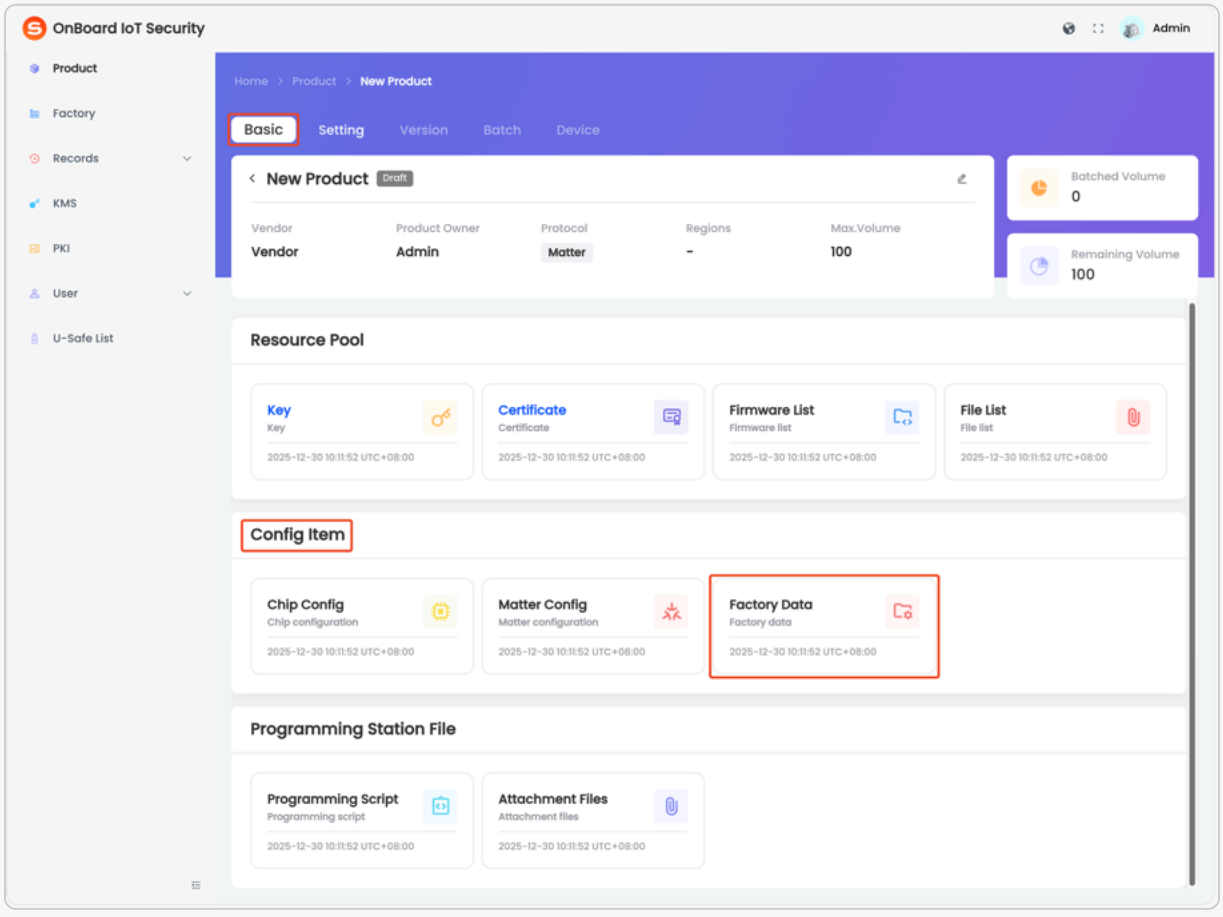
Information	Mandatory	Description
Discovery Mode	Yes	Supported discovery modes: ◦ BLE: Bluetooth Low Energy. ◦ On IP network: Discovery through an IP network.

3. Click **Save** to complete the setting of the Matter Config.

Factory Data (required)

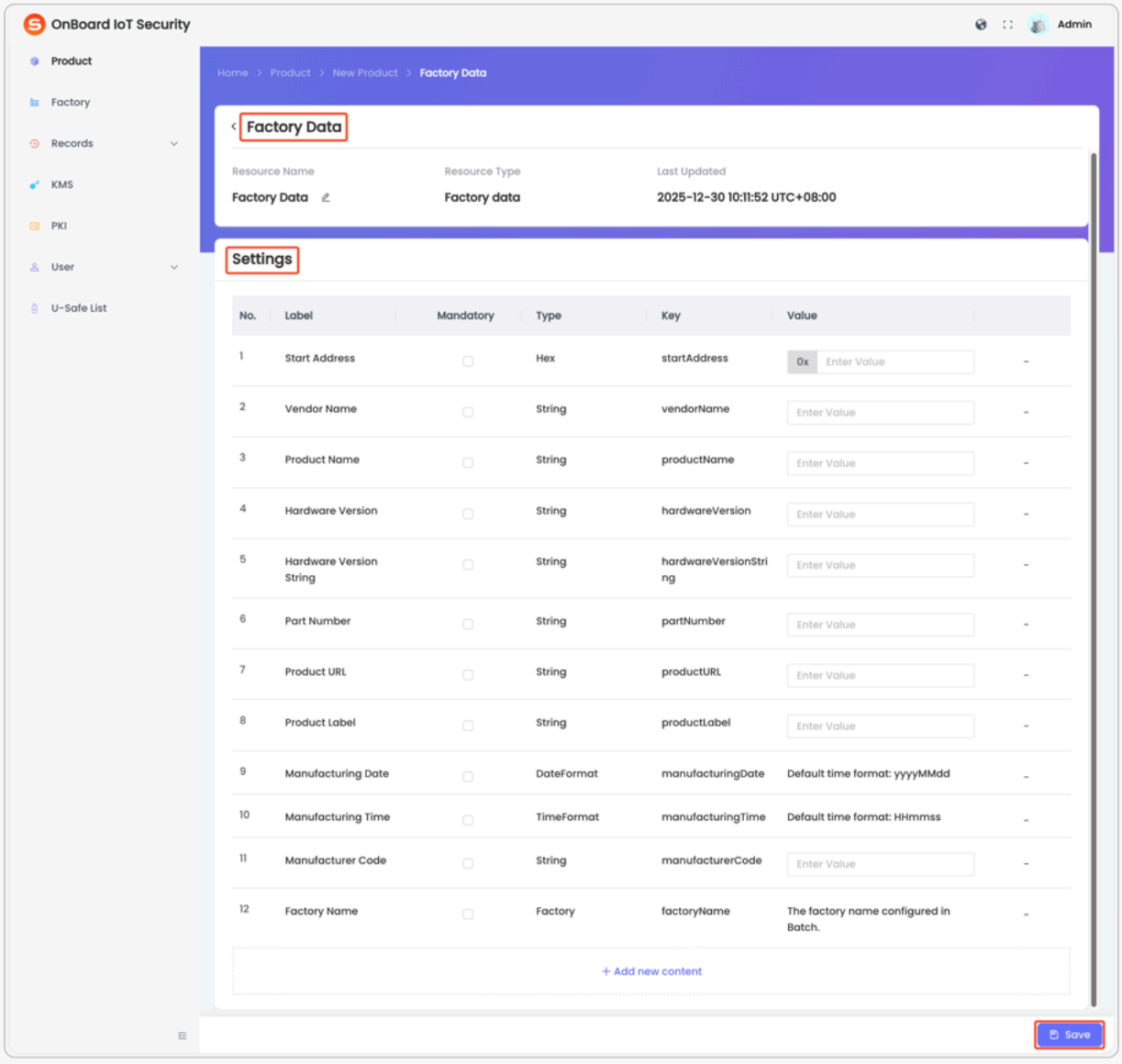
> **TIP**
>
> **Factory Data** allows you to configure factory-related information and supports all fields required by the Matter specification. You can change the Factory Data as needed when creating a version later.

1. On the product page, select **Factory Data** in the **Config Item** area to navigate to the **Factory Data** page.



2. On the **Factory Data** page, locate the **Settings** area and enter the Factory Data information of the product (values can be left empty).

> **NOTE**
>
> If the checkbox in the **Mandatory** column of a Factory Data item is selected, members must enter a value for that item when creating a version.



Description of Factory Data information:

> **TIP**
>
> The supported data types include: String, Number, Hex, Factory, DateFormat, TimeFormat, and File.

Information	Type	Description
Start Address	Hex	Start address.
Vendor Name	String	Vendor name.
Product Name	String	Product name.
Hardware Version	String	Hardware version.
Hardware Version String	String	Hardware version information presented as a string, which may include a more detailed version description. For example, HW_V1.0_20240101.
Part Number	String	Part number.
Product URL	String	Product link.
Product Label	String	Product label.
Manufacturing Date	DateFormat	Manufacturing date, default format is yyyyMMdd.

Information	Type	Description
Manufacturing Time	TimeFormat	Manufacturing time, default format is HHmmss.
Manufacturing Code	String	Manufacturing code.
Factory Name	Factory	The factory name configured in batch.

3. Click **Save** to complete the configuration of the Factory Data.

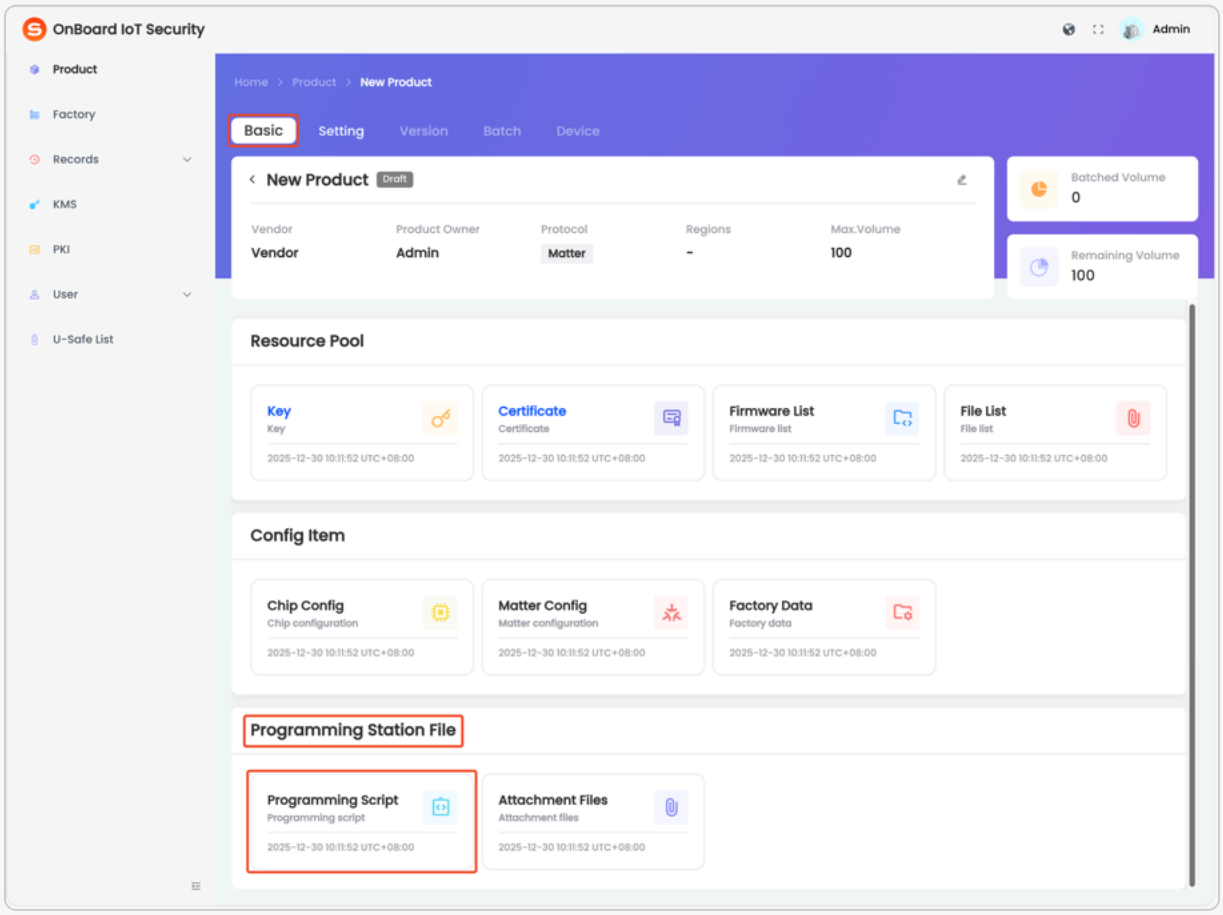
Step three: Upload programming station files

> **TIP**
>
> Files in the **Programming Station File** area will be used for batch creation. The files uploaded when configuring Programming Script and Attachment Files can be downloaded at any time from the configuration page of the corresponding resource.

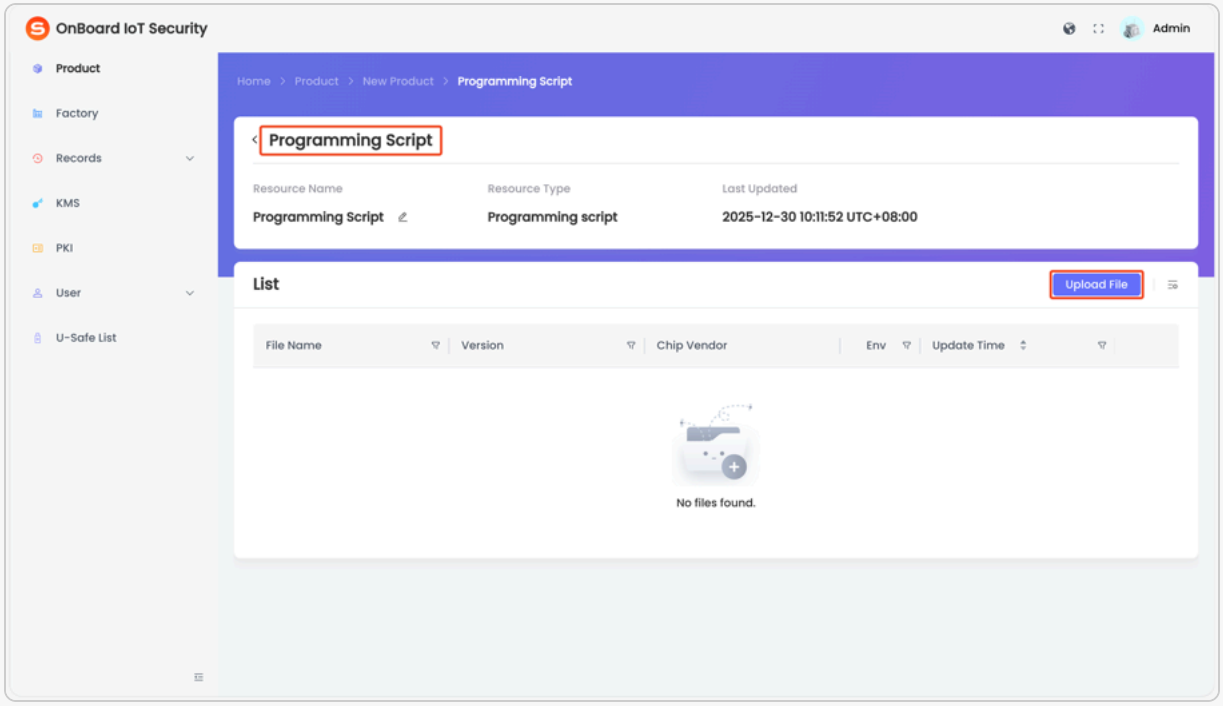
Programming Script (optional)

> **TIP**
>
> **Programming Script** allows you to upload script files used for device programming (any script format is supported). The uploaded programming scripts can be selected when creating a batch.
>
> You can select and upload the programming script as needed. If it is not required at the moment, you can also select it for a specific batch later in the Programming Station client.

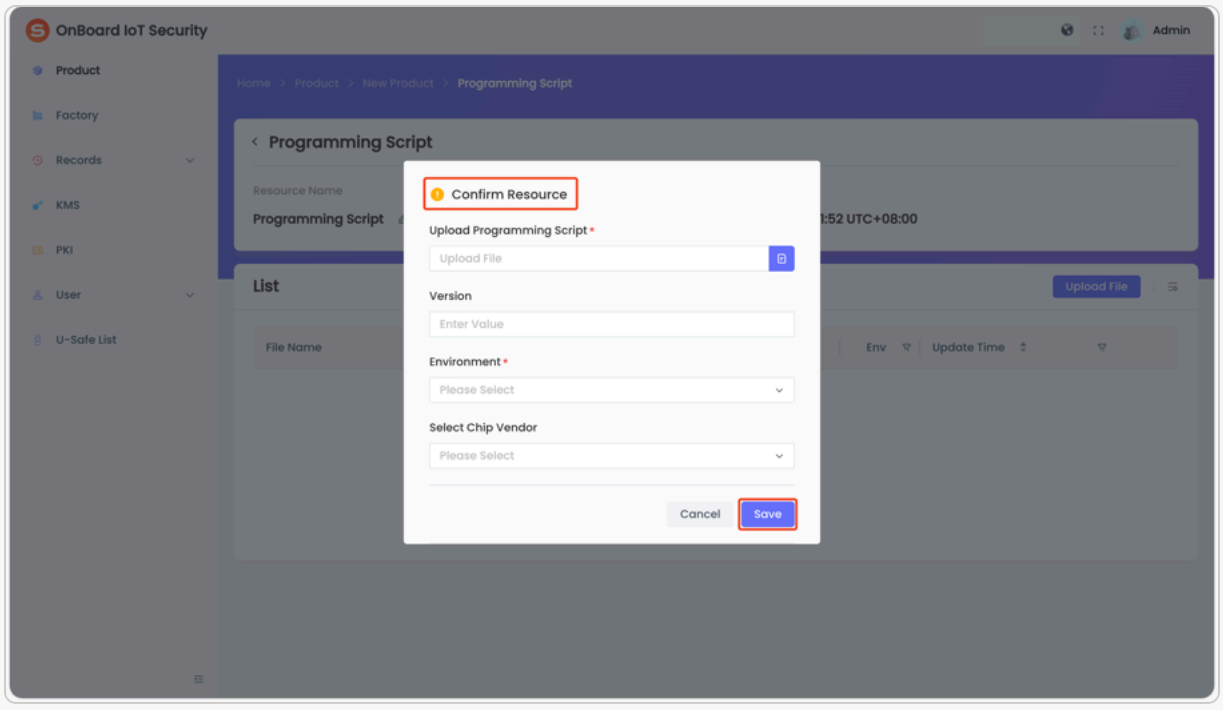
1. On the product page, select **Programming Script** in the **Programming Station File** area to navigate to the **Programming Script** page.



2. On the **Programming Script** page, click **Upload File**.



3. In the **Confirm Resource** window, select and enter the information related to the programming script.



Description of Programming Script information:

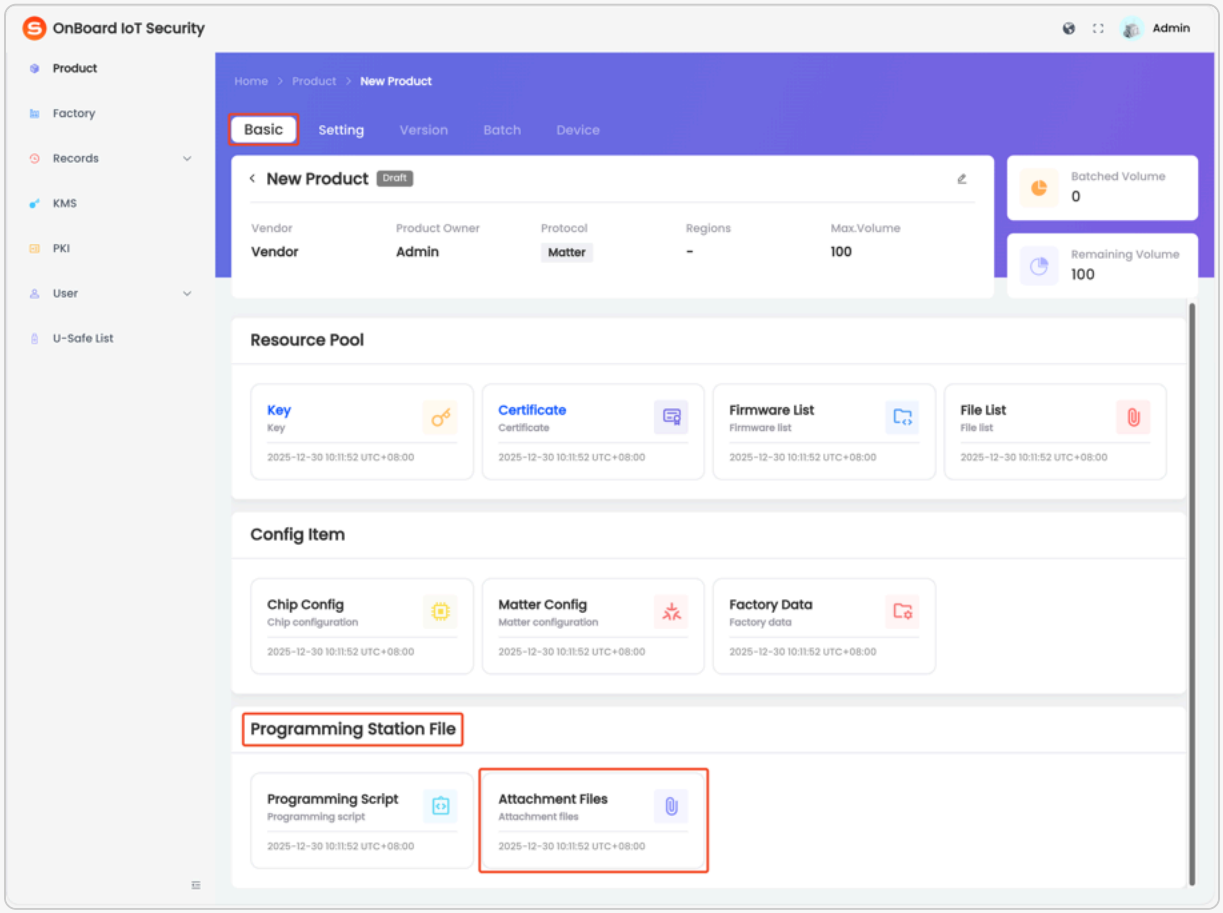
Information	Mandatory	Description
Upload Programming Script	Yes	Upload files by selecting the local programming script files.
Version	No	Indicate the version of the programming script.
Environment	Yes	Two environments can be selected: **TEST** and **PROD**.
Select Chip Vendor	No	Select the corresponding chip vendor.

4. Click **Save** to complete the configuration of the Programming Script.

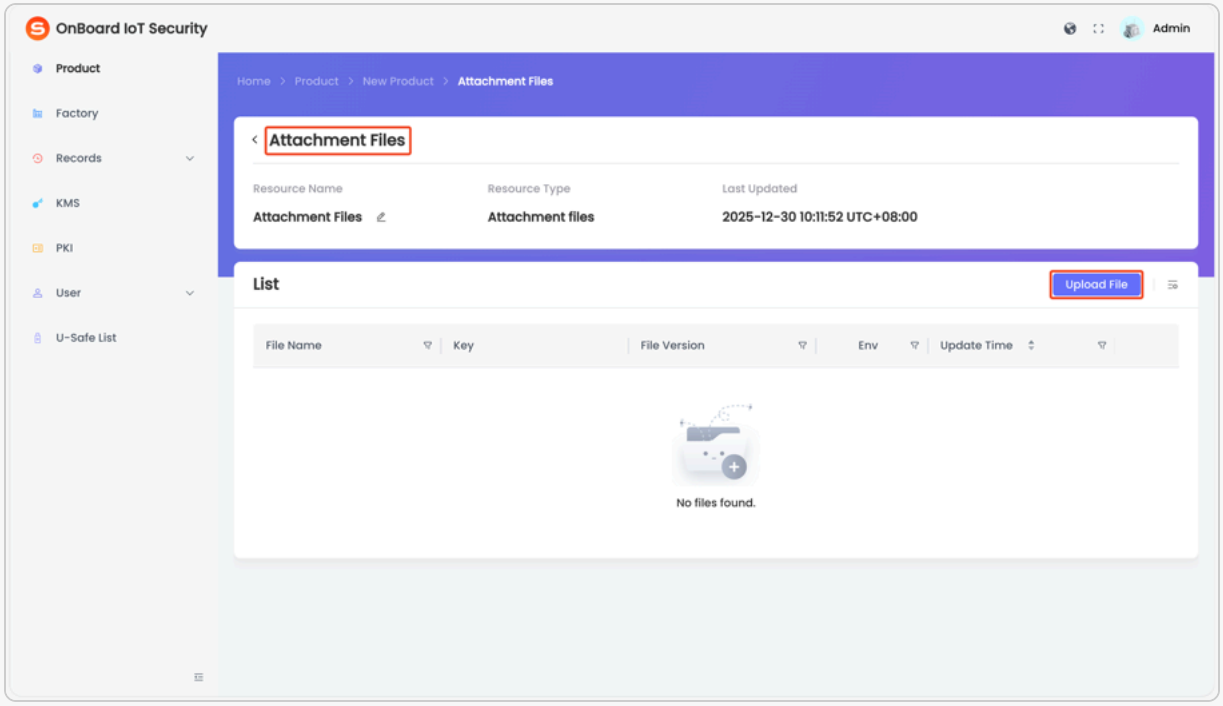
Attachment Files (optional)

> **TIP**
>
> **Attachment Files** allows you to upload other files required for batch creation (excluding programming scripts). Files in any format are supported. The uploaded files can be selected when creating a batch.

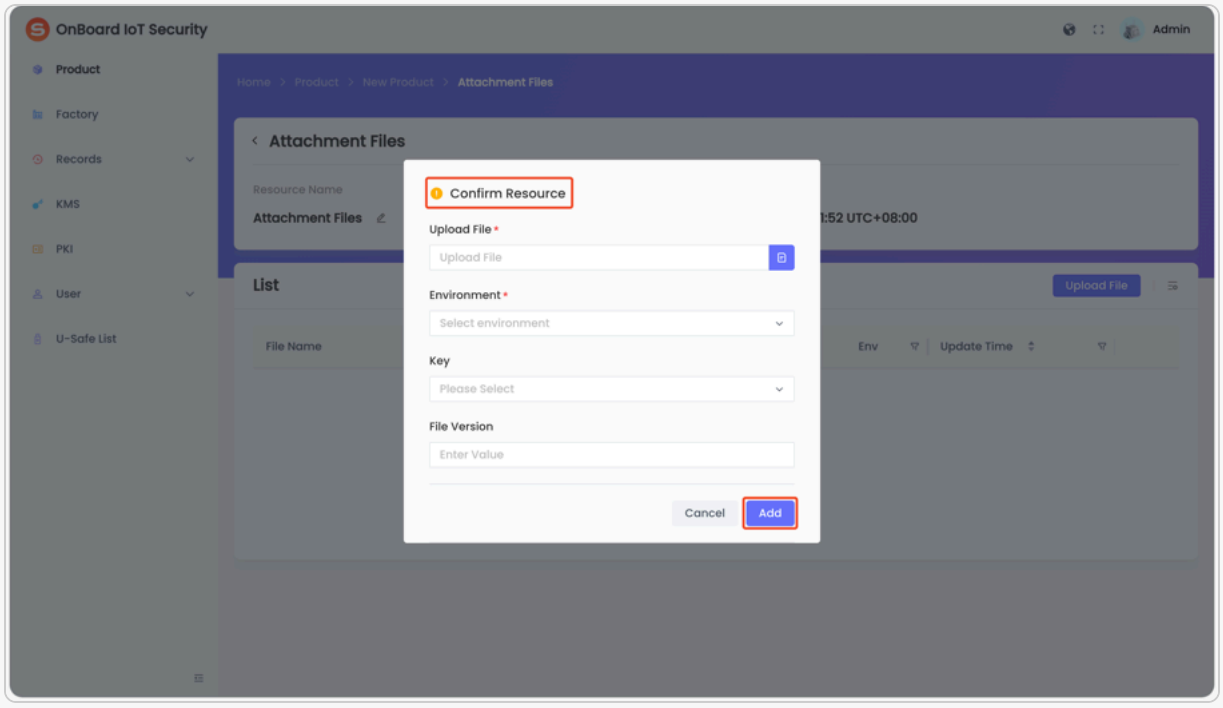
1. On the product page, select **Attachment Files** in the **Programming Station File** area to navigate to the **Attachment Files** page.



2. On the **Attachment Files** page, click **Upload File**.



3. In the **Confirm Resource** window, select and enter the relevant information.



Description of Attachment Files information:

Information	Mandatory	Description
Upload File	Yes	Upload files by selecting the local files.
Environment	Yes	Two environments can be selected: **TEST** and **PROD**.
Key	No	Assign keys to files. During later programming, the programming script can retrieve and use the corresponding files through their keys. **Note**: A file can be assigned multiple unique keys. Meanwhile, the same key can also be assigned to multiple files. You can add new keys or select from existing keys.
File Version	No	Identify the file version and use it to track file changes.

4. Click **Add** to complete the configuration of Attachment Files.

Version: v3.2.1

How to check products in OnBoard™ IoT Security

This article will guide you on how to check product basic information and configurations in OnBoard™ IoT Security (OBIS).

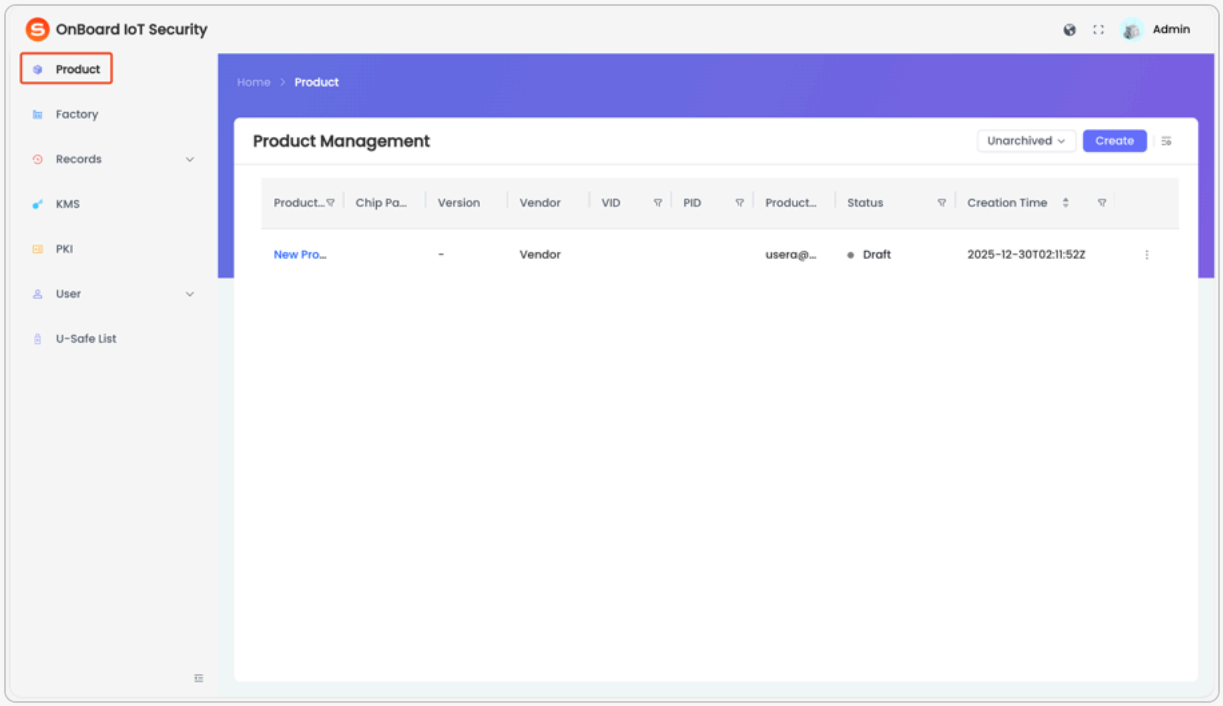
Prerequisites

Before getting started, please ensure that you have product role permissions for the corresponding product.

Instructions

1. Sign in to OBIS.
2. In the left menu, click **Product**, and then select the corresponding product to navigate to the product page.

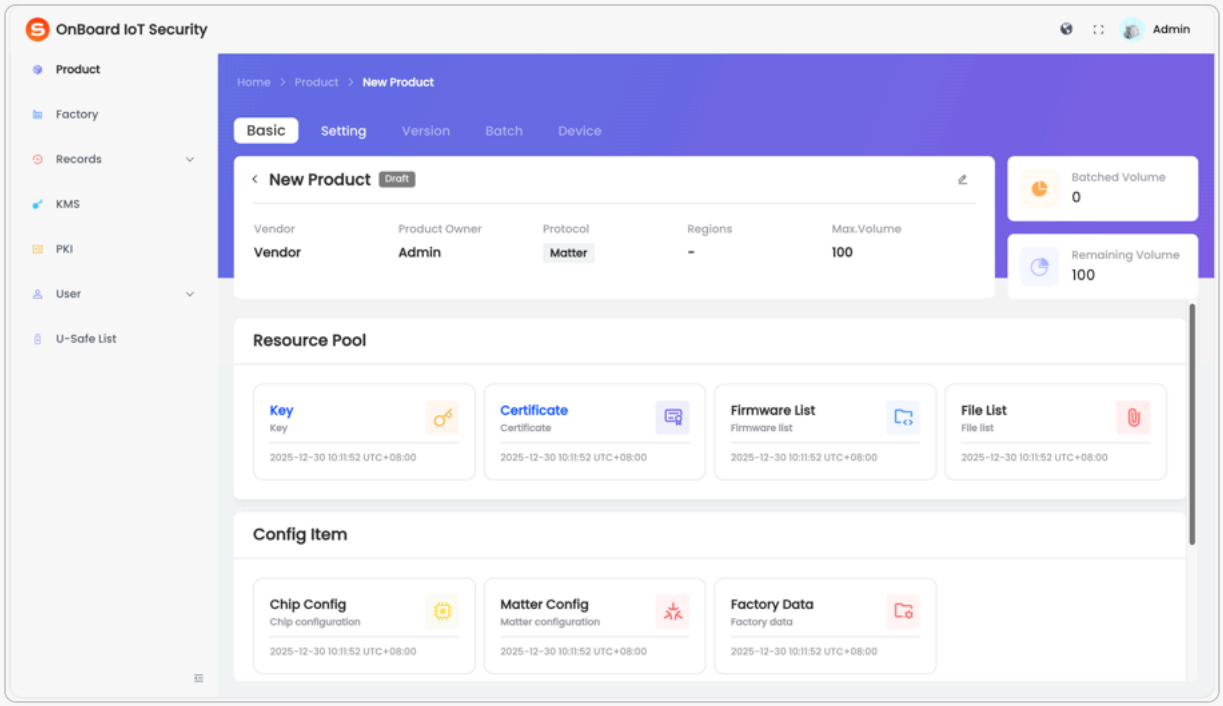
> **TIP**
>
> The product list includes both the products you are involved in and all products created within the tenant, with unarchived products displayed by default.



Description of product status:

Status	Description
Draft	Status before a version is created.
Released	Status after a version is created. **Note**: If all versions of a product are deleted, the product status will revert to **Draft**.

3. On the product page, you can check the product-related information (such as **Vendor**, **Product Owner**, and **Protocol**).


OnBoard IoT Security
Admin
Product
Factory
Records
KMS
PKI
User
U-Safe List
Home > Product > New Product
Basic
Setting
Version
Batch
Device
New Product
Draft
Vendor
Vendor
Product Owner
Admin
Protocol
Matter
Regions
-
Max.Volume
100
Batched Volume
0
Remaining Volume
100
Resource Pool
Key
Key
2025-12-30 10:11:52 UTC+08:00
Certificate
Certificate
2025-12-30 10:11:52 UTC+08:00
Firmware List
Firmware list
2025-12-30 10:11:52 UTC+08:00
File List
File list
2025-12-30 10:11:52 UTC+08:00
Config Item
Chip Config
Chip configuration
2025-12-30 10:11:52 UTC+08:00
Matter Config
Matter configuration
2025-12-30 10:11:52 UTC+08:00
Factory Data
Factory data
2025-12-30 10:11:52 UTC+08:00

Version: v3.2.1

How to manage products in OnBoard™ IoT Security

> **TIP**
>
> On the configuration page of the corresponding resource, you can download the files uploaded when configuring Firmware List, File List, Matter Config, Factory Data, Programming Script, and Attachment Files.

This article will guide you through the following operations in OnBoard™ IoT Security (OBIS):

- Edit product information
- Archive or unarchive a product

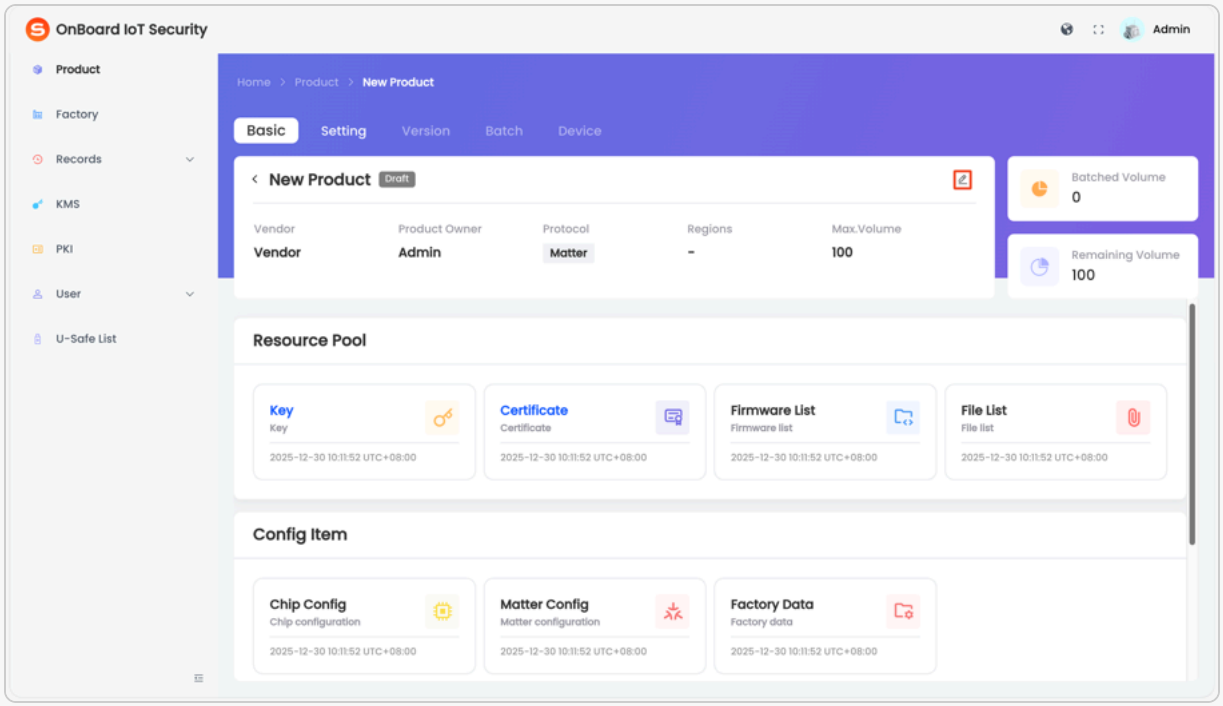
Prerequisites

Before getting started, please ensure that you have product role permissions for the corresponding product: Product Owner or Product Manager.

Edit product information

1. Sign in to OBIS.
2. In the left menu, click **Product**, and then select the corresponding product to navigate to the product page.

3. On the **Basic** tab, click the Edit icon to change the product information (such as **Product Name**, **Vendor**, and **Product Owner**).

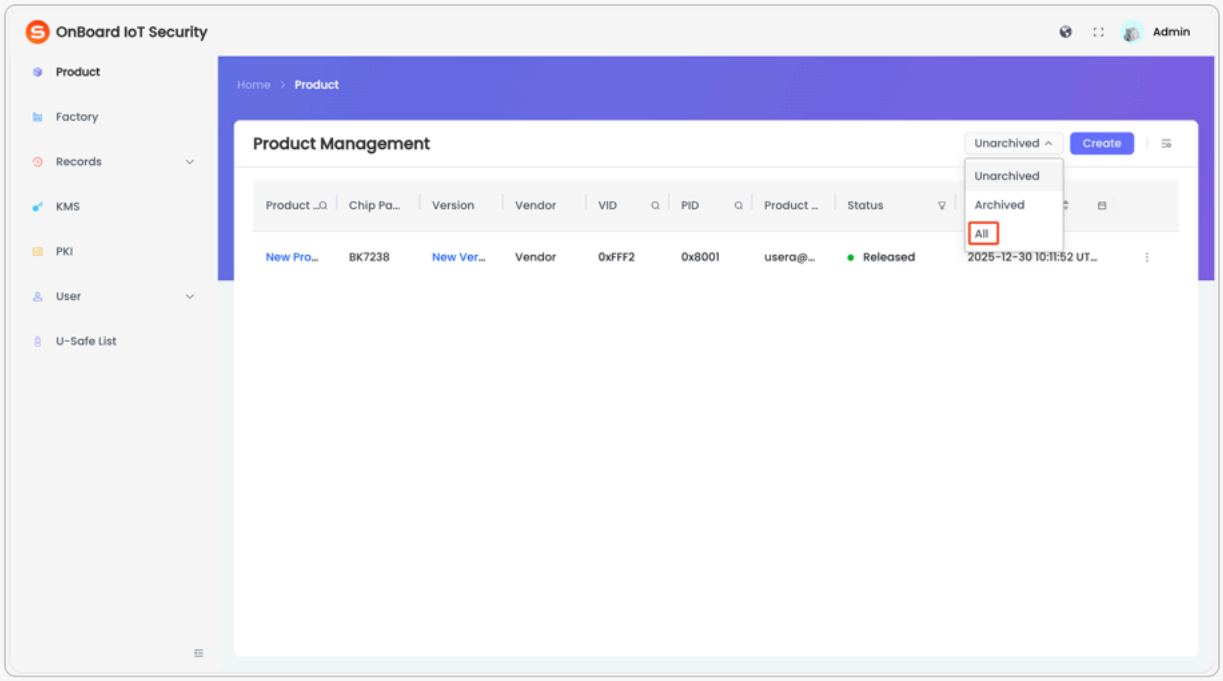


Archive or unarchive a product

> **TIP**
>
> The product list defaults to displaying unarchived products. Once a product is archived, no further editing can be made, including information modifications, resource configuration, version creation, and batch creation.

1. Sign in to OBIS.

2. In the left menu, click **Product**, and then select **All** in the top-right corner of the **Product Management** page.



3. Click the Options ⋮ icon for the corresponding product and then select **Archive** or **Unarchive** to archive or unarchive the product.

Version: v3.2.1

How to manage product members in OnBoard™ IoT Security

> **TIP**
>
> You can check members' activity records for the product in the **Logs** area of the Setting tab.

This article will guide you through the following operations in OnBoard™ IoT Security (OBIS) to manage user access and permissions for a product:

- Add a user
- Update user role
- Remove a user
- Add a user group
- Update user group role
- Remove a user group

Prerequisites

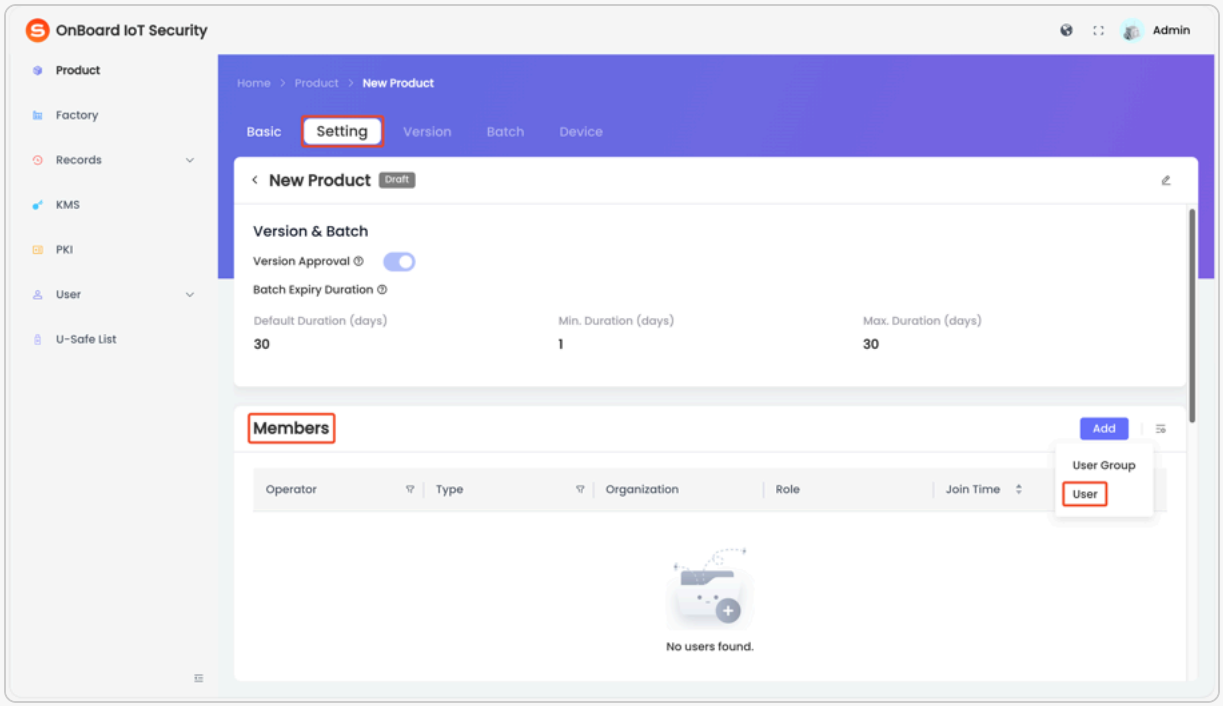
Before getting started, please ensure that:

- The users and user groups have already been created.
- You have product role permissions for the corresponding product: Product

Owner or Product Manager.

Add a user

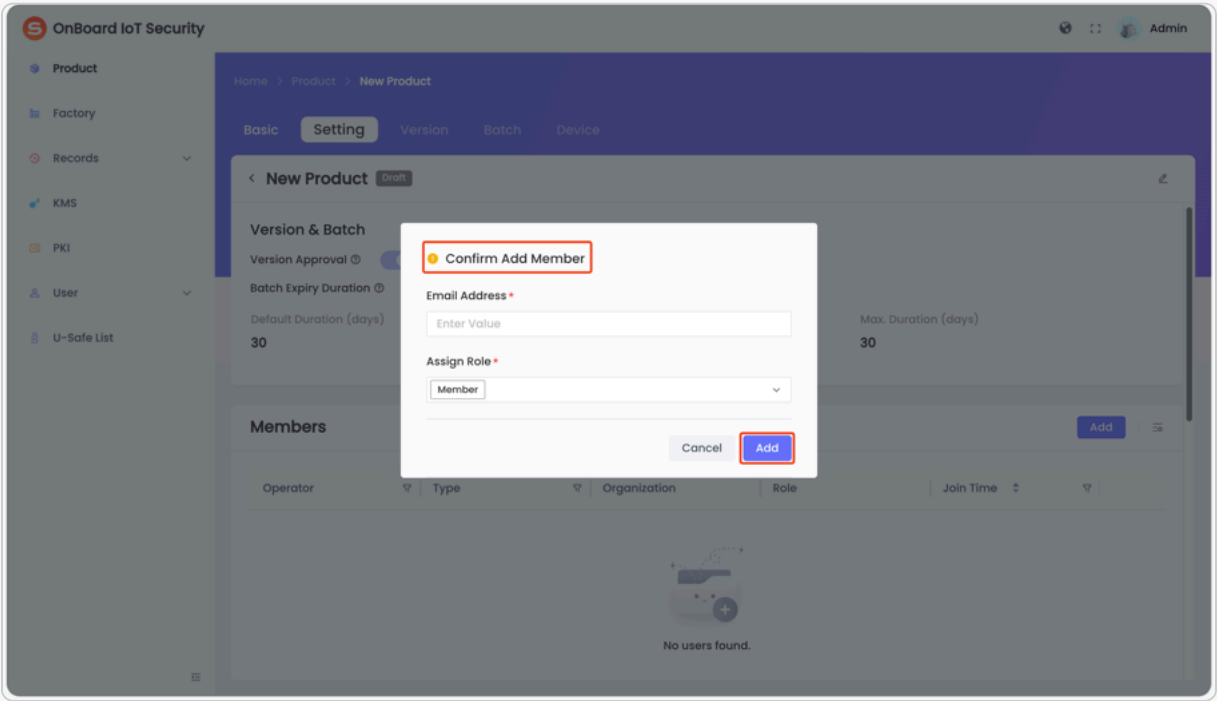
1. Sign in to OBIS.
2. In the left menu, click **Product**, and then select the corresponding product to navigate to the product page.
3. On the product page, select the **Setting** tab, click **Add** in the **Members** area and then select **User**.



4. In the **Confirm Add Member** window, enter the user's email address and then select the roles.

TIP

- A user can be assigned multiple roles. For more information, refer to the description of product role permissions.
- The added users will receive an email containing a link, through which they can use to access the product details.
- Only users with the **Active** status can be added. Users from other tenants can be added. For example, a Product Manager from Tenant A can add a user from Tenant B to their product team.



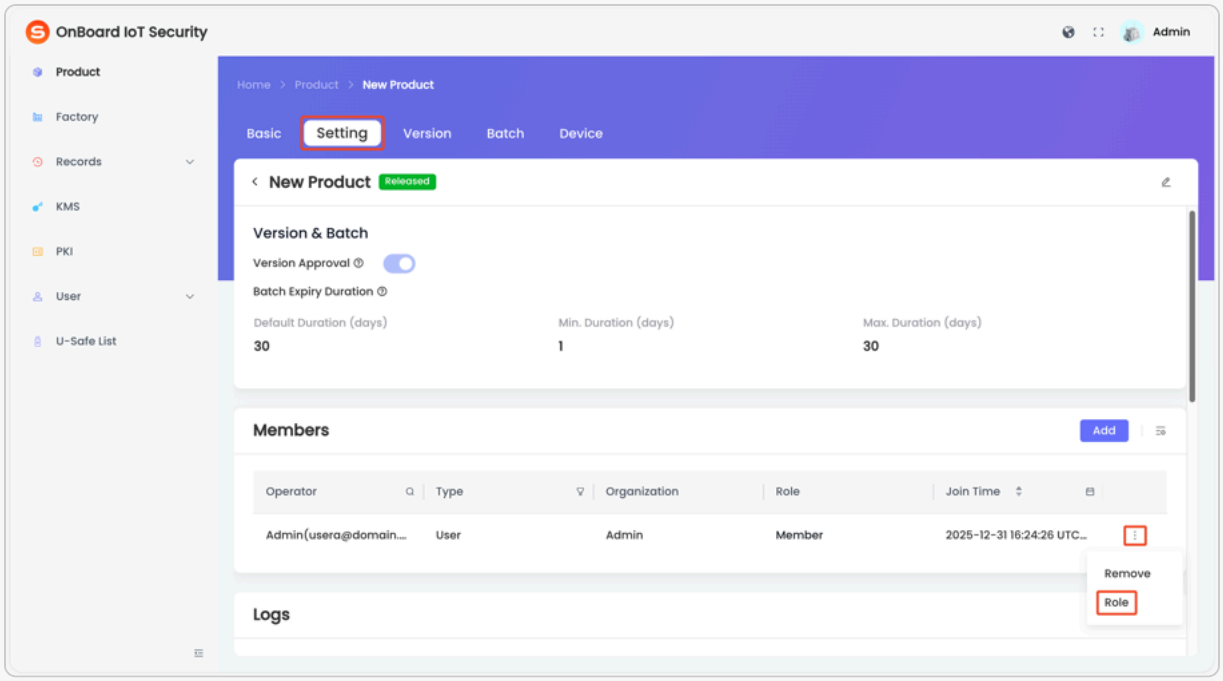
5. Click **Add** to complete adding a member.

Update user role

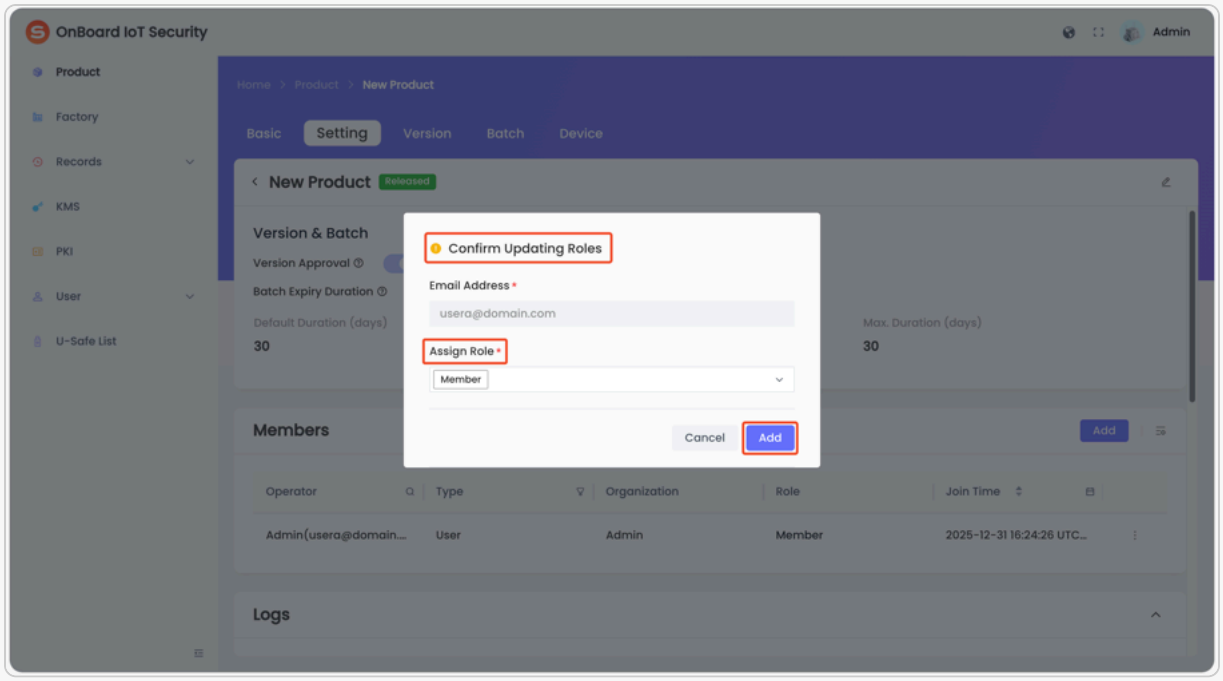
1. Sign in to OBIS.

2. In the left menu, click **Product**, and then select the corresponding product to navigate to the product page.

3. On the product page, select the **Setting** tab. In the **Members** area, click the Options ⋮ icon for the corresponding user and then select **Role**.



4. In the **Confirm Updating Roles** window, select the corresponding roles and then click **Add** to complete the update.

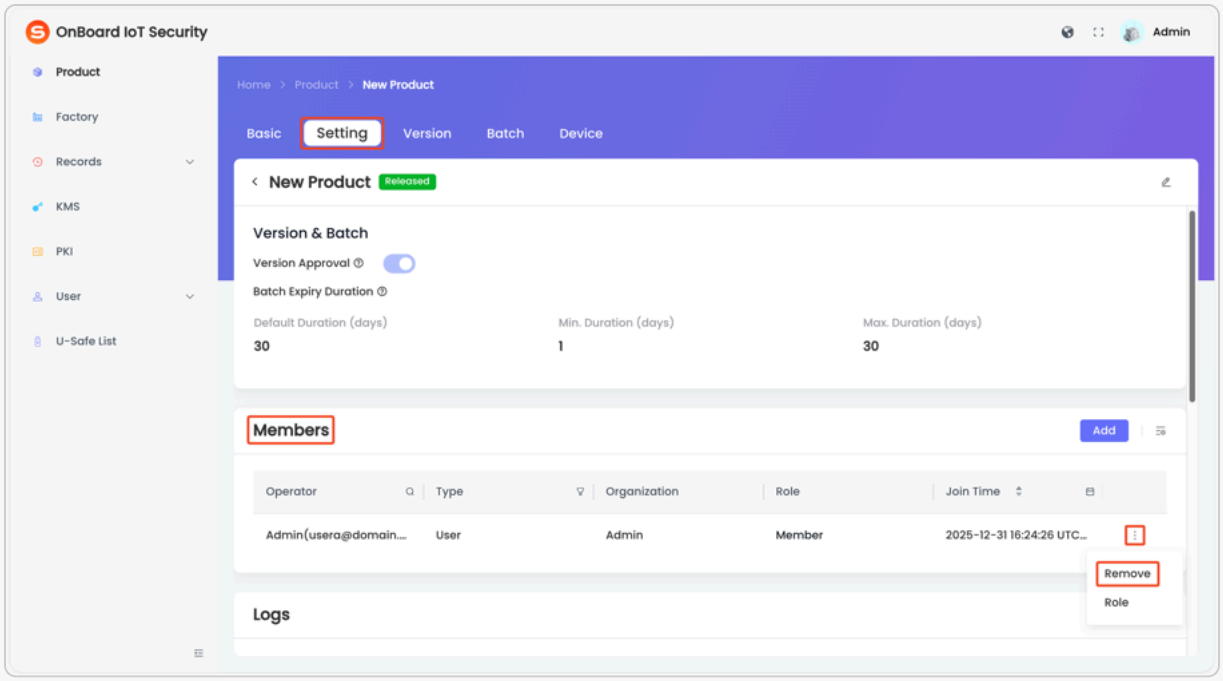


Remove a user

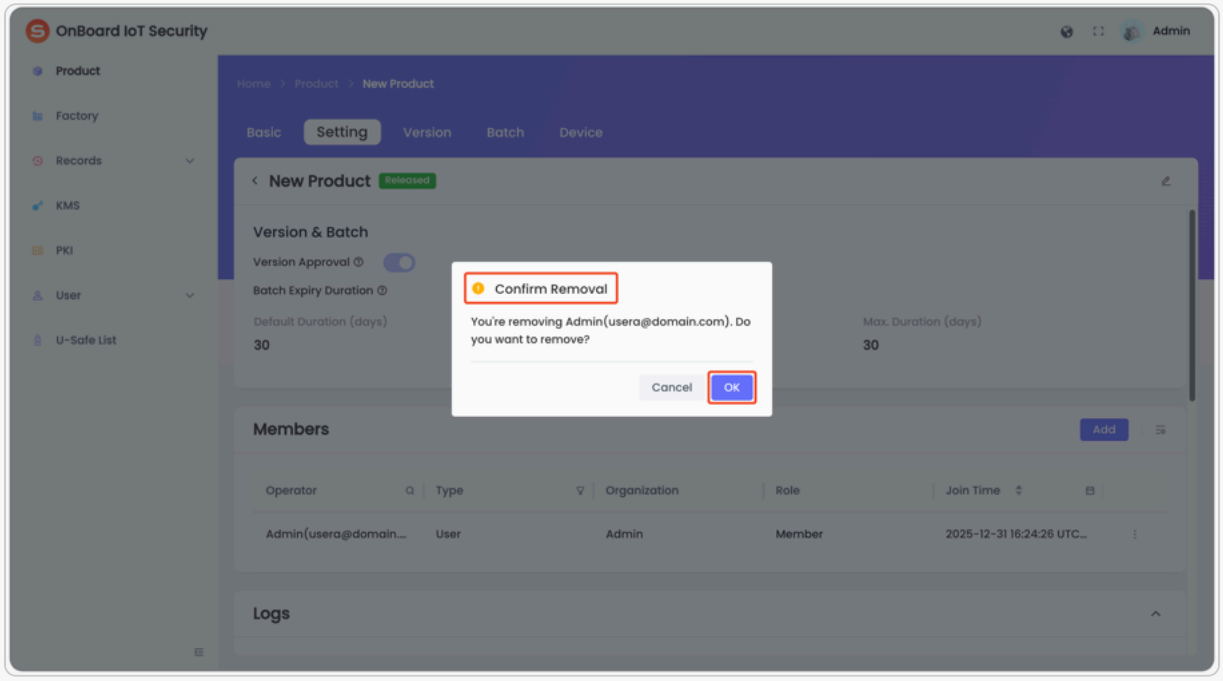
1. Sign in to OBIS.

2. In the left menu, click **Product**, and then select the corresponding product to navigate to the product page.

3. On the product page, select the **Setting** tab. In the **Members** area, click the Options ⋮ icon for the corresponding user and then select **Remove**.



4. In the **Confirm Removal** window, verify the information and then click **OK** to remove the user.

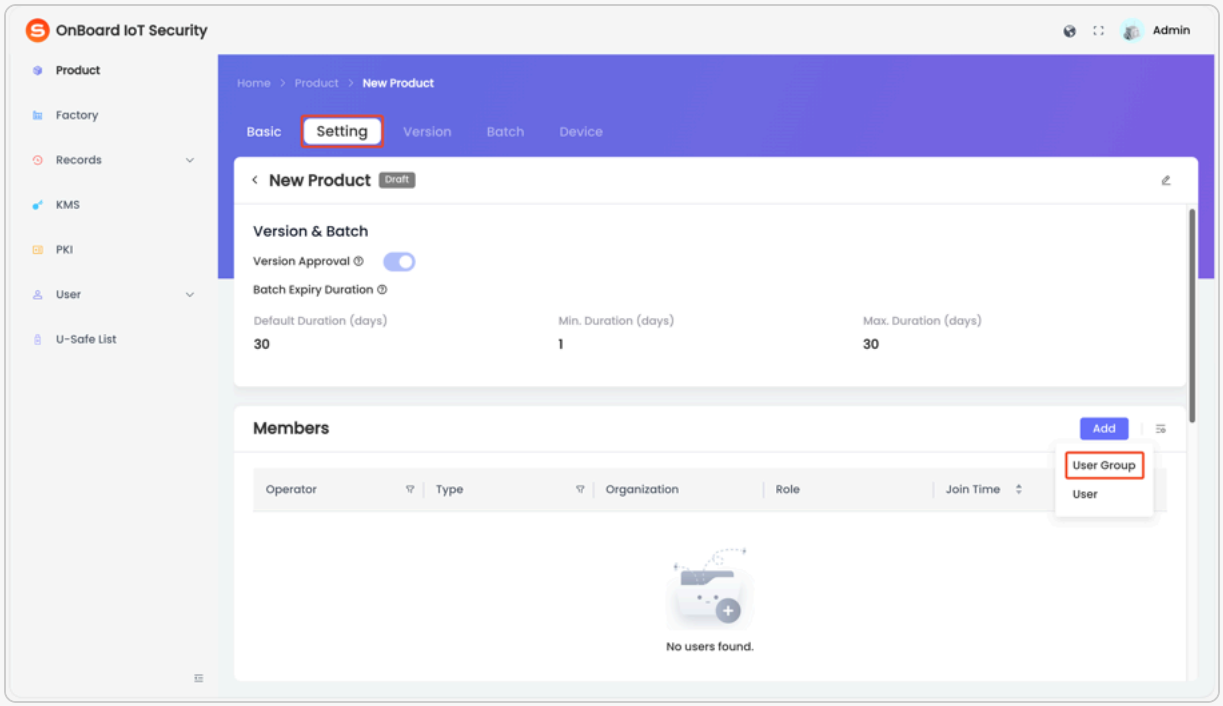


Add a user group

1. Sign in to OBIS.

2. In the left menu, click **Product**, and then select the corresponding product to navigate to the product page.

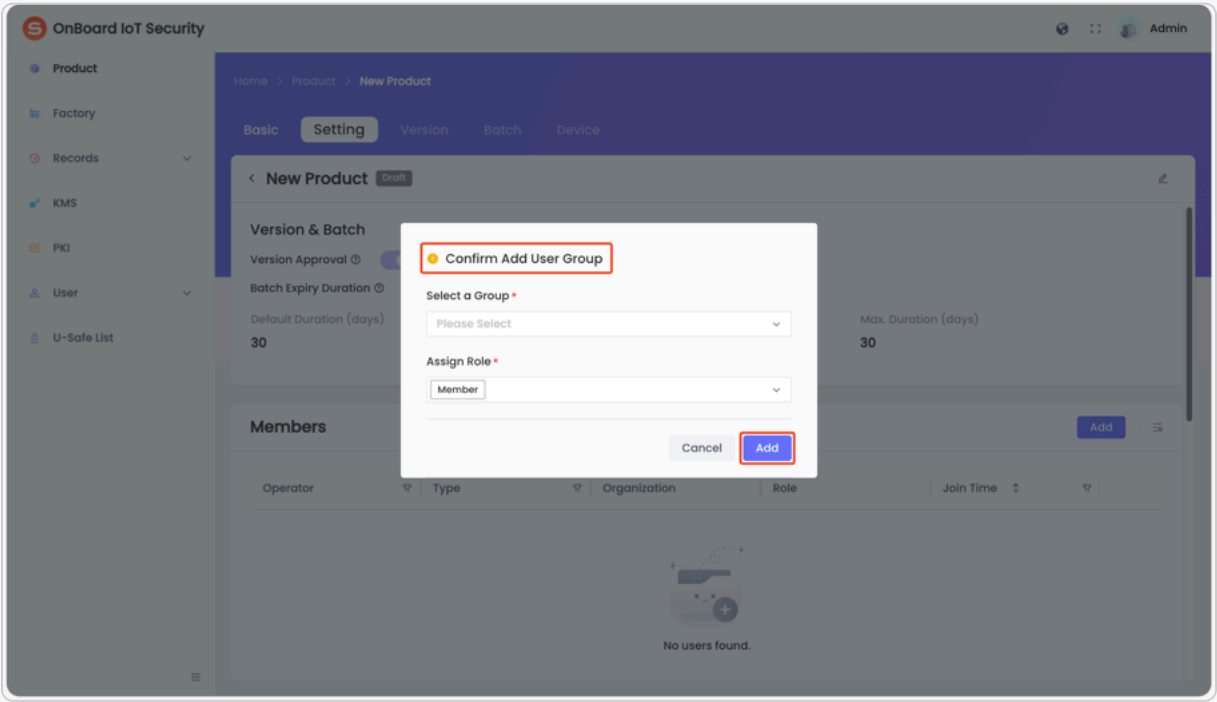
3. On the product page, select the **Setting** tab, click **Add** in the **Members** area and then select **User Group**.



4. In the **Confirm Add User Group** window, select the user group and roles.

TIP

- A user group can be assigned multiple roles. For more information, refer to the description of product role permissions.
- Users within the user group will receive an email containing a link, through which they can use to access the product details.
- Only user groups under the same tenant can be added.



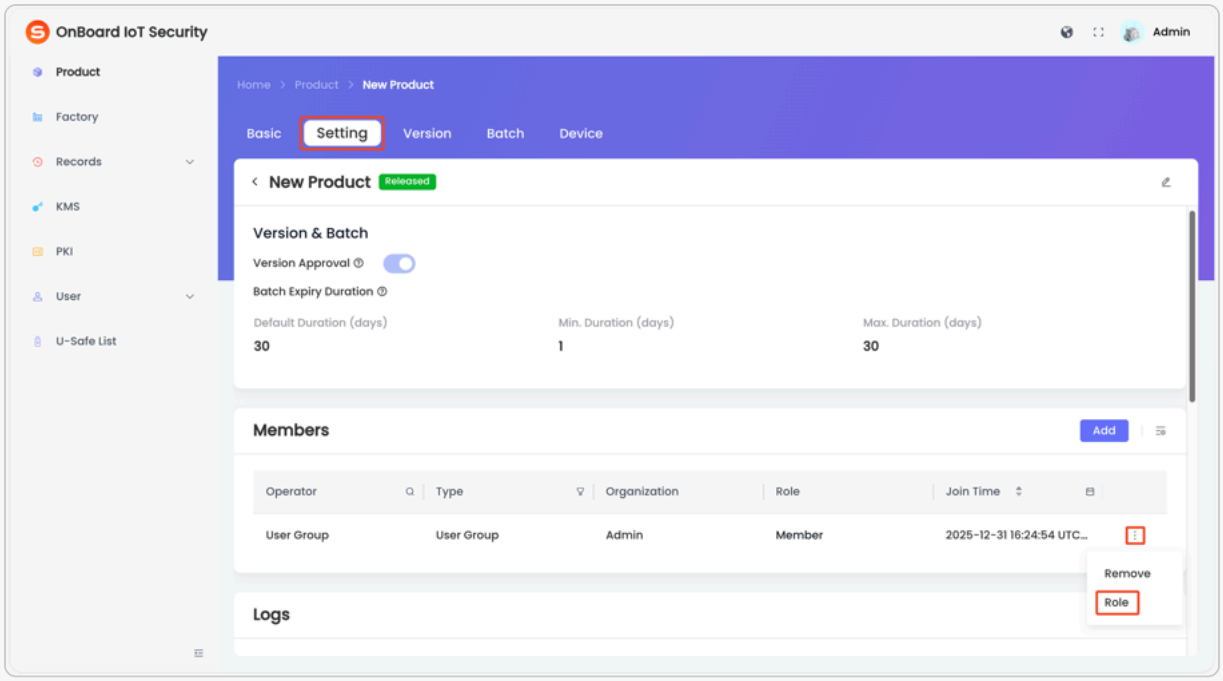
5. Click **Add** to complete adding a user group.

Update a user group role

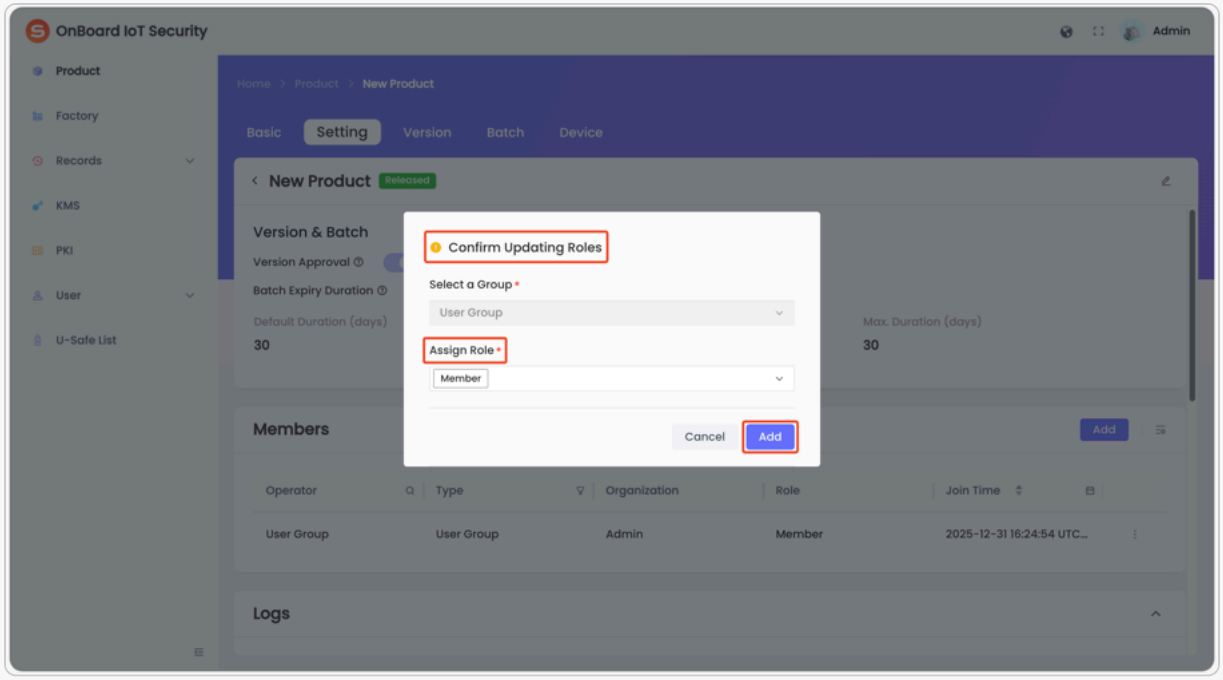
1. Sign in to OBIS.

2. In the left menu, click **Product**, and then select the corresponding product to navigate to the product page.

3. On the product page, select the **Setting** tab. In the **Members** area, click the Options ⋮ icon for the corresponding user group and then select **Role**.

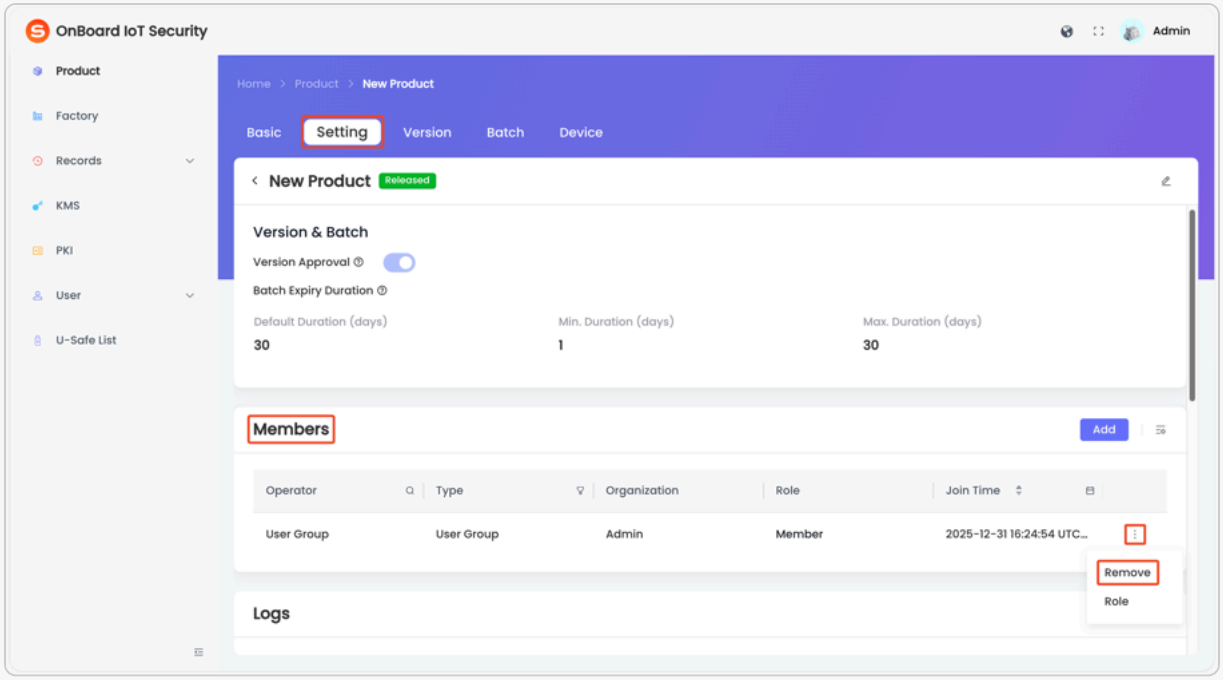


4. In the **Confirm Updating Roles** window, select the corresponding roles and then click **Add** to complete the update.

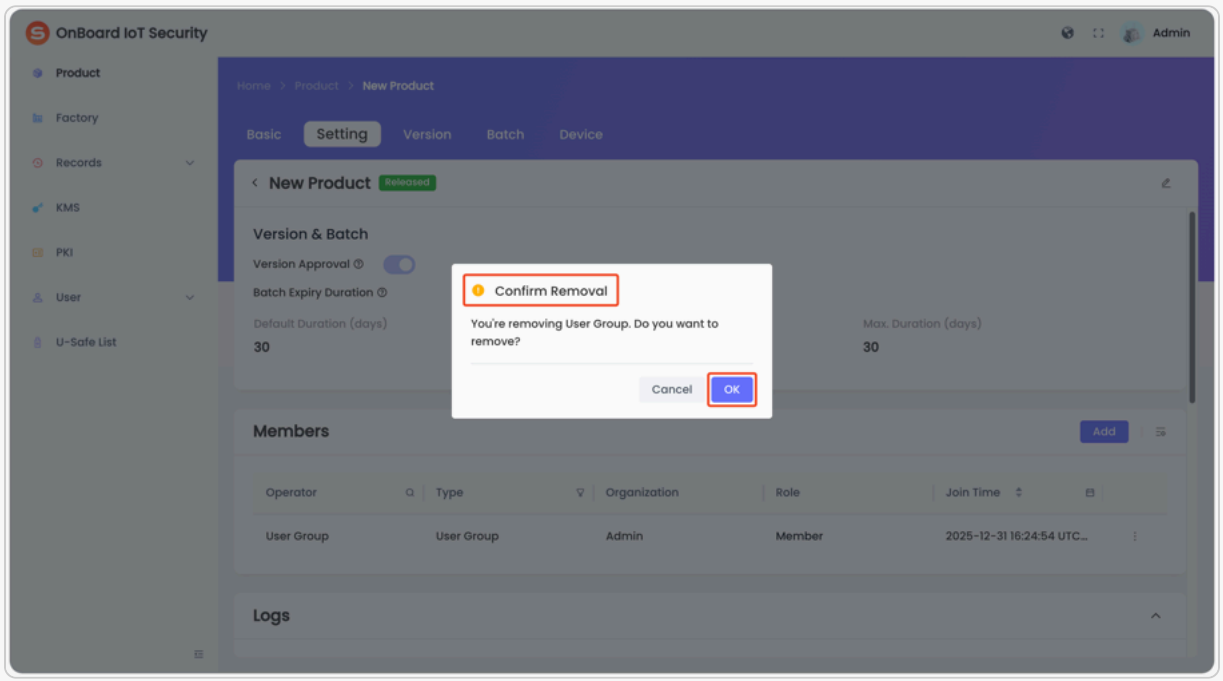


Remove a user group

1. Sign in to OBIS.
2. In the left menu, click **Product**, and then select the corresponding product to navigate to the product page.
3. On the product page, select the **Setting** tab. In the **Members** area, click the Options ⋮ icon for the corresponding user group and then select **Remove**.



4. In the **Confirm Removal** window, verify the information and then click **OK** to remove the user group.



Version: v3.2.1

How to create a version in OnBoard™ IoT Security

This article will guide you on how to create a version in OnBoard™ IoT Security (OBIS) to prepare for batch creation. Each product can create multiple versions.

Prerequisites

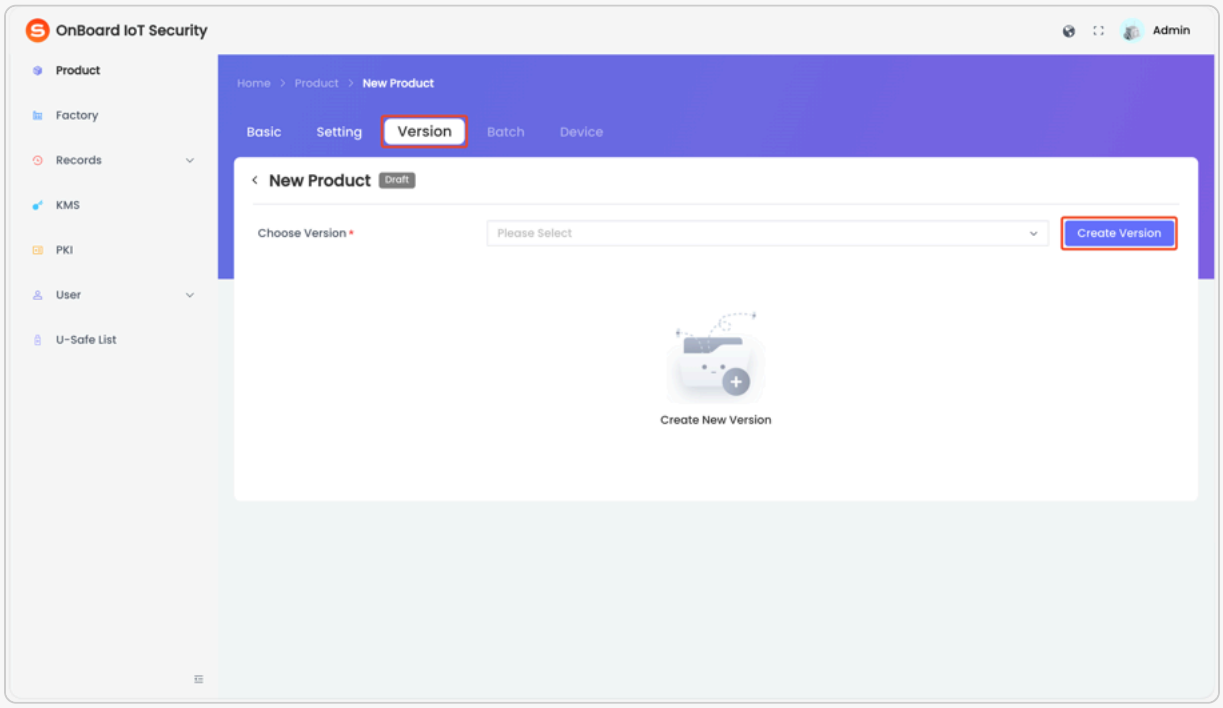
Before getting started, please ensure that:

- The product has already been configured.
- You have product role permissions for the corresponding product: Product Owner, Product Manager, or Version Manager.

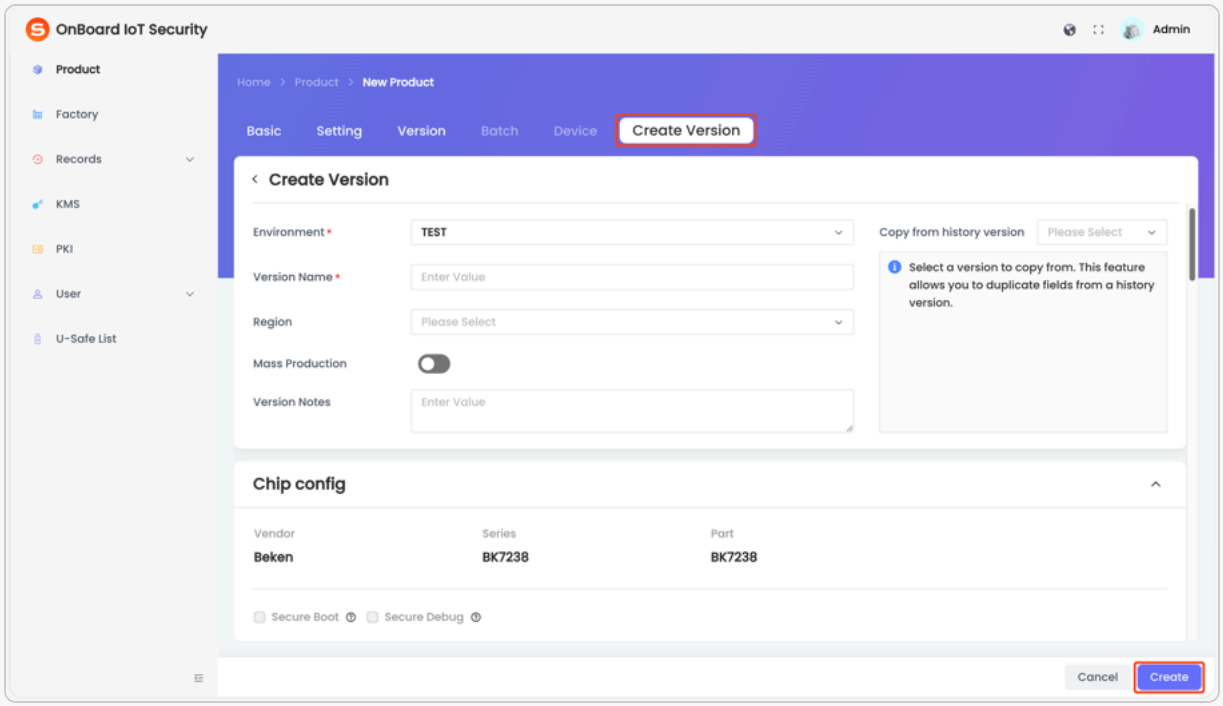
Instructions

1. Sign in to OBIS.
2. In the left menu, click **Product**, and then select the corresponding product to navigate to the product page.
3. On the product page, select the **Version** tab and then click **Create Version**. The page will navigate to the **Create Version** tab.

> **TIP**
>
> If the **Version** tab appears grayed out, it is currently unavailable. Please check whether all required resources and configuration items for your product have been configured. Once everything is properly configured, the **Version** tab will be enabled automatically.



4. On the **Create Version** tab, select and enter the version-related information.



Description of version information:

Information	Mandatory	Description
Environment	Yes	Two environments can be selected: **TEST** and **PROD**. **Note**: To configure a version in the PROD environment, the PROD DAC Template must be configured in the Matter configuration.
Version Name	Yes	The version name must be unique within the same environment.

Information	Mandatory	Description
Region	No	Select and set the region for the version. **Note**: Available regions depend on the supported regions configured when the product was created.
Mass Production	No	Indicate whether the version can be used for mass production. Typically, the **Mass Production** feature is enabled only for versions in the PROD environment after all testing has been completed.
Version Notes	No	Provide additional information related to the version, such as special requirements and important notes.
Copy from history version	No	You can select a historical version as a template for modification. **Note**: After copying from a historical version, the environment can't be changed.
Chip config	Yes	The Chip Config in the **Config Item** area. If the version is in the TEST environment and the **Secure Boot** or **Secure Debug** feature is enabled, you can uncheck these options; in the PROD environment, the settings can't be modified.

Information	Mandatory	Description
Matter config	Yes	The Matter Config in the **Config Item** area. The **CD File** and **PID** can be reselected, while other information can't be modified.
Firmware	Yes	The Firmware List in the **Resource Pool** area. Select the firmware files required for this version.
Other File	No	The File List in the **Resource Pool** area. Select the files required for this version.
Factory Data	Yes	The Factory Data in the **Config Item** area. All values can be modified. **Note**: All Factory Data items marked as required must be provided with valid values. Otherwise, the version can't be created.

5. After verifying that all information is correct, click **Create** at the bottom of the page to complete the version creation.

TIP

If version approval was enabled when the product was created, versions created in the PROD environment will require approval by the Product Owner or Product Manager.

- Before the version is approved, the version status is **Pending for approval** and it can't be used to create a batch.
- Once the version is approved, the version status changes to **Approved**.

Version: v3.2.1

How to check versions in OnBoard™ IoT Security

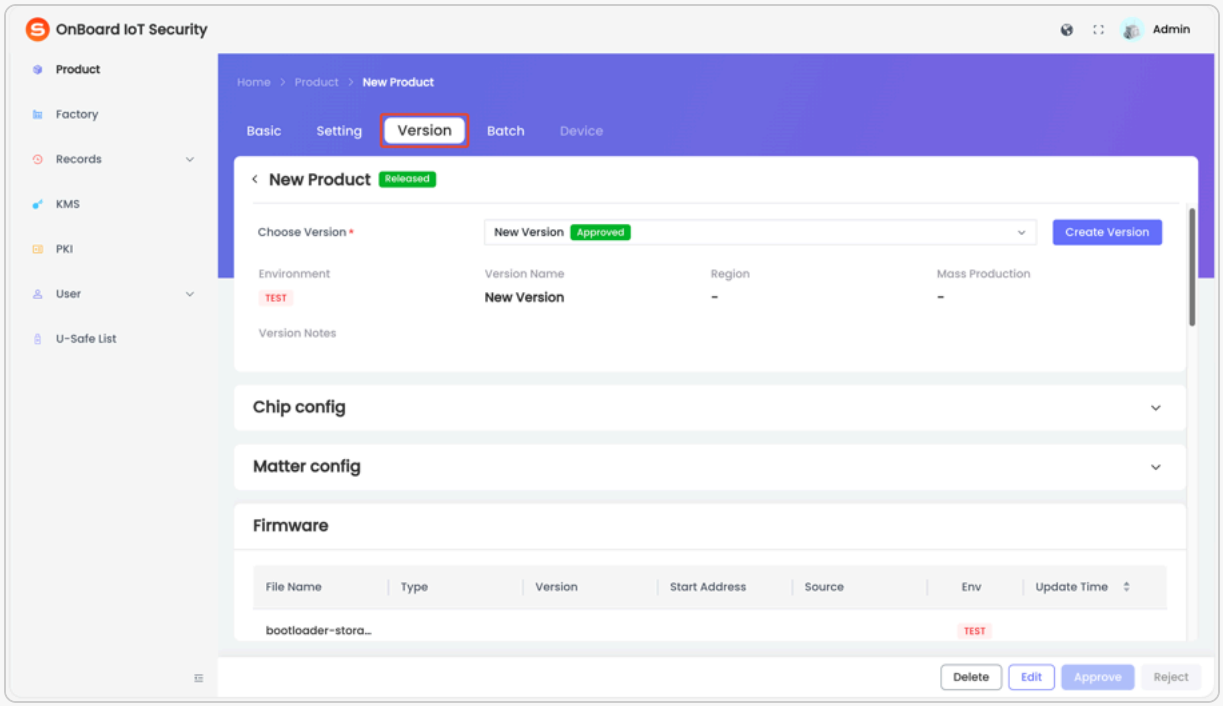
This article will guide you on how to check version information and configuration in OnBoard™ IoT Security (OBIS).

Prerequisites

Before getting started, please ensure that you have product role permissions for the corresponding product.

Instructions

1. Sign in to OBIS.
2. In the left menu, click **Product**, and then select the corresponding product to navigate to the product page.
3. On the product page, select the **Version** tab, and then select the corresponding version on the **Version** tab to check the version-related information (such as **Chip config**, **Matter config**, and **Firmware**).


OnBoard IoT Security
Admin
Product
Factory
Records
KMS
PKI
User
U-Safe List
Home > Product > New Product
Basic
Setting
Version
Batch
Device
New Product
Released
Choose Version *
New Version
Approved
Create Version
Environment
TEST
Version Name
New Version
Region
-
Mass Production
-
Version Notes
Chip config
Matter config
Firmware
File Name
Type
Version
Start Address
Source
Env
Update Time
bootloader-stora...
TEST
Delete
Edit
Approve
Reject

Version: v3.2.1

How to manage versions in OnBoard™ IoT Security

This article will guide you through the following operations in OnBoard™ IoT Security (OBIS):

- Approve or reject a version
- Edit a version
- Delete a version

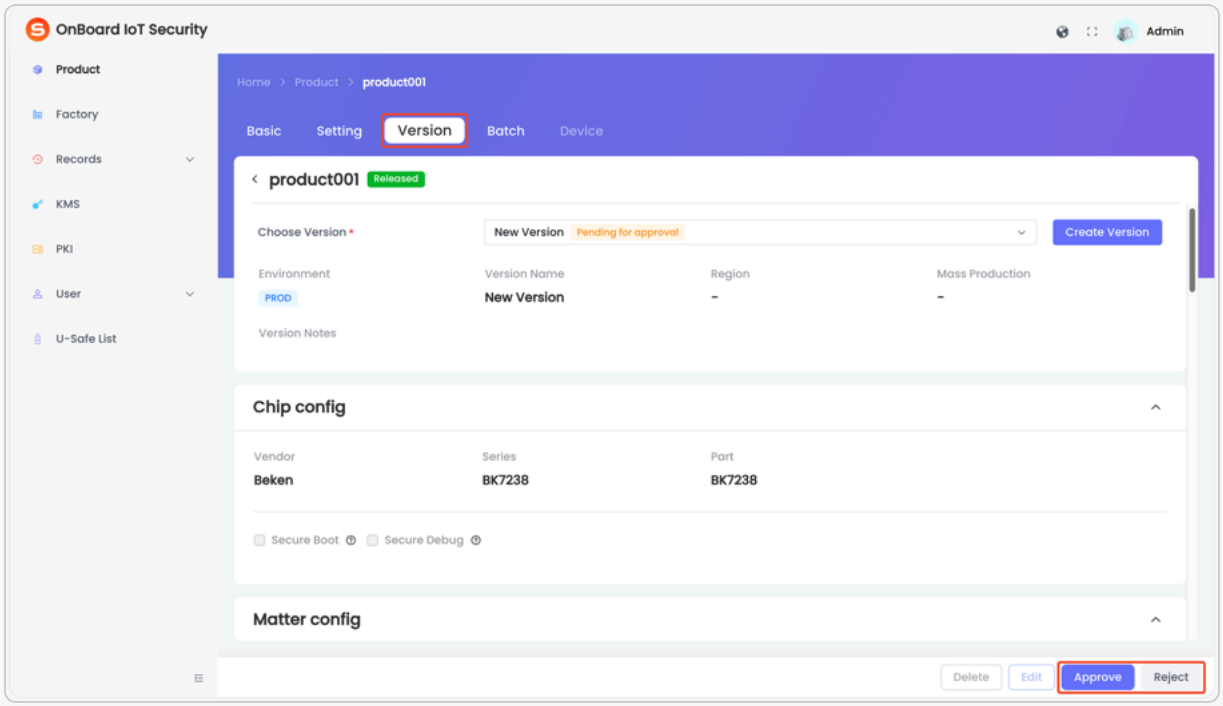
Prerequisites

Before getting started, please ensure that you have product role permissions for the corresponding product: Product Owner, Product Manager, or Version Manager.

Approve or reject a version

> **TIP**
>
> If version approval was enabled when the product was created, versions created in the PROD environment will require approval (approve or reject) by the Product Owner or Product Manager.
>
> - Once the version is approved, the version status will change to **Approved** and it can be used to create a batch.
> - Once the version is rejected, the version status will change to **Rejected**, and the version needs to be updated and resubmitted for approval.
>
> The Product Owner or Product Manager can disable it on the Setting tab of the product.

1. Sign in to OBIS.
2. In the left menu, click **Product**, and then select the corresponding product to navigate to the product page.
3. Select the Version tab, and select the corresponding version, and then click **Approve** or **Reject** at the bottom of the page.

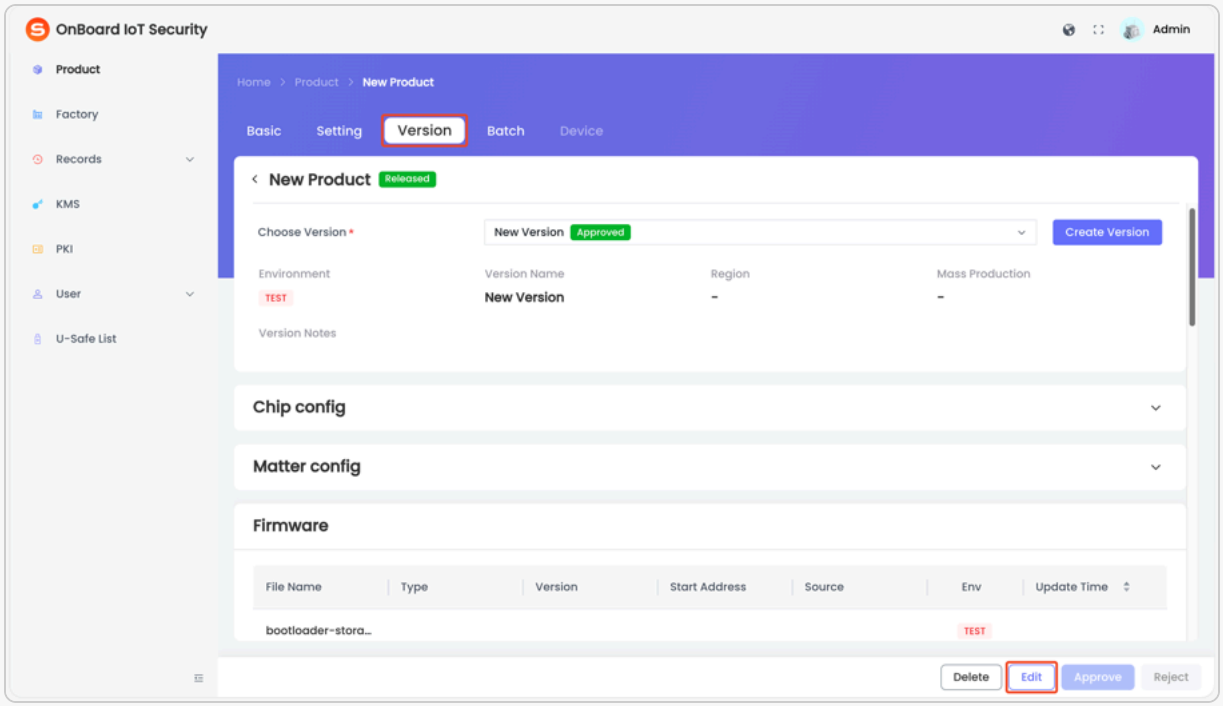


Edit a version

> **TIP**
>
> Before editing a version, please ensure that:
>
> - The version status is not **Pending for approval** (a version in **Pending for approval** can't be edited).
> - The version is not associated with any batches (once a batch has been created using this version, the version can no longer be edited).

1. Sign in to OBIS.
2. In the left menu, click **Product**, and then select the corresponding product to navigate to the product page.

3. Select the **Version** tab, and select the corresponding version, and then click **Edit** at the bottom of the page.



4. After verifying that the changes are correct, click **Save** to change the version information.

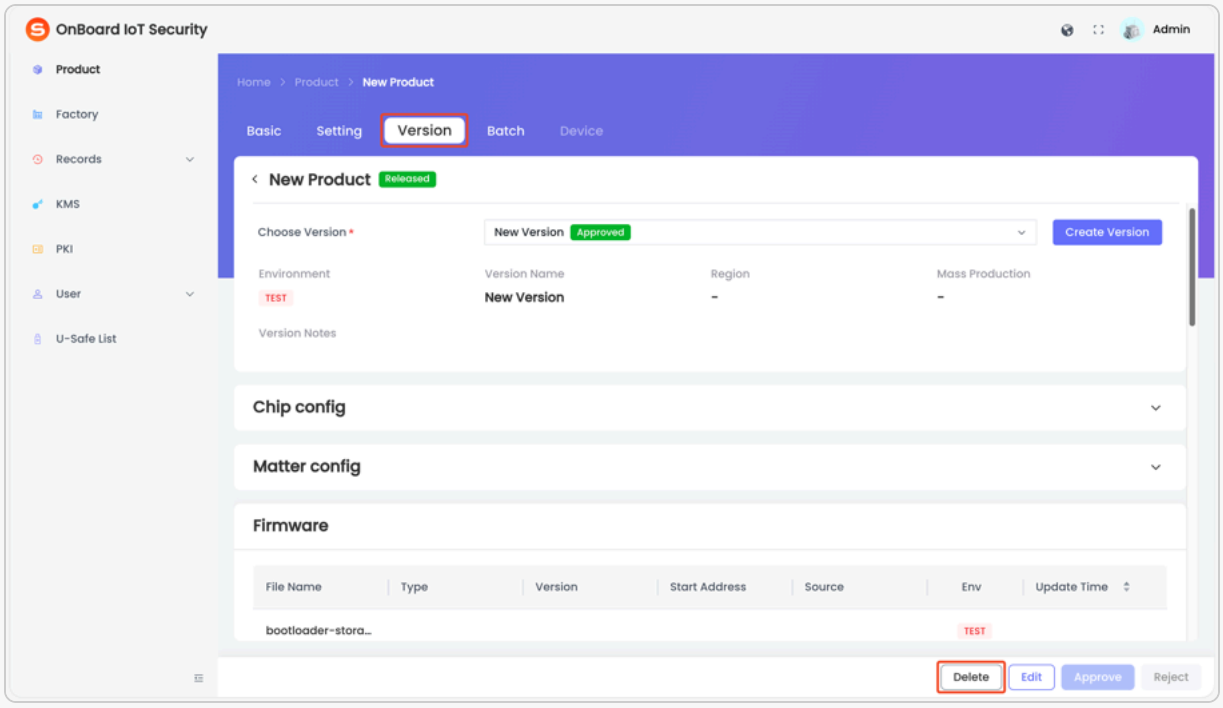
Delete a version

> **TIP**
>
> Before deleting a version, please ensure that:
>
> - The version status is not **Pending for approval** (a version in **Pending for approval** can't be deleted).
> - The version is not associated with any batches (once a batch has been created using this version, the version can no longer be deleted).

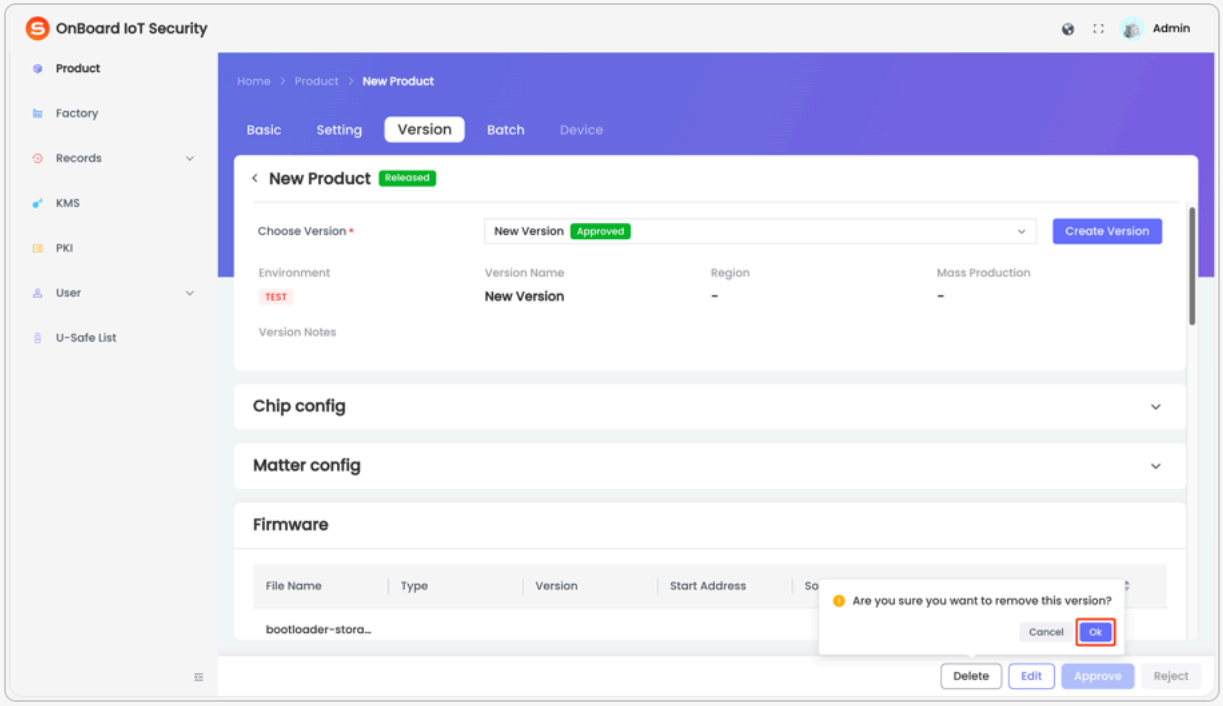
1. Sign in to OBIS.

2. In the left menu, click **Product**, and then select the corresponding product to navigate to the product page.

3. Select the **Version** tab, and select the corresponding version, and then click **Delete** at the bottom of the page.



4. In the pop-up window, click **OK** to delete the version.


OnBoard IoT Security
Admin
Product
Factory
Records
KMS
PKI
User
U-Safe List
Home > Product > New Product
Basic
Setting
Version
Batch
Device
New Product
Released
Choose Version *
New Version
Approved
Create Version
Environment
TEST
Version Name
New Version
Region
-
Mass Production
-
Version Notes
Chip config
Matter config
Firmware
File Name
Type
Version
Start Address
bootloader-stora...
Are you sure you want to remove this version?
Cancel
Ok
Delete
Edit
Approve
Reject

Version: v3.2.1

How to create a batch in OnBoard™ IoT Security

> **INFO**
>
> Currently, batches can't be modified or deleted after creation.

This article will guide you on how to create a batch in OnBoard™ IoT Security (OBIS), which can then be assigned to programming stations (local factory & cloud factory) for programming. Each version can create multiple batches, and a batch represents a production plan based on that version.

Prerequisites

Before getting started, please ensure that:

- The version has already been created and its status is **Approved** (If version approval was enabled when the product was created, versions created in the PROD environment will require approval by the Product Owner or Product Manager).
- You have product role permissions for the corresponding product: Product Owner, Product Manager, or Batch Manager.
- The factory has already been created, the HSM U-Safe has already been added (local factory), and the factory status is **Normal**.

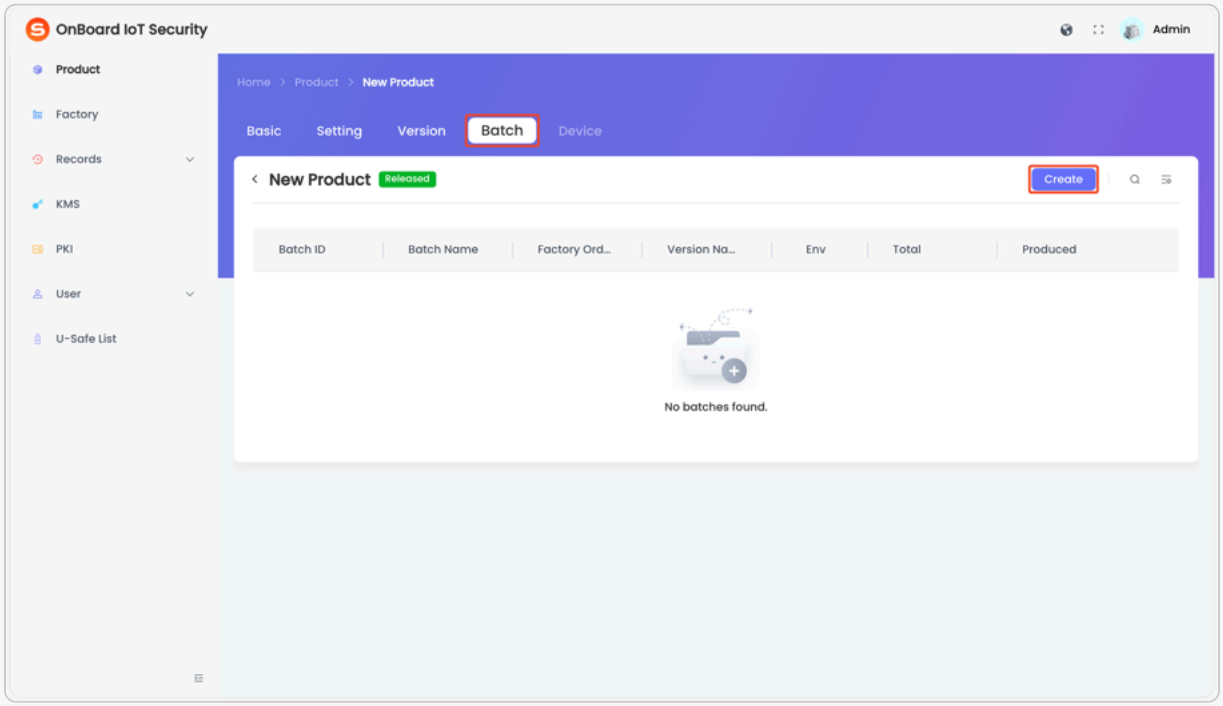
Instructions

1. Sign in to OBIS.

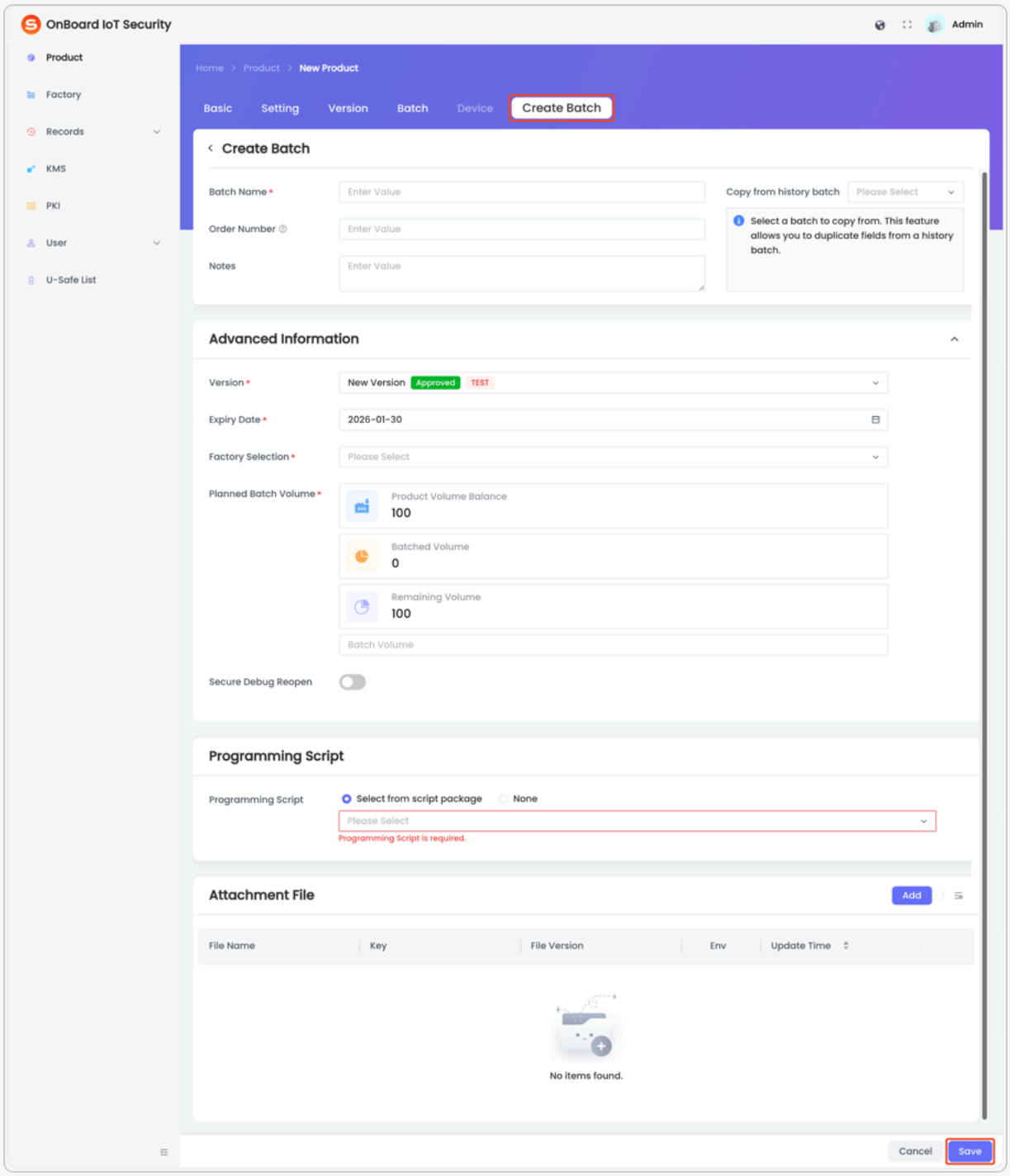
2. In the left menu, click **Product**, and then select the corresponding product to navigate to the product page.

3. On the product page, select the **Batch** tab and then click **Create**. The page will navigate to the **Create Batch** tab.

> **TIP**
>
> If the **Batch** tab appears grayed out, it is currently unavailable. Please check whether a version has been created. After a version is created, the **Batch** tab will be enabled automatically.



4. On the **Create Batch** tab, select and enter the batch-related information.



Description of batch information:

Information	Mandatory	Description
Batch Name	Yes	The batch name must be unique.
Order Number	No	Associate with a specific factory order.
Notes	No	Provide additional information related to the batch, such as special requirements and important Notes.
Copy from history batch	No	You can select a historical batch as a template for modification.
Version	Yes	Select the version with the **Approved** status.
Expiry Date	Yes	Specify the validity period or completion deadline for the batch. **Note**: Expired batches can no longer be used. In OnBoard Edge Service, the batch status will display as **Expired**. Available time depends on the Batch Expiry Duration configured when creating the product.
Factory Selection	Yes	Select a factory with the **Normal** status. **Note**: Batch data will be synchronized and updated in the factory's Batch List.

Information	Mandatory	Description
Planned Batch Volume	Yes	Set the batch volume. The **Product Volume Balance** (the Max.Volume configured when creating the product), **Batched Volume**, and **Remaining Volume** are displayed above. ◦ If the batch is associated with a version in the TEST environment, this volume will not be deducted from the total. ◦ If the batch is associated with a version in the PROD environment, this volume will be deducted (the unproduced volume of an expired batch will not be reclaimed). Additionally, the batch volume must be less than or equal to the **Product Volume Balance**.
Secure Debug Reopen	No	This feature uses the configured Secure Debug Key to unlock the chip's debug interface and erase the existing data, enabling the chip to be reprogrammed. **Note**: To enable the **Secure Debug Reopen** feature when creating a batch, the **Secure Debug** feature must be enabled when configuring the product. For more information about enabling the **Secure Debug** feature, refer to the description of Enablement Setting items.

Information	Mandatory	Description
Programming Script	Yes	Select the programming script uploaded when configuring the product as needed. If it is not required at the moment, you can select **None** and then select the programming script for a specific batch later in the Programming Station client.
Attachment File	No	Select the attachment file uploaded when configuring the product as needed.

5. After verifying that all information is correct, click **Save** at the bottom of the page to complete the batch creation.

> **TIP**
>
> Once the batch is created, it will be synchronized with the factory. If synchronization fails, navigate to the **Batch** tab of the product, click the Options ⋮ icon for the corresponding batch and then select **Resync**. For more information, refer to the description of batch status and batch synchronization status.

Version: v3.2.1

How to check batches in OnBoard™ IoT Security

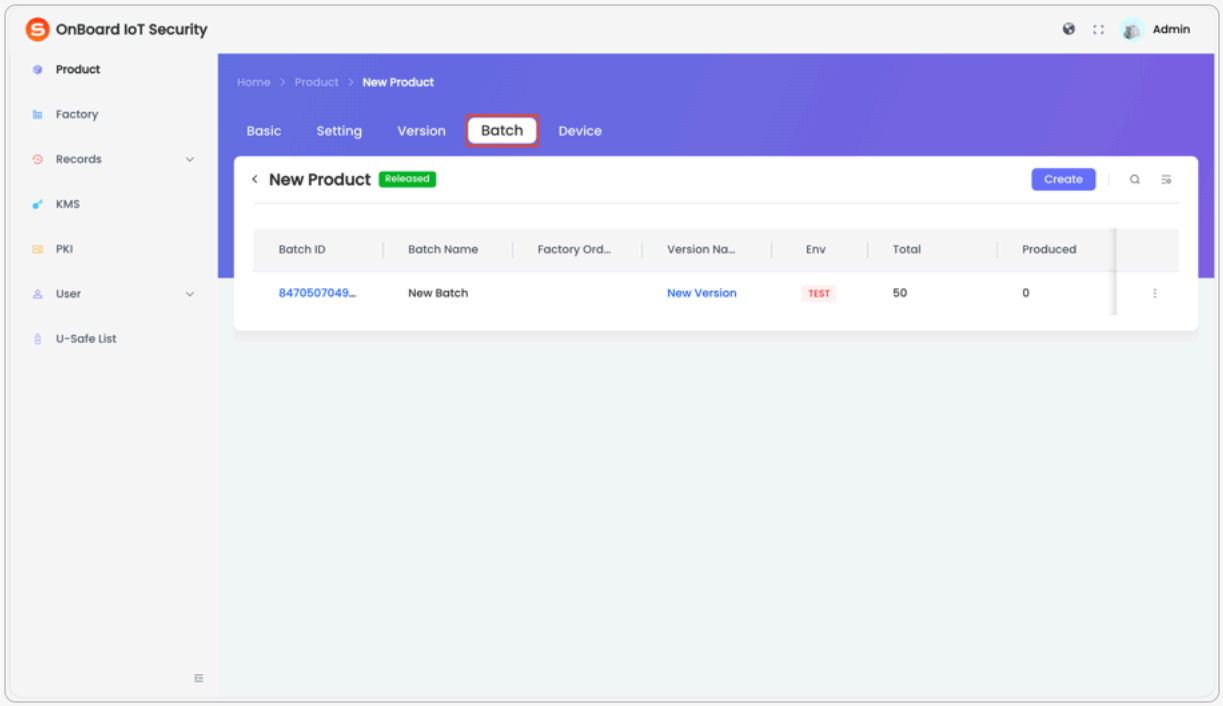
This article will guide you on how to check batch information and configuration in OnBoard™ IoT Security (OBIS).

Prerequisites

Before getting started, please ensure that you have product role permissions for the corresponding product.

Instructions

1. Sign in to OBIS.
2. In the left menu, click **Product**, and then select the corresponding product to navigate to the product page.
3. On the product page, select the **Batch** tab, and then select the corresponding batch to navigate to the batch page.



Description of **Batch Status**:

Status	Description
Generating	The status when the batch is being generated after creation.
Generate Failed	The status after the batch generation fails. You can try to regenerate it: On the **Batch** tab of the product, click the Options ⋮ icon for the corresponding batch and then select **Resync**.
Generated	The status after the batch is successfully generated.
Running	The status after the batch is successfully synchronized with the factory.

Status	Description
Stopped	The status after the batch is stopped.
Finished	The status after the batch programming is finished.
Expired	The batch has expired before the programming is finished.

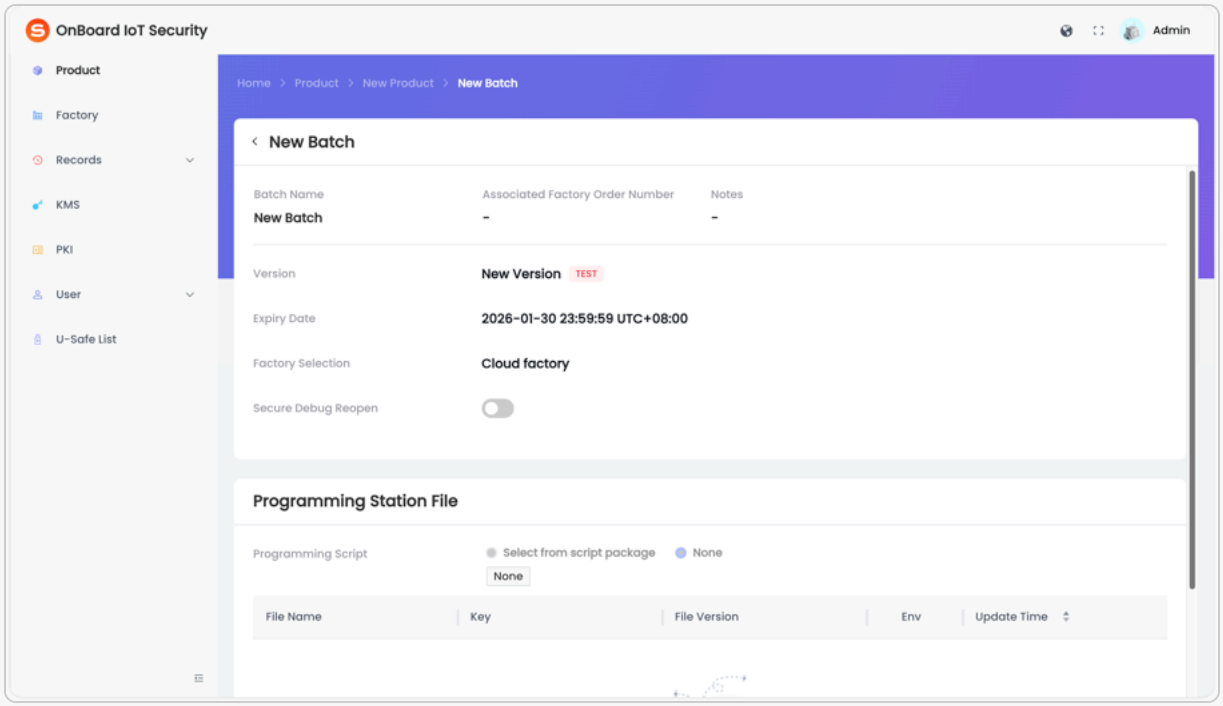
Description of **Batch Sync** (batch synchronization status):

Status	Description
Waiting	The status before the batch is successfully generated.
Syncing	The status when the batch is successfully generated and starts synchronizing with the factory.
Sync Failed	The status after synchronization fails. You can try resynchronizing: On the **Batch** tab of the product, click the Options ⋮ icon for the corresponding batch and then select **Resync**.
Synced	The status after the batch is successfully synchronized with the factory.
Deleted	The status after the batch is deleted.

Description of **Records** (programming data synchronization status):

Status	Description
No Records	The state before the batch starts programming.
Uploading	The status when the batch has started programming but is not yet completed, and the programming data is being uploaded.
Uploaded	The status after the batch programming is finished and all programming data has been fully uploaded.

4. On the batch page, you can check the batch-related information (such as **Batch Name**, **Version**, and **Expiry Date**).



Version: v3.2.1

How to check programmed devices in OnBoard™ IoT Security

This article will guide you on how to check the parameters and programming status for programmed devices in OnBoard™ IoT Security (OBIS).

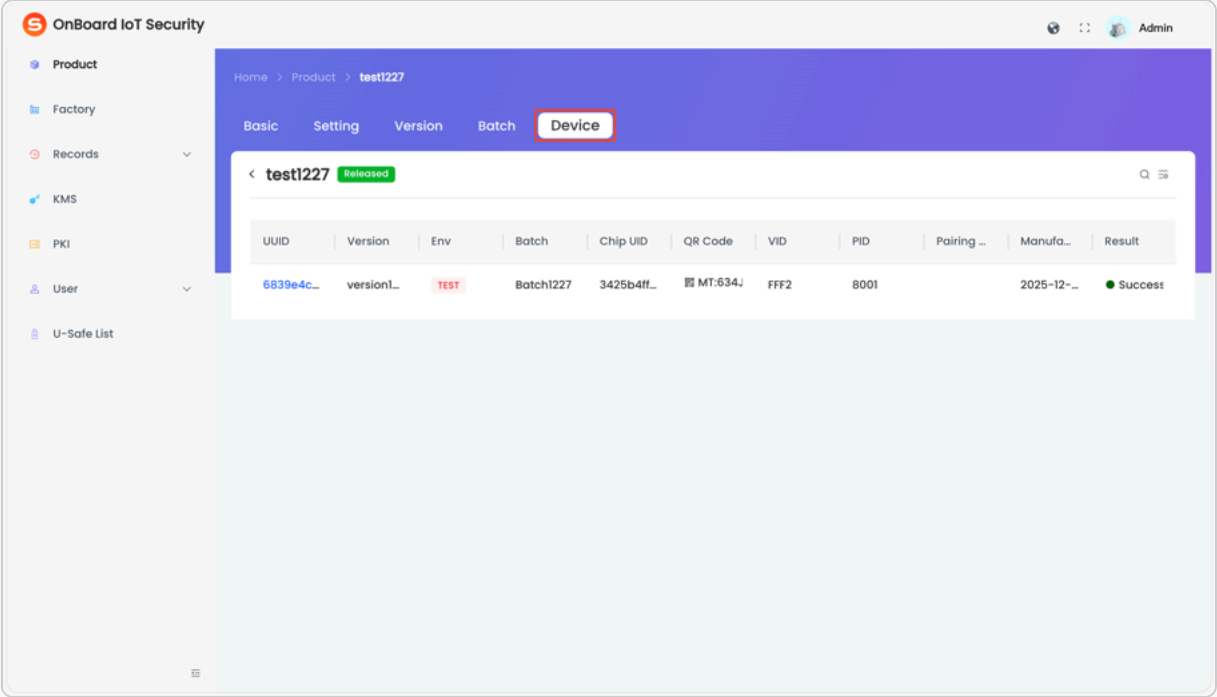
Prerequisites

Before getting started, please ensure that:

- The device has already been programmed.
- You have product role permissions for the corresponding product.

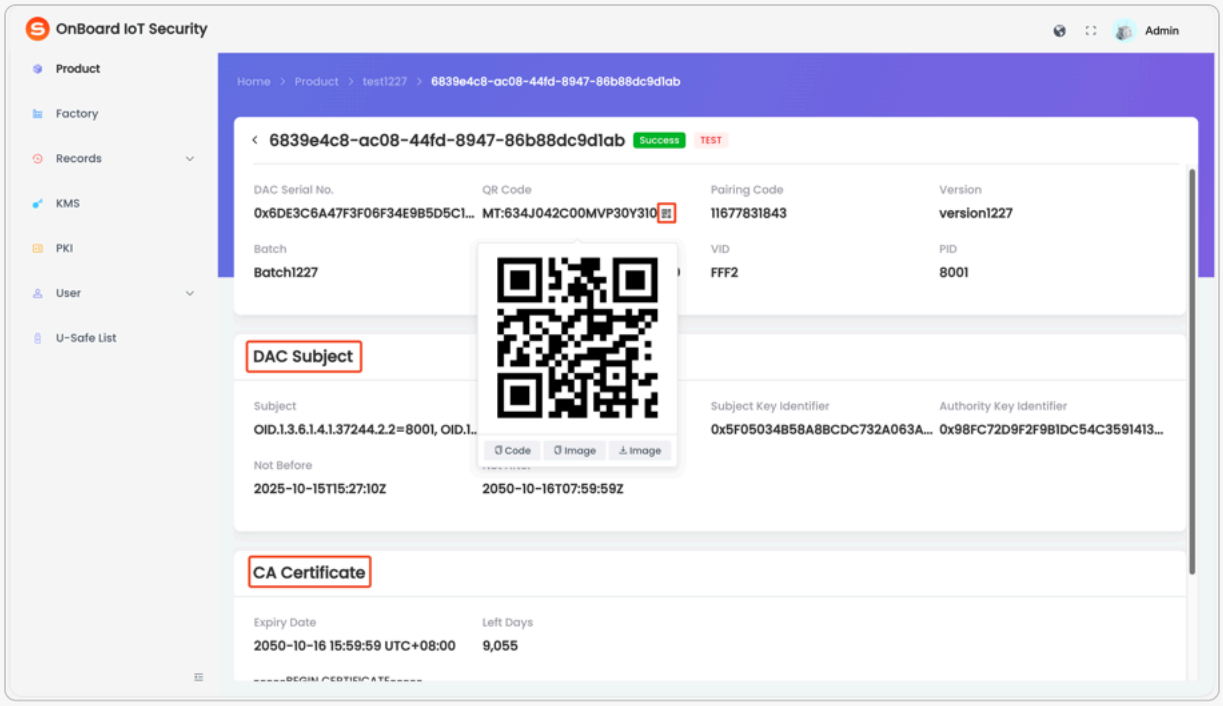
Instructions

1. Sign in to OBIS.
2. In the left menu, click **Product**, and then select the corresponding product to navigate to the product page.
3. On the product page, select the **Device** tab to check device programming information (such as **UUID**, **Version**, **Batch**, and **Result**).



4. On the **Device** tab, select the corresponding UUID to navigate to the device page, where you can check the device-related information (such as **DAC Serial No.**, **QR Code**, and **Pairing Code**).

> **TIP**
>
> Hover the mouse over the QR code icon to display the device QR code, which is used for secure and fast commissioning.
>
> - The **DAC Subject** area displays the information of the DAC subject (such as **Not Before** and **Not After**)
> - The **CA Certificate** area displays the encoding of the CA certificate. By parsing the encoding of the CA certificate, key certificate attributes such as the issuer information, validity period, and public key can be extracted.


OnBoard IoT Security
Admin
Product
Factory
Records
KMS
PKI
User
U-Safe List
Home > Product > test1227 > 6839e4c8-ac08-44fd-8947-86b88dc9d1ab
6839e4c8-ac08-44fd-8947-86b88dc9d1ab
Success
TEST
DAC Serial No.
0x6DE3C6A47F3F06F34E9B5D5C1...
QR Code
MT:634J042C00MVP30Y310
Pairing Code
11677831843
Version
version1227
Batch
Batch1227
VID
FFF2
PID
8001
Code
Image
Image
DAC Subject
Subject
OID.1.3.6.1.4.1.37244.2.2=8001, OID.1..
Subject Key Identifier
0x5F05034B58A8BCDC732A063A...
Authority Key Identifier
0x98FC72D9F2F9B1DC54C3591413...
Not Before
2025-10-15T15:27:10Z
2050-10-16T07:59:59Z
CA Certificate
Expiry Date
2050-10-16 15:59:59 UTC+08:00
Left Days
9,055

Version: v3.2.1

How to create a factory in OnBoard™ IoT Security

> **INFO**
>
> Currently, factories can't be deleted after creation.

This article will guide you on how to create a factory in OnBoard™ IoT Security (OBIS) to configure its basic information, preparing for configuring factory managers, HSM U-Safe, Identity U-Safe, and programming stations.

Factory type

OBIS supports the creation of local factories and cloud factories:

- Local factory: Requires an OnBoard Edge Service deployment on the factory side, typically hosted on a local factory server, to provide on-site production management capabilities. Each local factory is associated with a dedicated OnBoard Edge Service. Once the batch information is synchronized to OnBoard Edge Service, the factory can execute offline programming without requiring a real-time network connection during the process.
- Cloud factory: Production management capabilities are provided through the cloud. A real-time network connection is required during the programming process to ensure batch information synchronization, production data upload, and status updates.

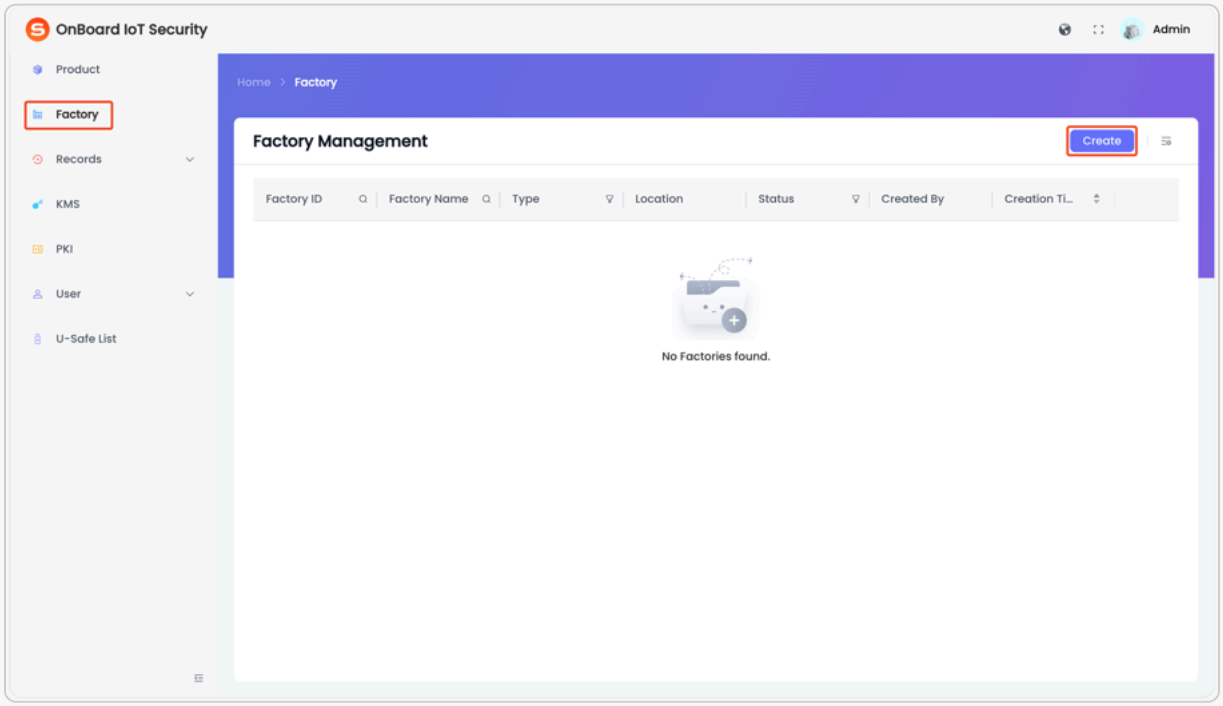
> **TIP**
>
> You can contact Snowball for assistance in deploying OnBoard Edge Service.

Prerequisites

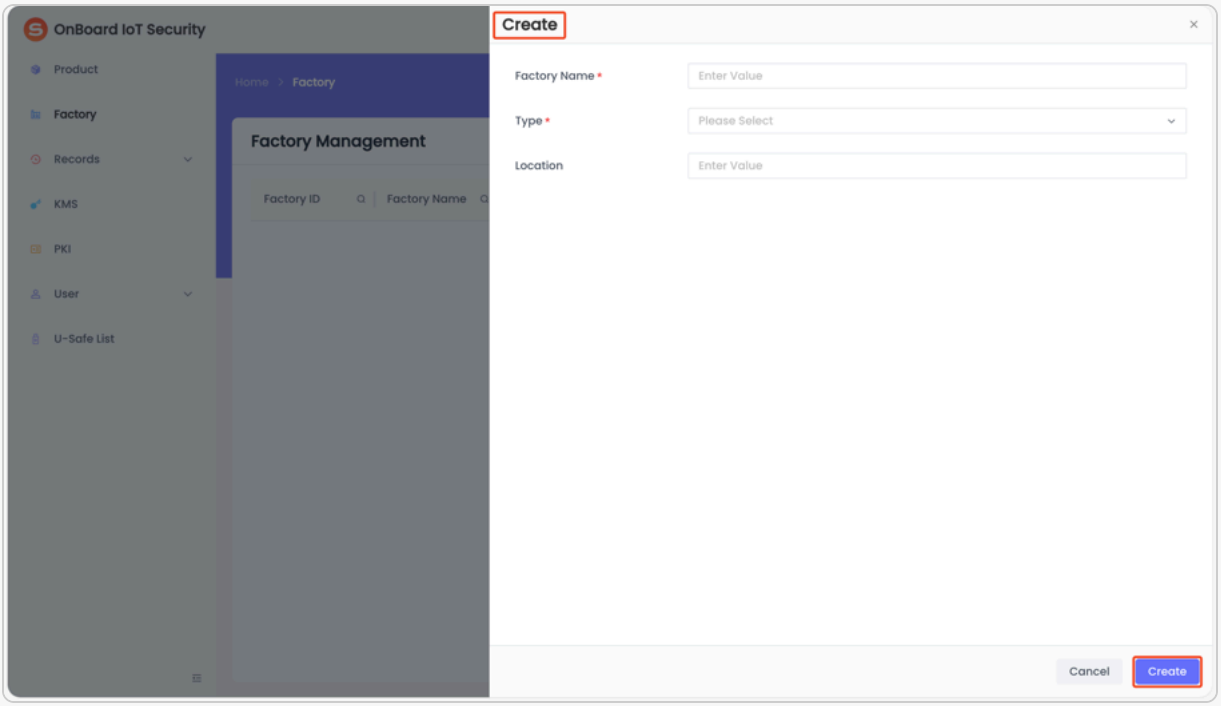
Before getting started, please ensure that you have OBIS role permissions as a System Manager or Factory Manager.

Instructions

1. Sign in to OBIS.
2. In the left menu, click **Factory**, and then click **Create**.



3. On the **Create** panel, select and enter the required factory basic information.



Description of factory basic information:

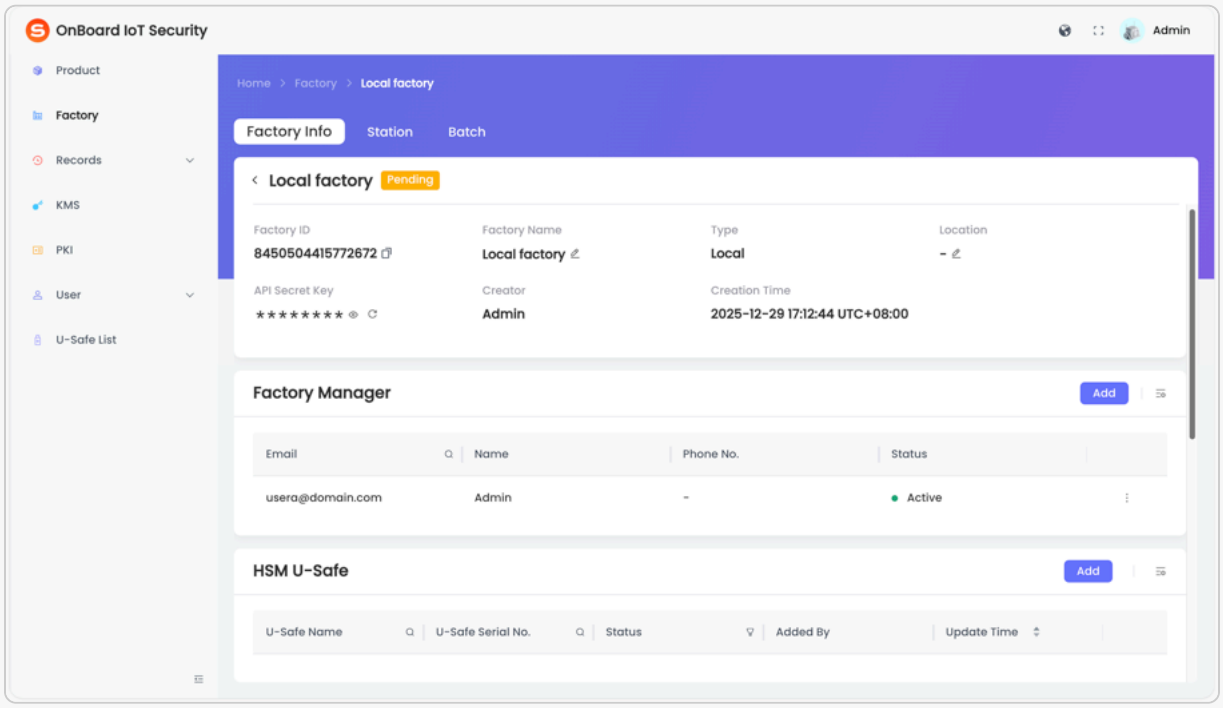
Information	Mandatory	Description
Factory Name	Yes	The factory name must be unique.
Type	Yes	Two types can be selected: **Local** and **Cloud**.
Location	No	It is recommended to provide the factory address.

4. Click **Create** to complete the factory creation. After creation, select the corresponding factory ID on the **Factory Management** page, and then you can check the factory-related information on the factory page.

TIP

After the factory is created, you can only modify the **Factory Name** and **Location** on the factory page.

- The initial status of a local factory is **Pending**. Once an HSM U-Safe is added to the local factory, the status will change from **Pending** to **Normal**. For more information, refer to the description of factory status.
- The initial status of a cloud factory is **Normal**.



Version: v3.2.1

How to configure and manage factories in OnBoard™ IoT Security

This article will guide you through the following operations in OnBoard™ IoT Security (OBIS):

- Add a Factory Manager
- Reset the Factory Manager password
- Delete a Factory Manager
- Add an HSM U-Safe
- Unbind, lock, or unlock an HSM U-Safe
- Add an Identity U-Safe
- Unbind, lock, or unlock an Identity U-Safe
- Lock or unlock a factory

Prerequisites

Before getting started, please ensure that:

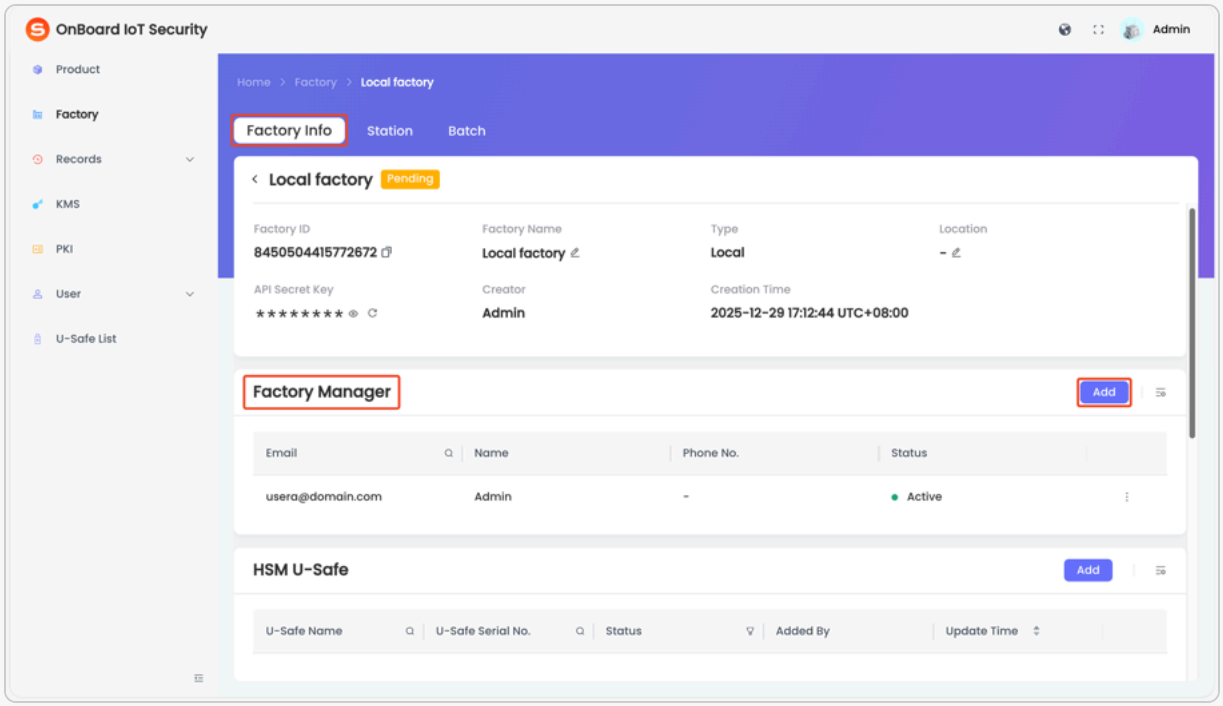
- You have one of the following role permissions:
  - OBIS role: System Manager or Factory Manager
  - Factory role: Factory Manager corresponding to the factory
- You have obtained the HSM U-Safe and Identity U-Safe required for identity

authentication from Snowball.

Add a Factory Manager

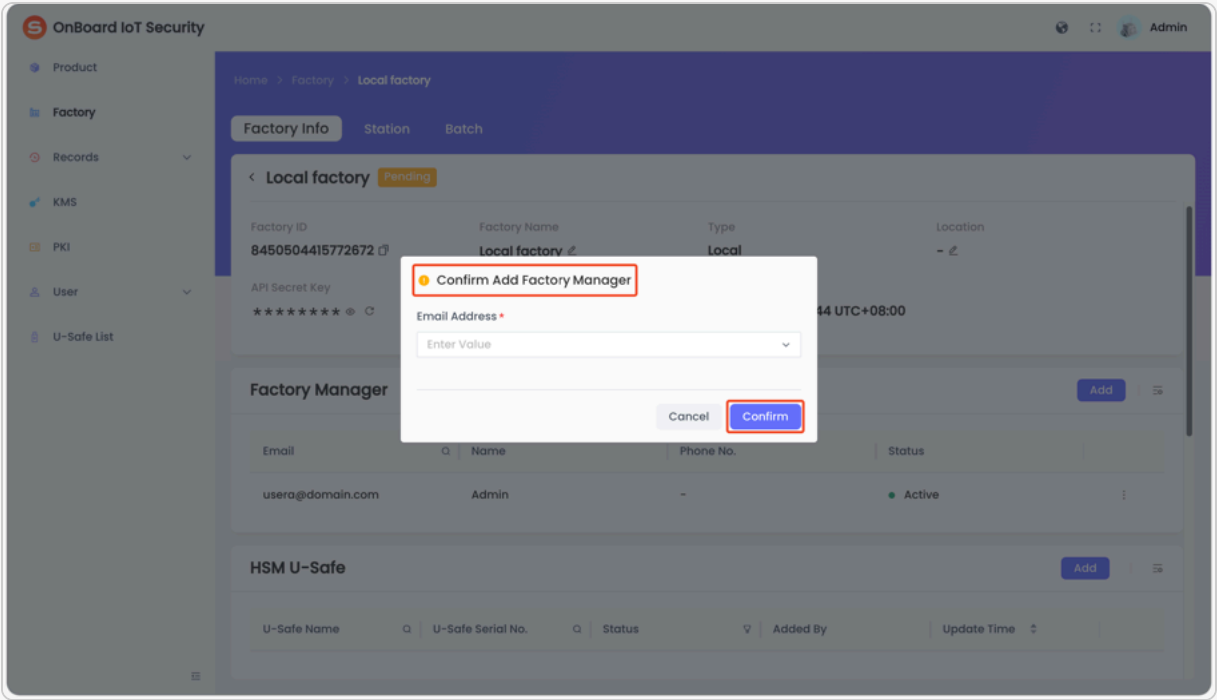
> **TIP**
>
> Factory managers can manage and operate factory-related business, and sign in to OnBoard Edge Service using either the "Email + Password" or "Email + Verification code" method.
>
> **Notes**:
>
> - When the Internet is disconnected, OnBoard Edge Service only supports identity verification using the "Email + Password" method.
> - After a Factory Manager is added, the system automatically generates an initial password and sends it to the corresponding email address. Currently, this randomly generated password can be reset, but custom modification is not yet supported.

1. Sign in to OBIS.
2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.
3. On the factory page, locate the **Factory Manager** area and then click **Add**.



4. In the **Confirm Add Factory Manager** window, select and enter the email address of the new Factory Manager.

> **TIP**
>
> The Factory Manager being added must be an **Active** user within the same tenant.



5. Click **Confirm** to complete adding a Factory Manager.

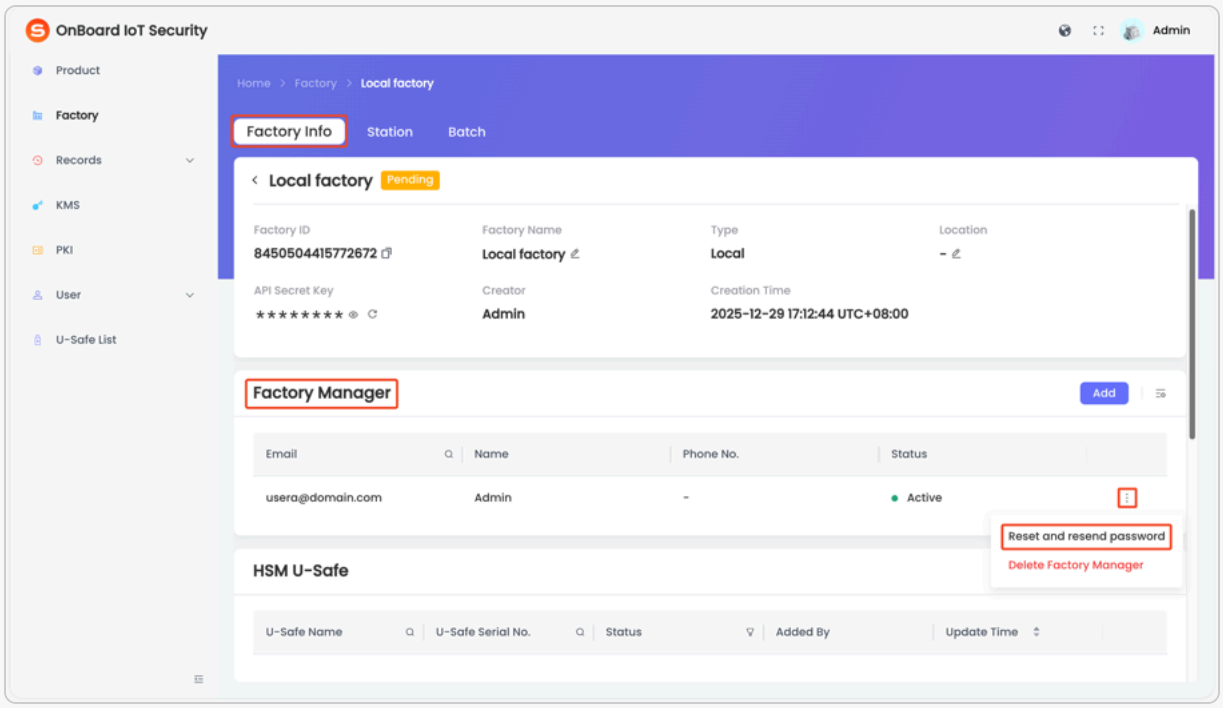
Reset the Factory Manager password

> **TIP**
>
> After the password is reset, the system automatically generates a random password and sends it to the corresponding email address.

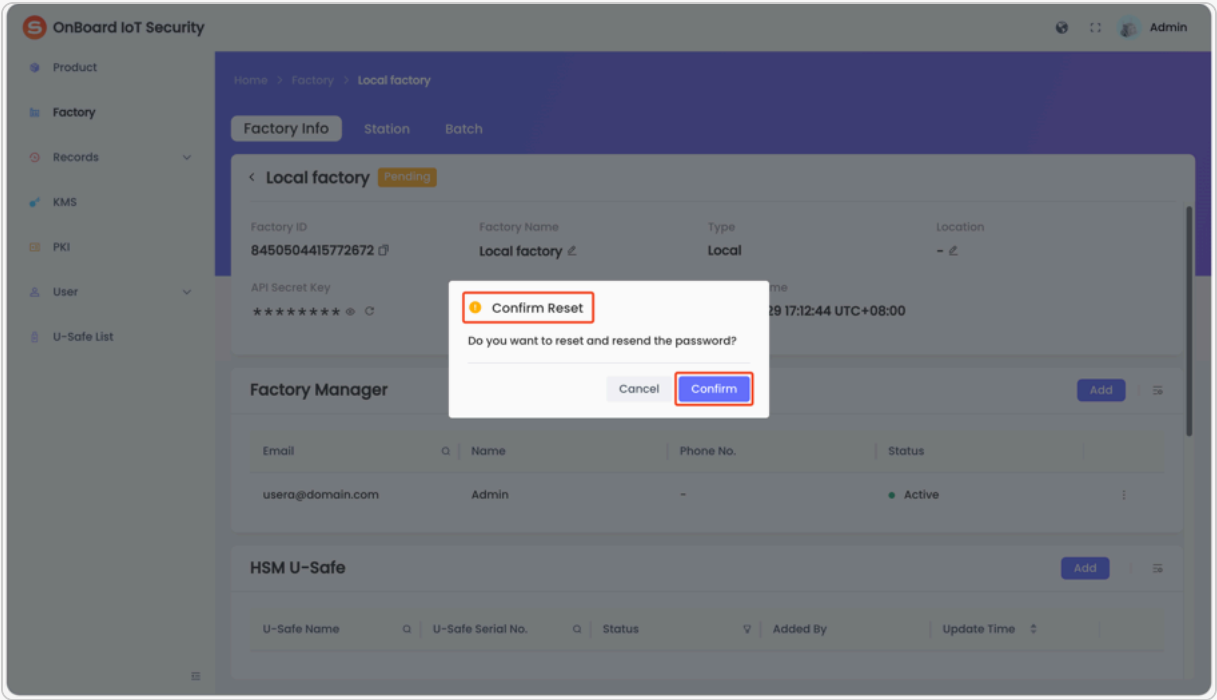
1. Sign in to OBIS.

2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.

3. On the factory page, locate the **Factory Manager** area, and then click the Options ⋮ icon for the corresponding Factory Manager and select **Reset and resend password**.



4. In the **Confirm Reset** window, click **Confirm** to reset the password.

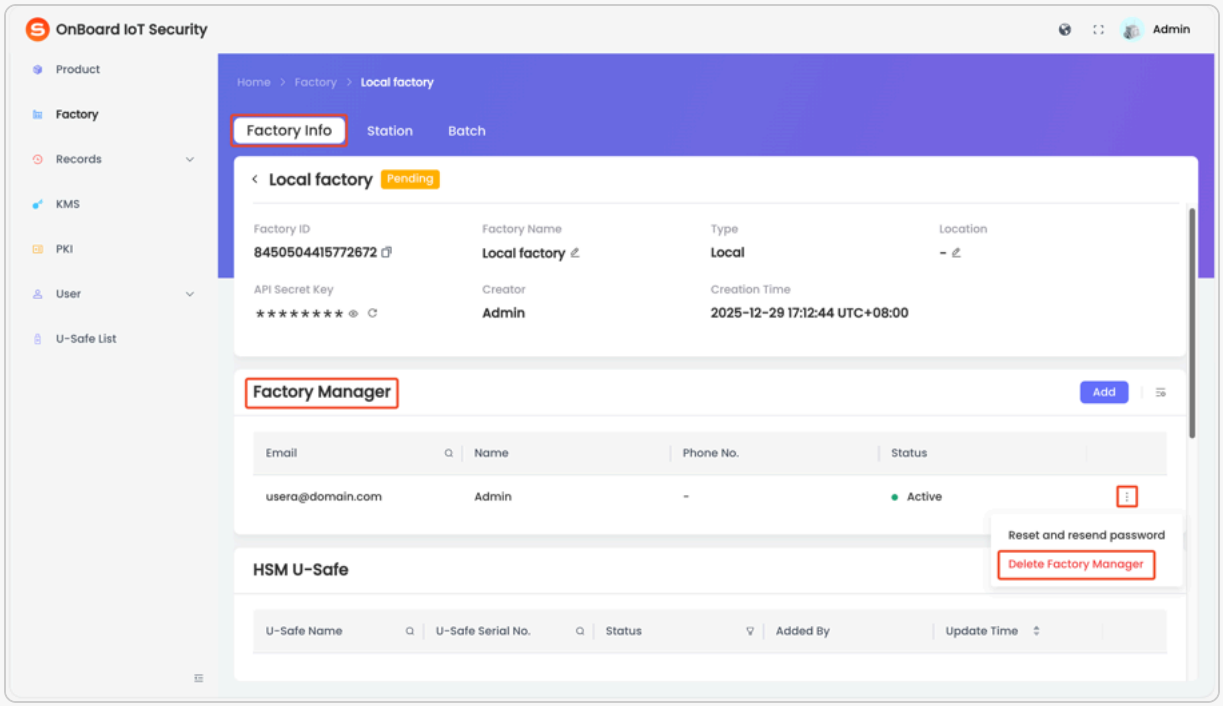


Delete a Factory Manager

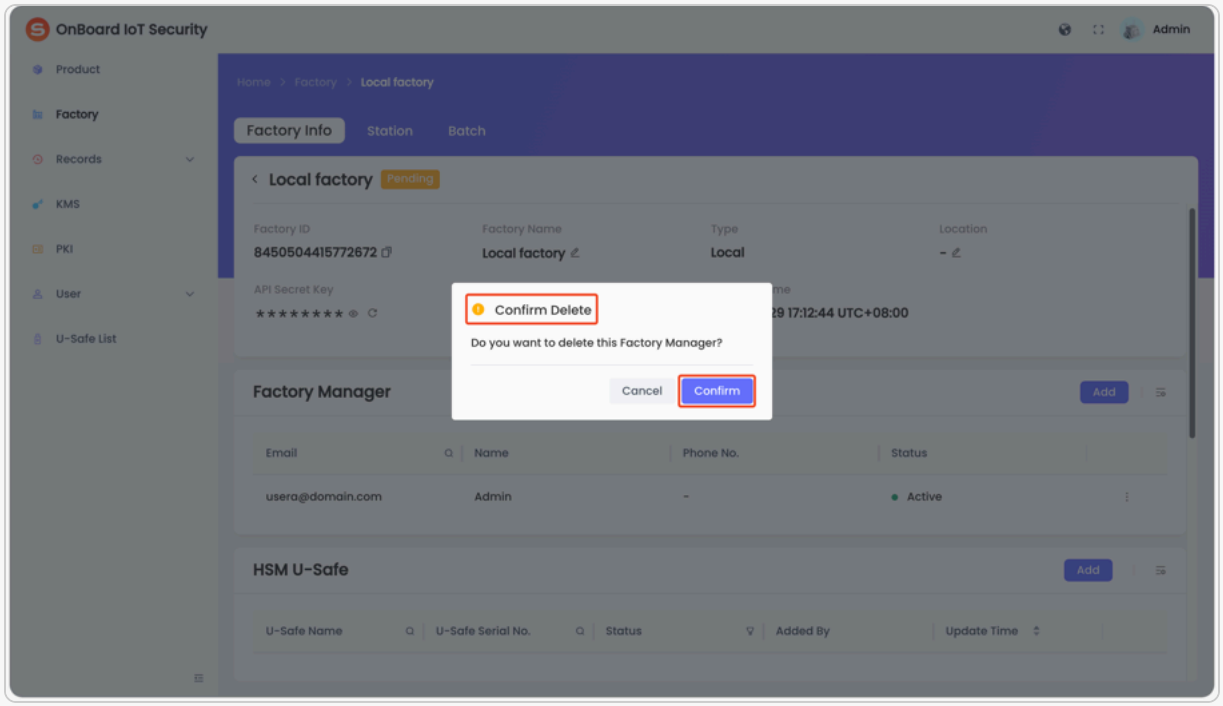
1. Sign in to OBIS.

2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.

3. On the factory page, locate the **Factory Manager** area, and then click the Options ⋮ icon for the corresponding Factory Manager and select **Delete Factory Manager**.



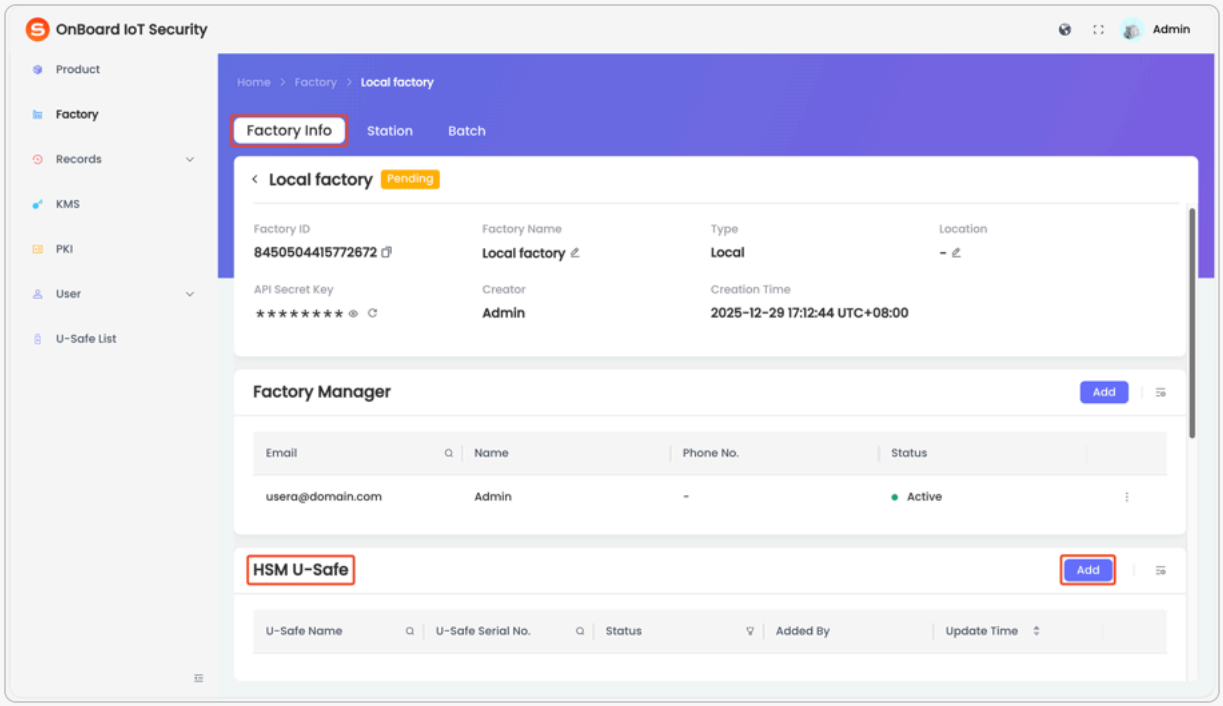
4. In the **Confirm Delete** window, click **Confirm** to delete the Factory Manager.



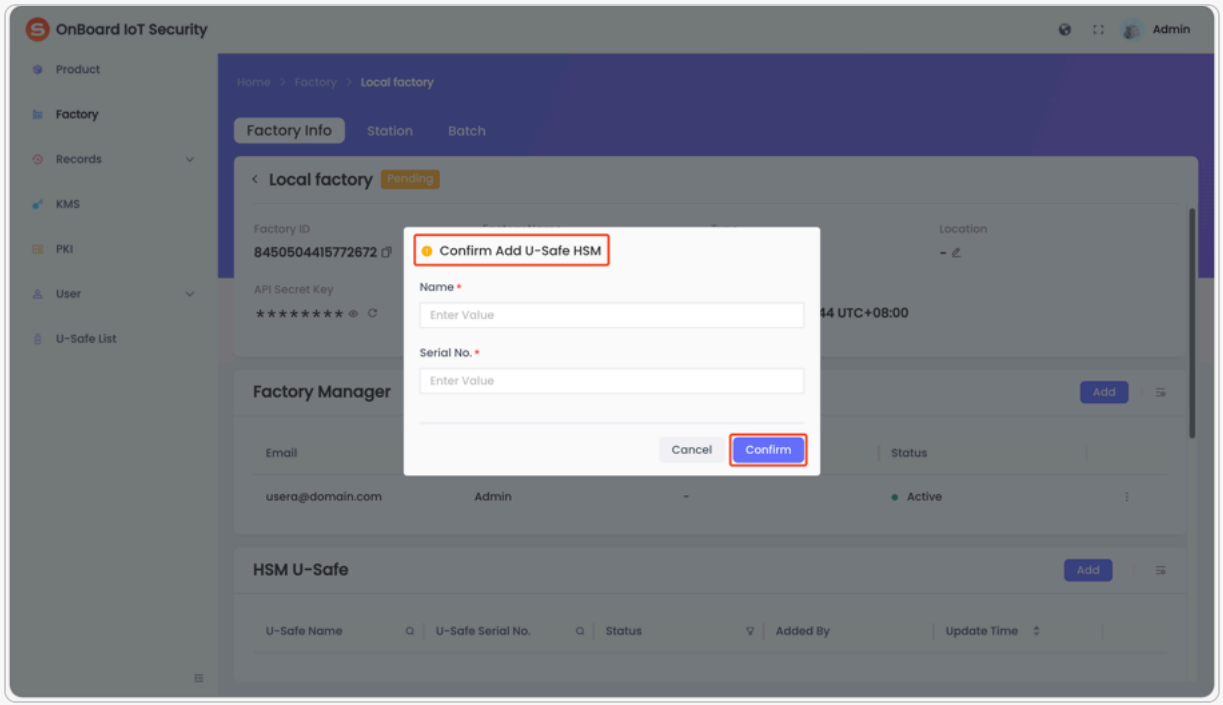
Add an HSM U-Safe

> **TIP**
>
> The purpose of adding an HSM U-Safe is to bind it to the corresponding factory and use it for signing in to OnBoard Edge Service. Meanwhile, secure key storage and device programming support are achieved through the HSM U-Safe management.
>
> Once an HSM U-Safe is added to the factory, the factory status will change from **Pending** to **Normal**.

1. Sign in to OBIS.
2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.
3. On the factory page, locate the **HSM U-Safe** area and then click **Add**.



4. In the **Confirm Add U-Safe HSM** window, enter the **Name** (customizable) and **Serial No.** (which can be found on the device) of the HSM U-Safe you want to bind.

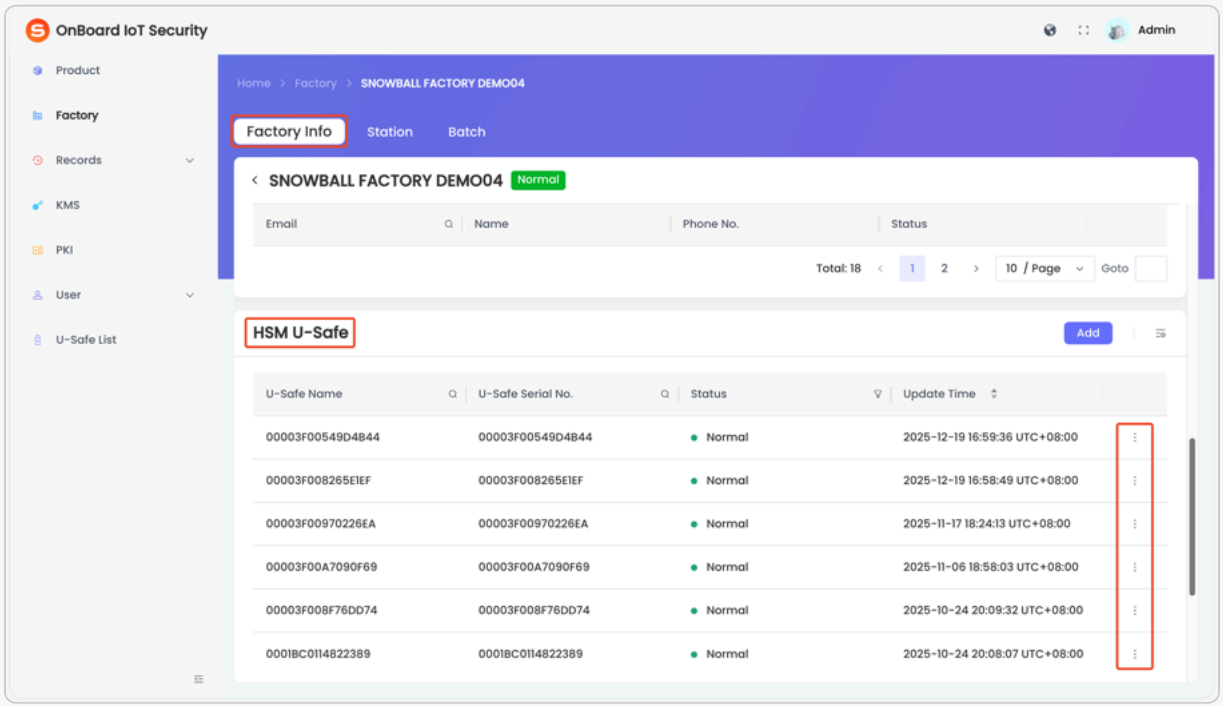


5. Click **Confirm** to bind the HSM U-Safe.

Unbind, lock, or unlock an HSM U-Safe

> **TIP**
>
> Lock the HSM U-Safe to prevent its use and unbinding. Once locked, its status will change to **Locked**.

1. Sign in to OBIS.
2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.
3. On the factory page, locate the **HSM U-Safe** area, and then click the Options icon for the corresponding HSM U-Safe and select **Unbind**, **Lock**, or **Unlock**.

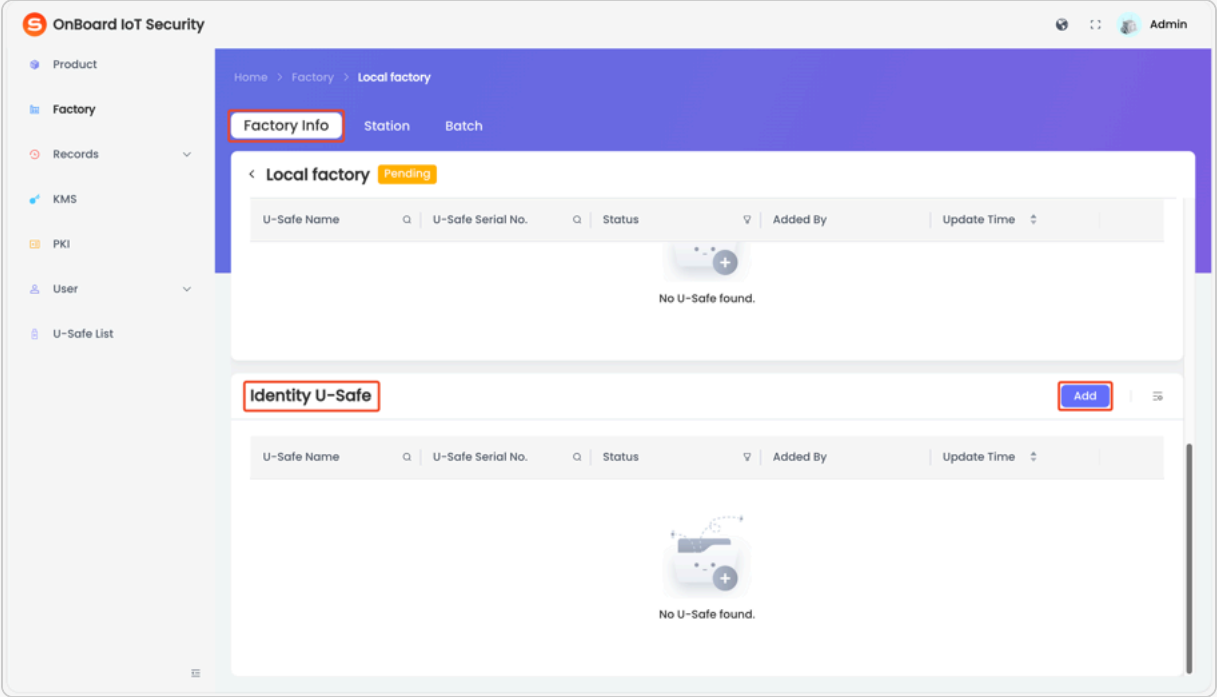


4. In the pop-up window, click **Confirm** to unbind, lock, or unlock the HSM U-Safe.

Add an Identity U-Safe

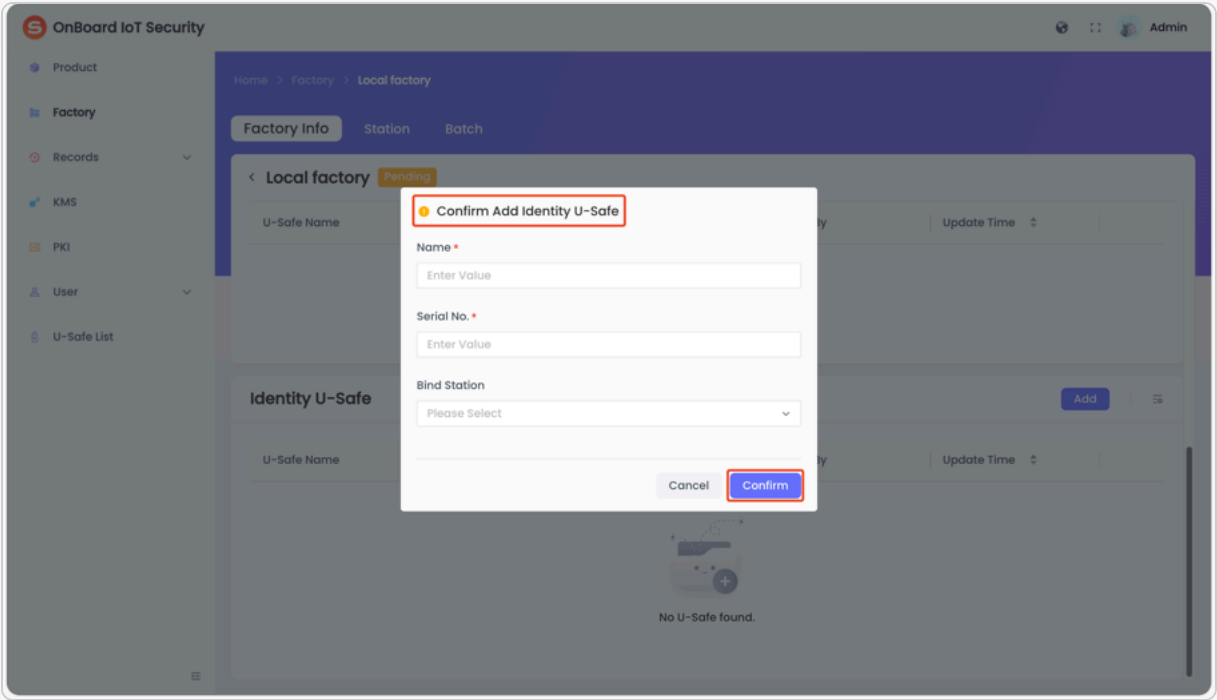
> **TIP**
>
> The purpose of adding an Identity U-Safe is to bind it to the corresponding factory. After binding it to a programming station in OBIS or OnBoard Edge Service, it can be used to sign in to the Programming Station client. Meanwhile, login authentication for the programming station is achieved through the Identity U-Safe management.

1. Sign in to OBIS.
2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.
3. On the factory page, locate the **Identity U-Safe** area and then click **Add**.



4. In the **Confirm Identity U-Safe** window, enter the **Name** (customizable) and **Serial No.** (which can be found on the device) of the Identity U-Safe you want to bind.

> **TIP**
>
> - If a programming station has already been added, you can select and bind the corresponding programming station.
> - If a programming station hasn't been added yet, you can bind it when adding a programming station.



5. Click **Confirm** to bind the Identity U-Safe.

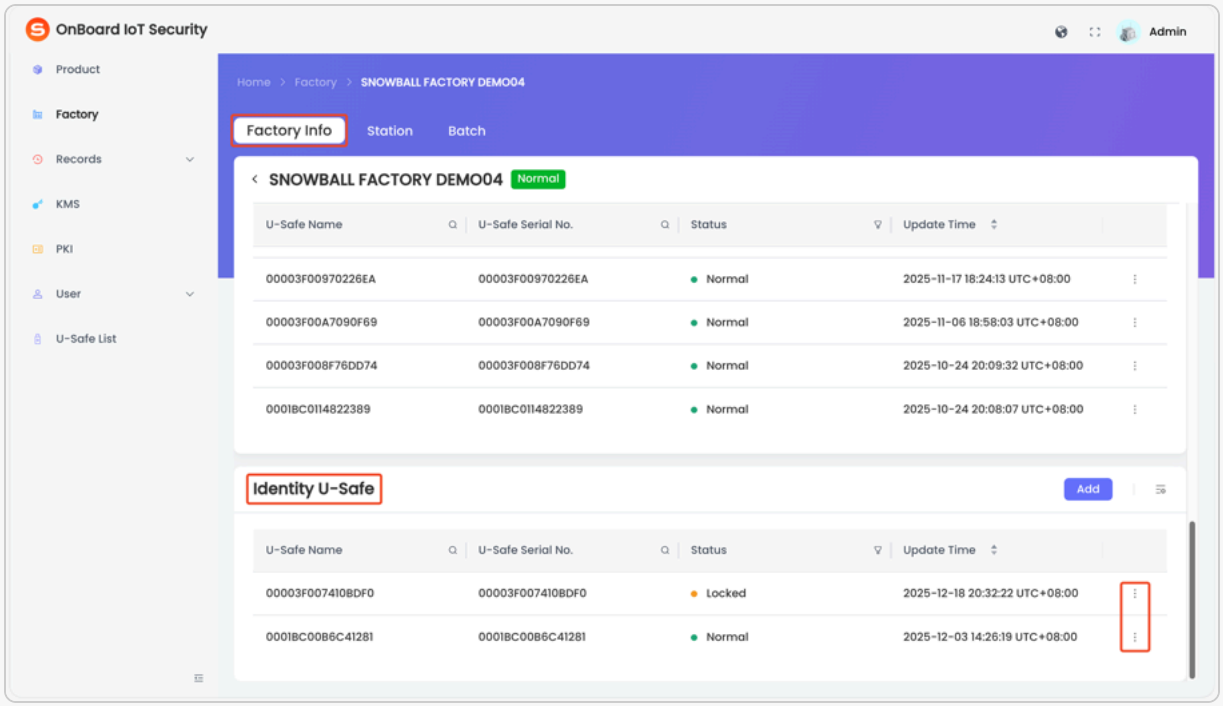
Unbind, lock, or unlock an Identity U-Safe

> **TIP**
>
> Lock the Identity U-Safe to prevent its use and unbinding. Once locked, its status will change to **Locked**.

1. Sign in to OBIS.

2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.

3. On the factory page, locate the **Identity U-Safe** area, and then click the Options ⋮ icon for the corresponding Identity U-Safe and select **Unbind**, **Lock**, or **Unlock**.

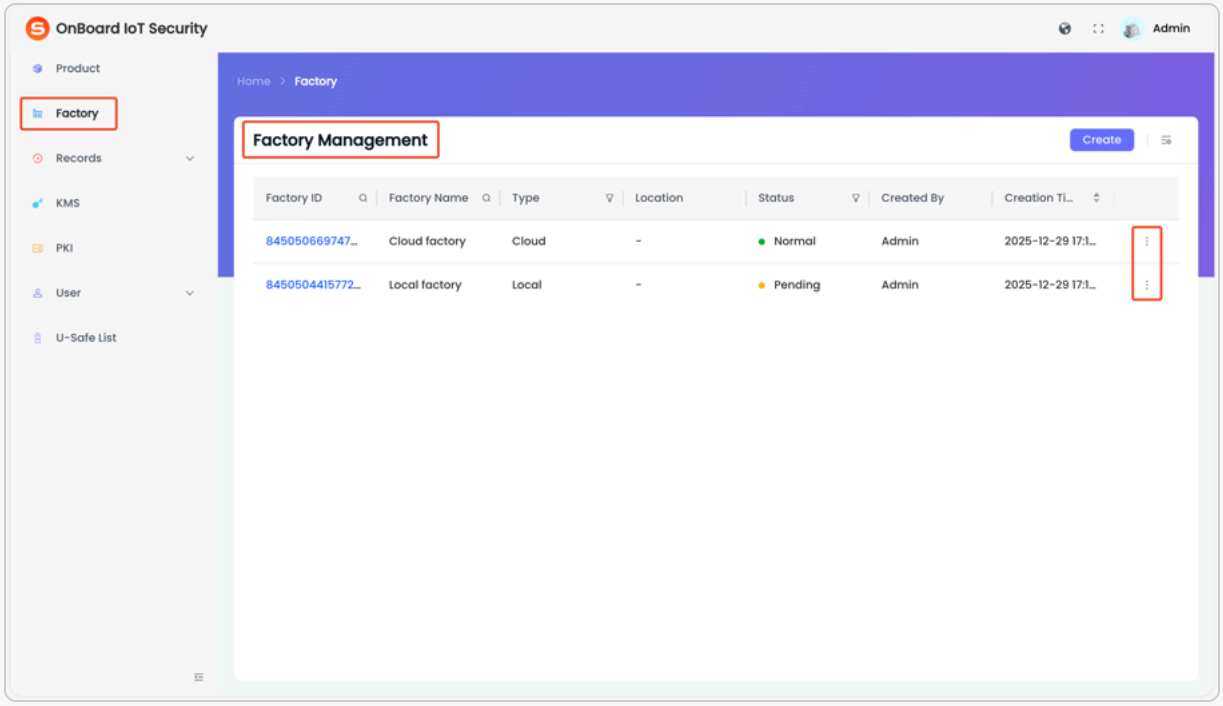


4. In the pop-up window, click **Confirm** to unbind, lock, or unlock the Identity U-Safe.

Lock or unlock a factory

> **TIP**
>
> Lock the factory to prevent its use. Once locked, its status will change to **Locked**.

1. Sign in to OBIS.
2. In the left menu, click **Factory**, and then click the Options ⋮ icon for the corresponding factory and select **Lock** or **Unlock** to lock or unlock the factory.



3. In the pop-up window, click **OK** to lock or unlock the factory.

Version: v3.2.1

How to configure and manage programming stations in OnBoard™ IoT Security

> **TIP**
>
> You can configure and manage the programming station in OnBoard™ IoT Security (OBIS) or OnBoard Edge Service as needed.

This article will guide you through the following operations in OBIS:

- Create a programming station
- Bind an Identity U-Safe to a programming station
- Unbind an Identity U-Safe from a programming station
- Delete a programming station
- Check the programming station password

Prerequisites

Before getting started, please ensure that:

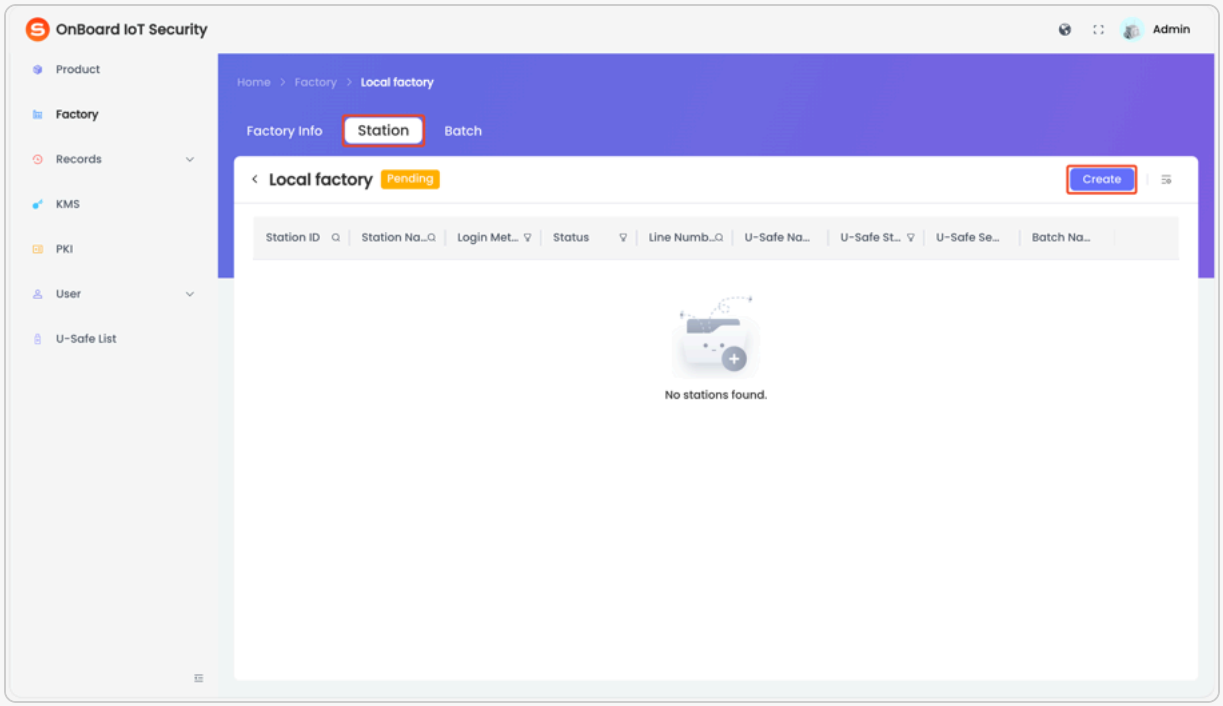
- The factory has already been configured.
- You have one of the following role permissions:

    - OBIS role: System Manager or Factory Manager
    - Factory role: Factory Manager corresponding to the factory
- You have obtained the Identity U-Safe for identity authentication from Snowball.

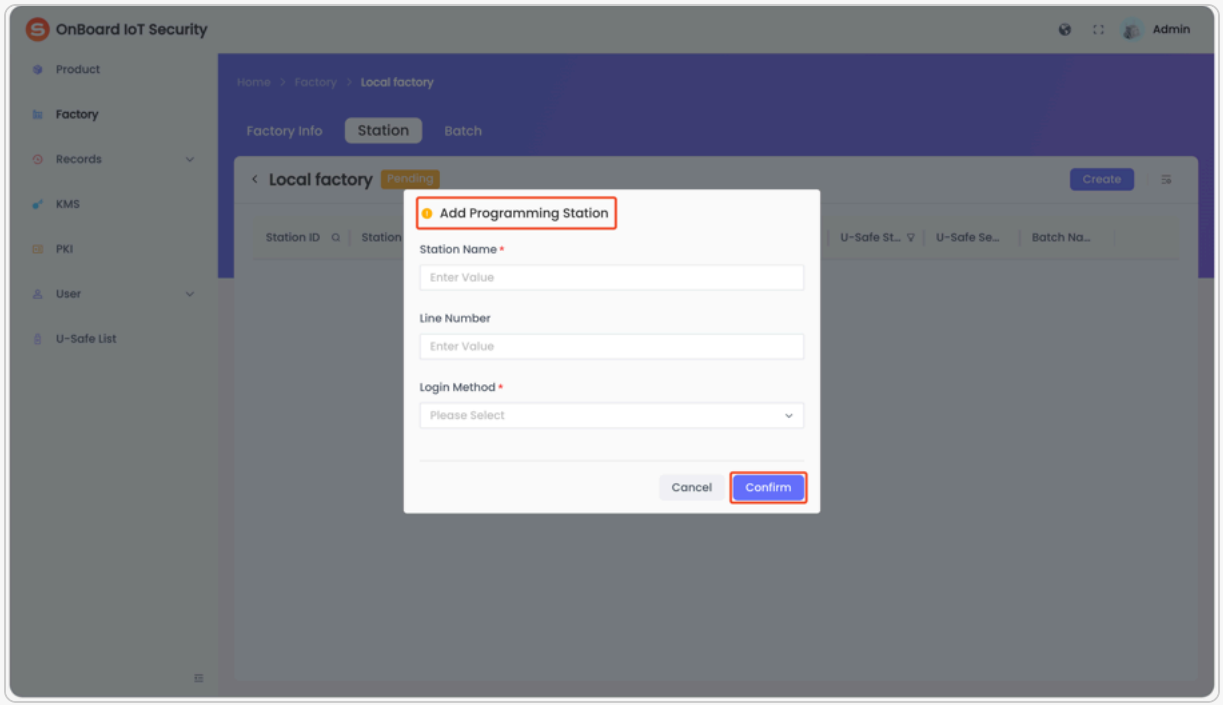
Create a programming station

> **TIP**
>
> After creating a programming station for the factory, you can assign specific programming stations to batches in OnBoard Edge Service as needed. Before programming a device, it is necessary to query and retrieve the corresponding batch information and programming data based on the programming station ID.
>
> Two programming methods are supported:
>
> - Using the Programming Station client
> - Using the OBIS Programming CLI

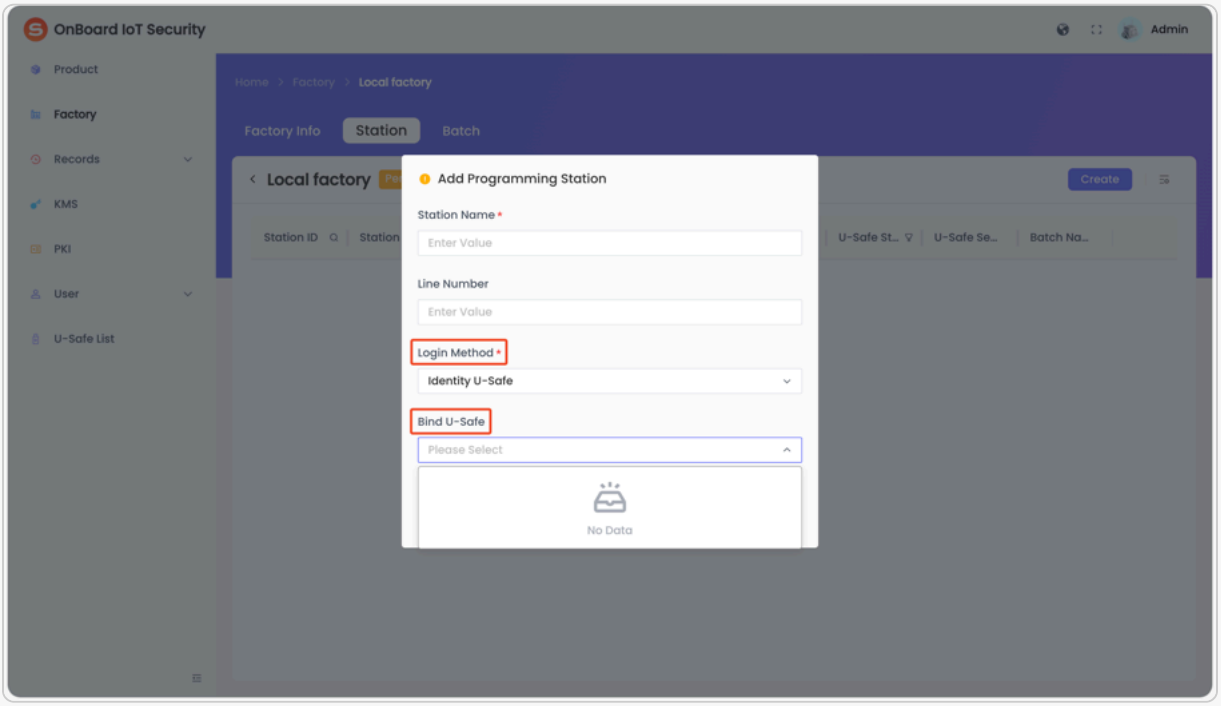
1. Sign in to OBIS.
2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.
3. On the factory page, select the **Station** tab and then click **Create**.



4. In the **Add Programming Station** window, enter the **Station Name** and **Line Number**, and then select the **Login Method**. The methods include: **No Password**, **Password**, or **Identity U-Safe**.



> **TIP**
>
> If the login method is **Identity U-Safe**, you can select an already added Identity U-Safe from the **Bind U-Safe** drop-down list to bind it, or you can bind it after adding the programming station.



5. Click **Confirm** to create the programming station.

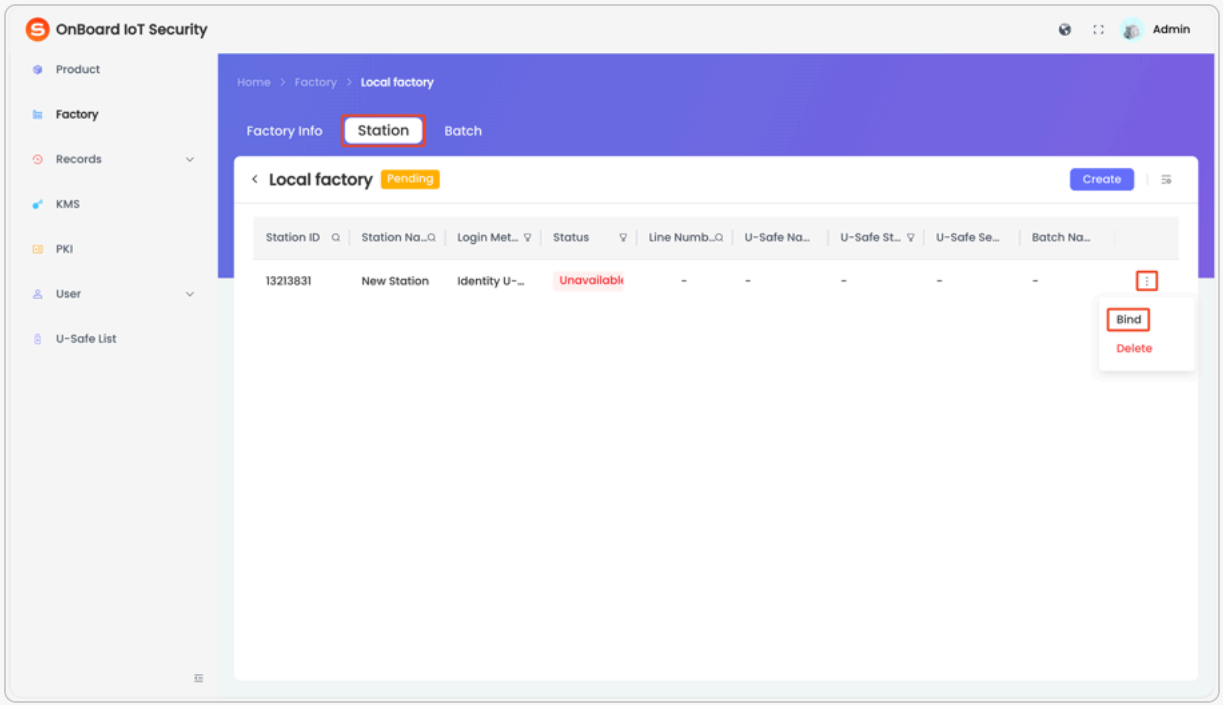
Bind an Identity U-Safe to a programming station

> **TIP**
>
> You can use the Identity U-Safe to sign in to the Programming Station client after binding.

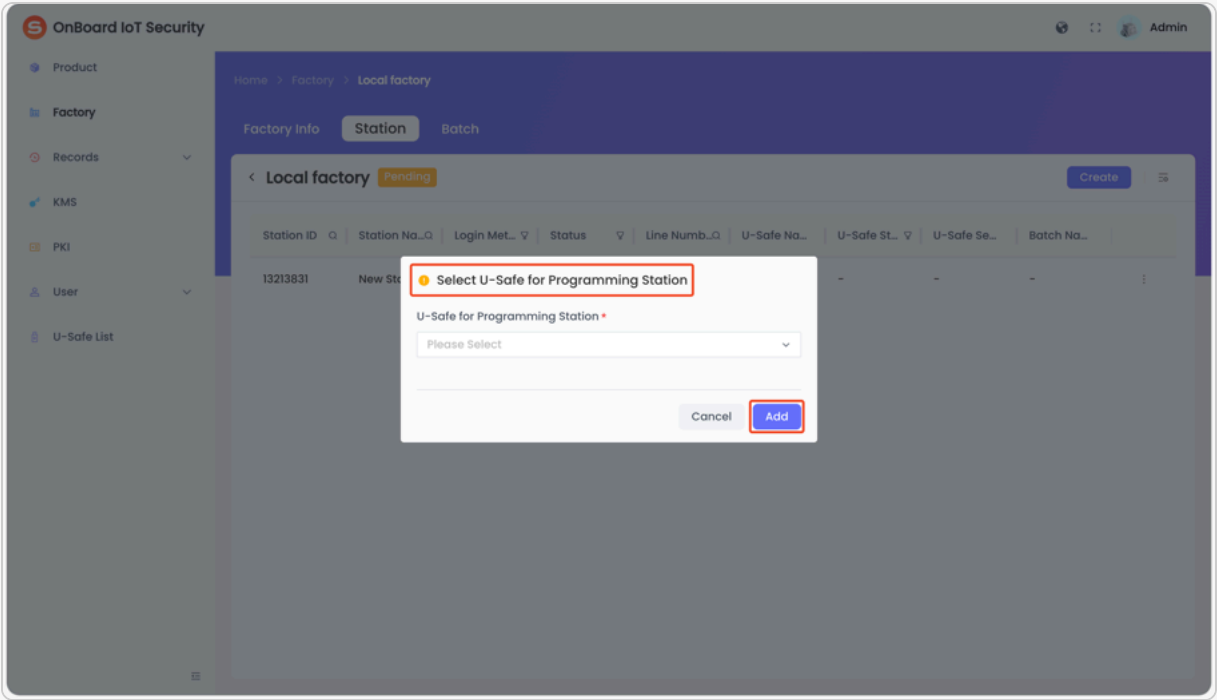
1. Sign in to OBIS.

2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.

3. On the factory page, select the **Station** tab, and then click the Options ⋮ icon for the corresponding programming station and select **Bind**.



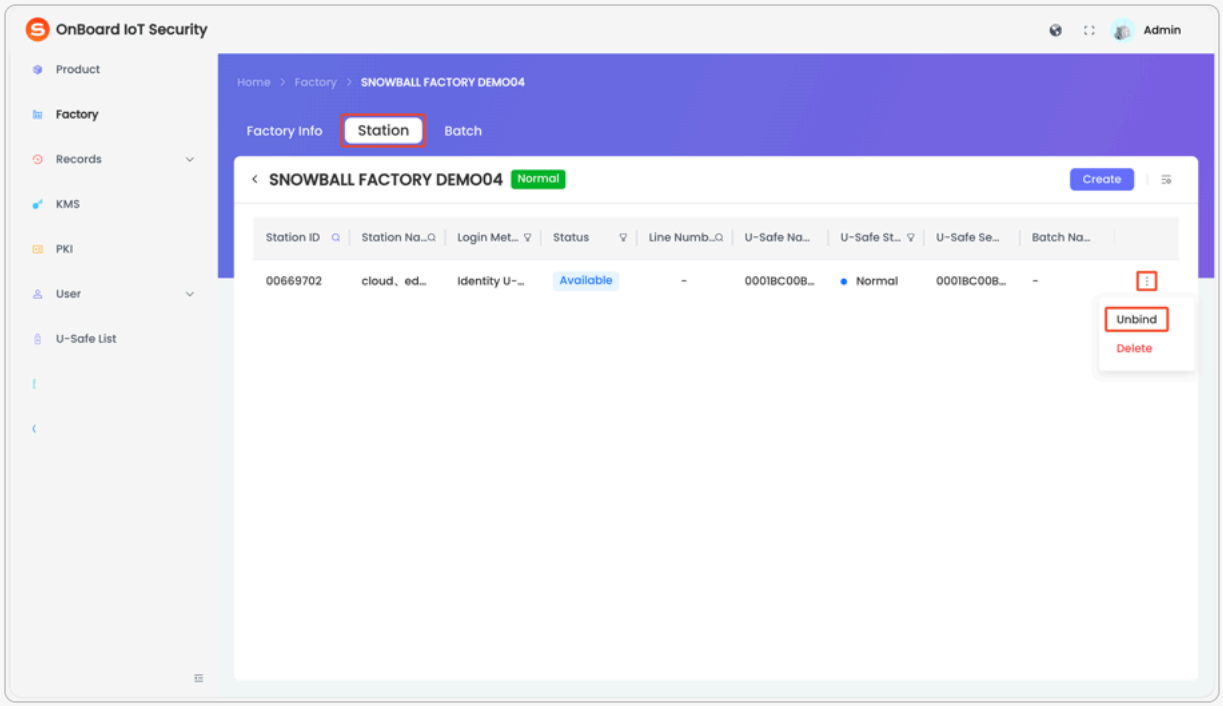
4. In the **Select U-Safe for Programming Station** window, select the Identity U-Safe you want to bind and then click **Add** to bind it to the programming station.



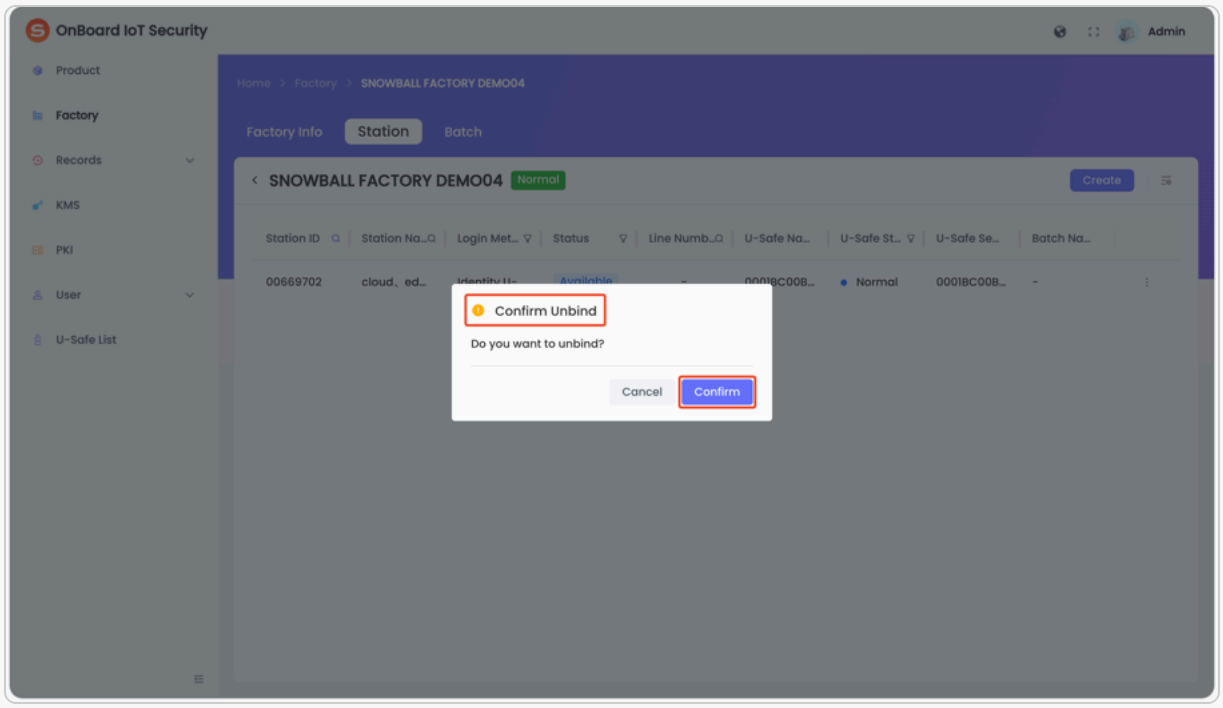
Unbind an Identity U-Safe from a programming station

> **WARNING**
>
> After unbinding, the Identity U-Safe can no longer be used to sign in to the Programming Station client.

1. Sign in to OBIS.
2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.
3. On the factory page, select the **Station** tab, and then click the Options ⋮ icon for the corresponding programming station and select **Unbind**.

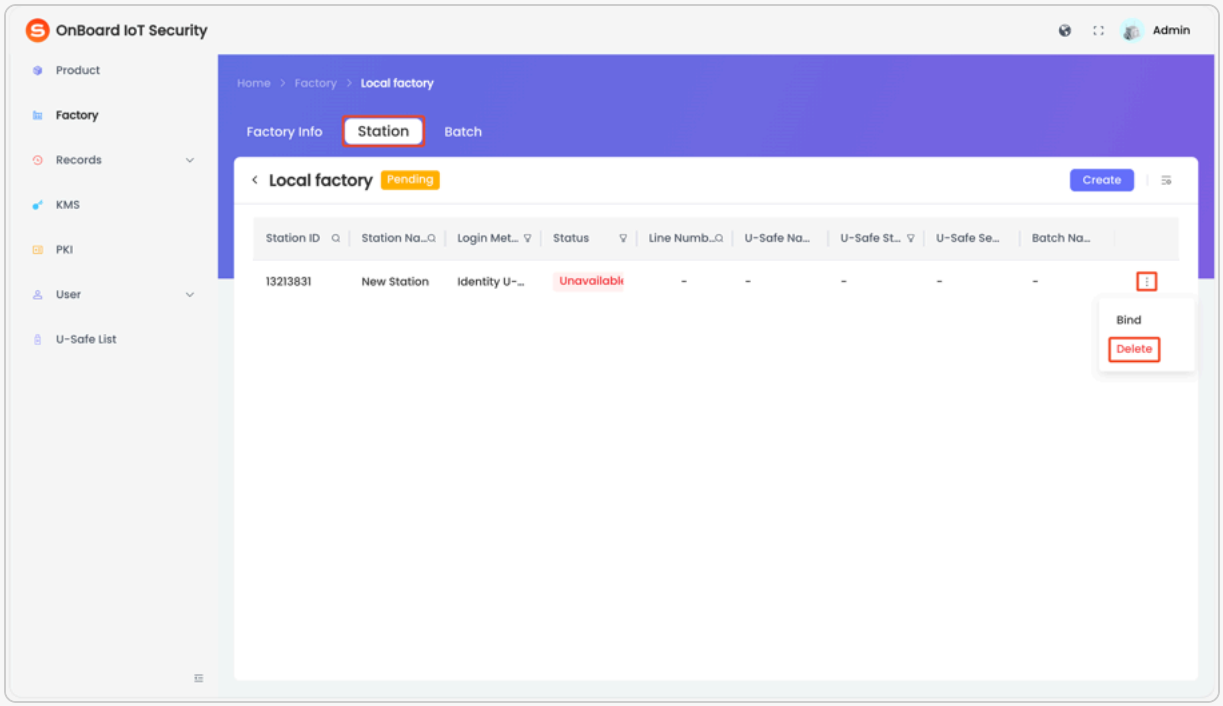


4. In the **Confirm Unbind** window, click **Confirm** to unbind the Identity U-Safe for the programming station.

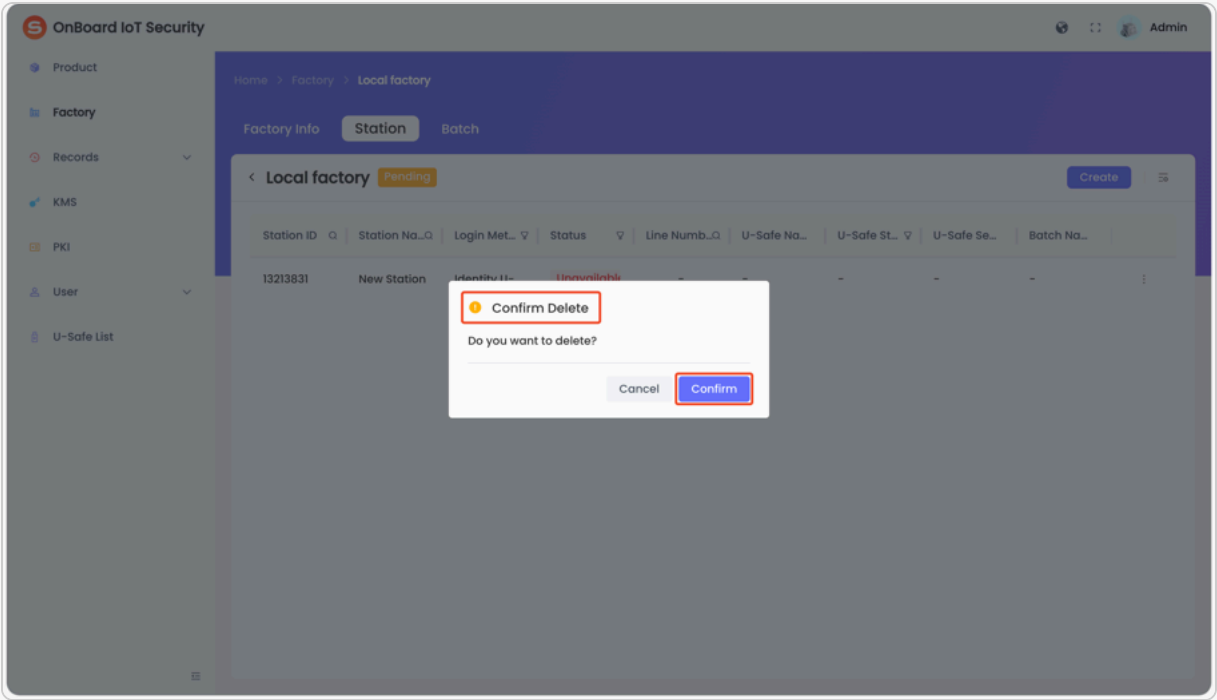


Delete a programming station

1. Sign in to OBIS.
2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.
3. On the factory page, select the **Station** tab, and then click the Options ⋮ icon for the corresponding programming station and select **Delete**.



4. In the **Confirm Delet** window, click **Confirm** to delete the programming station.

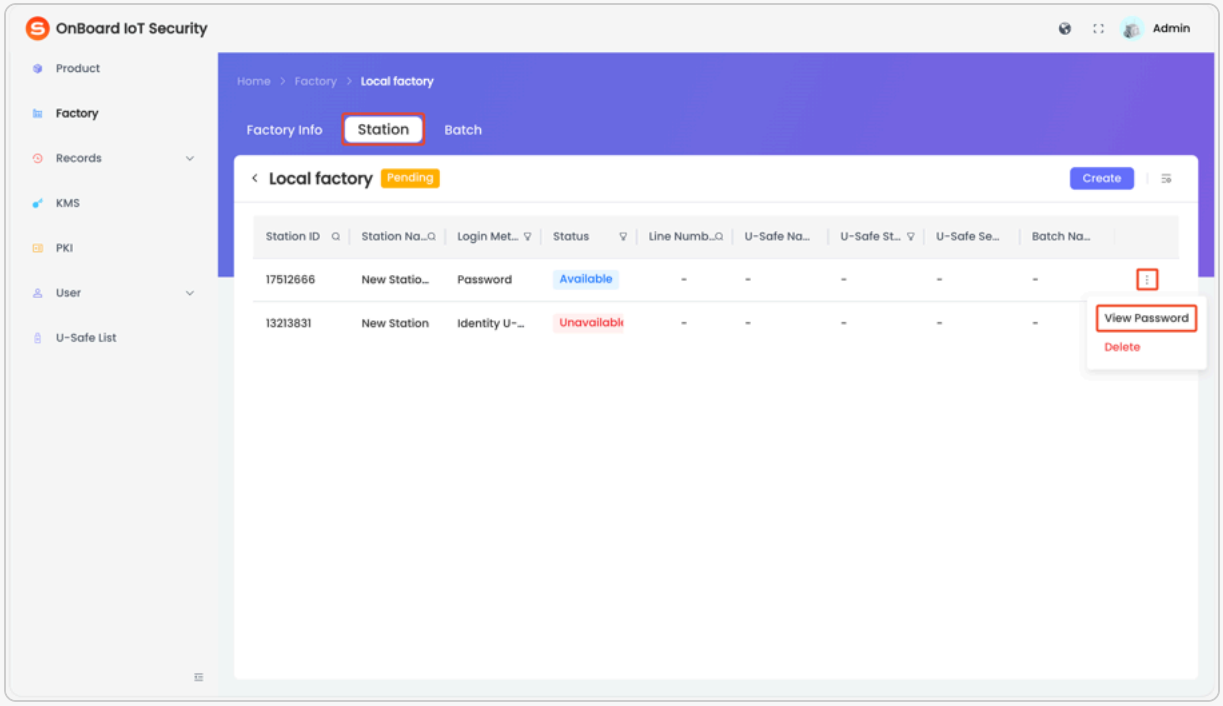


Check the programming station password

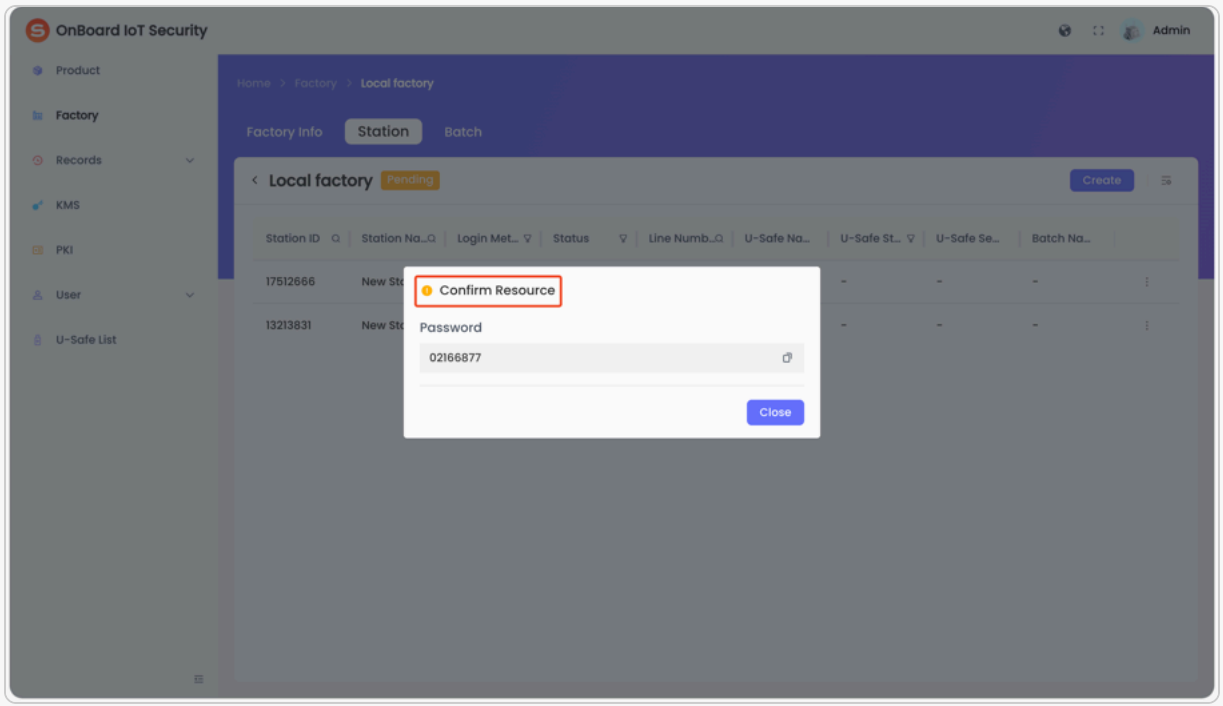
> **TIP**
>
> When the login method for the programming station is **Password**, the system will generate a random password, which can be used to sign in to the Programming Station client. Currently, the randomly generated password can't be changed.

1. Sign in to OBIS.
2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.

3. On the factory page, select the **Station** tab, and then click the Options ⋮ icon for the corresponding programming station and select **View Password**.



4. In the **Confirm Resource** window, you can check and copy the sign-in password for the programming station.



Version: v3.2.1

How to configure and manage factories in OnBoard™ IoT Security

This article will guide you through the following operations in OnBoard™ IoT Security (OBIS):

- Add a Factory Manager
- Reset the Factory Manager password
- Delete a Factory Manager
- Add an Identity U-Safe
- Unbind, lock, or unlock an Identity U-Safe
- Lock or unlock a factory

Prerequisites

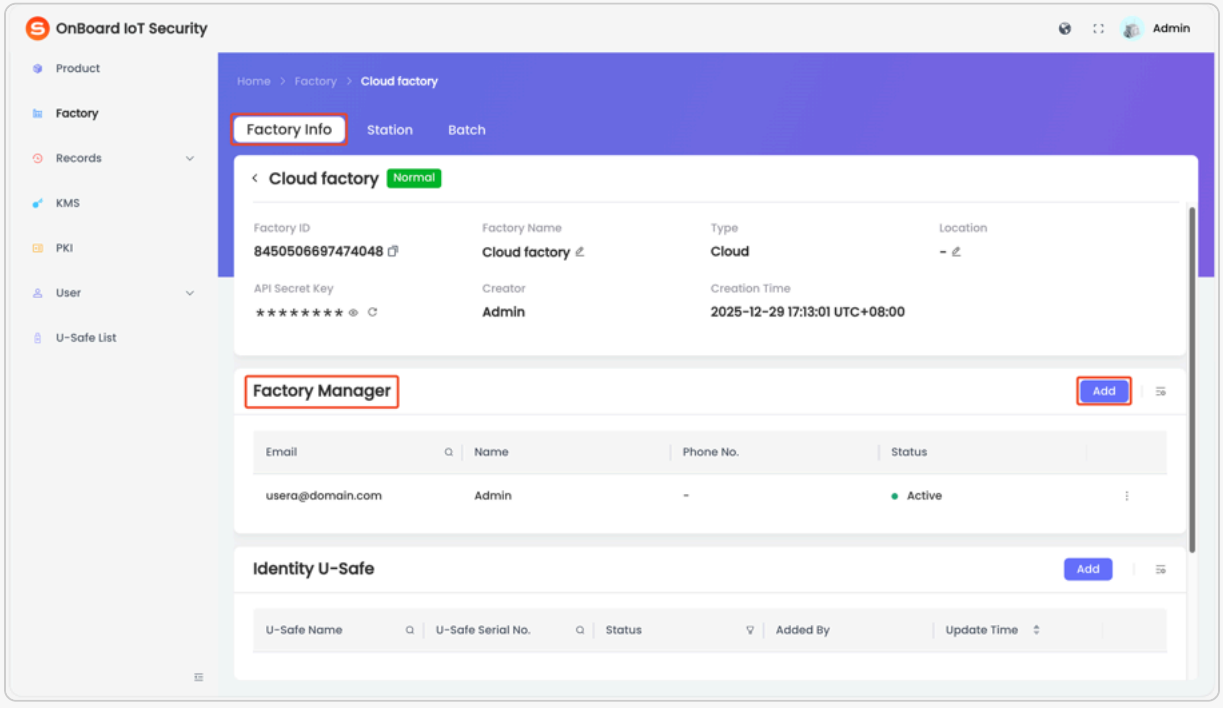
Before getting started, please ensure that:

- You have one of the following role permissions:
  - OBIS role: System Manager or Factory Manager
  - Factory role: Factory Manager corresponding to the factory
- You have obtained the Identity U-Safe required for identity authentication from Snowball.

Add a Factory Manager

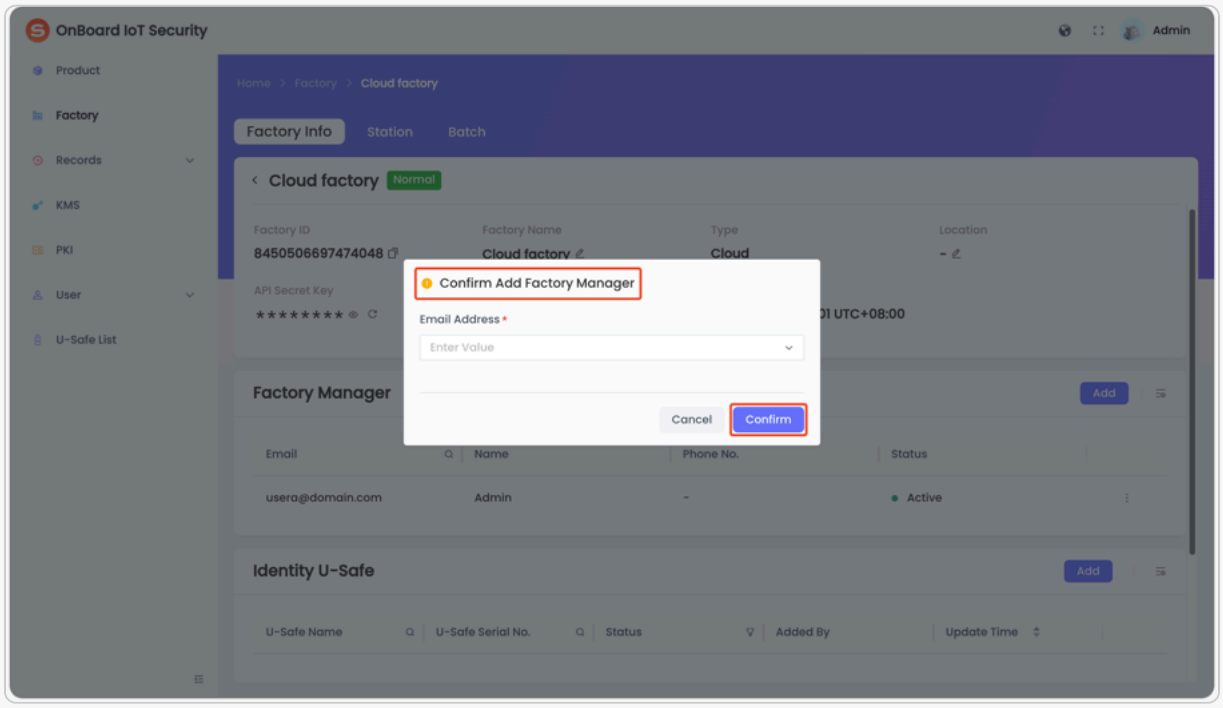
> **TIP**
>
> Factory managers can manage and operate factory-related businesses. After a Factory Manager is added, the system automatically generates an initial password and sends it to the corresponding email address. Currently, this randomly generated password can be reset, but custom modification is not yet supported.

1. Sign in to OBIS.
2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.
3. On the factory page, locate the **Factory Manager** area and then click **Add**.



4. In the **Confirm Add Factory Manager** window, select and enter the email address of the new Factory Manager.

> **TIP**
>
> The Factory Manager being added must be an **Active** user within the same tenant.



5. Click **Confirm** to complete adding a Factory Manager.

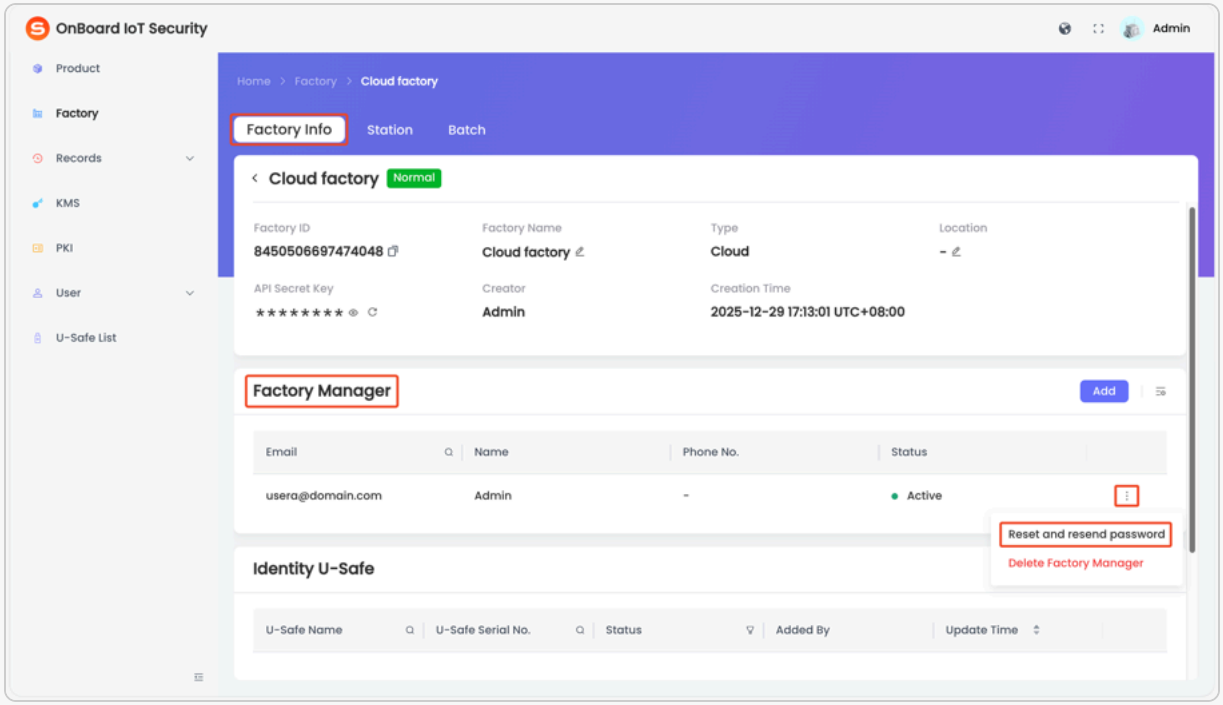
Reset the Factory Manager password

> **TIP**
>
> After the password is reset, the system automatically generates a random password and sends it to the corresponding email address.

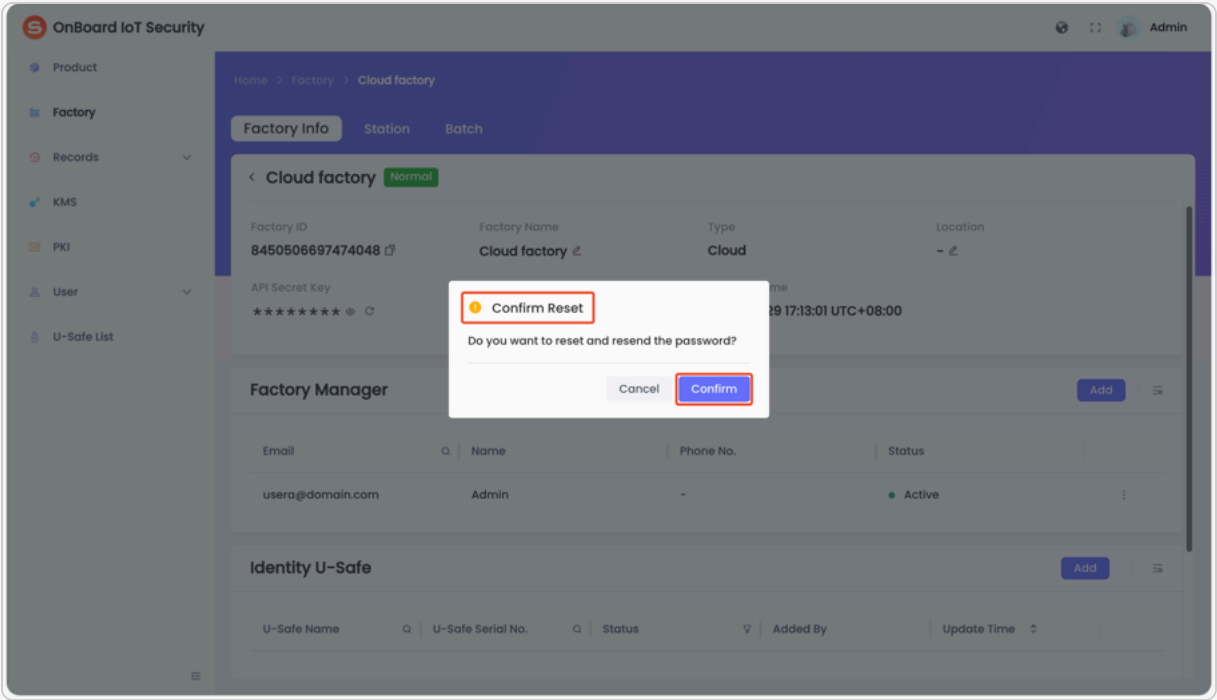
1. Sign in to OBIS.

2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.

3. On the factory page, locate the **Factory Manager** area, and then click the Options ⋮ icon for the corresponding Factory Manager and select **Reset and resend password**.



4. In the **Confirm Reset** window, click **Confirm** to reset the password.

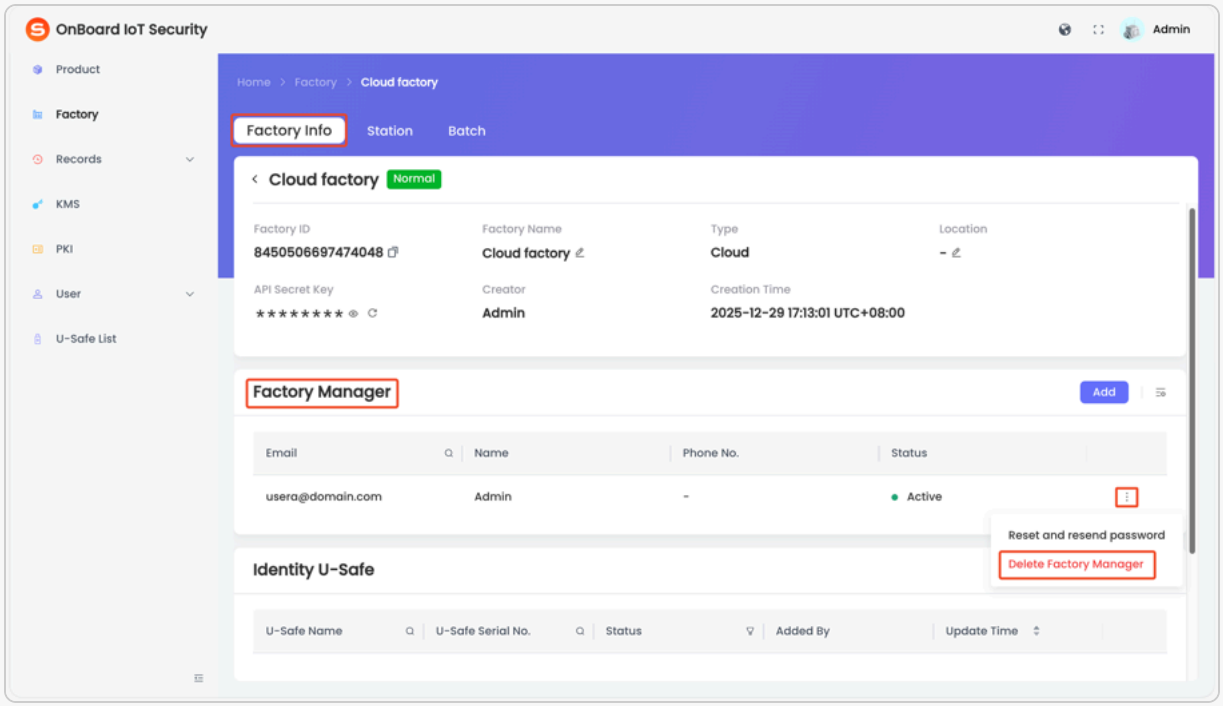


Delete a Factory Manager

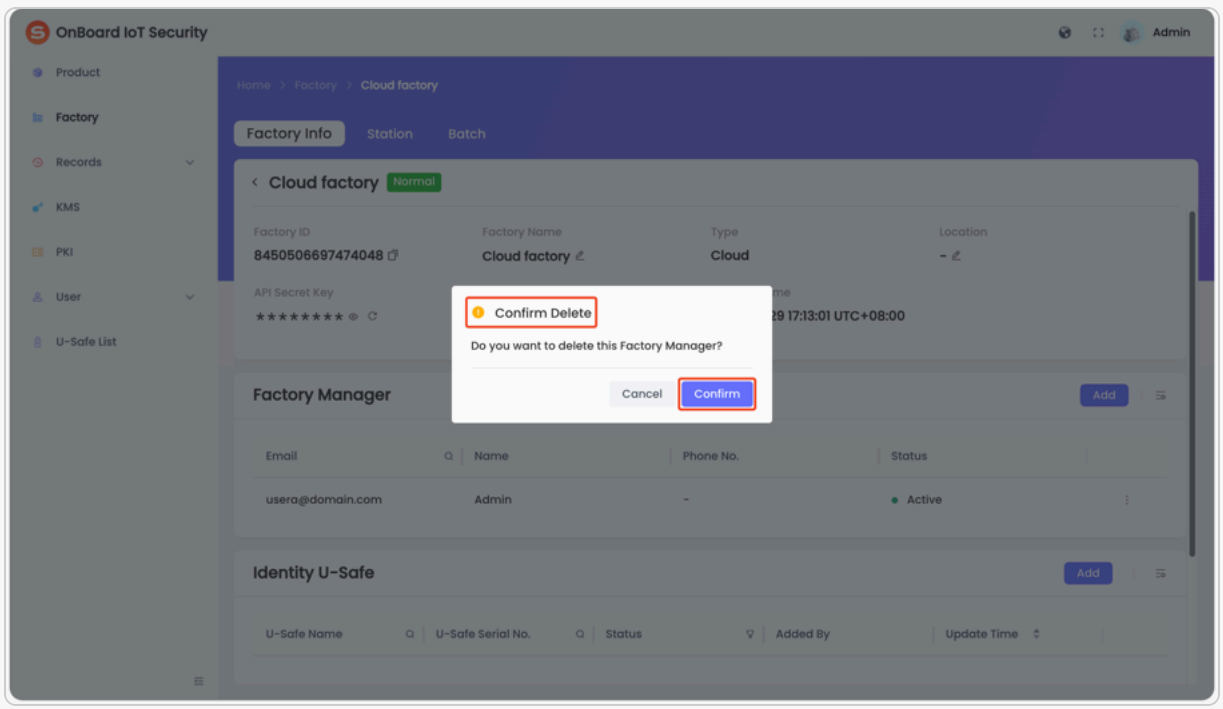
1. Sign in to OBIS.

2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.

3. On the factory page, locate the **Factory Manager** area, and then click the Options ⋮ icon for the corresponding Factory Manager and select **Delete Factory Manager**.



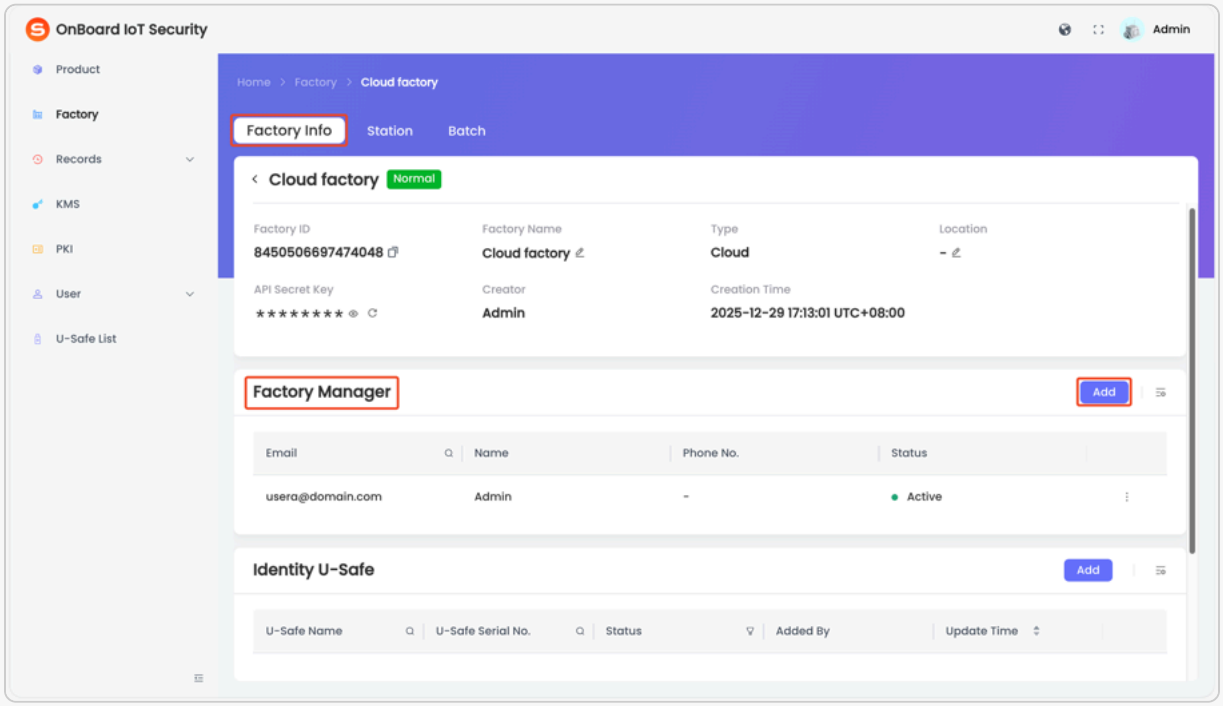
4. In the **Confirm Delete** window, click **Confirm** to delete the Factory Manager.



Add an Identity U-Safe

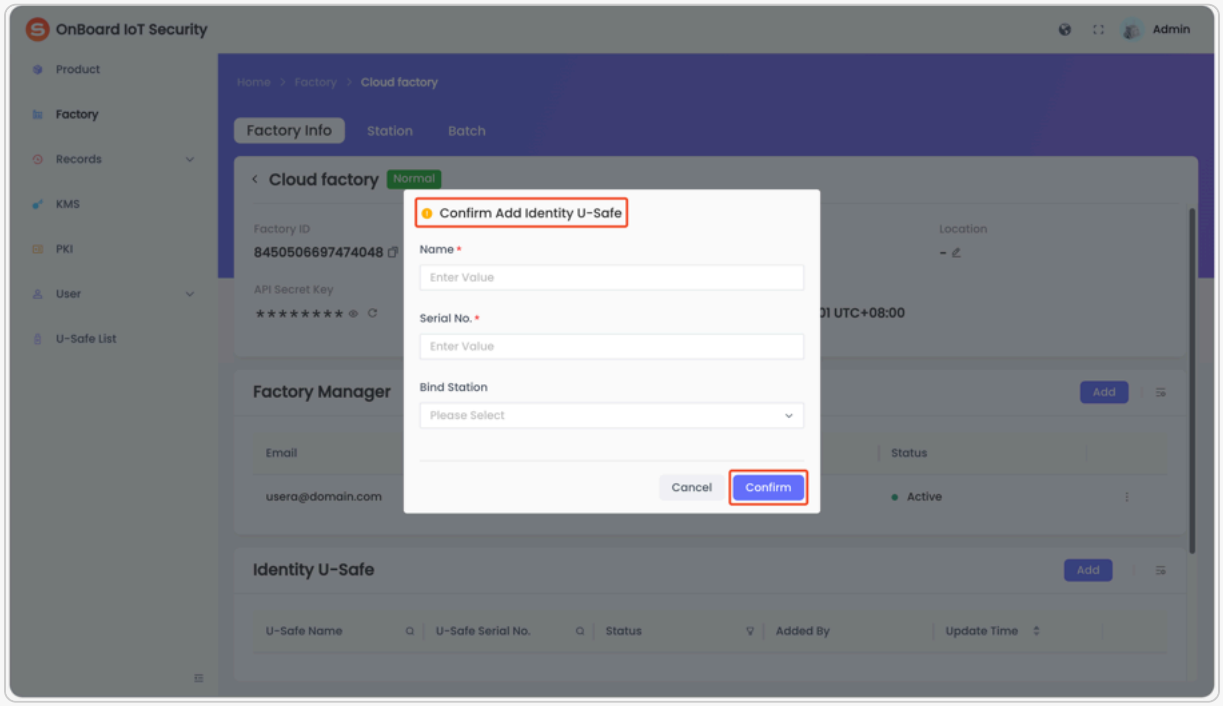
> **TIP**
>
> The purpose of adding an Identity U-Safe is to bind it to the corresponding factory. After binding it to a programming station, it can be used to sign in to the Programming Station client. Meanwhile, login authentication for the programming station is achieved through the Identity U-Safe management.

1. Sign in to OBIS.
2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.
3. On the factory page, locate the **Identity U-Safe** area and then click **Add**.



4. In the **Confirm Add Identity U-Safe** window, enter the **Name** (customizable) and **Serial No.** (which can be found on the device) of the Identity U-Safe you want to bind.

> **TIP**
>
> - If a programming station has already been added, you can select and bind the corresponding programming station.
> - If a programming station hasn't been added yet, you can bind it when adding a programming station.

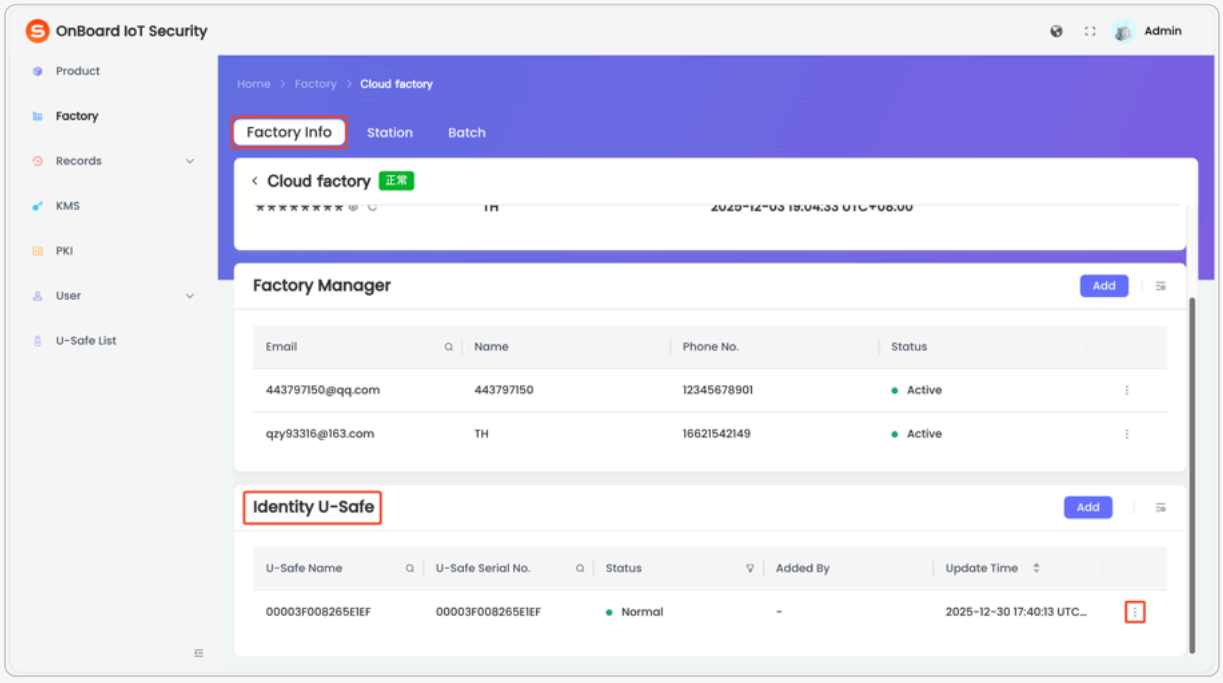


5. Click **Confirm** to bind the Identity U-Safe.

Unbind, lock, or unlock an Identity U-Safe

> **TIP**
>
> Lock the Identity U-Safe to prevent its use and unbinding. Once locked, its status will change to **Locked**.

1. Sign in to OBIS.
2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.
3. On the factory page, locate the **Identity U-Safe** area, and then click the Options ⋮ icon for the corresponding Identity U-Safe and select **Unbind**, **Lock**, or **Unlock**.

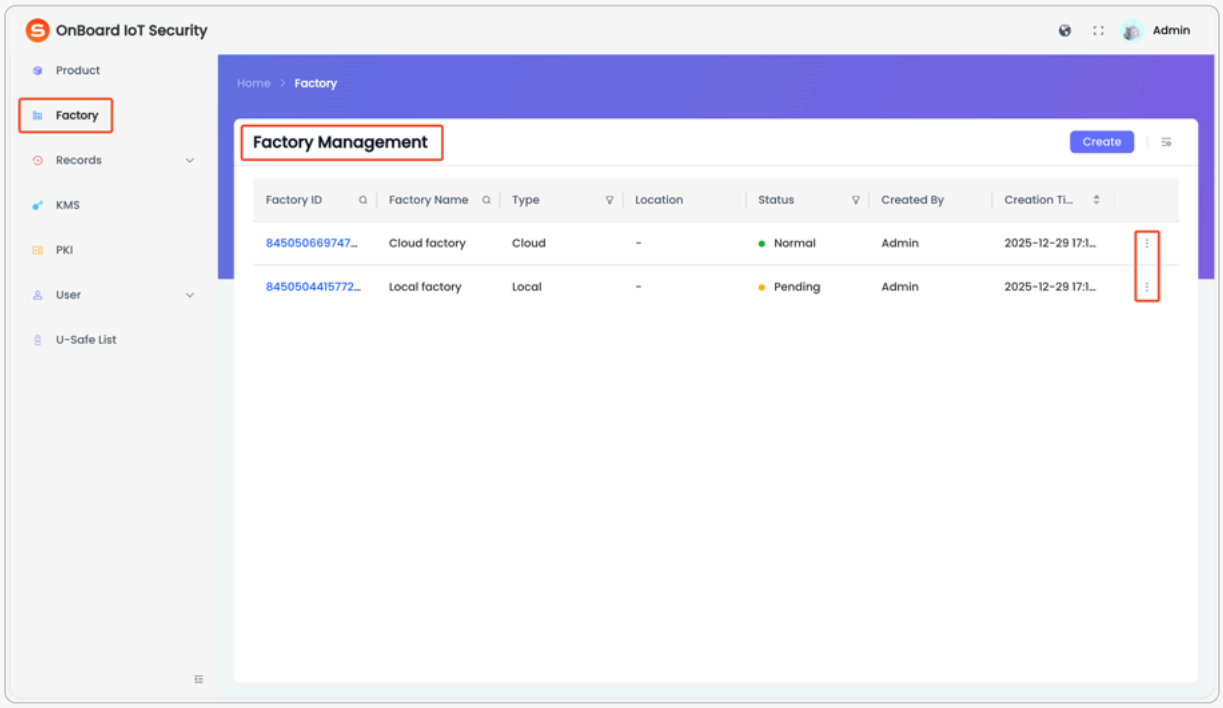


4. In the pop-up window, click **Confirm** to unbind, lock, or unlock the Identity U-Safe.

Lock or unlock a factory

> **TIP**
>
> Lock the factory to prevent its use. Once locked, its status will change to **Locked**.

1. Sign in to OBIS.
2. In the left menu, click **Factory**, and then click the Options ⋮ icon for the corresponding factory and select **Lock** or **Unlock**.



3. In the pop-up window, click **OK** to lock or unlock the factory.

Version: v3.2.1

How to configure and manage programming stations in OnBoard™ IoT Security

This article will guide you through the following operations in OnBoard™ IoT Security (OBIS):

- Create a programming station
- Bind an Identity U-Safe to a programming station
- Unbind an Identity U-Safe from a programming station
- Delete a programming station
- Check the programming station password

Prerequisites

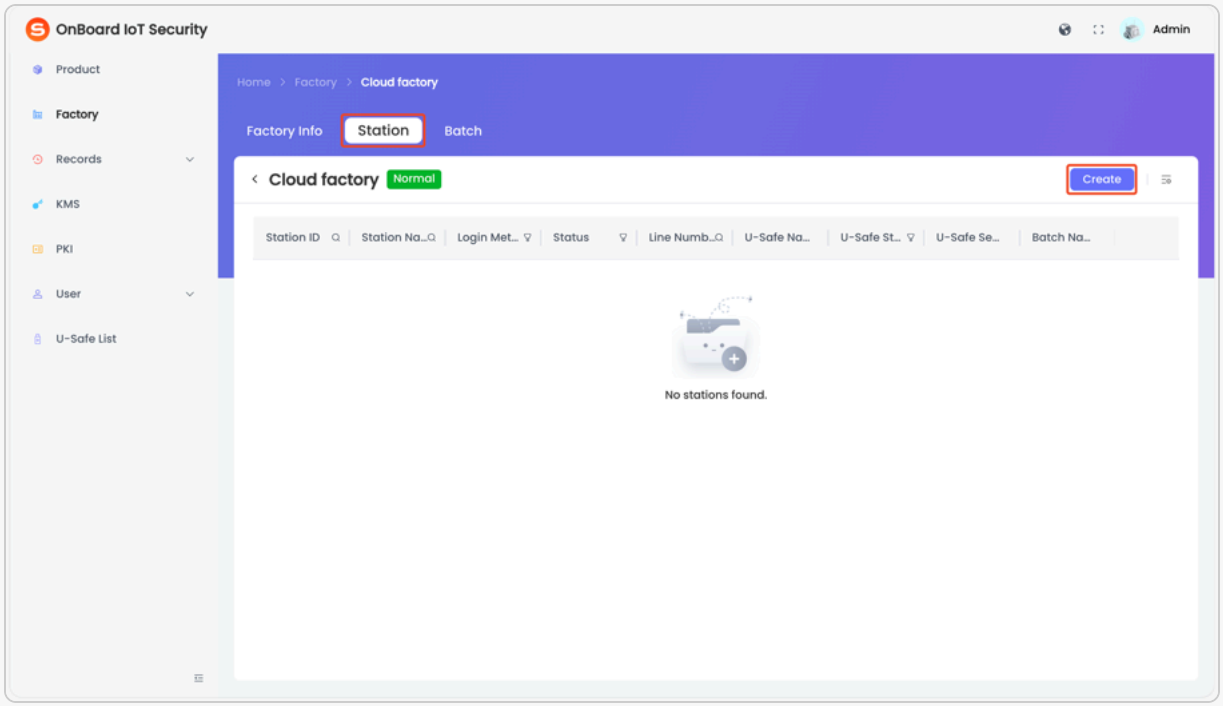
Before getting started, please ensure that:

- The factory has already been configured.
- You have one of the following role permissions:
  - OBIS role: System Manager or Factory Manager
  - Factory role: Factory Manager corresponding to the factory
- You have obtained the Identity U-Safe for identity authentication from Snowball.

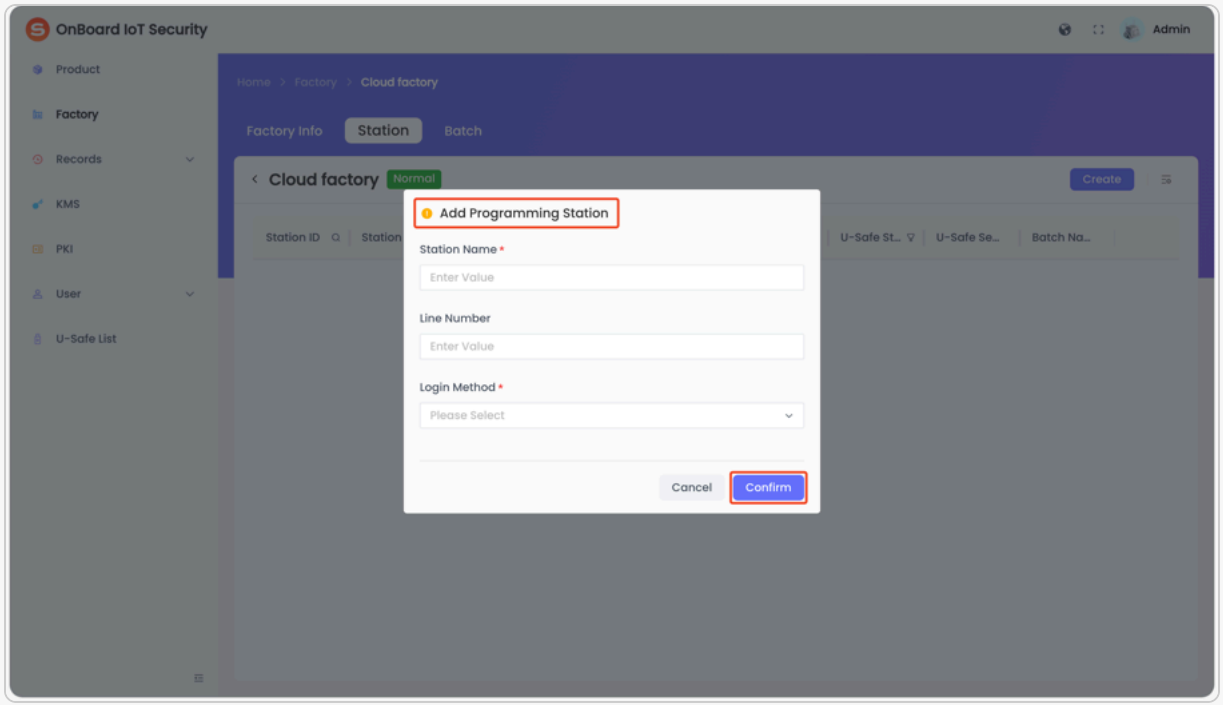
Create a programming station

> **TIP**
>
> After creating a programming station for the factory, you can assign specific programming stations to batches as needed. Before programming a device, it is necessary to query and retrieve the corresponding batch information and programming data based on the programming station ID.
>
> Two programming methods are supported:
>
> - Using the Programming Station client
> - Using the OBIS Programming CLI

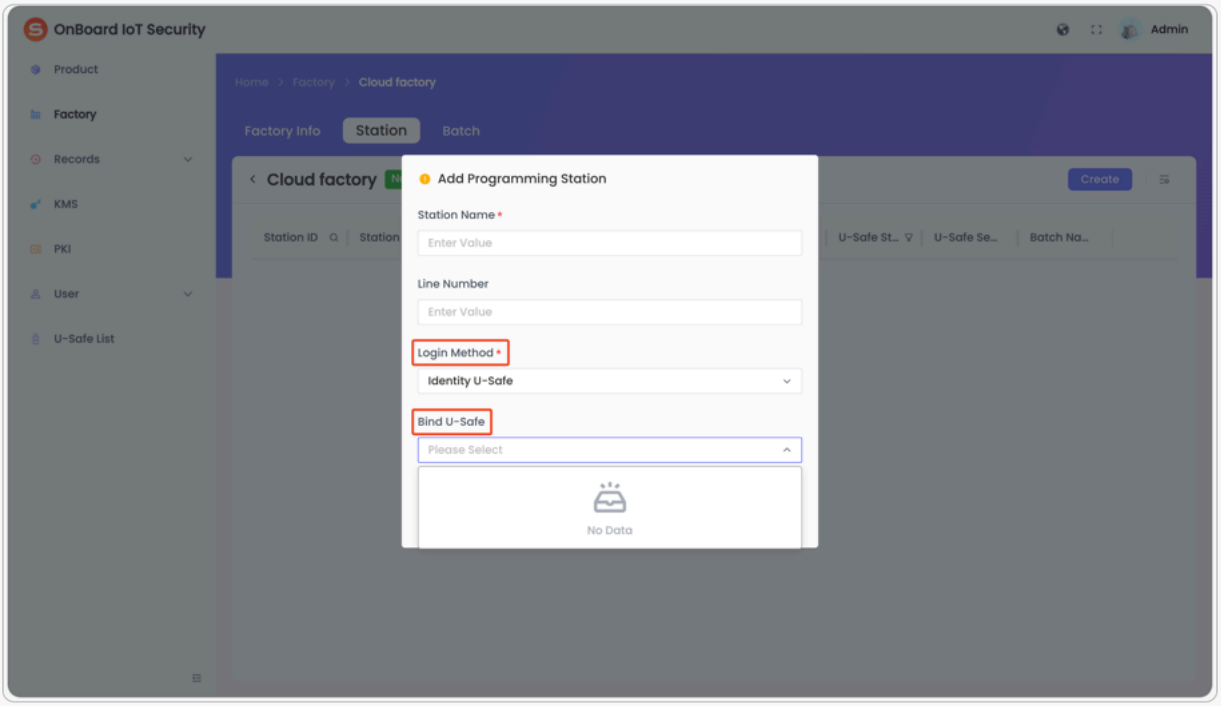
1. Sign in to OBIS.
2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.
3. On the factory page, select the **Station** tab and then click **Create**.



4. In the **Add Programming Station** window, enter the **Station Name** and **Line Number**, and then select the **Login Method**. The methods include: **Password** or **Identity U-Safe**.



> **TIP**
>
> If the login method is **Identity U-Safe**, you can select an already added Identity U-Safe from the **Bind U-Safe** drop-down list to bind it, or you can bind it after adding the programming station.



5. Click **Confirm** to create the programming station.

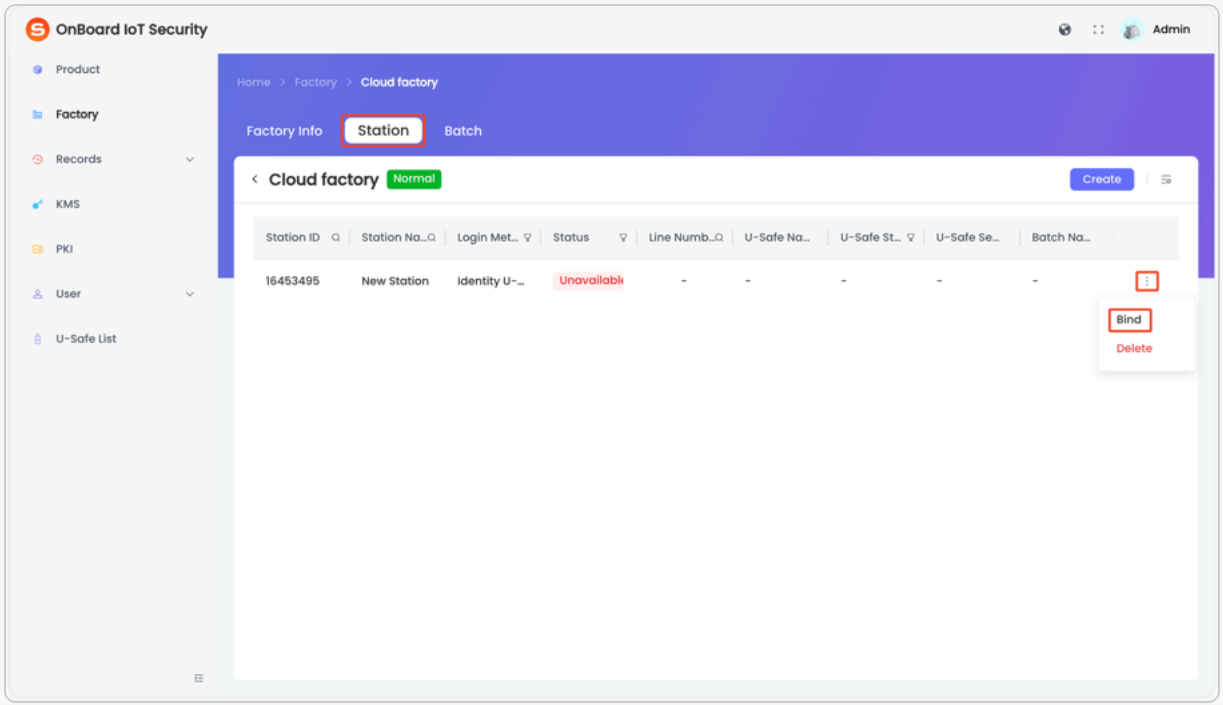
Bind an Identity U-Safe to a programming station

> **TIP**
>
> You can use the Identity U-Safe to sign in to the Programming Station client after binding.

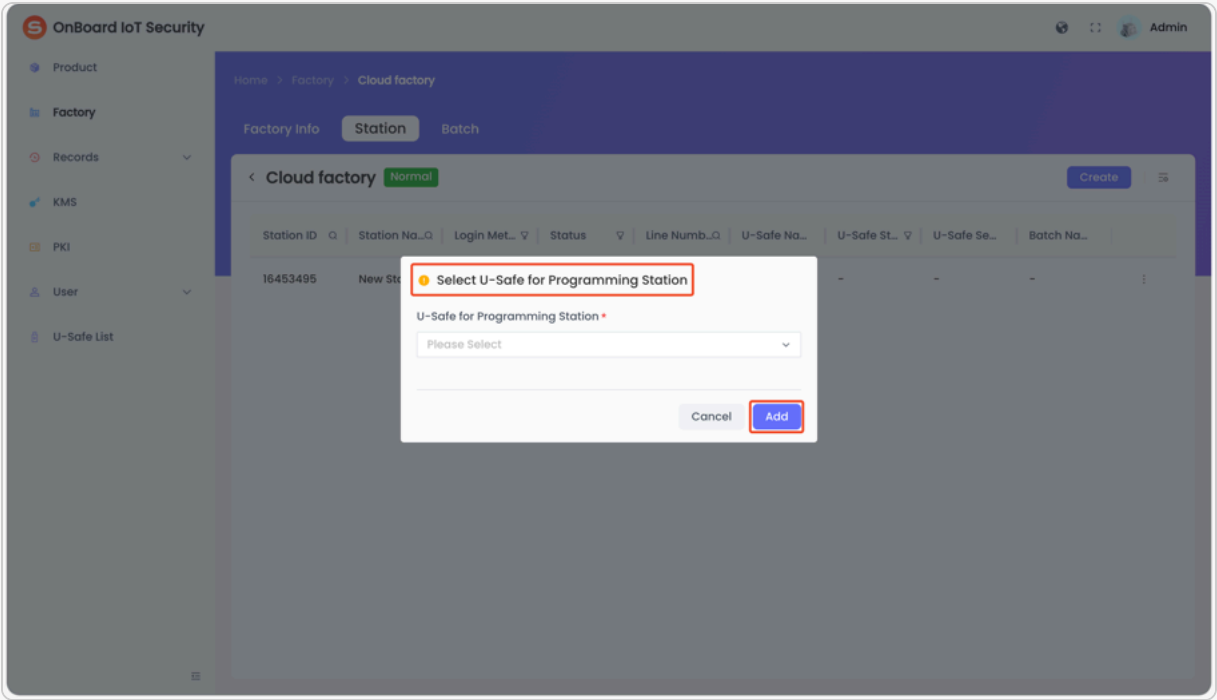
1. Sign in to OBIS.

2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.

3. On the factory page, select the **Station** tab, and then click the Options ⋮ icon for the corresponding programming station and select **Bind**.



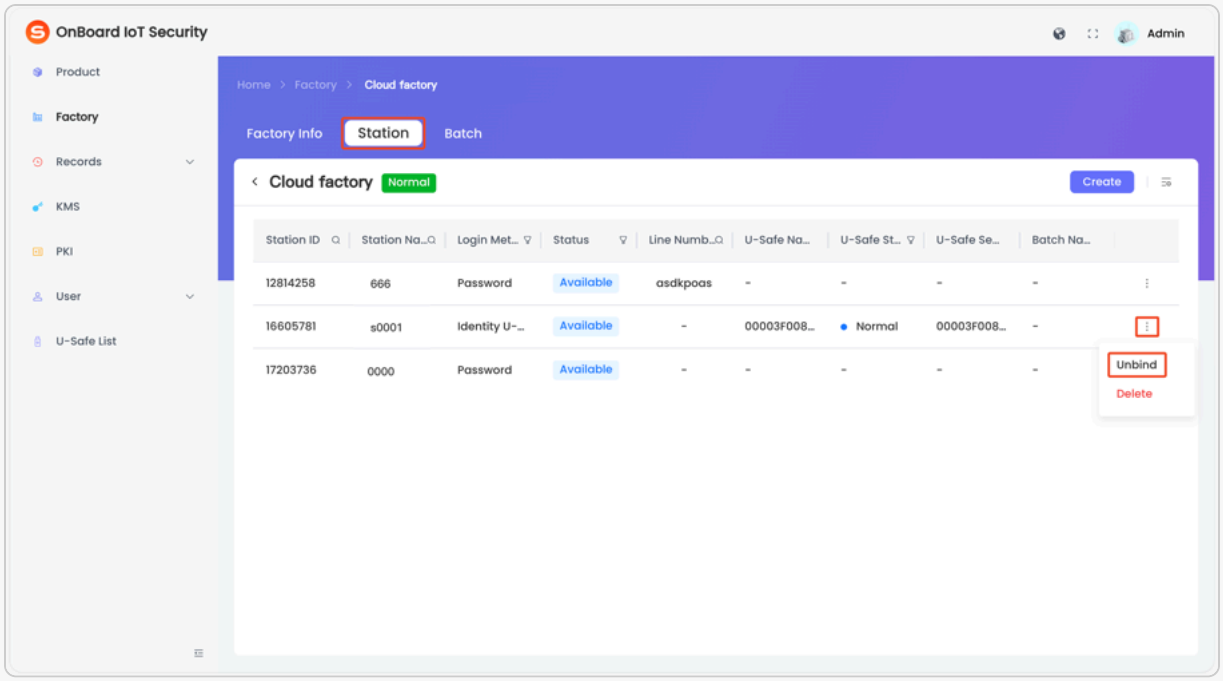
4. In the **Select U-Safe for Programming Station** window, select the Identity U-Safe you want to bind and then click **Add** to bind it to the programming station.



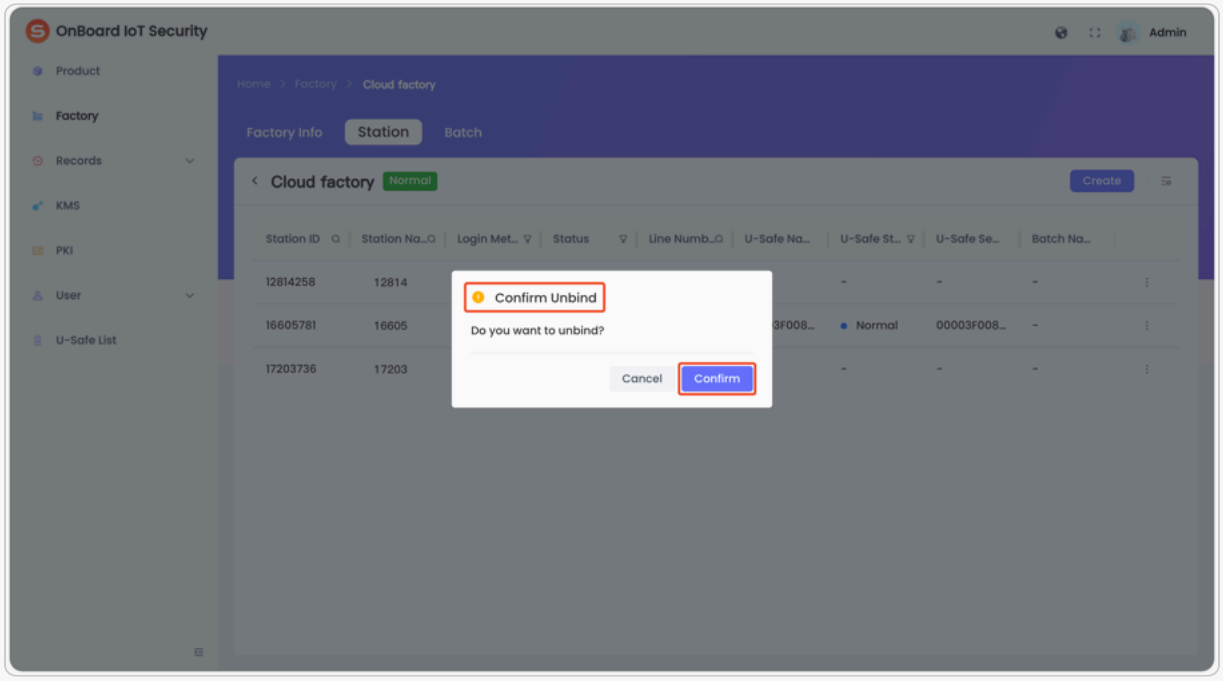
Unbind an Identity U-Safe from a programming station

> **WARNING**
>
> After unbinding, the Identity U-Safe can no longer be used to sign in to the Programming Station client.

1. Sign in to OBIS.
2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.
3. On the factory page, select the **Station** tab, and then click the Options ⋮ icon for the corresponding programming station and select **Unbind**.

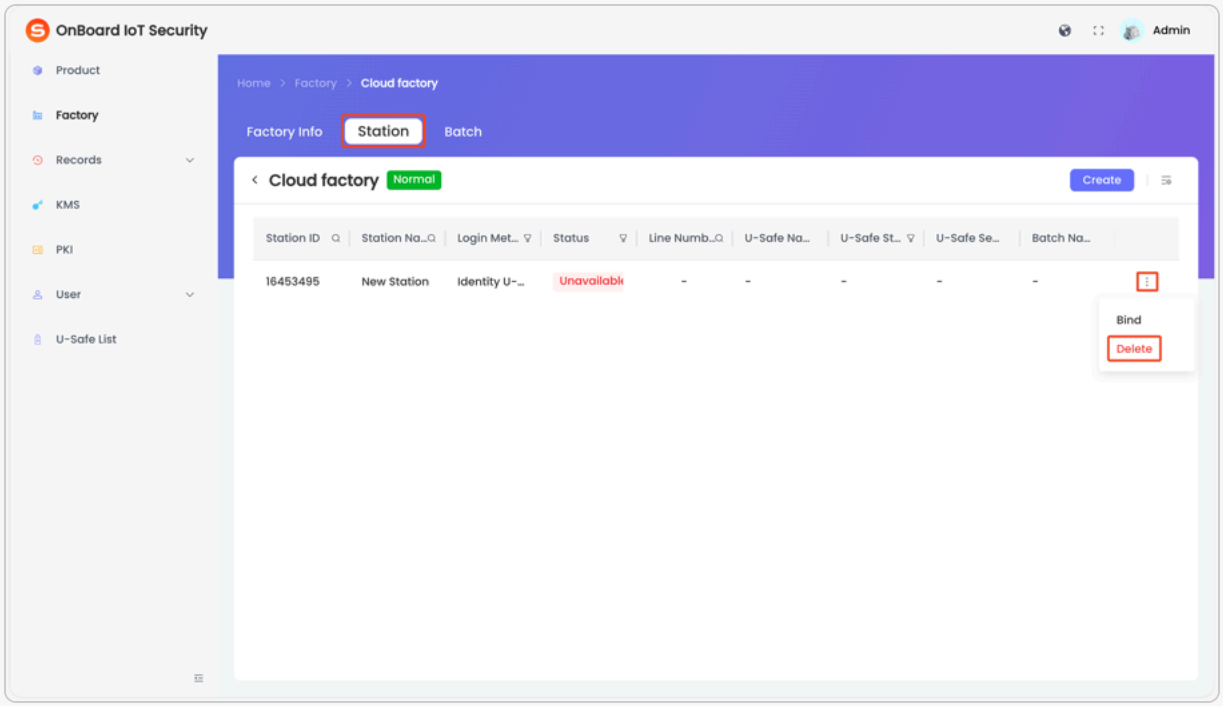


4. In the **Confirm Unbind** window, click **Confirm** to unbind the Identity U-Safe for the programming station.

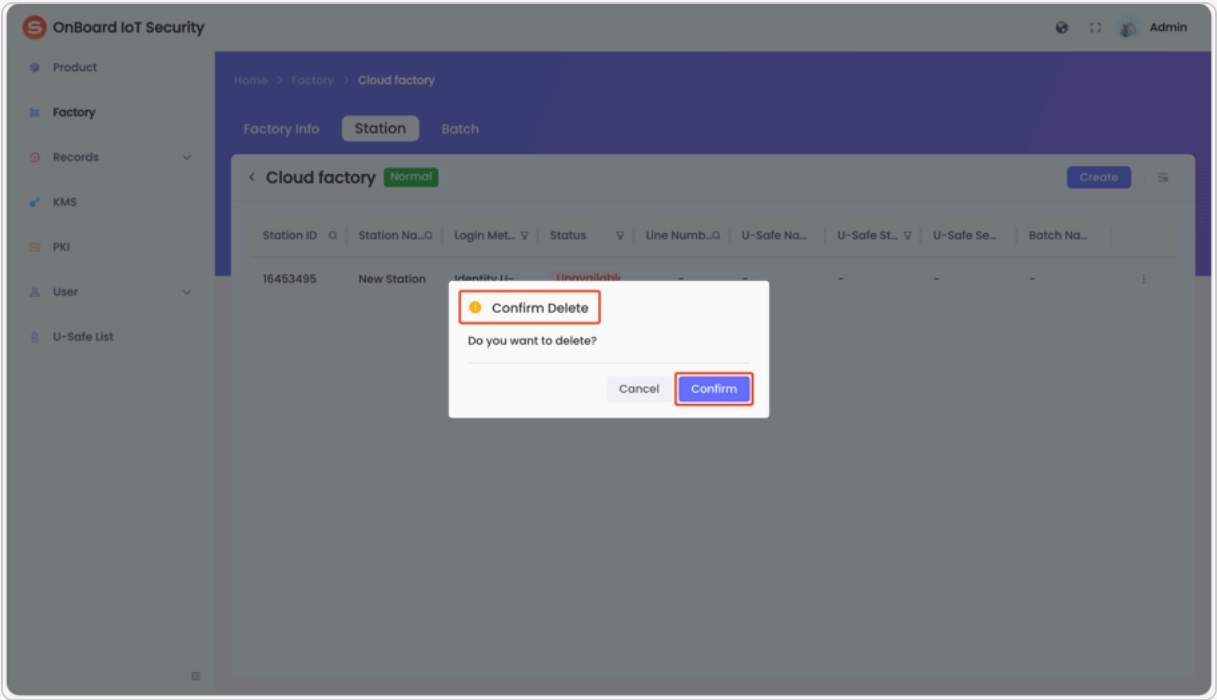


Delete a programming station

1. Sign in to OBIS.
2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.
3. On the factory page, select the **Station** tab, and then click the Options ⋮ icon for the corresponding programming station and select **Delete**.



4. In the **Confirm Delete** window, click **Confirm** to delete the programming station.

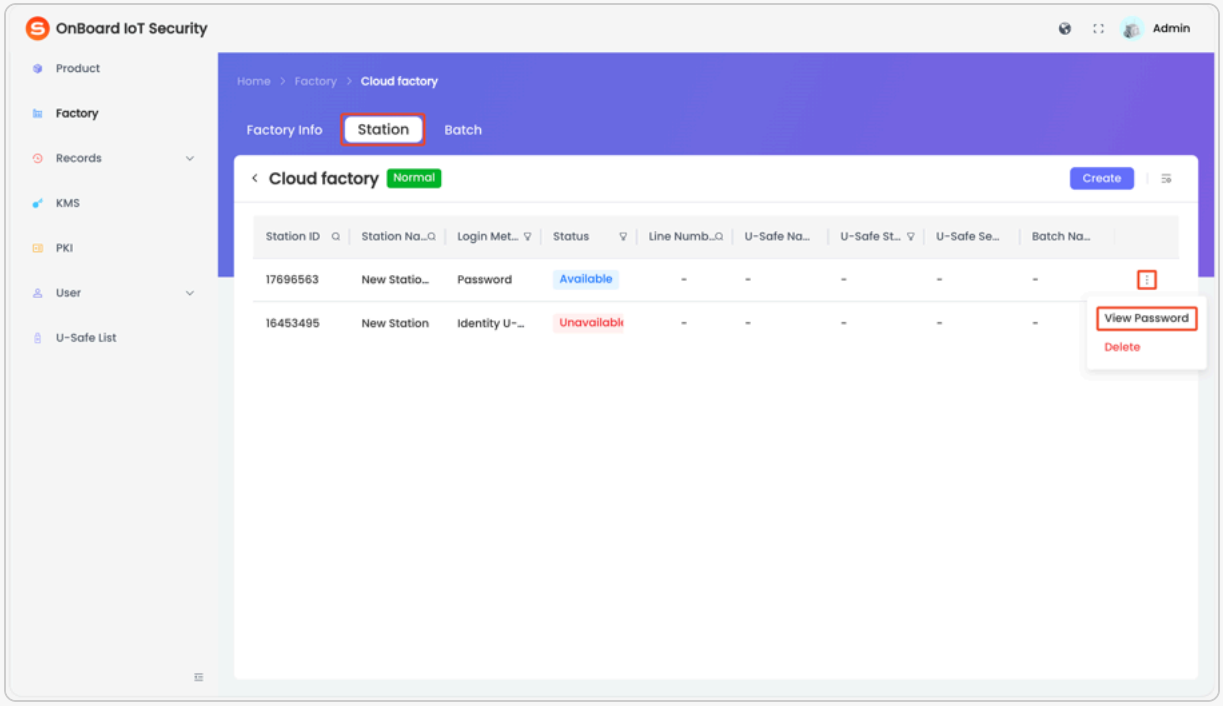


Check the programming station password

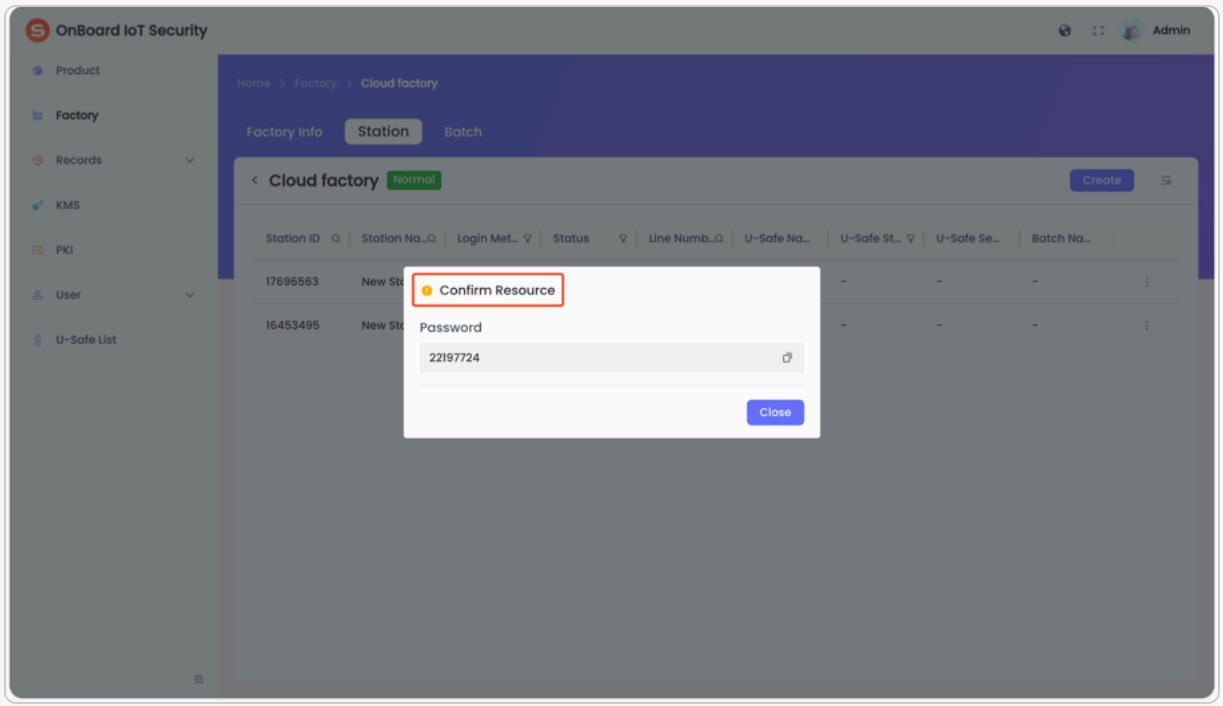
> **TIP**
>
> When the login method for the programming station is **Password**, the system will generate a random password, which can be used to sign in to the Programming Station client. Currently, the randomly generated password can't be changed.

1. Sign in to OBIS.
2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.

3. On the factory page, select the **Station** tab, and then click the Options ⋮ icon for the corresponding programming station and select **View Password**.



4. In the **Confirm Resource** window, you can check and copy the sign-in password for the programming station.



Version: v3.2.1

How to manage a batch and assign a programming station in OnBoard Edge Service

This article will guide you through the following operations in OnBoard™ IoT Security (OBIS):

- Check batch status
- Assign a programming station to a batch
- Unassign a programming station from a batch
- Turn off and delete a batch

Prerequisites

Before getting started, please ensure that:

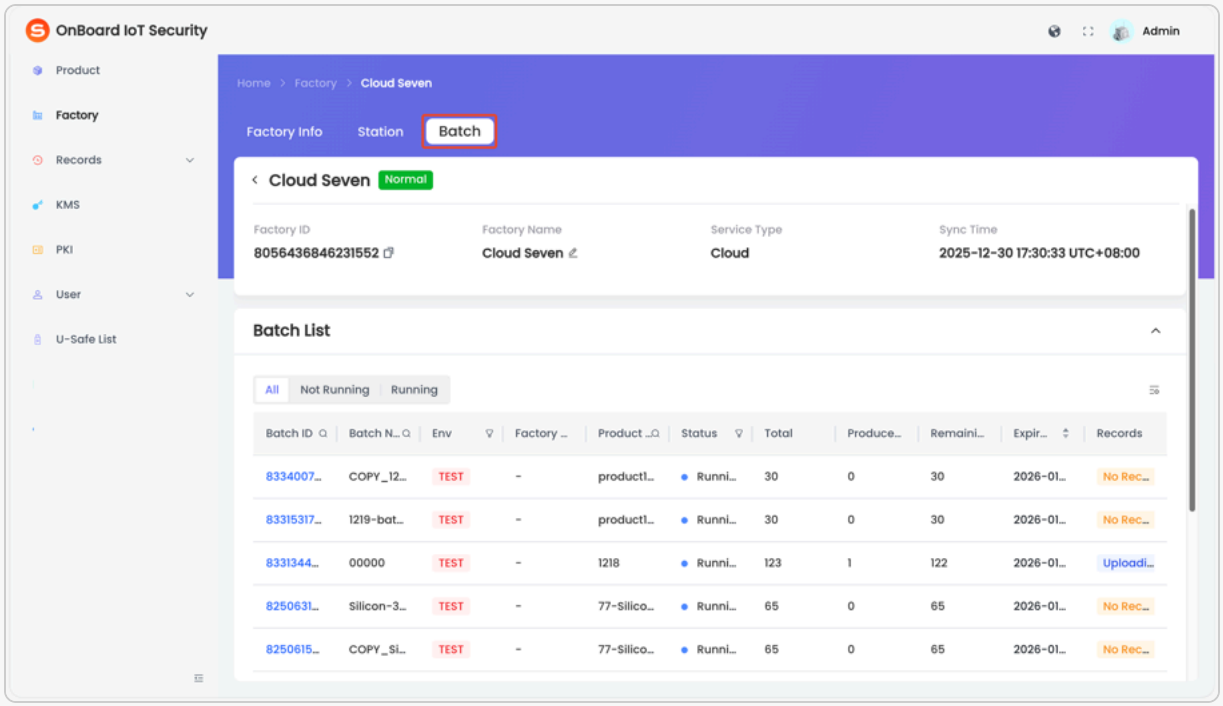
- The factory has already been configured.
- You have one of the following role permissions:
  - OBIS role: System Manager or Factory Manager
  - Factory role: Factory Manager corresponding to the factory

Check batch status

1. Sign in to OBIS.

2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.

3. On the factory page, select the **Batch** tab, and then you can check the batch status.



Description of batch status:

Status	Can be turned on	Description
Running	-	The default state after creating a batch. You can bind it to a programming station.
Stopped	Yes	The batch is currently turned off. You can turn it on to bind it to a programming station.
Finished	No	The status after batch programming is completed.
Expired	No	The batch has expired before the programming was completed. **Note**: After a batch expires, most functions related to that batch will no longer operate normally, including programming in the Programming Station client. It is recommended to delete the expired batches.

Assign a programming station to a batch

> **TIP**
>
> A batch can be assigned to multiple programming stations, while a programming station can only program one batch at the same time.

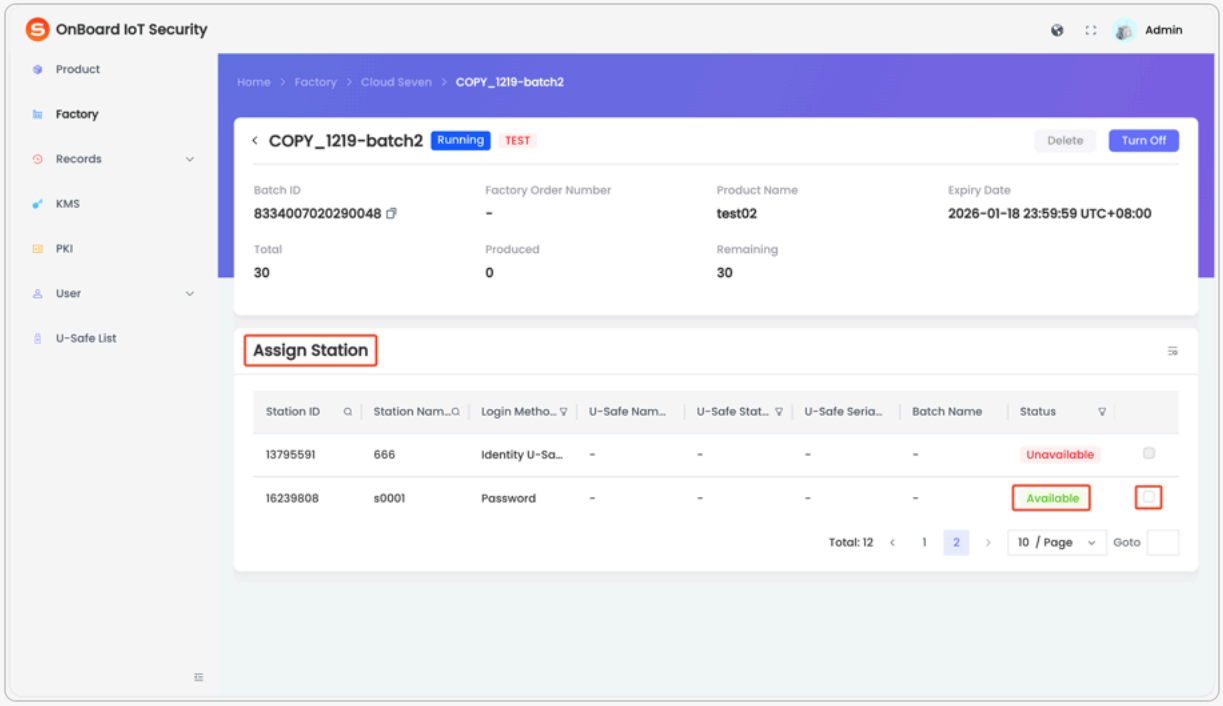
Assign a programming station with the **Available** status to a batch for programming:

1. Sign in to OBIS.

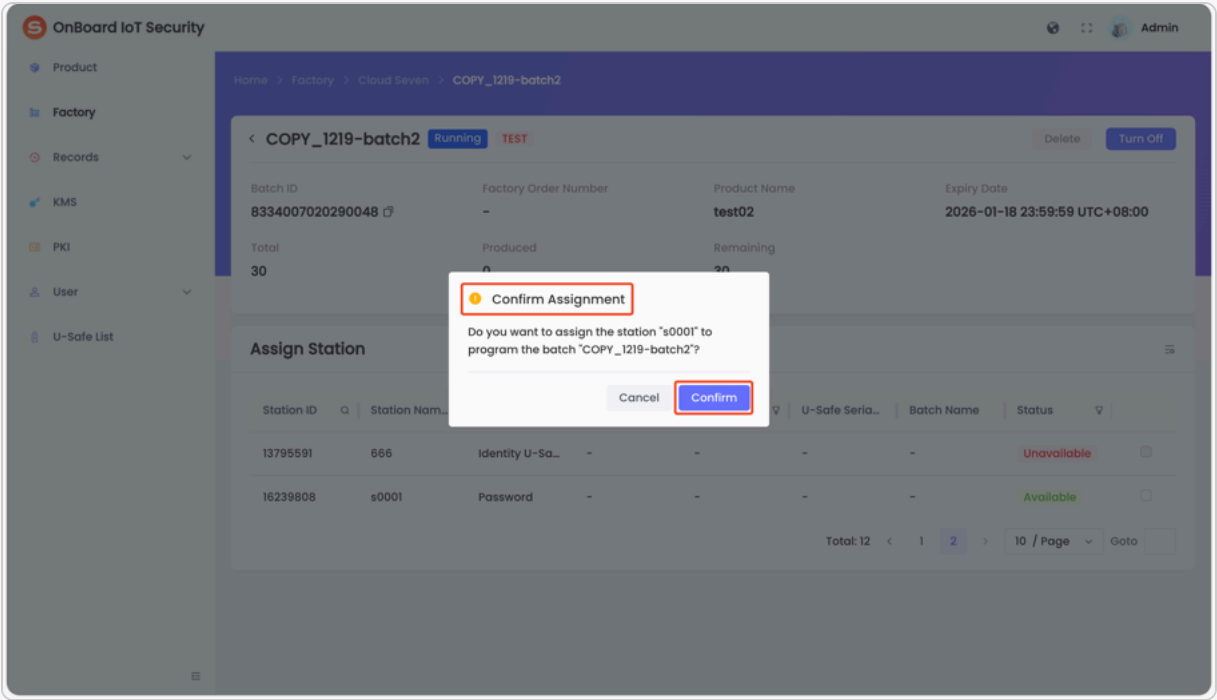
2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.

3. On the factory page, select the **Batch** tab and click the batch ID to navigate to the batch page.

4. In the **Assign Station** area of the batch page, confirm that the batch status is **Running**, and then select the programming station with the **Available** status for assignment.



5. In the **Confirm Assignment** window, click **Confirm** to complete the assignment of the programming station. After the assignment, the programming station status will change to **Busy**.

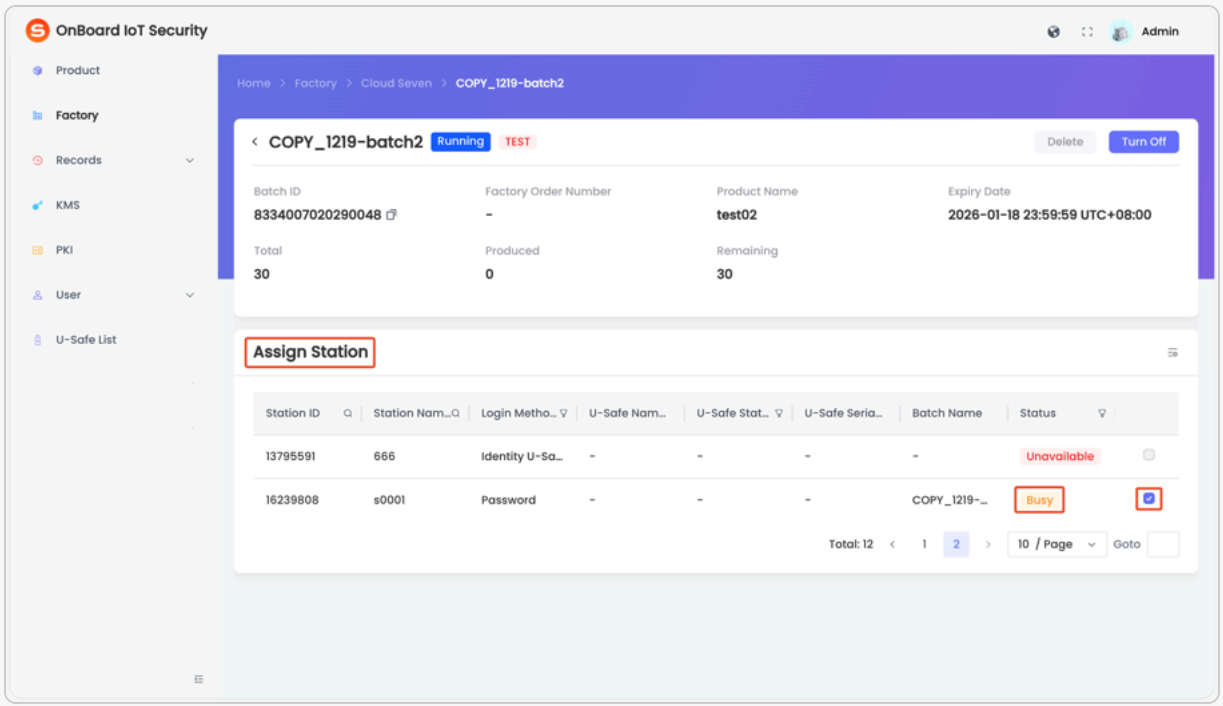


Unassign a programming station from a batch

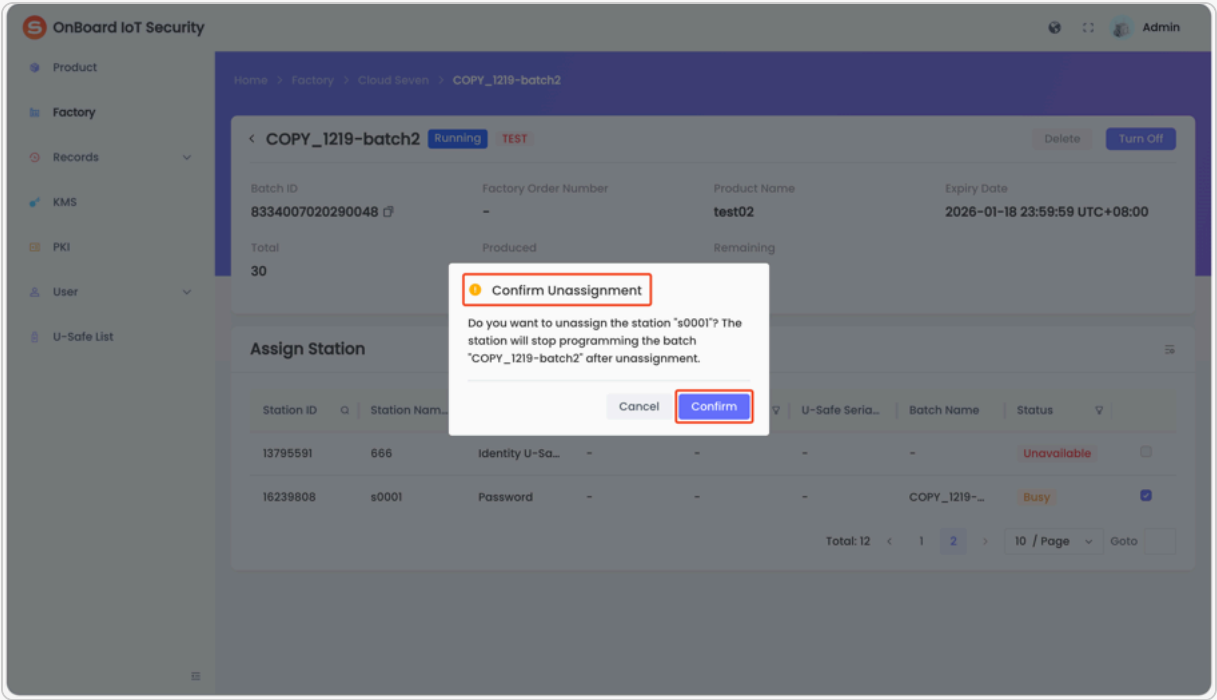
> **WARNING**
>
> After unassignment, the corresponding programming station can no longer be used for programming this batch.

1. Sign in to OBIS.
2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.
3. On the factory page, select the **Batch** tab and click the batch ID to navigate to the batch page.

4. In the **Assign Station** area of the batch page, locate the **Operation** column for the corresponding batch and uncheck the programming station with the **Busy** status.



5. In the **Confirm Unassignment** window, click **Confirm** to complete the unassignment of the programming station. After unassignment, the programming station status will change to **Available**.

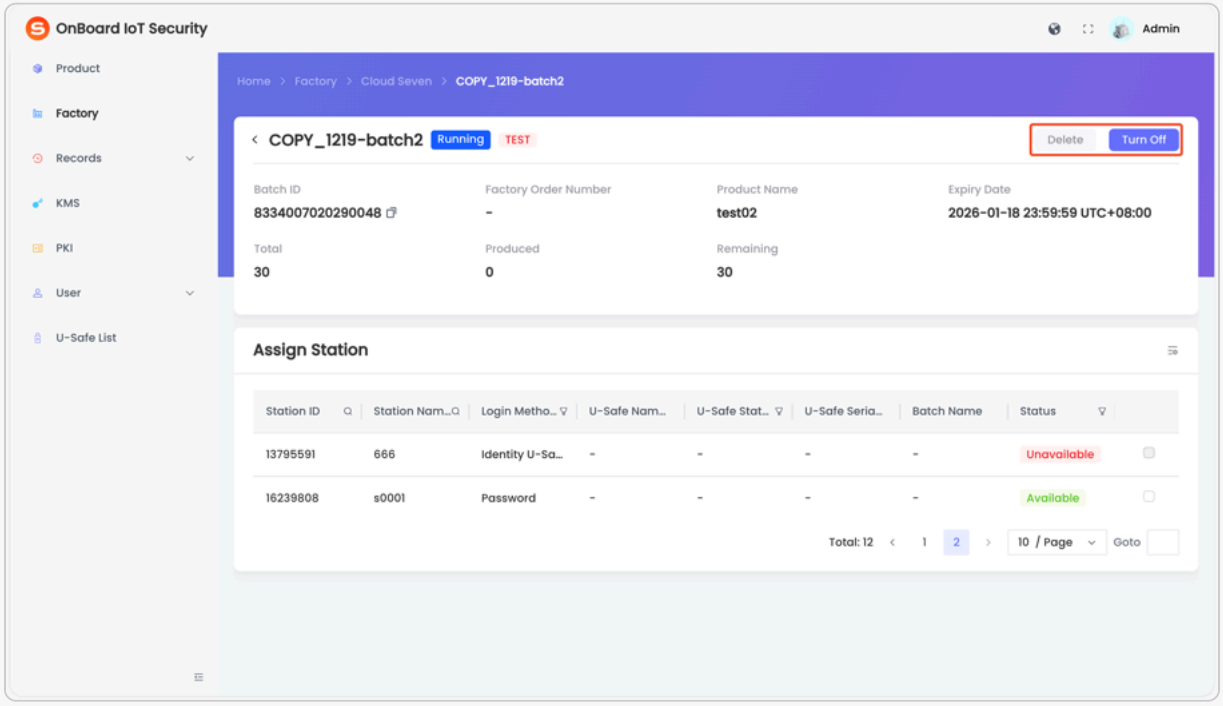


Turn off and delete a batch

> **INFO**
>
> After a batch is deleted, any programming stations assigned to that batch will be automatically unassigned.

1. Sign in to OBIS.
2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.
3. On the factory page, select the **Batch** tab and click the batch ID to navigate to the batch page.
4. On the batch page, click **Turn Off** or **Delete**.

> **TIP**
>
> If the **Delete** button appears grayed out, it is currently unavailable. Please click **Turn Off** to turn off the batch. Once the batch is in a non-running status, it can be deleted.



Version: v3.2.1

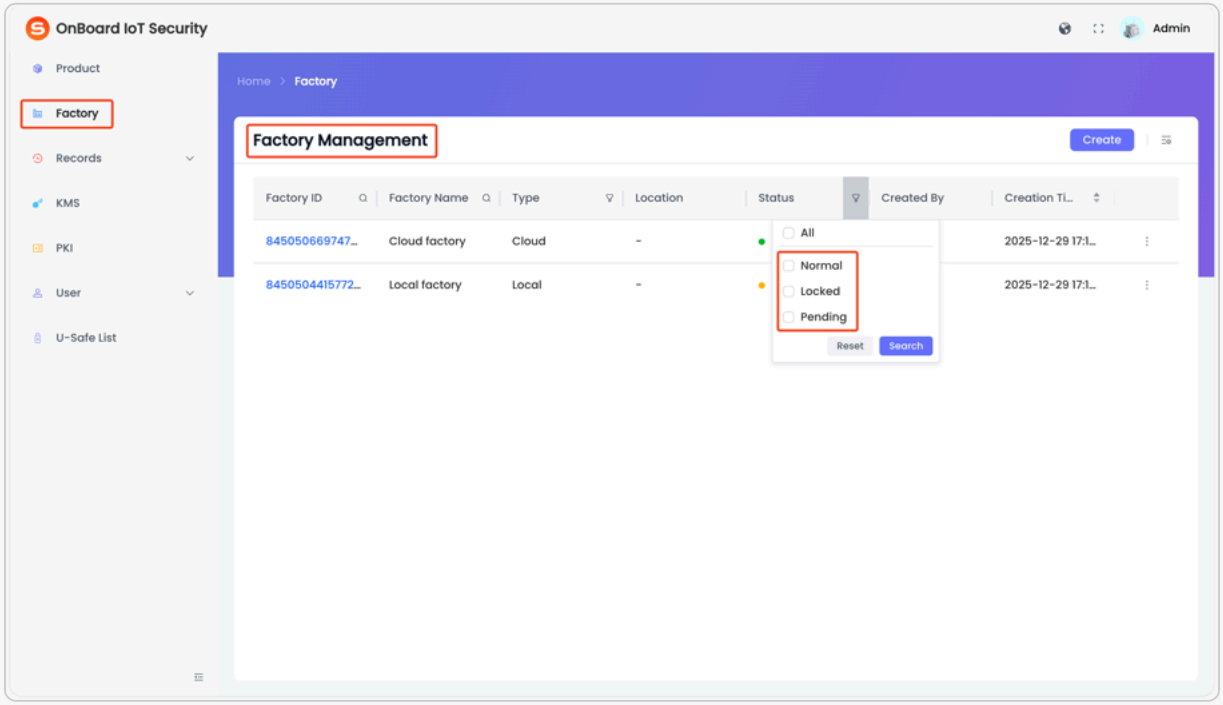
How to check factories in OnBoard™ IoT Security

This article will guide you on how to check factory information, factory status, programming station information, and batch information in OnBoard™ IoT Security (OBIS).

Check factory status

1. Sign in to OBIS.

2. In the left menu, click **Factory**. On the **Factory Management** page, the **Status** column will display three statuses: **Normal**, **Locked**, and **Pending**.

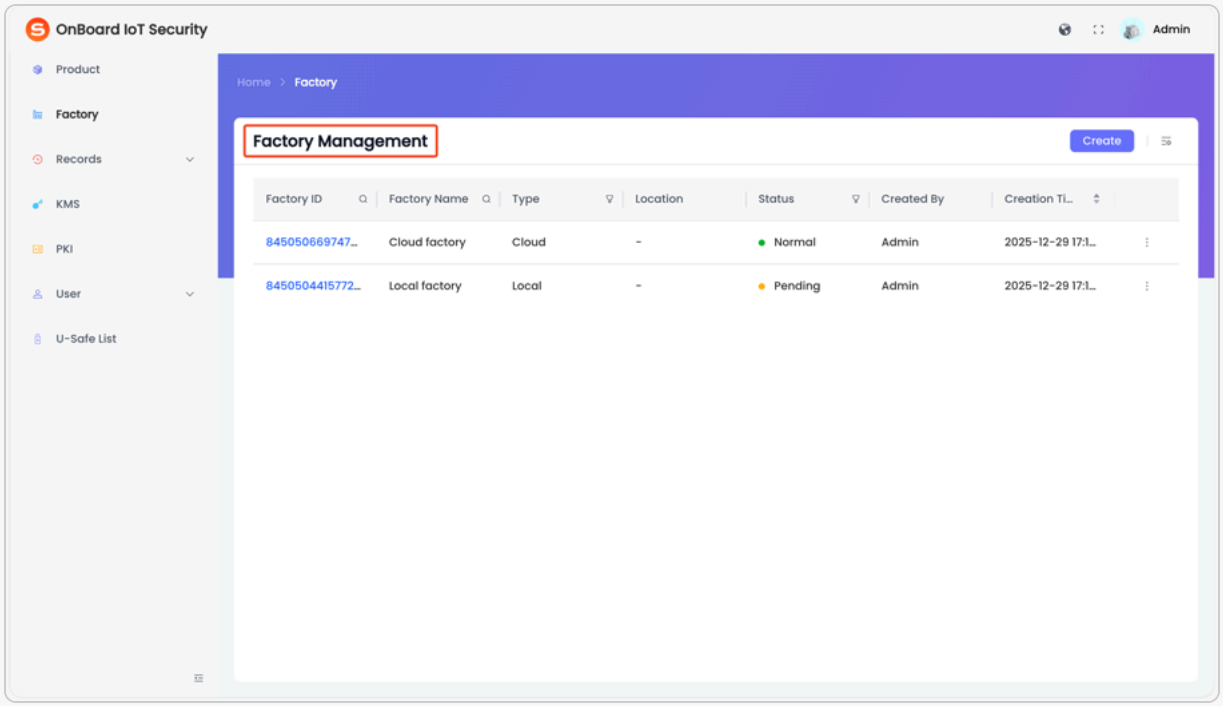


Description of factory status:

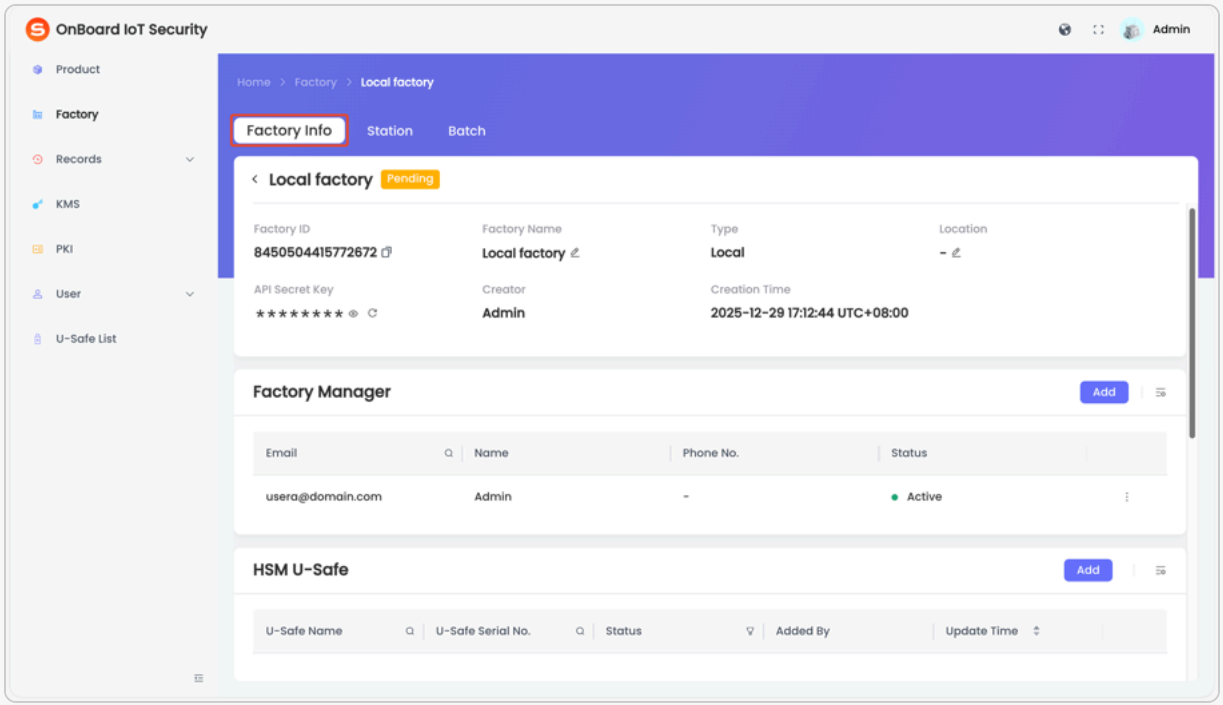
Status	Description
Normal	The factory can be selected when creating a batch. ◦ Local factory: Once an HSM U-Safe is added to the local factory, the status will change from **Pending** to **Normal**. ◦ Cloud factory: The initial status of a cloud factory is **Normal**.
Locked	The factory has been locked, and can't be used normally.
Pending	The factory status after the factory creation but without adding an HSM U-Safe, and the factory can't be selected when creating a batch. **Note**: This status applies only to local factories.

Check the factory information

1. Sign in to OBIS.
2. In the left menu, click **Factory**, and then you can check the factory list on the **Factory Management** page.



3. Select the corresponding factory ID to navigate to the factory page, and then you can check the factory-related information (such as **Factory ID**, **Factory Name**, and **API Secret Key**).

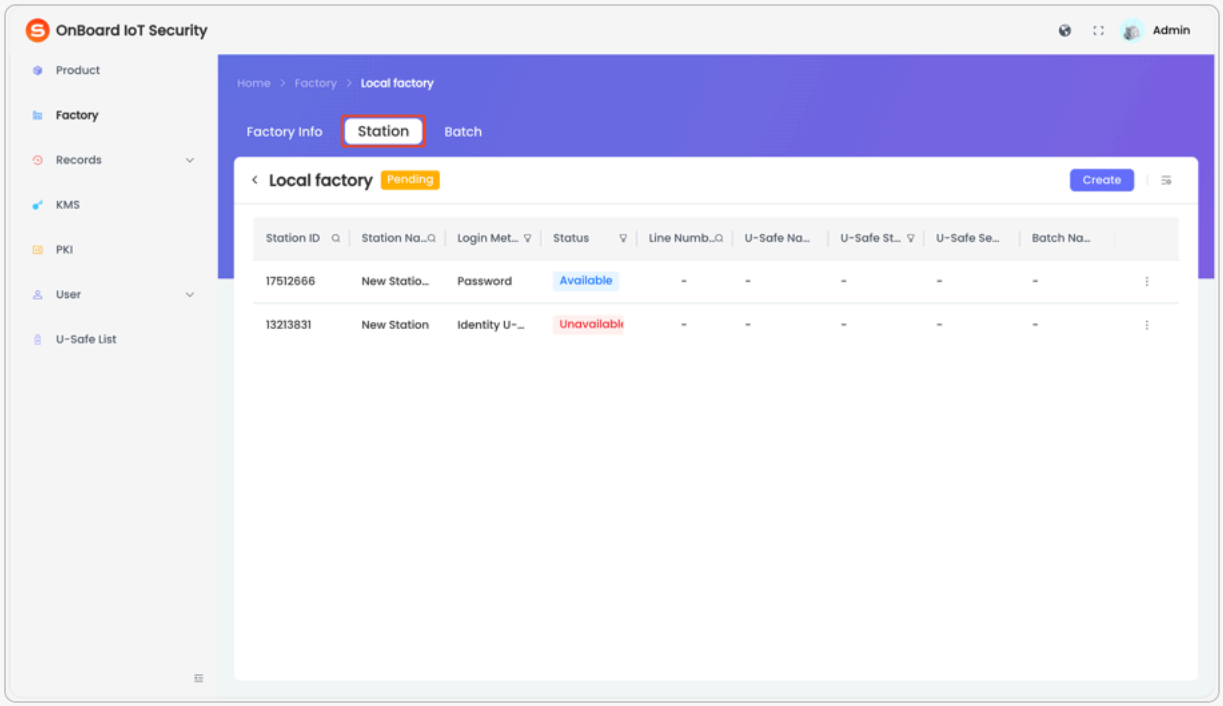


Check the programming station information

1. Sign in to OBIS.

2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.

3. On the factory page, select the **Station** tab, and then you can check the programming station information (such as **Station ID**, **Station Name**, and **Login Method**).



Check the batch information

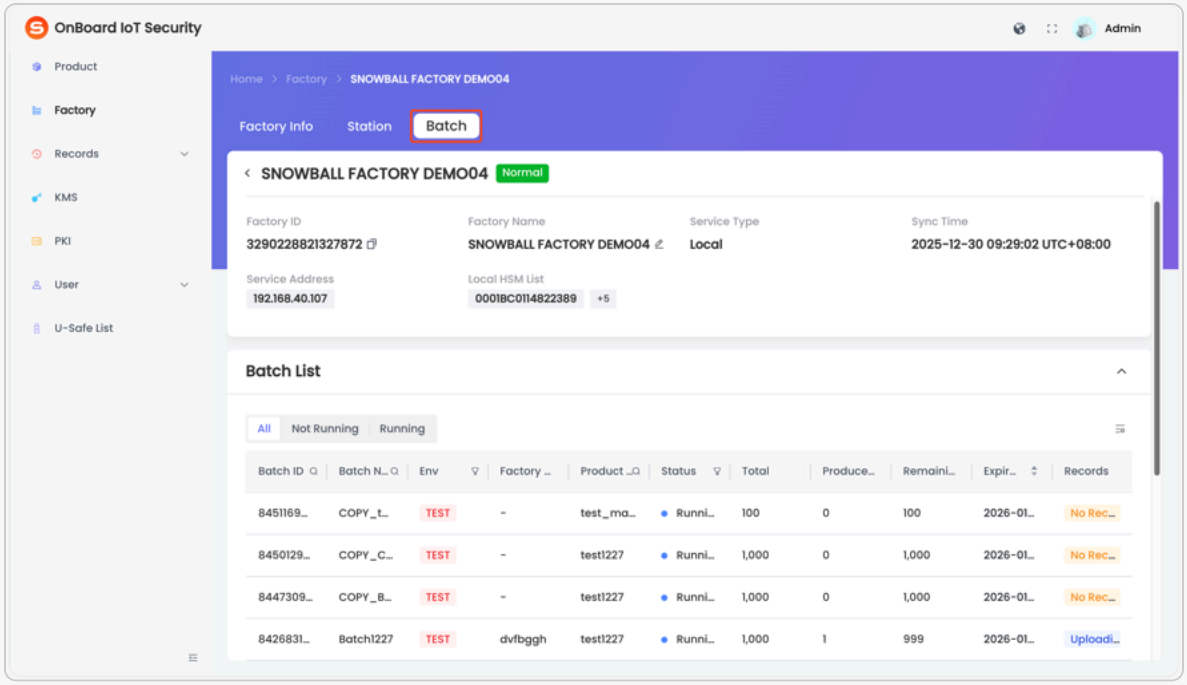
Local factory | Cloud factory

> **TIP**
>
> By checking the batch information, you can get the service addresses used to sign in to OnBoard Edge Service and the Programming Station client.
>
> After selecting the corresponding factory when creating a batch, the **Batch** tab on the factory page will synchronize the batch data.

1. Sign in to OBIS.
2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.
3. On the factory page, select the **Batch** tab, and then you can check the following information:
    - Service Info: such as **Factory ID**, **Factory Name**, **Service Type**, **Sync Time**, **Service Address**, and **Local HSM List**.

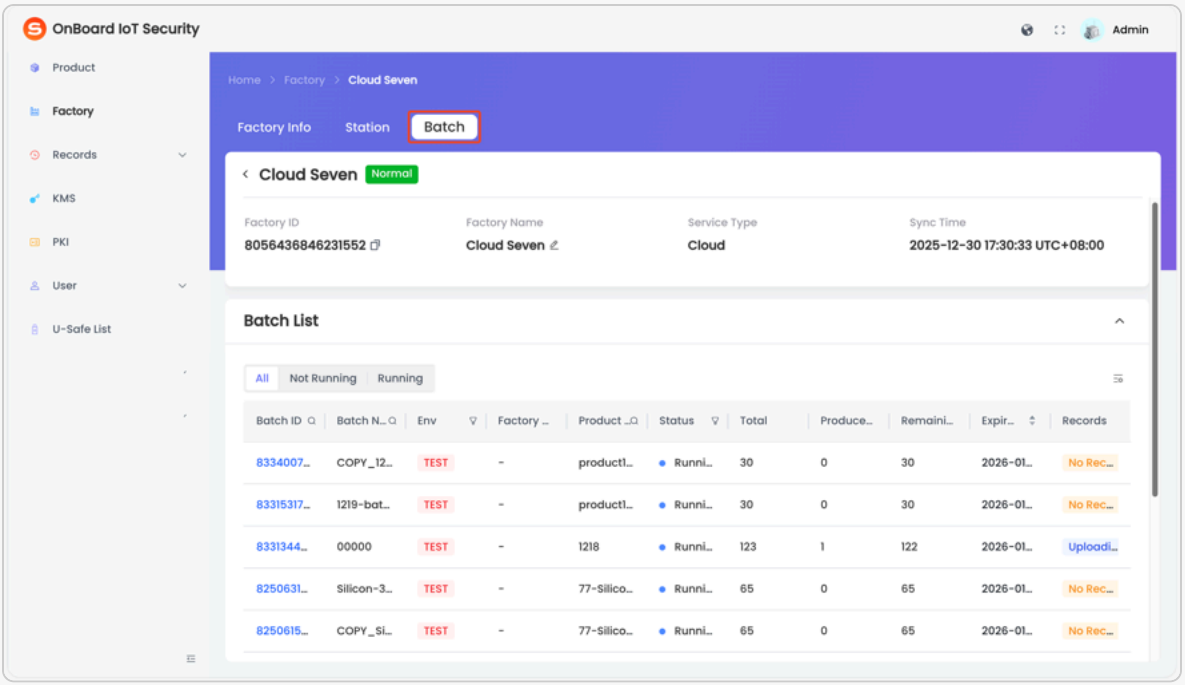
      > **TIP**
      >
      > After contacting Snowball for assistance in deploying OnBoard Edge Service, locate the **Service Address** on the **Batch** tab of the factory page. This service address will be used to sign in to OnBoard Edge Service and the Programming Station client.

- Batch List: such as **Batch ID**, **Batch Name**, and **Env**.



> **TIP**
>
> After selecting the corresponding factory when creating a batch, the **Batch** tab on the factory page will synchronize the batch data.

1. Sign in to OBIS.
2. In the left menu, click **Factory**, and then select the corresponding factory ID to navigate to the factory page.
3. On the factory page, select the **Batch** tab, and then you can check the following information:
    - Service Info: such as **Factory ID**, **Factory Name**, **Service Type**, and **Sync Time**.
    - Batch List: such as **Batch ID**, **Batch Name**, and **Env**.



4. Select the corresponding batch ID to navigate to the batch page, and then you can check the batch-related information.

Version: v3.2.1

How to check device history in OnBoard™ IoT Security

This article will guide you on how to check the device history in OnBoard™ IoT Security (OBIS), including the device parameters and programming status.

Prerequisites

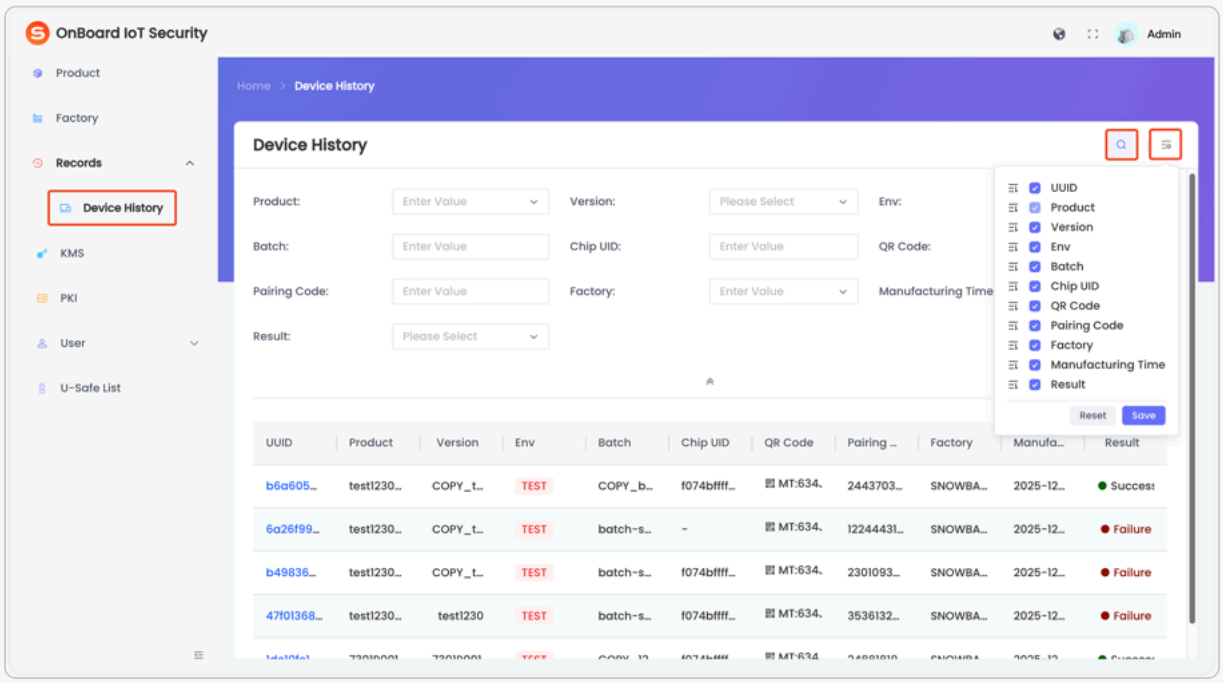
Before getting started, please ensure that the device has already been programmed.

Instructions

1. Sign in to OBIS.
2. In the left menu, click **Records** > **Device History**, and then you can check the device history on the **Device History** page.
3. Click the Search icon, set the search criteria and then click **Search**. The device history list will return all device histories that match the criteria.

> **TIP**
>
> Currently, device history can be searched by **Product**, **Version**, **Env**, **Batch**, **Chip UID**, **QR Code**, **Pairing Code**, **Factory**, **Manufacturing Time**, and **Result**.

Meanwhile, you can click the Column Options icon to customize which columns are shown and how they are ordered.



Description of device history information:

Information	Description
UUID	Universally Unique Identifier.
Product	The product name of the corresponding device.
Version	The version name of the corresponding device.

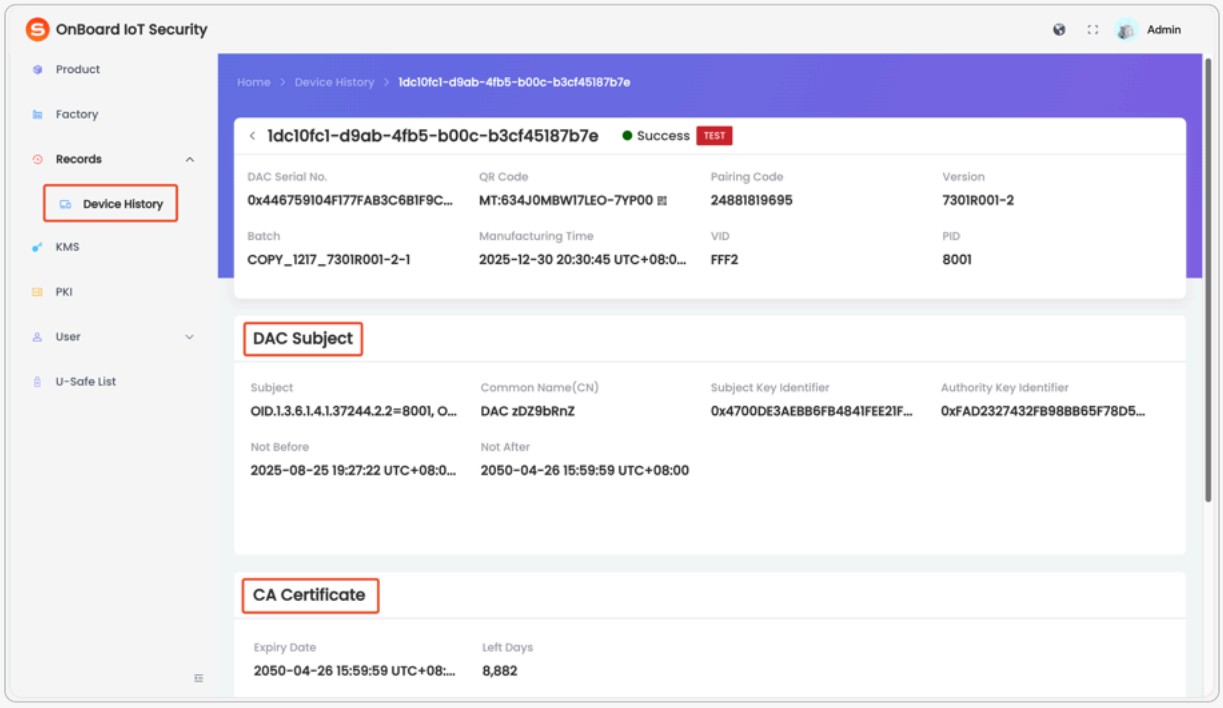
Information	Description
Env	The environment of the corresponding version, such as **TEST** and **PROD**.
Batch	The batch name of the corresponding device.
Chip UID	The Unique Identifier of the chip inside the device.
QR Code	The device's QR code, used for secure and fast commissioning.
Pairing Code	A temporary verification code manually entered by the user to complete device pairing.
Factory	The factory name of the corresponding device.
Manufacturing Time	The date and time when the device was manufactured, in the format **YYYY-MM-DD HH:MM:SS**.
Result	The device status, such as **Pending**, **Success**, and **Failure**.

4. Select the corresponding UUID to navigate to the device page, where you can check the device-related information (such as **DAC Serial No.**, **QR Code**, and **Pairing Code**).

TIP

Hover the mouse over the QR code icon to display the device QR code, which is used for secure and fast commissioning.

- The **DAC Subject** area displays the information of the DAC subject (such as **Not Before** and **Not After**).
- The **CA Certificate** area displays the encoding of the CA certificate. By parsing the encoding of the CA certificate, key certificate attributes such as the issuer information, validity period, and public key can be extracted.



Version: v3.2.1

Overview of KMS in OnBoard™ IoT Security

KMS (Key Management Service) in OBIS (OnBoard™ IoT Security) is the core module responsible for key security, which provides secure management and access control of keys. It supports creating, importing, integrating, and managing various types of keys, and leverages HSM (Hardware Security Module) to ensure high-security key generation, storage, and usage.

- Supports both symmetric and asymmetric keys, as well as various key algorithms.
- Supports constraining key usage, environment, validity period, and access permissions.
- Supports creating, importing, and integrating keys by using multiple key sources.
- Supports the use of KMS in multiple scenarios:
  - Resource encryption and decryption
  - Firmware signing and verification
  - Secure boot
  - Secure debug
  - HMAC (Hash-based message authentication code) authentication

The following table summarizes the core capabilities of KMS:

Capability	Description
Key Lifecycle Management	• Symmetric and asymmetric key lifecycle management
Key State & Storage	• Key state management (Active, Suspended, Revoked) • Hardware-backed secure storage and cryptographic execution
Enterprise BYOK (Key Import)	• OEM generates and imports keys into HSM • KMS governs usage via authorization policies
Cloud-Native BYOK (External KMS)	• Keys remain in customer's cloud KMS (e.g., AWS KMS) • KMS invokes operations through secure authenticated APIs

KMS does not mandate where keys must live. It mandates how cryptographic capabilities are governed, constrained, and audited.

Version: v3.2.1

How to create a key in OnBoard™ IoT Security

This article will guide you on how to create a symmetric and asymmetric key in OnBoard™ IoT Security (OBIS).

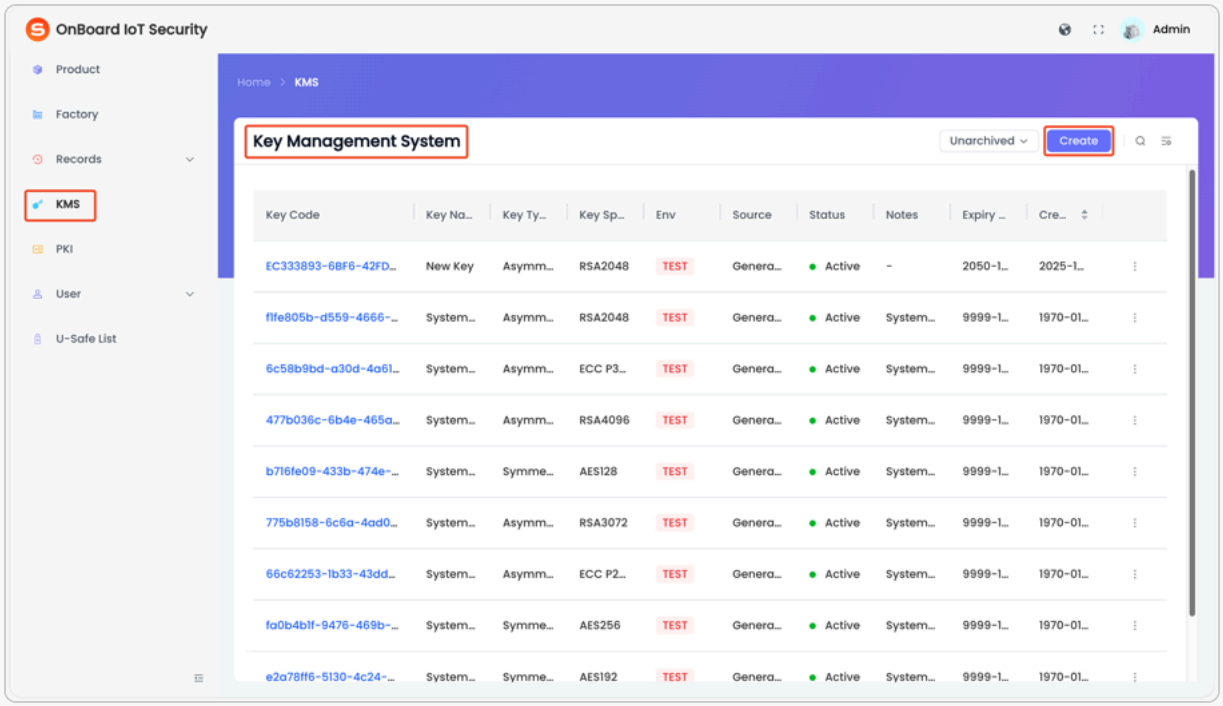
Prerequisites

Before getting started, please ensure that you have OBIS role permissions as a System Manager or Resource Manager.

Instructions

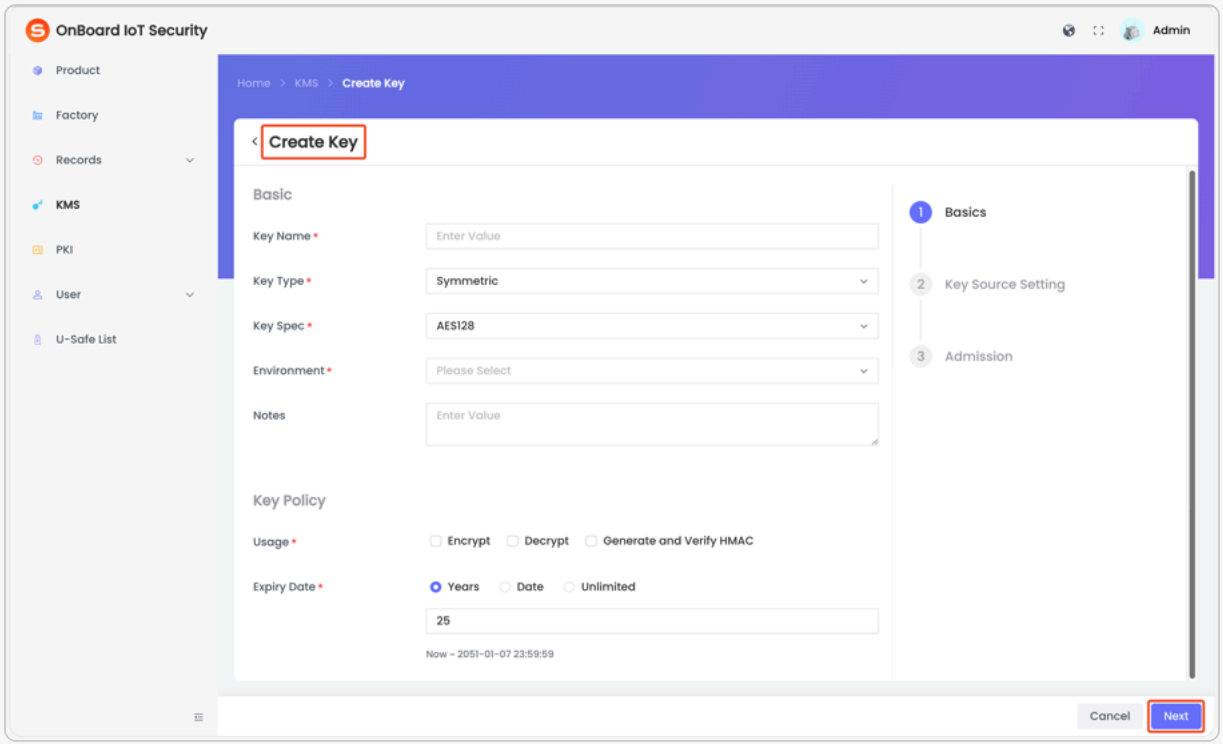
> **TIP**
>
> After a key is created, it can be selected when configuring Key for a product.

1. Sign in to OBIS.
2. In the left menu, click **KMS**, and then click **Create** on the **Key Management System** page.



Step one: Configure the basic information

1. On the **Create Key** page, select and enter the key basic information.



Description of key basic information:

Information	Mandatory	Description
Key Name	Yes	The key name must be unique.
Key Type	Yes	Two key types can be selected: **Symmetric** and **Asymmetric**.
Key Spec	Yes	Select encryption algorithms for symmetric and asymmetric keys: ◦ Symmetric key: **AES128**, **AES192**, or **AES256** ◦ Asymmetric key: **RSA2048**, **RSA3072**, **RSA4096**, **ECC P256**, or **ECC P384**

Information	Mandatory	Description
Environment	Yes	Two environments can be selected: **TEST** and **PROD**.
Notes	No	Provide additional information related to the key, such as special requirements and important notes.
Usage	Yes	Limit the operations the key is allowed to perform (multiple selections allowed): ◦ **Sign**: Allows the use of an asymmetric key to generate digital signatures. ◦ **Verify**: Allow the use of an asymmetric key to verify digital signatures. ◦ **Encrypt**: Allow the use of a symmetric or asymmetric key to encrypt data. ◦ **Decrypt**: Allow the use of a symmetric or asymmetric key to decrypt data. ◦ **Generate and Verify HMAC**: Allows the use of a symmetric key to generate and verify message authentication codes to ensure data integrity. **Note**: For asymmetric keys, once **Sign** or **Decrypt** is selected, the Import Public Key option in the **Key Source** area will be unavailable.
Expiry Date	Yes	Set the expiry date of the key.

2. After verifying that the basic information is correct, click **Next** to navigate to the **Key Source Setting** page.

Step two: Configure the key source

> **TIP**
>
> On the **Key Source Setting** page, you can click **Skip** to temporarily bypass key source configuration and proceed directly to configure the key admission. Before the key source configuration is completed, the key will remain in the **Pending** status and can't be used.
>
> If you select to skip the key source configuration, you can configure it later by selecting the corresponding key on the **Key Management System** page. On the key page, click the link in the prompt to complete **these final steps**.

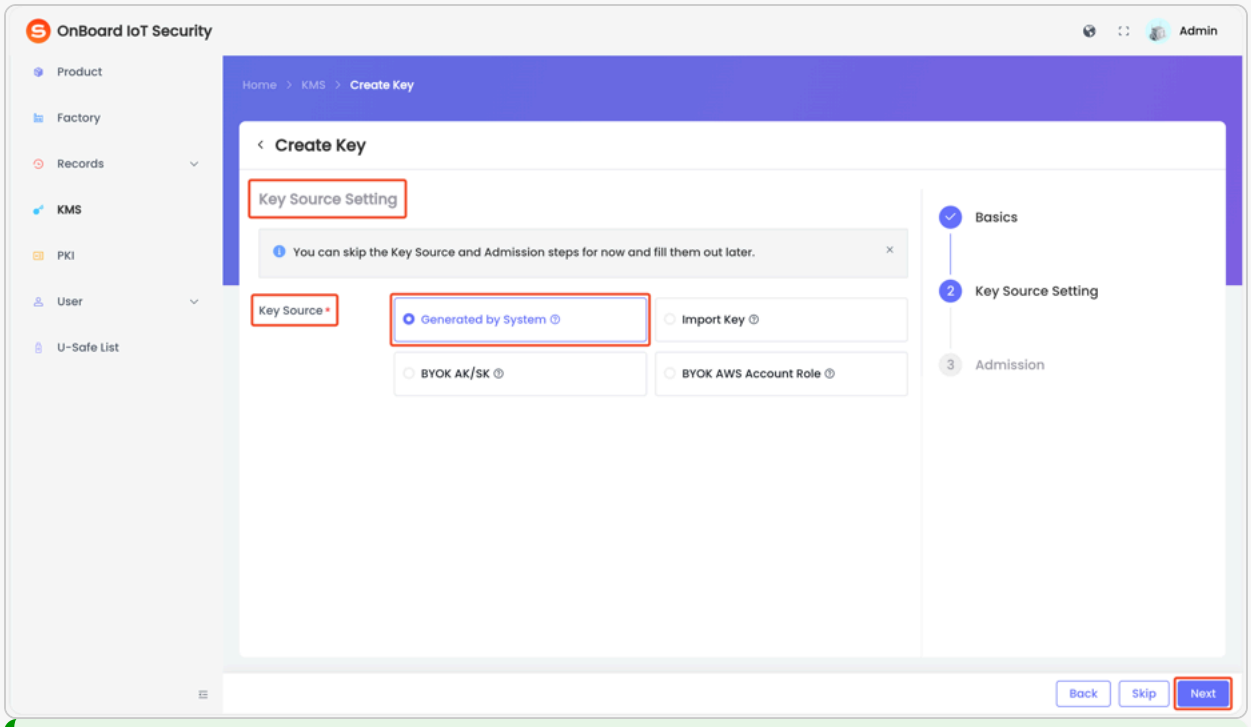
Symmetric key

Generated by System | Import Key | BYOK AK/SK

BYOK AWS Account Role

> **TIP**
>
> The system securely generates a random key pair inside a Hardware Security Module (HSM).

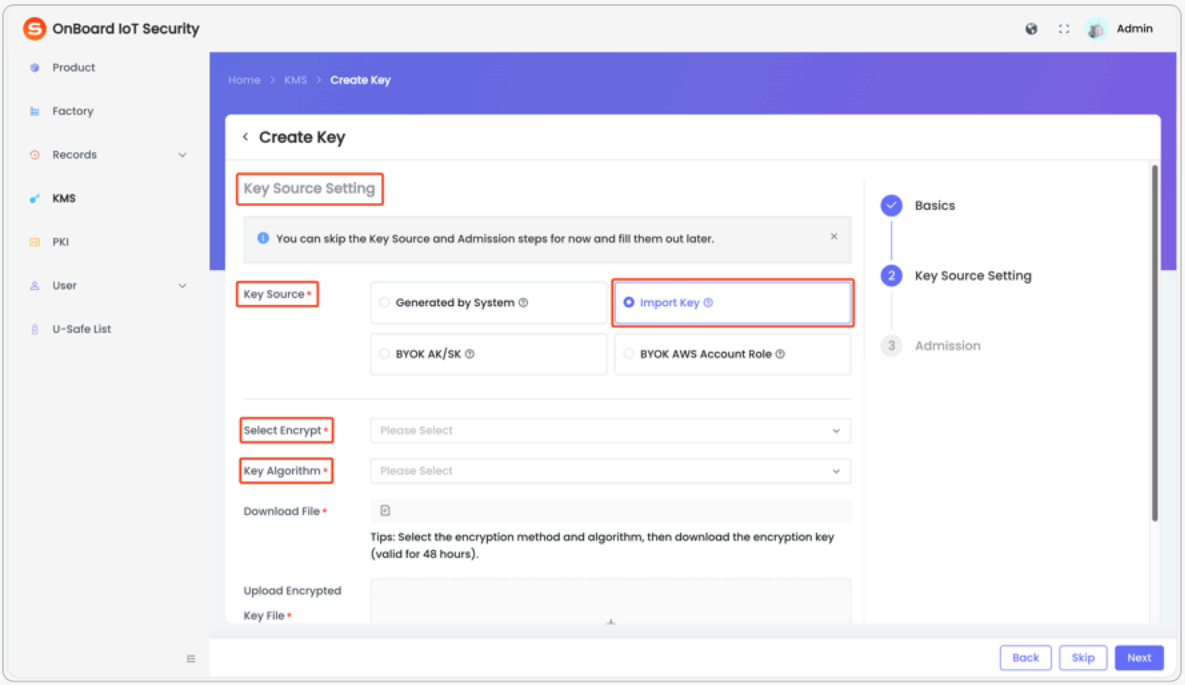
On the **Key Source Setting** page, select **Generated by System** in the **Key Source** area, and then click **Next** to navigate to the **Admission** page.



TIP

Manually upload an existing symmetric key generated outside the system.

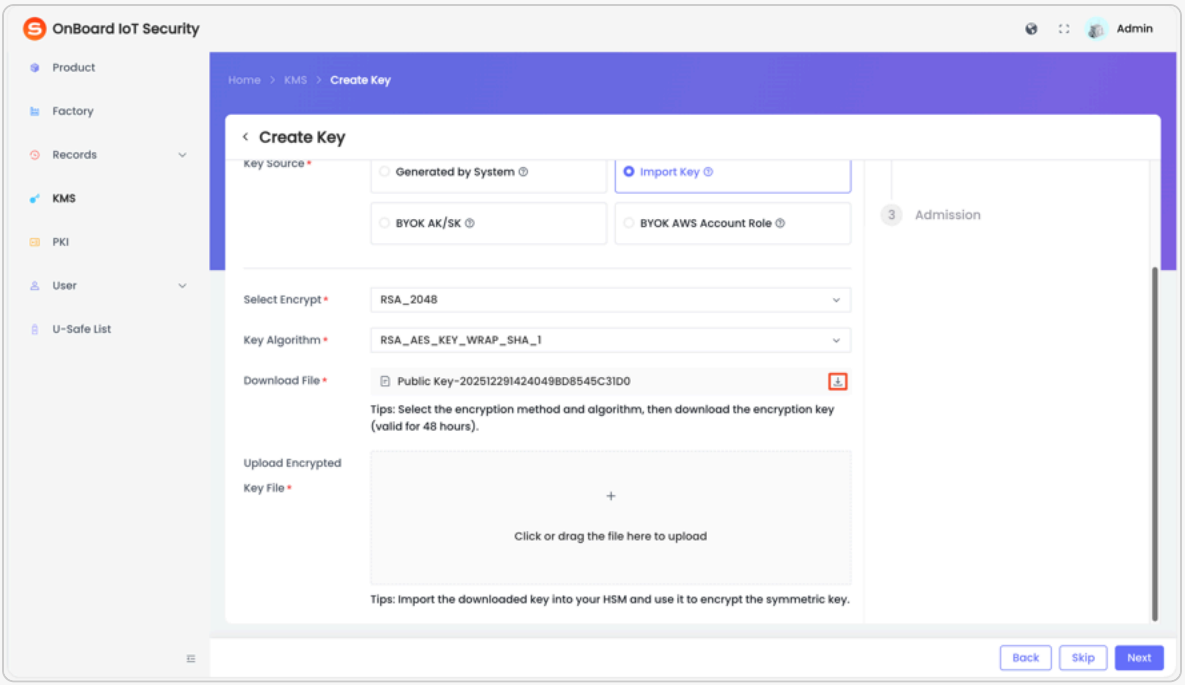
1. On the **Key Source Setting** page, select **Import Key** in the **Key Source** area. Select the encryption method and key algorithm.



Description of Import Key information:

Information	Mandatory	Description
Select Encrypt	Yes	Select the encryption algorithm. Currently, only **RSA_2048** is supported, which provides a baseline level of security and is suitable for scenarios such as key encryption and digital signatures.
Key Algorithm	Yes	Select the key algorithm: ◦ **RSA_AES_KEY_WRAP_SHA_1** ◦ **RSA_AES_KEY_WRAP_SHA_256**

2. Click the Download icon to download the encryption key file.



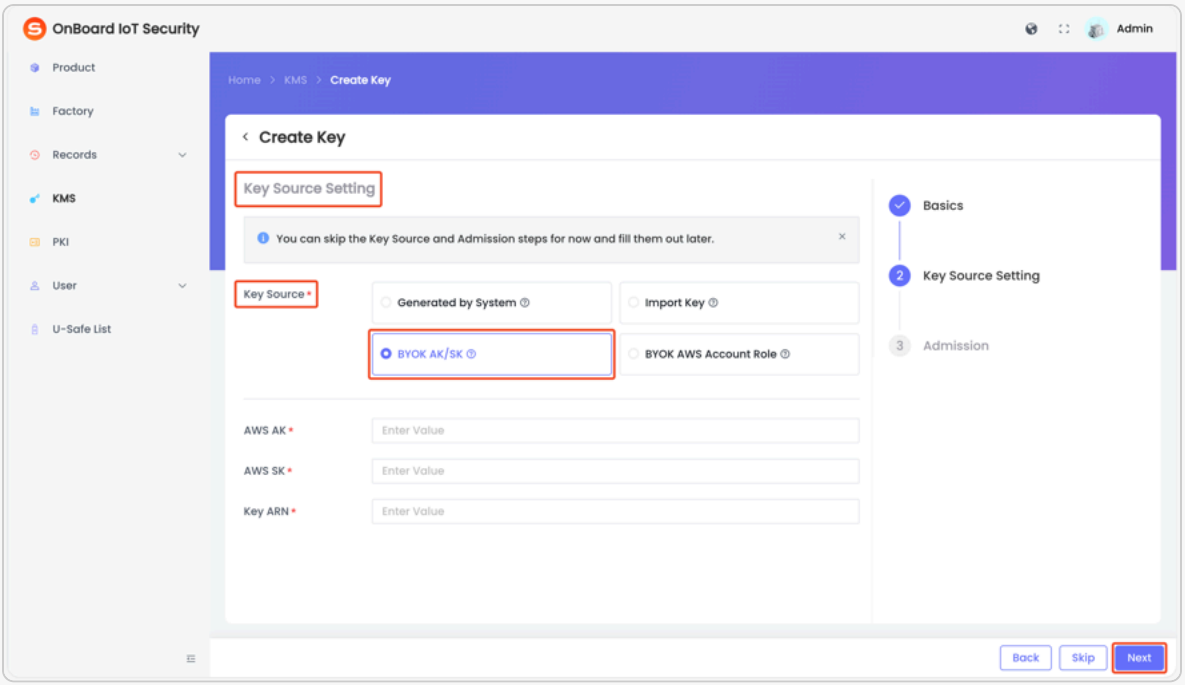
3. Import the downloaded encryption key into your HSM and use it to encrypt the symmetric key.

4. After the symmetric key is encrypted, select and upload the encrypted key file in **.json** format in the **Upload Encrypted Key File** area.

5. After verifying that the information is correct, click **Next** to navigate to the **Admission** page.

> **TIP**
>
> Access external keys from third-party KMS using Access Key and Secret Key credentials.

1. On the **Key Source Setting** page, select **BYOK AK/SK** in the **Key Source** area, and then enter the key-related information.



Description of AWS AK/SK information:

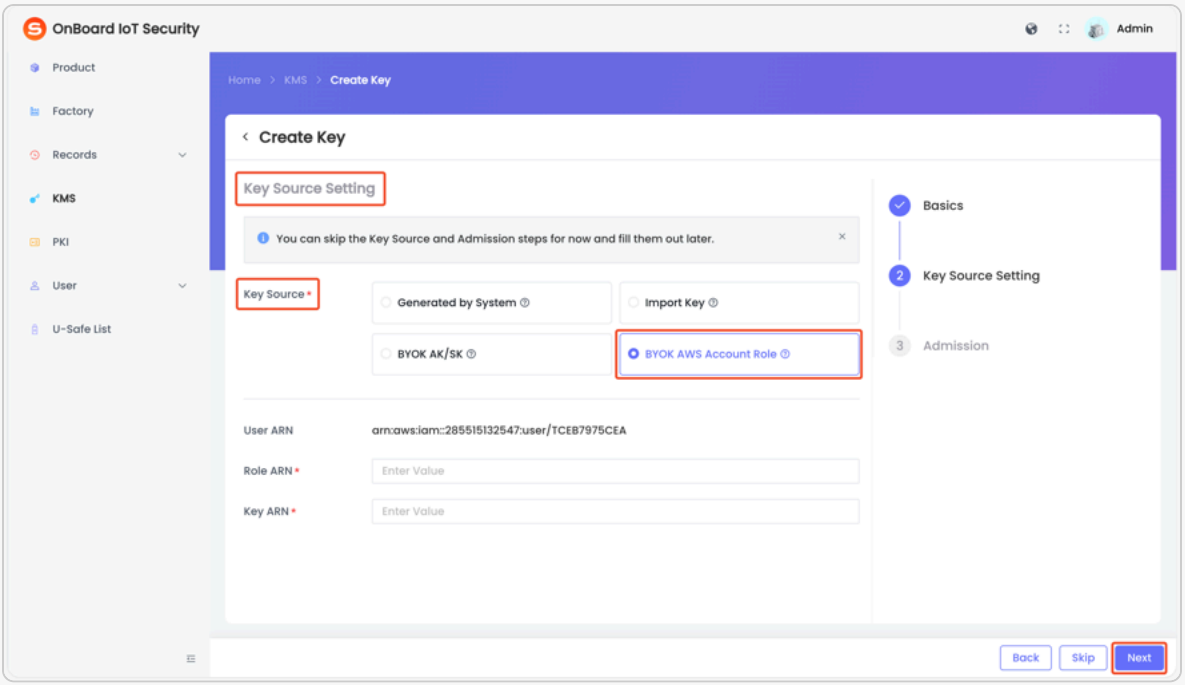
Information	Mandatory	Description
AWS AK	Yes	AK (Access Key ID): ◦ Identity identifier: Uniquely identifies a user in AWS IAM (Identity and Access Management), similar to a username. ◦ Public attribute: Can be safely shared (for example, configured in code or the CLI), but must be used together with the SK to complete authentication.

Information	Mandatory	Description
AWS SK	Yes	SK (Secret Access Key): ◦ Confidential credential: Used to generate request signatures and verify identity, similar to a password. ◦ Strict confidentiality: Disclosure may result in full account compromise and should be protected with the same care as a password.
Key ARN	Yes	ARN (Amazon Resource Name): uniquely identifies an encryption key in AWS KMS, used for data encryption and decryption.

2. After verifying that the information is correct, click **Next** to navigate to the **Admission** page.

> **TIP**
>
> Connect to AWS KMS using IAM Role-based access—secure and ideal for cloud-native deployments.

1. On the **Key Source Setting** page, select **BYOK AWS Account Role** in the **Key Source** area, and then enter the key-related information.



Description of AWS Account Role information:

Information	Mandatory	Description
User ARN	No	Uniquely identifies a user in AWS IAM (such as an administrator or developer), used for permission assignment and resource access control.
Role ARN	Yes	Uniquely identifies a role in AWS IAM, used for temporary permission assignment (such as cross-account access for EC2 instances or Lambda functions).

Information	Mandatory	Description
Key ARN	Yes	Uniquely identifies an encryption key in AWS KMS, used for data encryption and decryption.

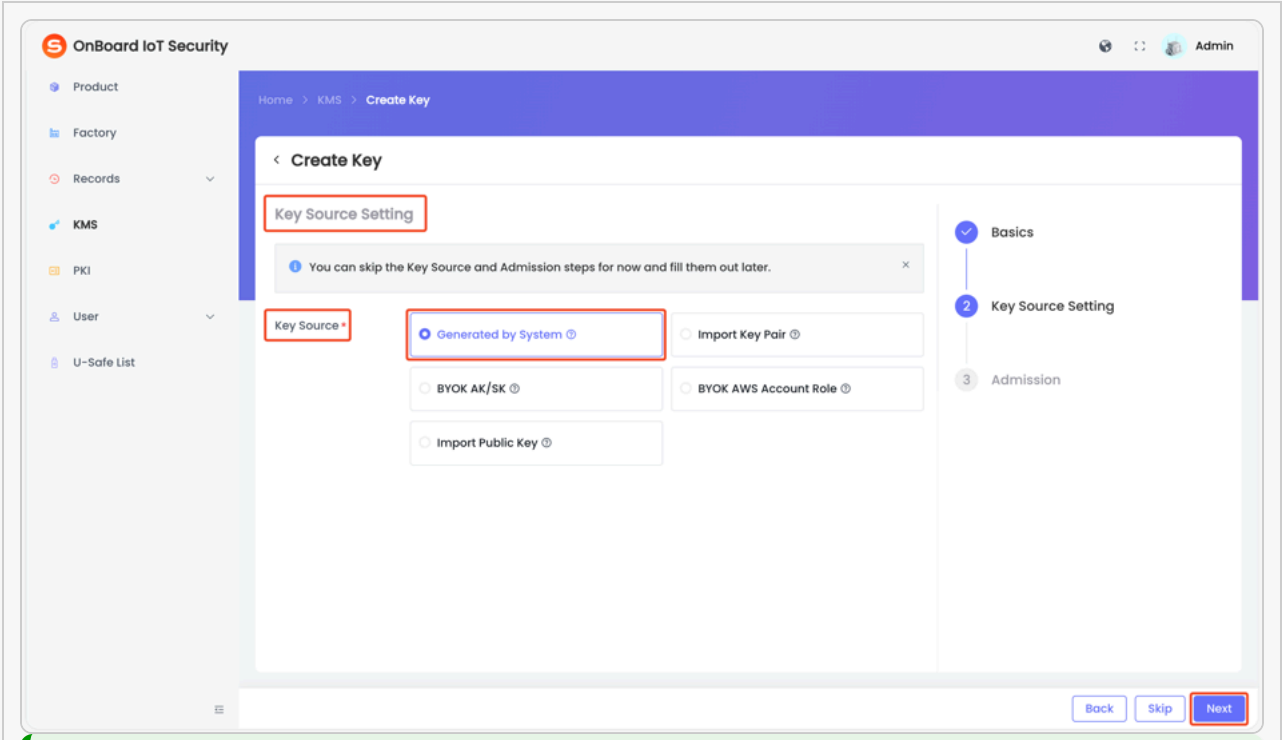
2. After verifying that the information is correct, click **Next** to navigate to the **Admission** page.

Asymmetric key

Generated by System | Import Key Pair | BYOK AK/SK | BYOK AWS Account Role | Import Public Key

> **TIP**
>
> The system securely generates a random key pair inside a Hardware Security Module (HSM).

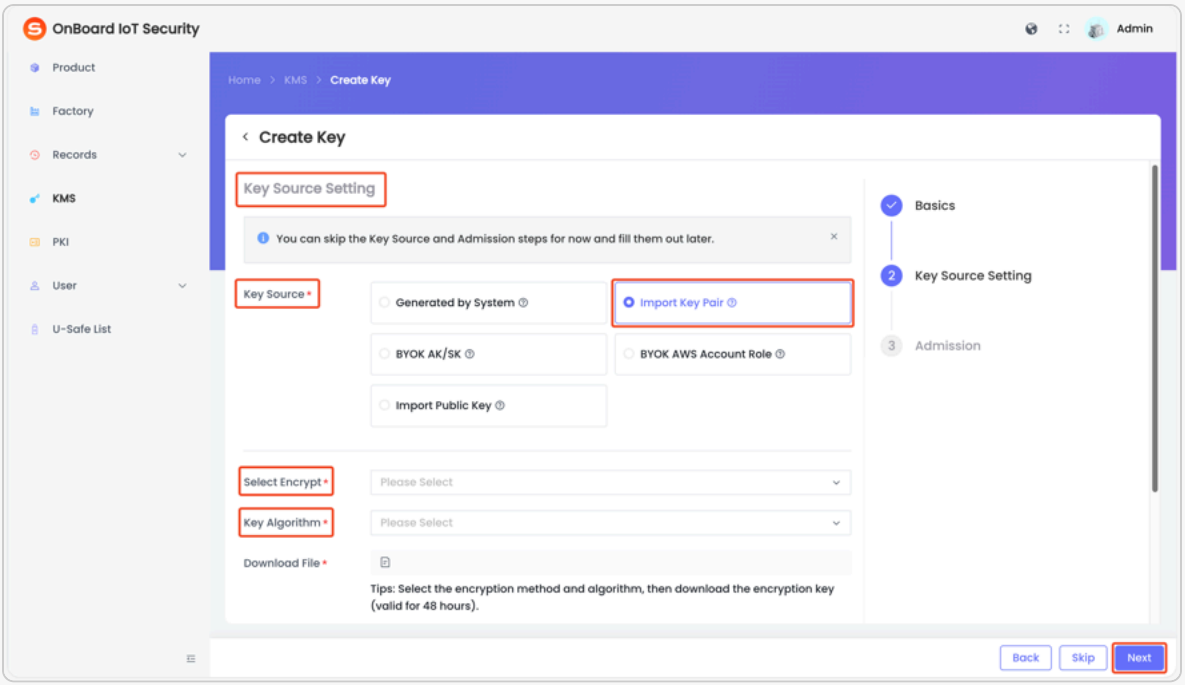
On the **Key Source Setting** page, select **Generated by System** in the **Key Source** area, and then click **Next** to navigate to the **Admission** page.



TIP

Manually upload an existing asymmetric key pair generated outside the system.

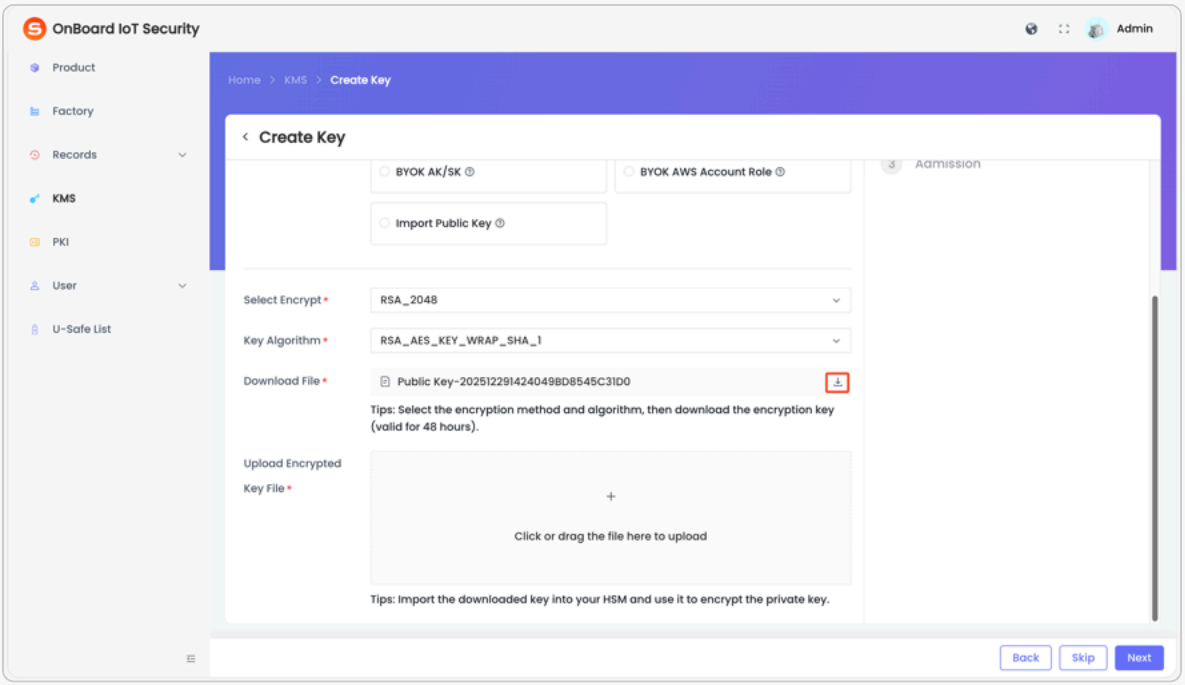
1. On the **Key Source Setting** page, select **Import Key Pair** in the **Key Source** area. Select the encryption method and key algorithm.



Description of Import Key Pair information:

Information	Mandatory	Description
Select Encrypt	Yes	Select the encryption algorithm. Currently, only **RSA_2048** is supported, which provides basic security strength and is suitable for scenarios such as key encryption and digital signatures.
Key Algorithm	Yes	Select the key algorithm: ◦ **RSA_AES_KEY_WRAP_SHA_1** ◦ **RSA_AES_KEY_WRAP_SHA_256**

2. Click the Download icon to download the encryption key file.



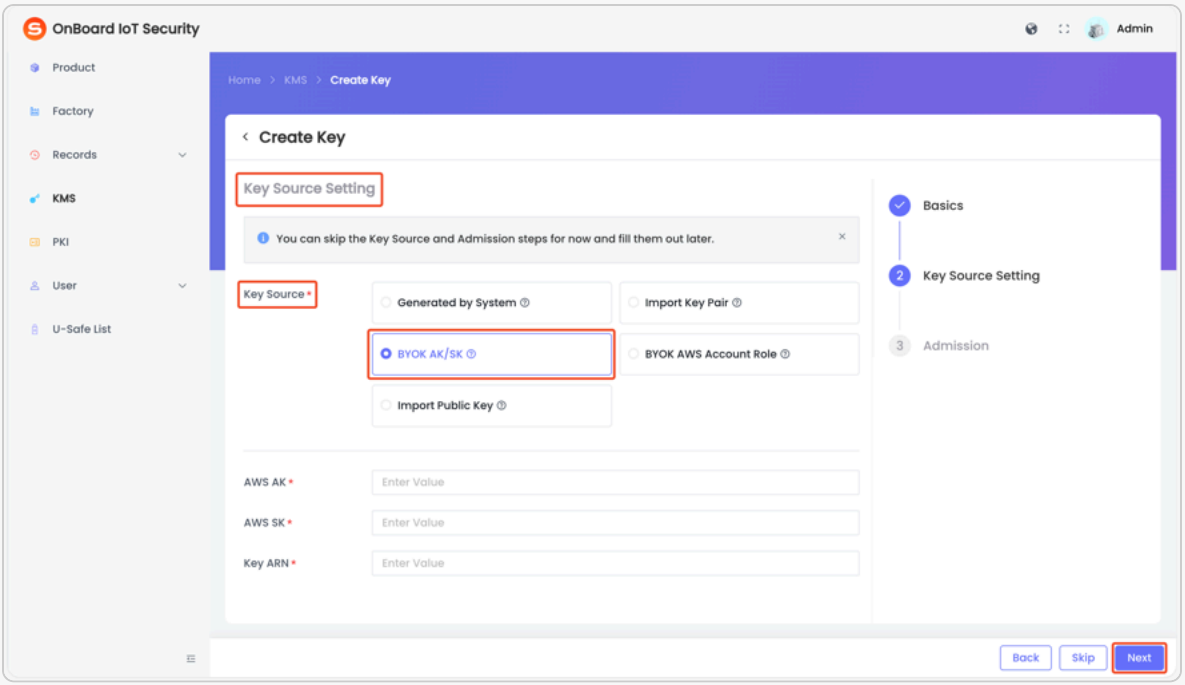
3. Import the downloaded encryption key into your HSM and use it to encrypt the private key.

4. After private key is encrypted, select and upload the encrypted key file in **.json** format in the **Upload Encrypted Key File** area.

5. After verifying that the information is correct, click **Next** to navigate to the **Admission** page.

> **TIP**
>
> Access external keys from third-party KMS using Access Key and Secret Key credentials.

1. On the **Key Source Setting** page, select **BYOK AK/SK** in the **Key Source** area, and then enter the key-related information.



Description of AWS AK/SK information:

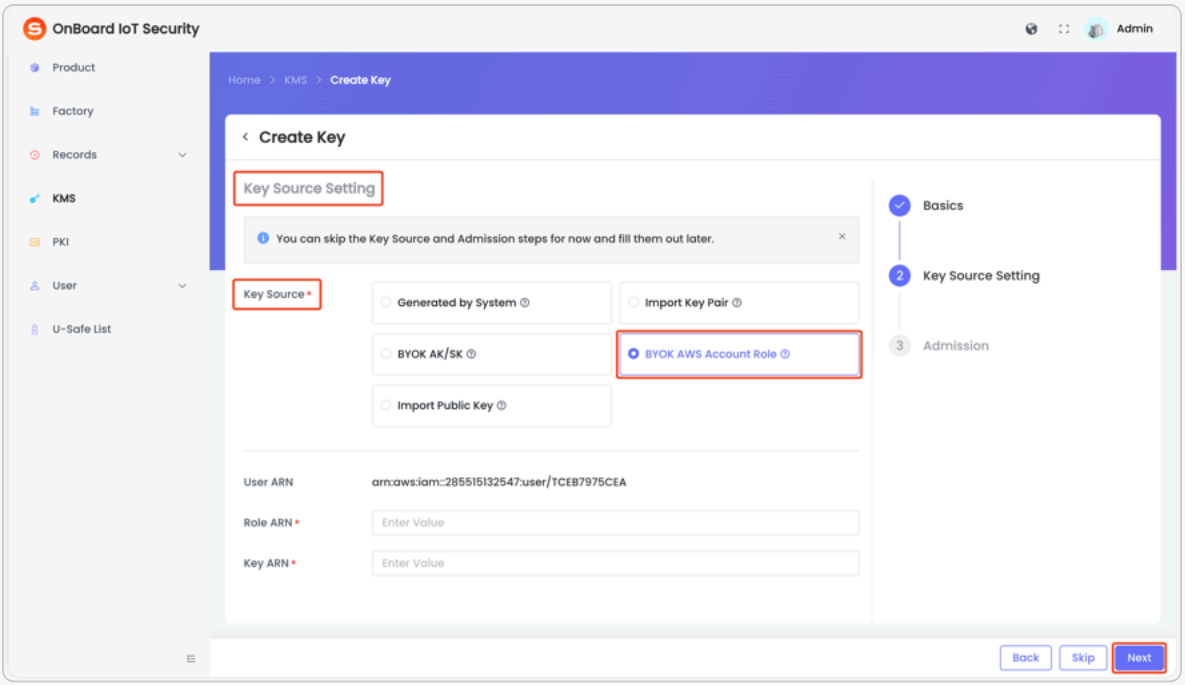
Information	Mandatory	Description
AWS AK	Yes	AK (Access Key ID): ◦ Identity identifier: Uniquely identifies a user in AWS IAM (Identity and Access Management), similar to a username. ◦ Public attribute: Can be safely shared (for example, configured in code or the CLI), but must be used together with the SK to complete authentication.

Information	Mandatory	Description
AWS SK	Yes	SK (Secret Access Key): ◦ Confidential credential: Used to generate request signatures and verify identity, similar to a password. ◦ Strict confidentiality: Disclosure may result in full account compromise and should be protected with the same care as a password.
Key ARN	Yes	ARN (Amazon Resource Name): uniquely identifies an encryption key in AWS KMS, used for data encryption and decryption.

2. After verifying that the information is correct, click **Next** to navigate to the **Admission** page.

> **TIP**
>
> Connect to AWS KMS using IAM Role-based access—secure and ideal for cloud-native deployments.

1. On the **Key Source Setting** page, select **BYOK AWS Account Role** in the **Key Source** area, and then enter the key-related information.



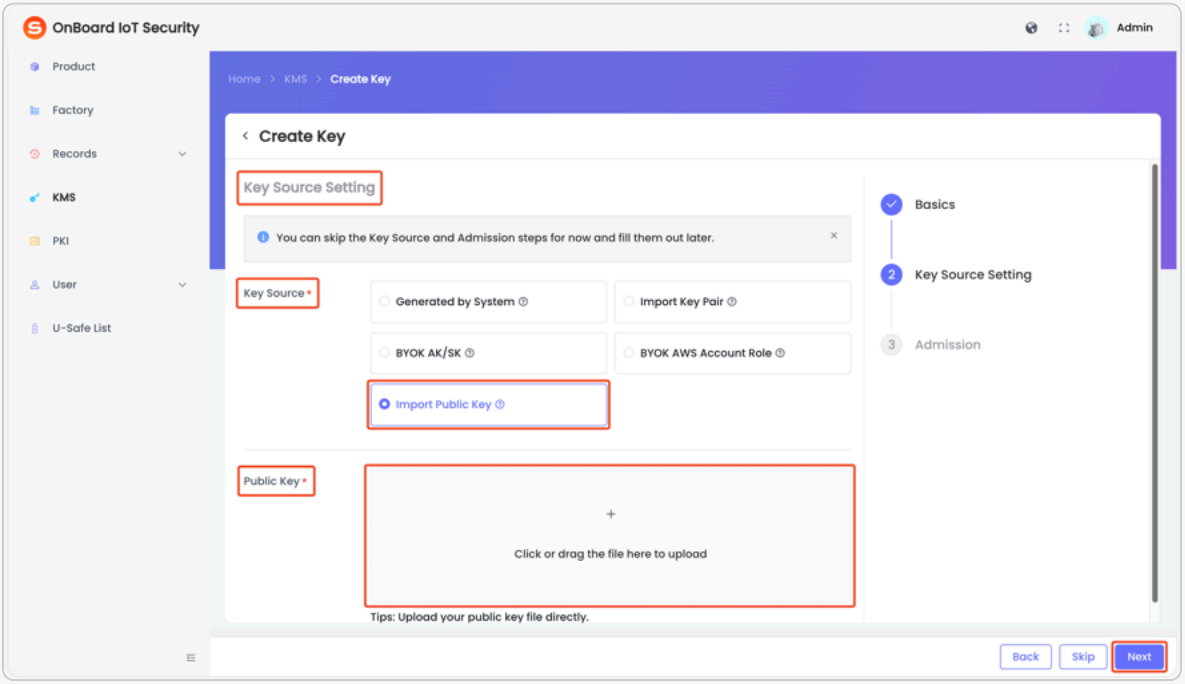
Description of AWS Account Role information:

Information	Mandatory	Description
User ARN	No	Uniquely identifies a user in AWS IAM (such as an administrator or developer), used for permission assignment and resource access control.
Role ARN	Yes	Uniquely identifies a role in AWS IAM, used for temporary permission assignment (such as cross-account access for EC2 instances or Lambda functions).
Key ARN	Yes	Uniquely identifies an encryption key in AWS KMS, used for data encryption and decryption.

2. After verifying that the information is correct, click **Next** to navigate to the **Admission** page.

> **TIP**
>
> Import only a public key for verifying signatures or device identities—no private key required.

1. On the **Key Source Setting** page, select **Import Public Key** in the **Key Source** area. In the **Public Key** area, upload a public key file in **.json** format.

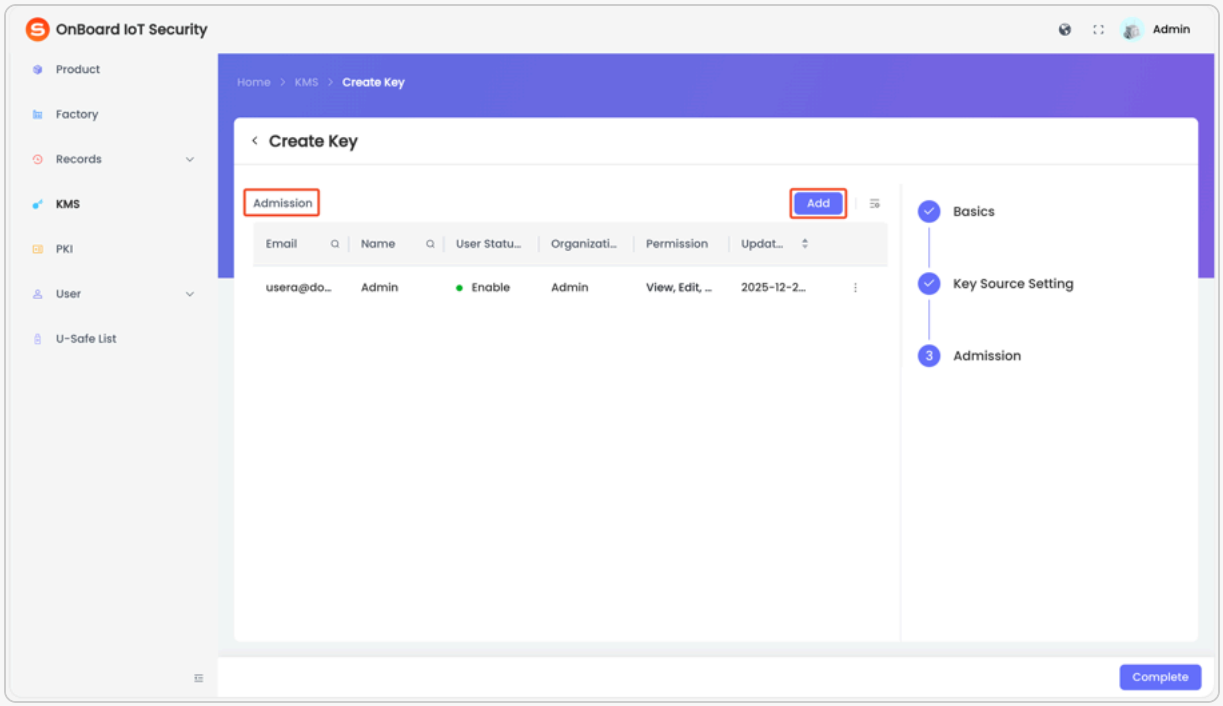


2. After verifying that the file is correct, click **Next** to navigate to the **Admission** page.

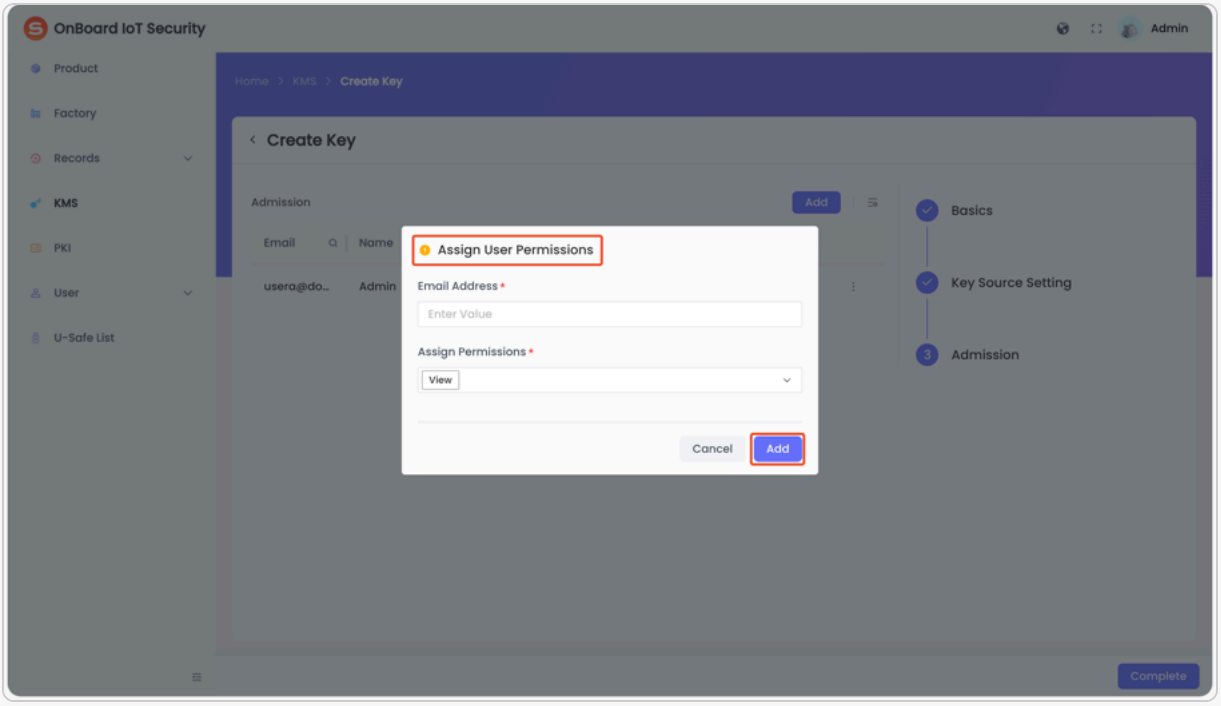
Step three: Configure the admission

After the key source configuration is completed, you can adjust and set user permissions (View, Edit, Use, Archive, Lock, Grant, and Delete) for the key by following these steps:

1. On the **Admission** page, click **Add**.



2. In the **Assign User Permissions** window, enter the email address and assign the corresponding permissions, and then click **Add**.



3. Click **Complete**. Once the key is successfully created, it can be checked on the **Key Management System** page.

Version: v3.2.1

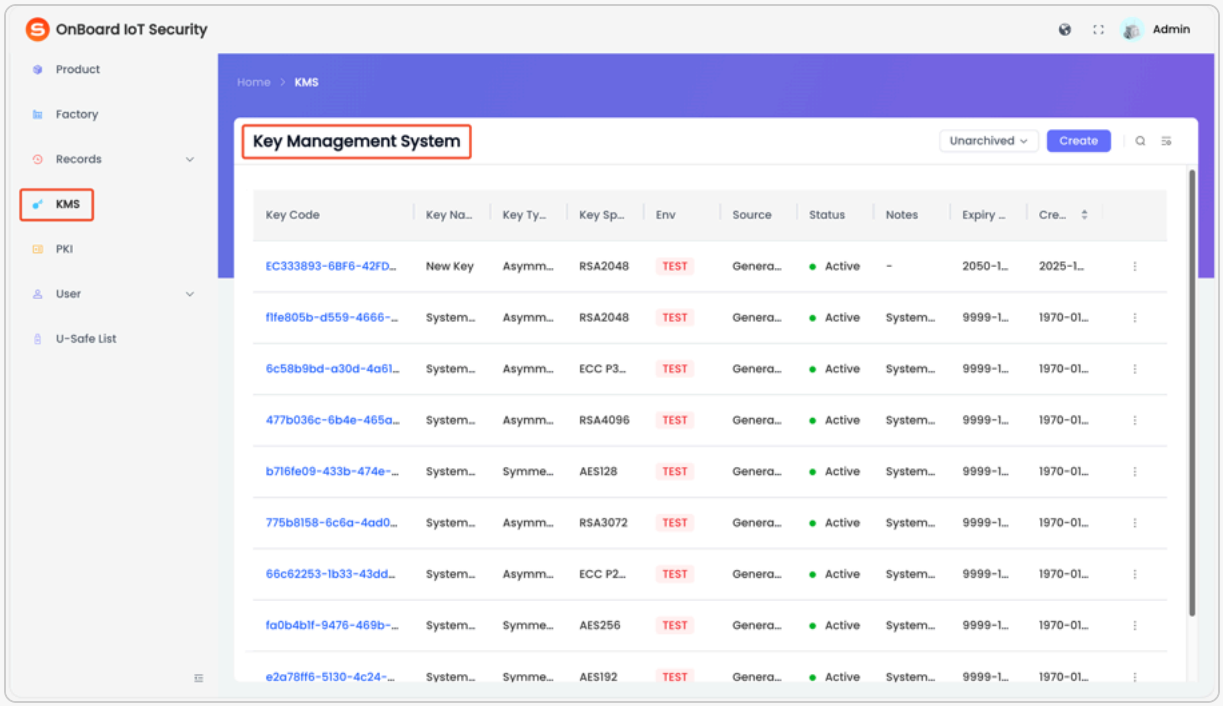
How to check keys in OnBoard™ IoT Security

This article will guide you on how to check key information and logs in OnBoard™ IoT Security (OBIS).

Instructions

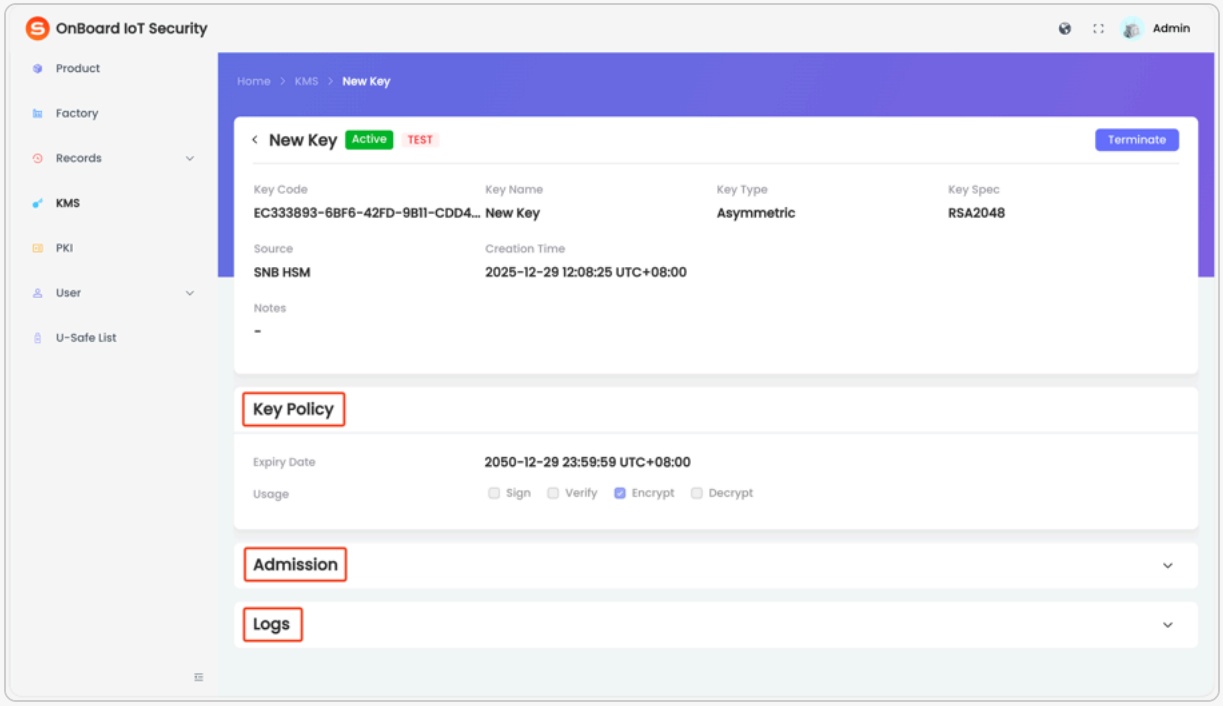
1. Sign in to OBIS.

2. In the left menu, click **KMS** to navigate to the **Key Management System** page, and then you can check the key information (such as **Key Code**, **Key Name**, and **Key Type**). At the same time, you can search for the corresponding key based on the key information and filter to display both archived and unarchived keys.



3. On the **Key Management System** page, select the corresponding key code. On the key page, you can check the key-related information (such as Basic info, Key Policy, Admission, and Logs).

> **TIP**
>
> In the **Admission** area, you can manage the key permissions.
>
> In the **Logs** area, you can check the historical operations of the key.


OnBoard IoT Security
Admin
Product
Factory
Records
KMS
PKI
User
U-Safe List
Home > KMS > New Key
New Key
Active
TEST
Terminate
Key Code
EC333893-6BF6-42FD-9B11-CDD4...
Key Name
New Key
Key Type
Asymmetric
Key Spec
RSA2048
Source
SNB HSM
Creation Time
2025-12-29 12:08:25 UTC+08:00
Notes
-
Key Policy
Expiry Date
2050-12-29 23:59:59 UTC+08:00
Usage
Sign
Verify
Encrypt
Decrypt
Admission
Logs

Version: v3.2.1

How to manage keys in OnBoard™ IoT Security

This article will guide you through the following operations in OnBoard™ IoT Security (OBIS):

- Manage key permissions
- Archive or unarchive a key
- Complete a key
- Download a key
- Lock or unlock a key
- Delete a key

Prerequisites

Before getting started, please ensure that you have OBIS role permissions as a System Manager or Resource Manager, or have operational permissions for the corresponding key.

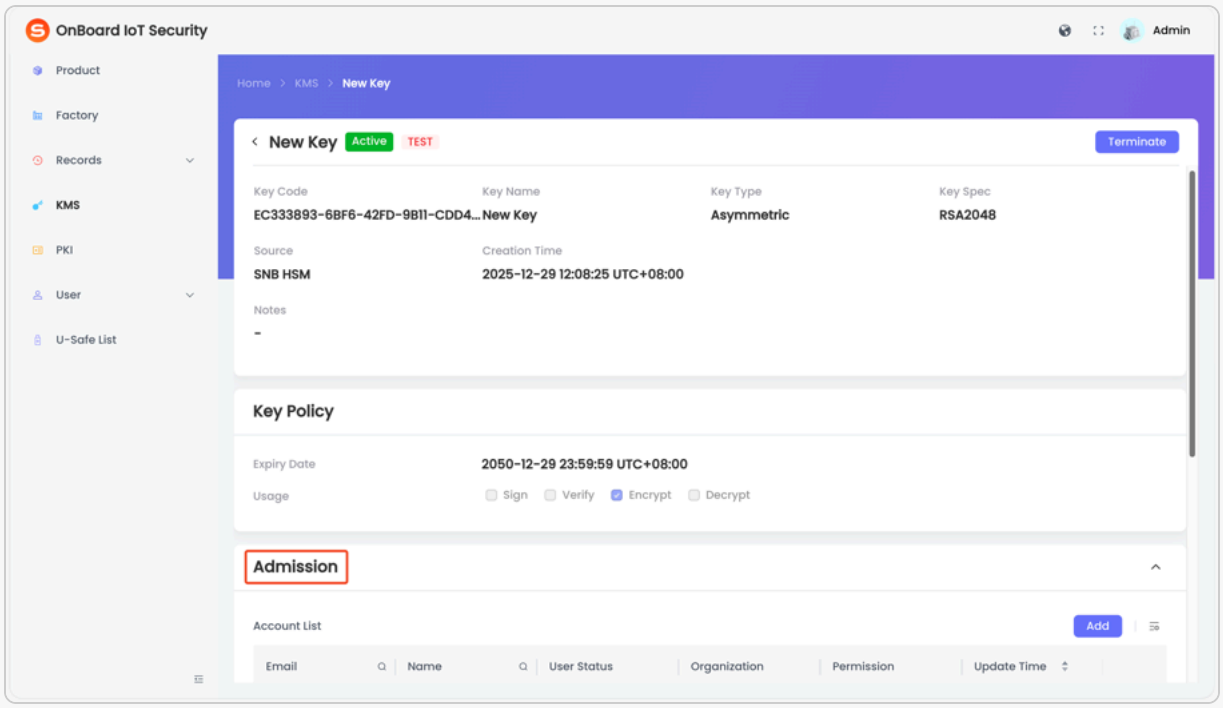
Manage key permissions

You can adjust and set user permissions (View, Edit, Use, Lock, Archive, Grant, and Delete) for keys by following these steps:

1. Sign in to OBIS.

2. In the left menu, click **KMS**, and then select the corresponding key code to navigate to the key page.

3. On the key page, you can manage the key permissions in the **Admission** area.

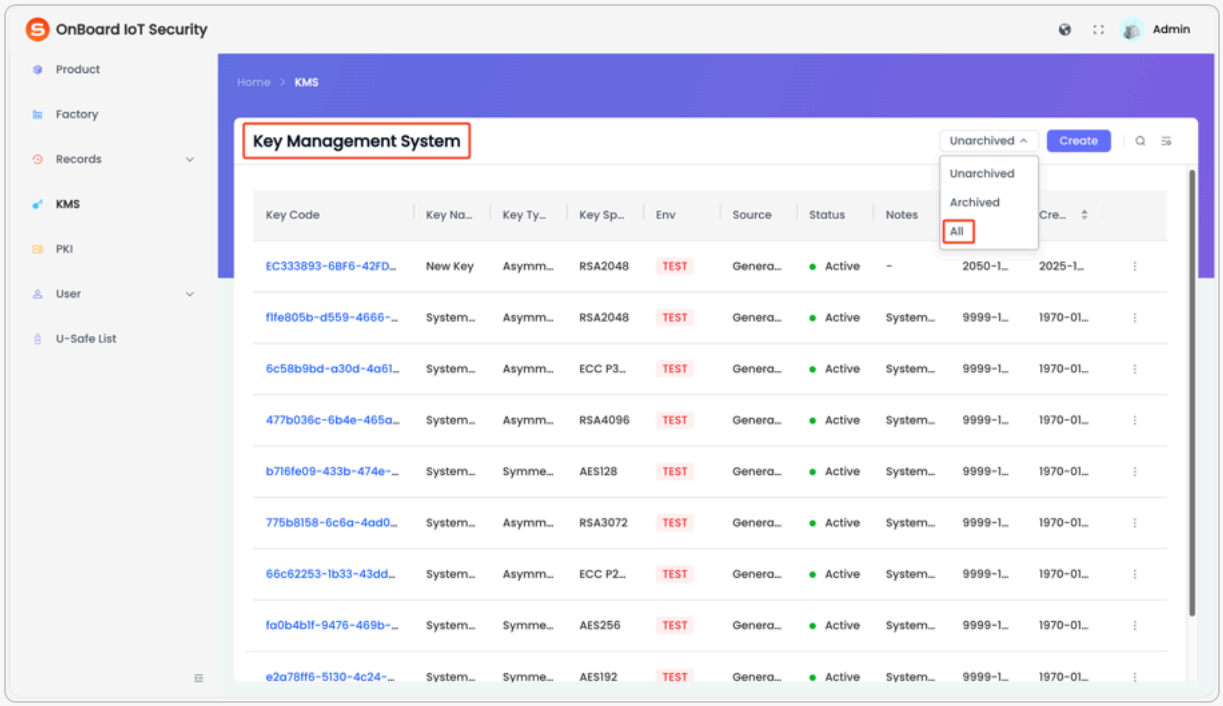


Archive or unarchive a key

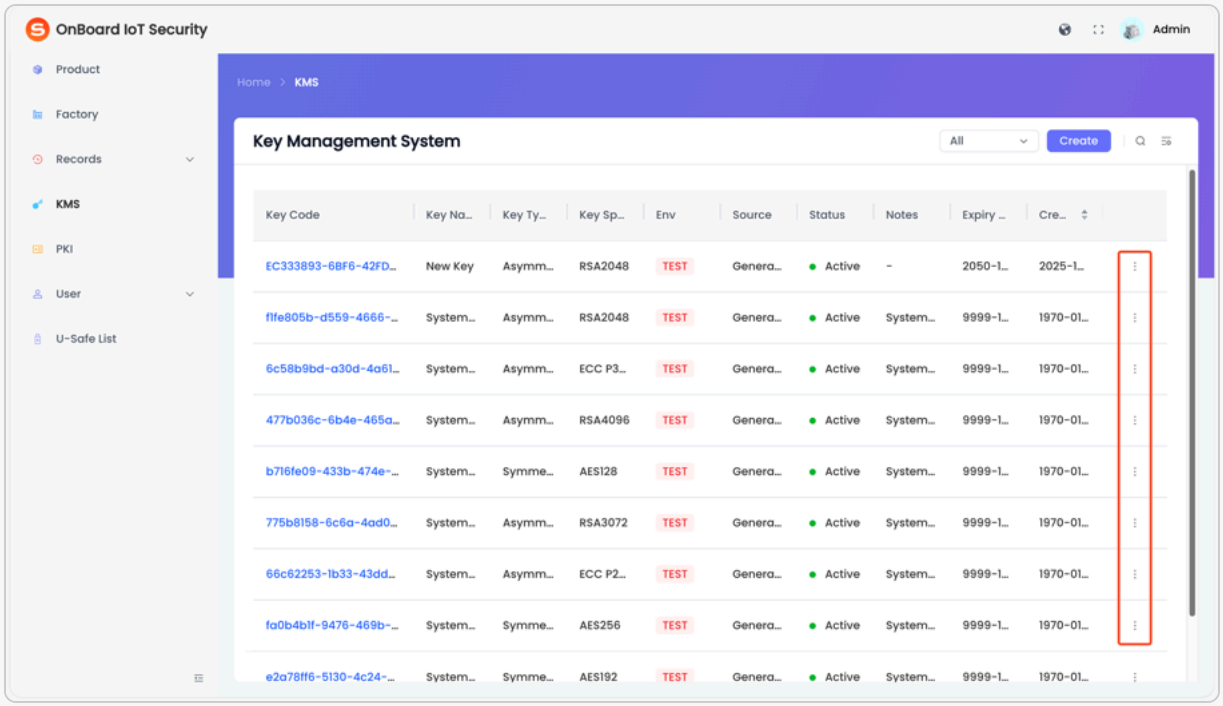
> **TIP**
>
> The key list defaults to displaying unarchived keys. After a key is archived, the key will be displayed at the bottom of the key list when configuring Key for the product.

1. Sign in to OBIS.

2. In the left menu, click **KMS**, and then select **All** in the top-right corner of the **Key Management System** page.



3. Click the Options ⋮ icon for the corresponding key and then select **Archive** or **Unarchive** to archive or unarchive the key.

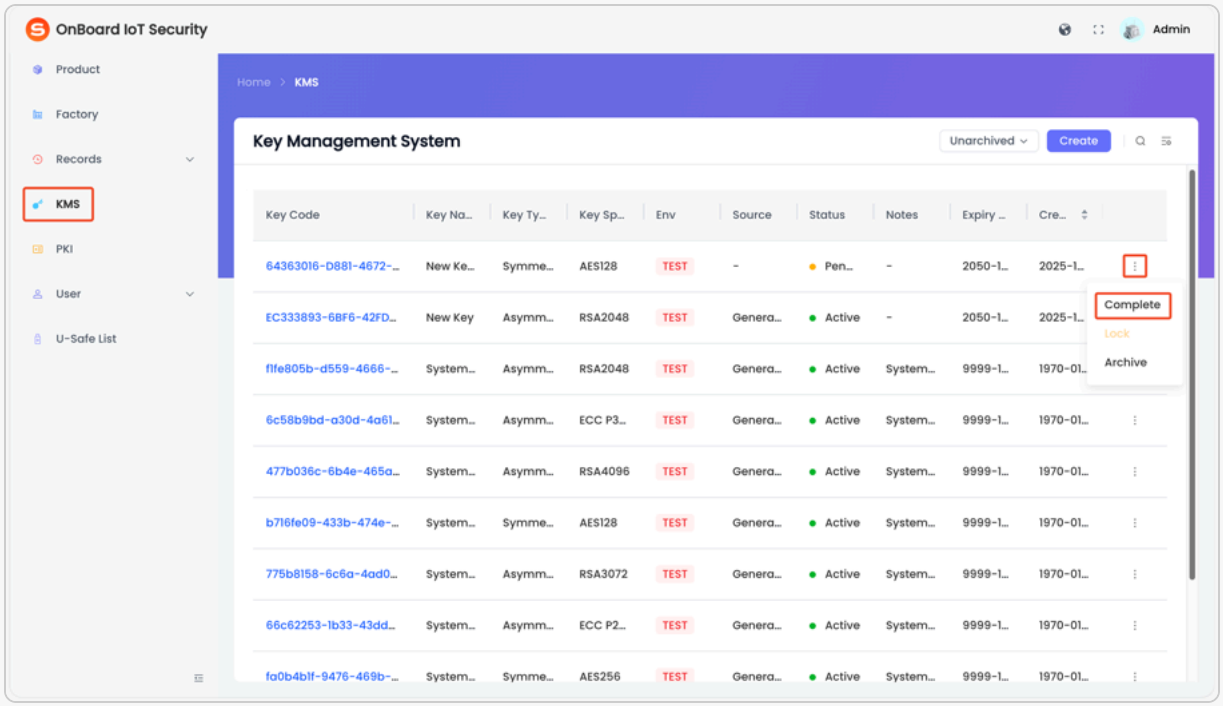


Complete a key

> **TIP**
>
> After completing the key creation, the key can be selected when configuring Key for the product.

1. Sign in to OBIS.

2. In the left menu, click **KMS**, and then click the Options ⋮ icon for the corresponding key and select **Complete**. Then, refer to the steps for creating a key to complete the key creation.

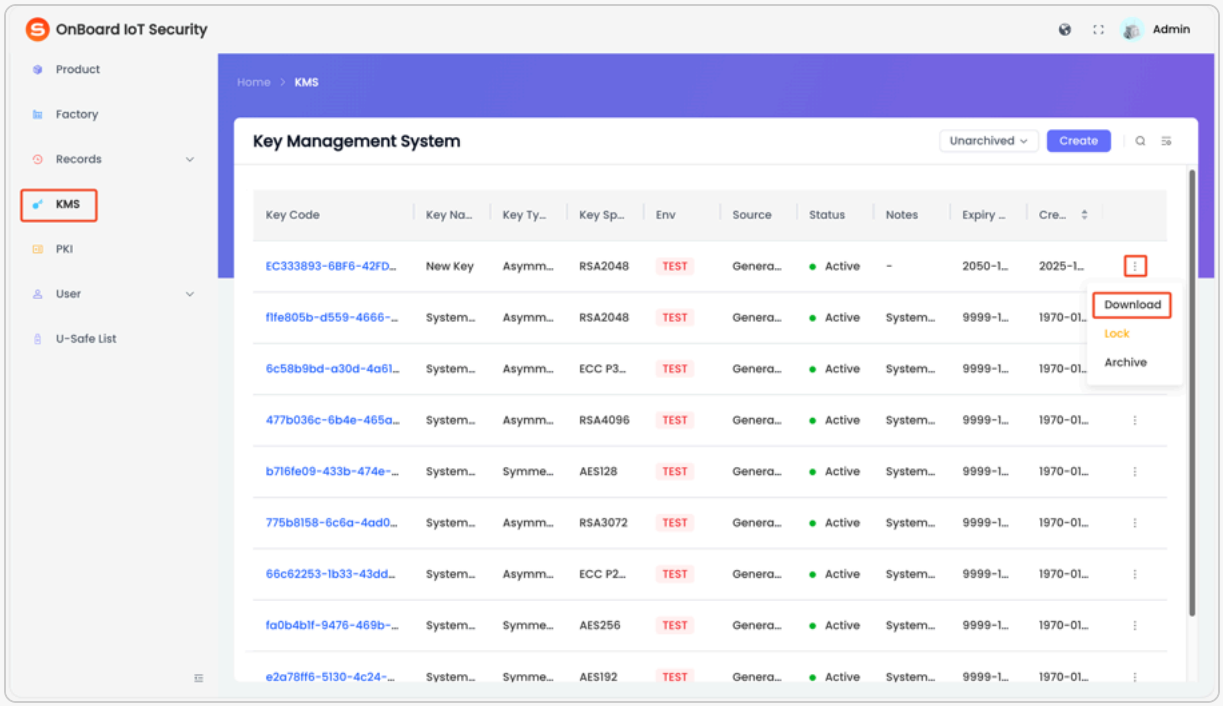


Download a key

> **TIP**
>
> Download the public key of the asymmetric key for encryption and verification.

1. Sign in to OBIS.

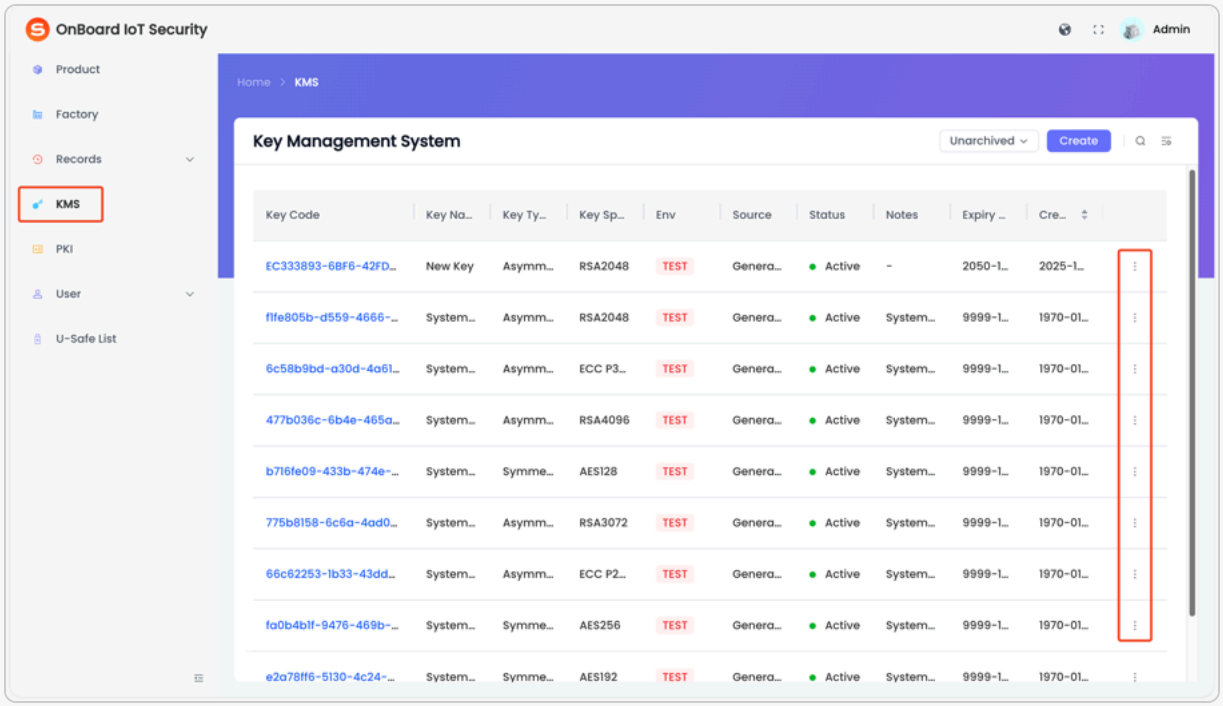
2. In the left menu, click **KMS**, and then click the Options ⋮ icon for the corresponding asymmetric key and select **Download** to download the public key of the asymmetric key.



Lock or unlock a key

> **TIP**
>
> Before locking or unlocking a key, please ensure that the key has not been archived. After the key is locked, the key can't be selected when configuring Key for the product.

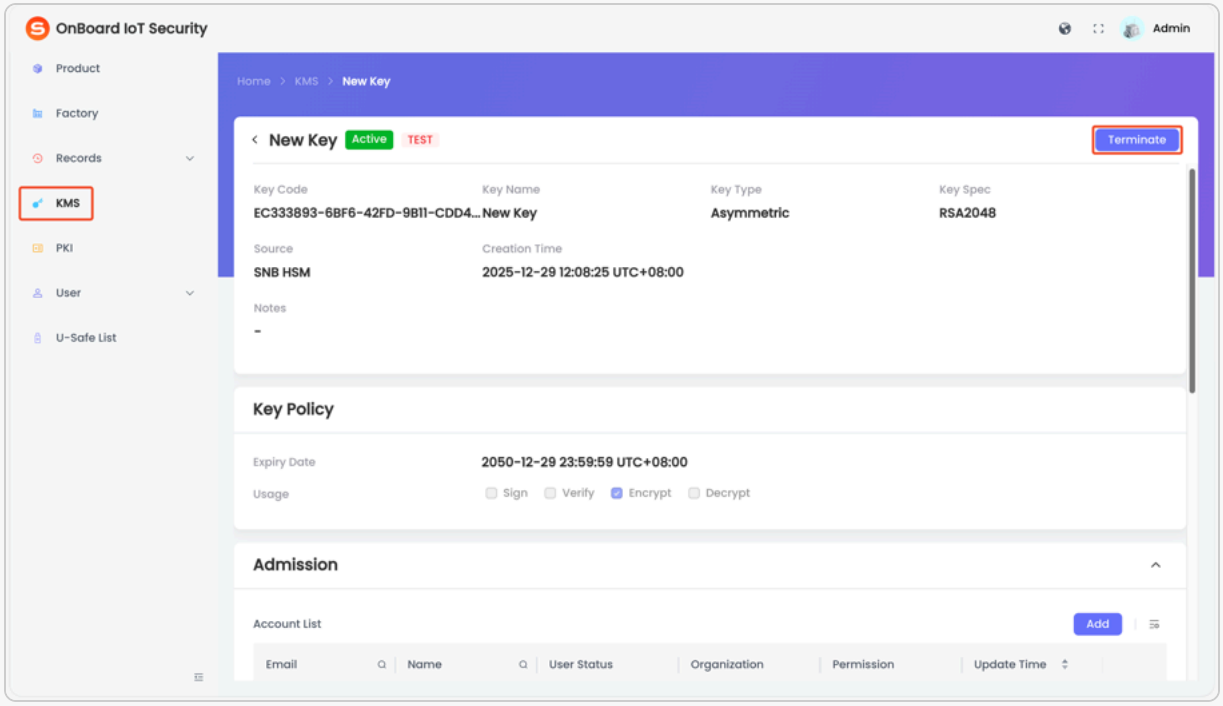
1. Sign in to OBIS.
2. In the left menu, click **KMS**, and then click the Options ⋮ icon for the corresponding key and select **Lock** or **Unlock** to lock or unlock the key.



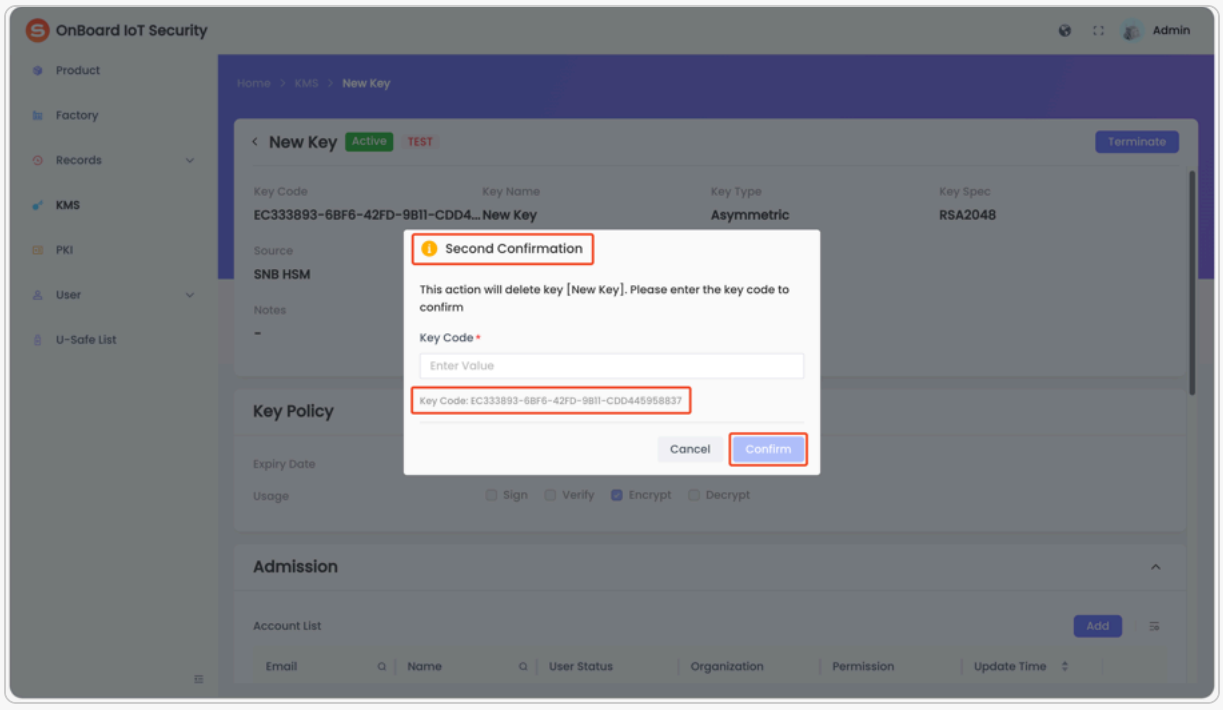
Delete a key

> **INFO**
>
> Before deleting a key, please ensure that the key is not referenced by any resources. If the key is in use, you will see the message "Key cannot be deleted, the Key is currently used by the following list" along with a list of the referencing resources.

1. Sign in to OBIS.
2. In the left menu, click **KMS**, and then select the corresponding key code to navigate to the key page.
3. On the key page, click **Terminate**.



4. In the **Second Confirmation** window, enter the key code prompted at the bottom of the window, and then click **Confirm** to delete the key.



Version: v3.2.1

Overview of PKI in OnBoard™ IoT Security

PKI (Public Key Infrastructure) in OBIS (OnBoard™ IoT Security) is the core module responsible for certificate security, which provides certificate generation, issuance, and revocation services to ensure secure communication for IoT devices. It is mainly used for:

- Pre-provisioning certificate templates for devices in bulk.
- Using command-line tools to issue end-entity templates.
- Using command-line tools to verify the validity of certificates and certificate chains.

In addition, OBIS provides certificate templates for the Matter protocol, enabling rapid generation of corresponding certificate chains. The following table summarizes the core capabilities of PKI:

Capability	Description
CA Hierarchy Management	• Root CA and Intermediate CA hierarchy management • Certificate chain validation and trust path verification

Capability	Description
Certificate Lifecycle	• Certificate issuance, renewal, and revocation • Certificate templates and policy management • Certificate validity and status monitoring
External PKI Integration (Root CA Delegation)	• OEM retains Root CA ownership in enterprise PKI • PKI operates issuing CAs under delegated authority

Version: v3.2.1

How to create a certificate in OnBoard™ IoT Security

This article will guide you through the following operations in OnBoard™ IoT Security (OBIS):

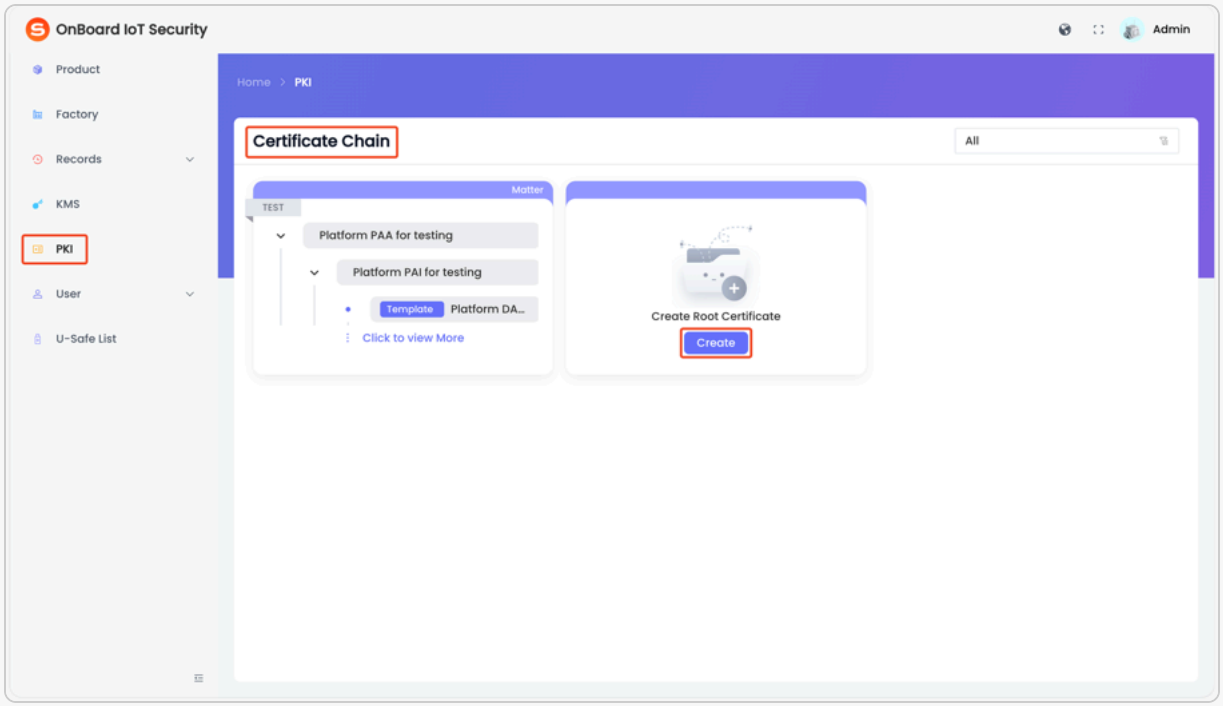
- Create a certificate chain
- Create a custom certificate
- Create a certificate based on the Matter template

Prerequisites

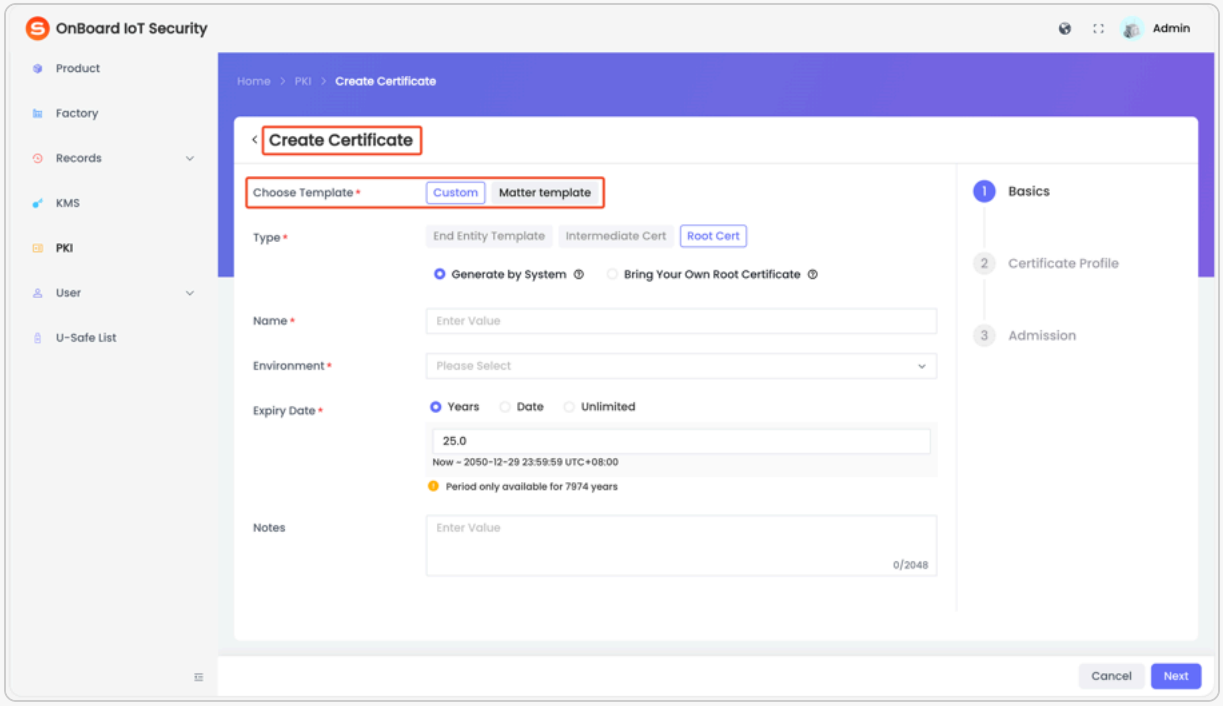
Before getting started, please ensure that you have OBIS role permissions as a System Manager or Resource Manager.

Create a certificate chain

1. Sign in to OBIS.
2. In the left menu, click **PKI**, and then click **Create** on the **Certificate Chain** page.



3. On the **Create Certificate** page, you can create a certificate in either the **Custom** or **Matter template** format as needed.



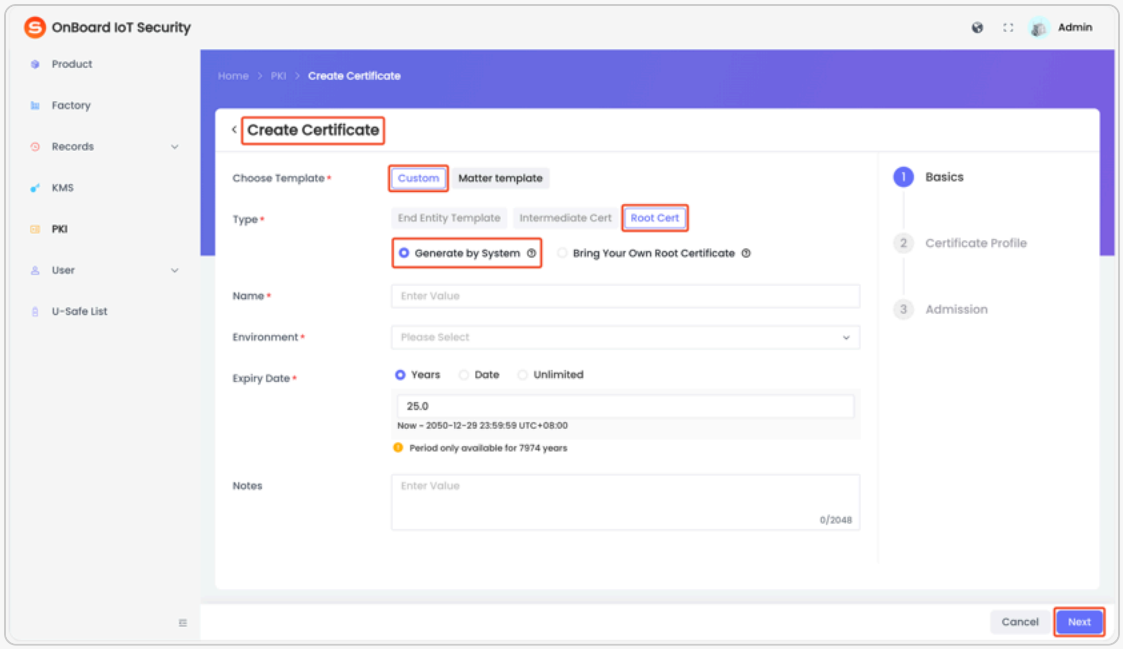
Create a custom certificate

> **TIP**
>
> After a custom certificate or certificate template is created, it can be selected when configuring Certificate for a product.

Create a root certificate | Create an intermediate certificate | Create an end entity template

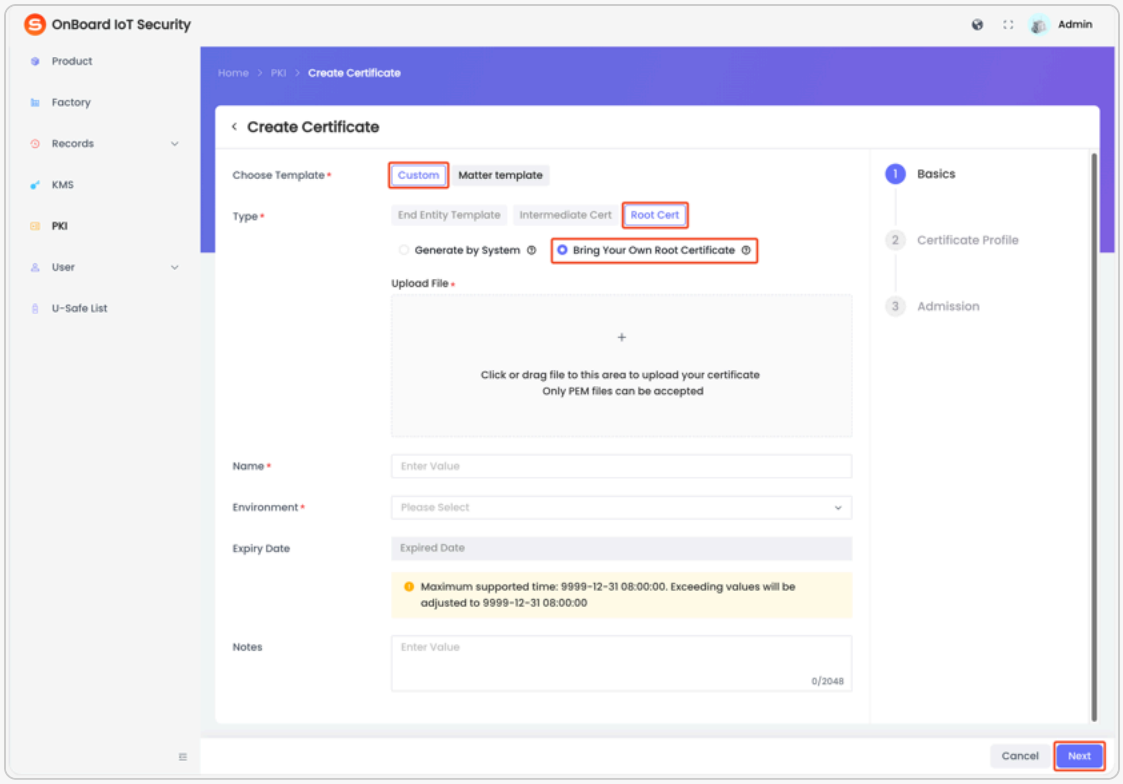
1. On the **Create Certificate** page, select the **Custom** for the format and the **Root Cert** for the type.
2. Select **Generate by System** or **Bring Your Own Root Certificate** as needed.
    - If you are not using your own root certificate, select **Generate by System**, then select and enter the basic information.

      > **TIP**
      >
      > The system securely generates a random key pair in the HSM and self-signs the Root Certificate. The Root Certificate (public part only) can be exported.



- If you are using your own root certificate, select **Bring Your Own Root Certificate**, then select and enter the basic information.

> **TIP**
>
> Manually upload your own root certificate (with public key only) generated outside the system for verifying signature.



Description of root certificate basic information:

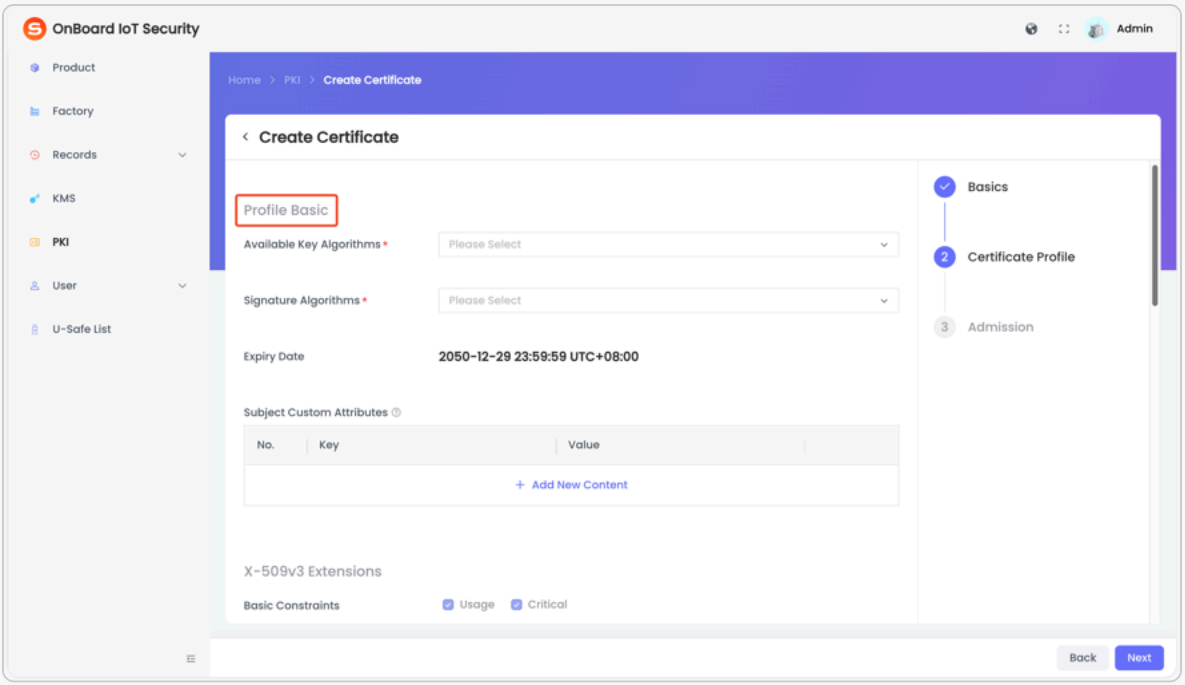
Information	Generate by System	Bring Your Own Root Certificate
Upload file	No file upload is required.	In the **Upload File** area, click the Add ➕ icon to select and upload a root certificate in **.pem** format, or directly drag and drop your own root certificate into the upload area.
Name	The root certificate name must be unique in the same environment.	

Information	Generate by System	Bring Your Own Root Certificate
Environment	Two environments can be selected: **TEST** and **PROD**.	
Expiry Date	Set the expiry date of the root certificate.	Automatically parsed from the certificate content and can't be modified.
Notes	Provide additional information related to the root certificate, such as special requirements and important notes.	

3. After verifying that the information is correct, click **Next** and configure according to the following information.

> **TIP**
>
> If you are using your own root certificate, the related configuration will be automatically parsed and generated based on the certificate content.

Description of Profile Basic information

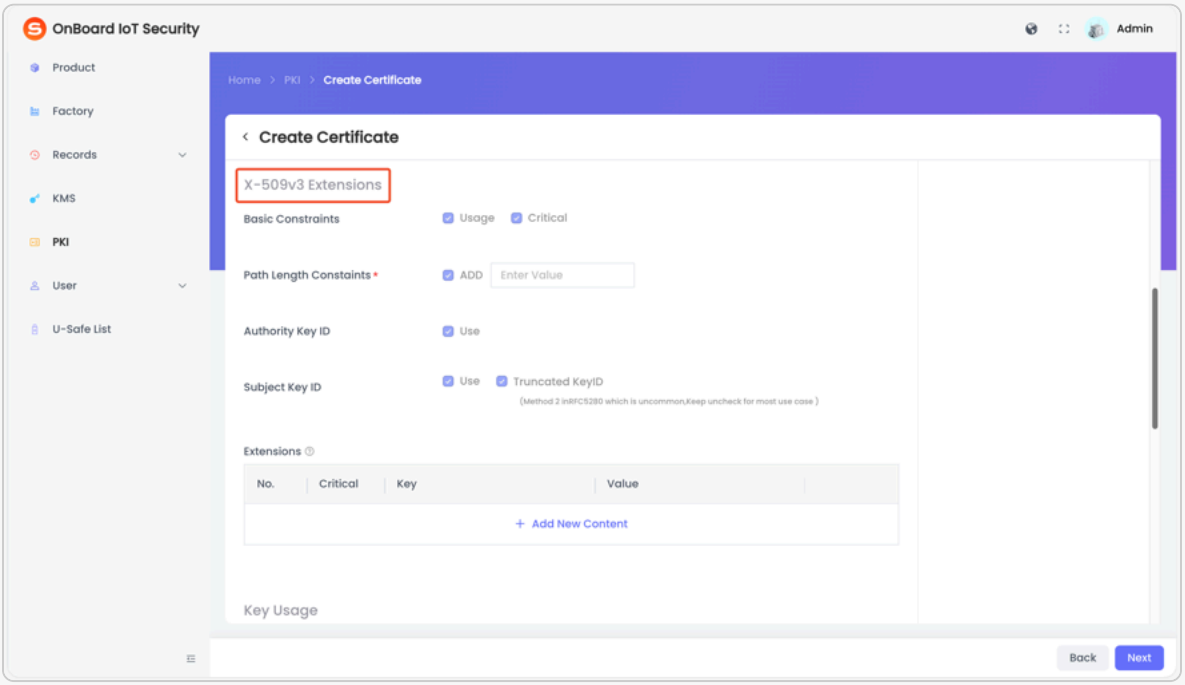


Information	Mandatory	Description
Available Key Algorithms	Yes	Available algorithms for the certificates. Options include: ◦ **RSA_2048** ◦ **RSA_3072** ◦ **RSA_4096** ◦ **EC_secp384r1** ◦ **EC_prime256v1**

Information	Mandatory	Description
Signature Algorithms	Yes	Signature algorithms for the certificates. Options include: ◦ **SHA256WITHRSA** ◦ **SHA384WITHRSA** ◦ **SHA512WITHRSA**
Expiry Date	No	Inherits the Expiry Date set in the previous step and can't be modified.
Subject Custom Attributes	No	Configure subject custom attributes for the certificate by adding specific keys and values.

Description of X-509v3 Extensions information

> **TIP**
>
> If the **Critical** checkbox for a certain item is selected, the system must recognize and support that item when processing the certificate; otherwise, the certificate will be rejected.

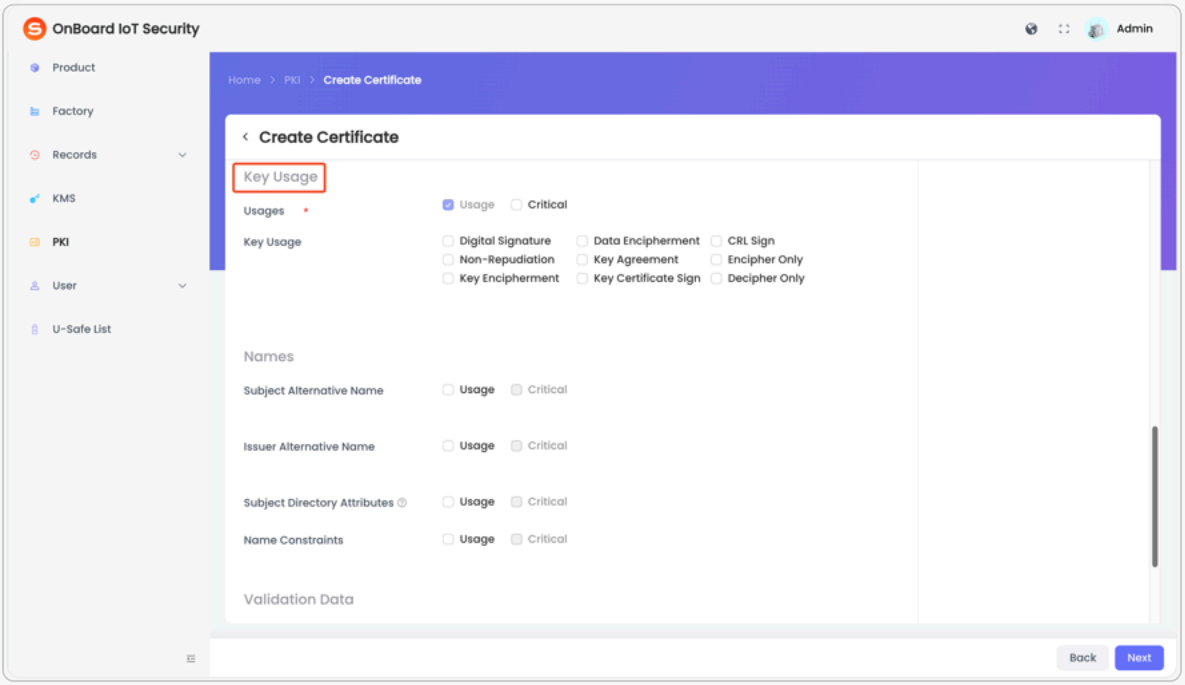


Information	Mandatory	Description
Basic Constraints	No	Automatically parsed from the certificate content and can't be modified.
Path Length Constaints	Yes	Limits the number of sub-certificates.

Information	Mandatory	Description
Authority Key ID	No	Automatically parsed from the certificate content and can't be modified.
Subject Key ID	No	Automatically parsed from the certificate content and can't be modified.
Extensions	No	Configure the X-509v3 extension attributes for the certificate by adding specific keys and values.

Description of Key Usage information

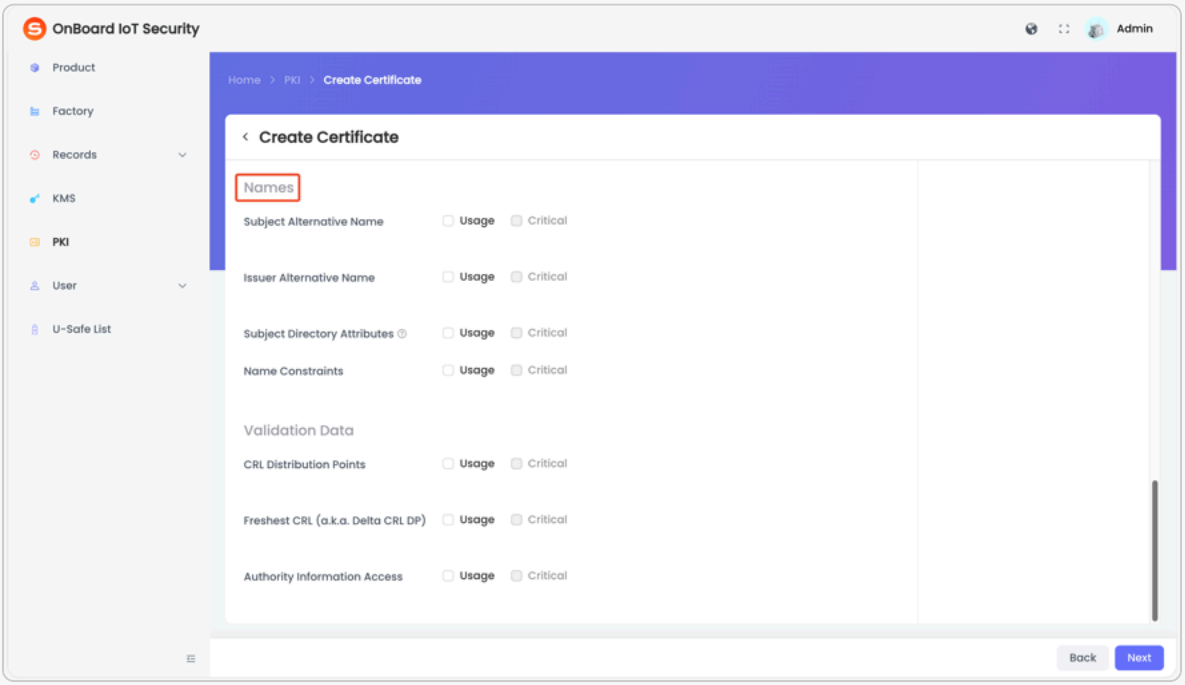
> **TIP**
>
> If the **Critical** checkbox for a certain item is selected, the system must recognize and support that item when processing the certificate; otherwise, the certificate will be rejected.



Information	Mandatory	Description
Usages	Yes	Enabled by default and can't be modified.
Key Usage	Yes	Optional key usages, such as **Digital Signature**, **Data Encipherment**, and **CRL Sign**.

Description of Names information

> **TIP**
>
> If the **Critical** checkbox for a certain item is selected, the system must recognize and support that item when processing the certificate; otherwise, the certificate will be rejected.

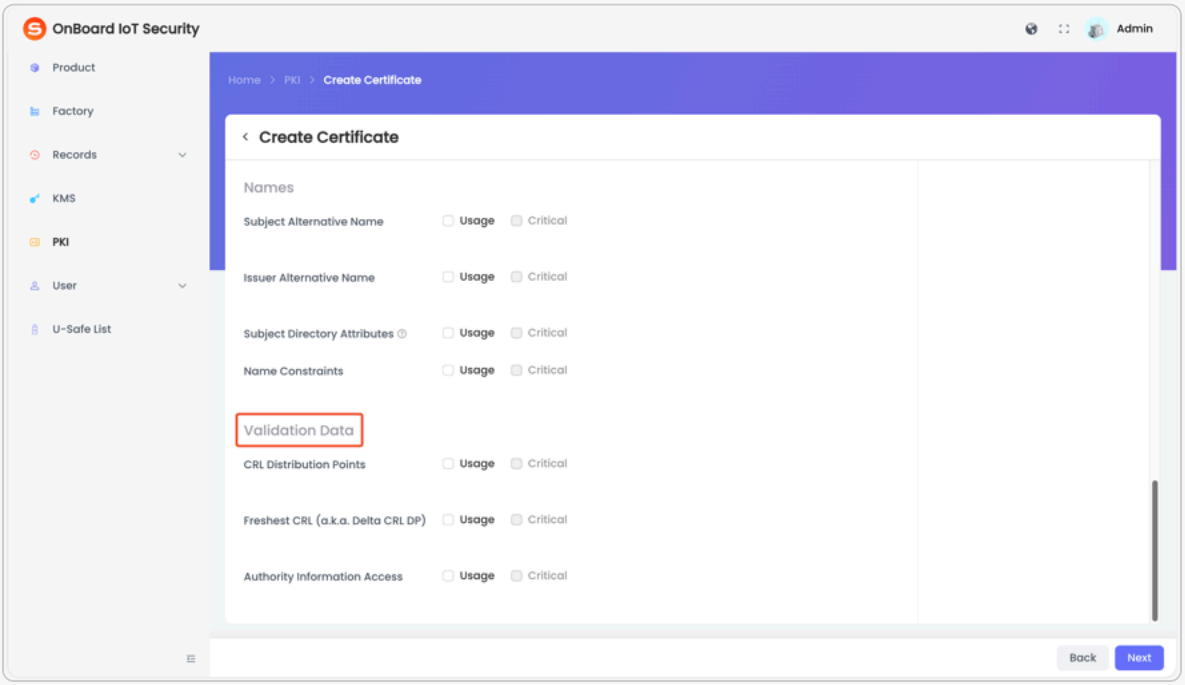


Information	Mandatory	Description
Subject Alternative Name	No	After selecting **Usage**, a subject alternative name must be added.
Issuer Alternative Name	No	After selecting **Usage**, an issuer alternative name must be added.

Information	Mandatory	Description
Subject Directory Attributes	No	After selecting **Usage**, specific keys and values must be added.
Name Constraints	No	After selecting **Usage**, specific values must be added to the Black List.

Description of Validation Data information

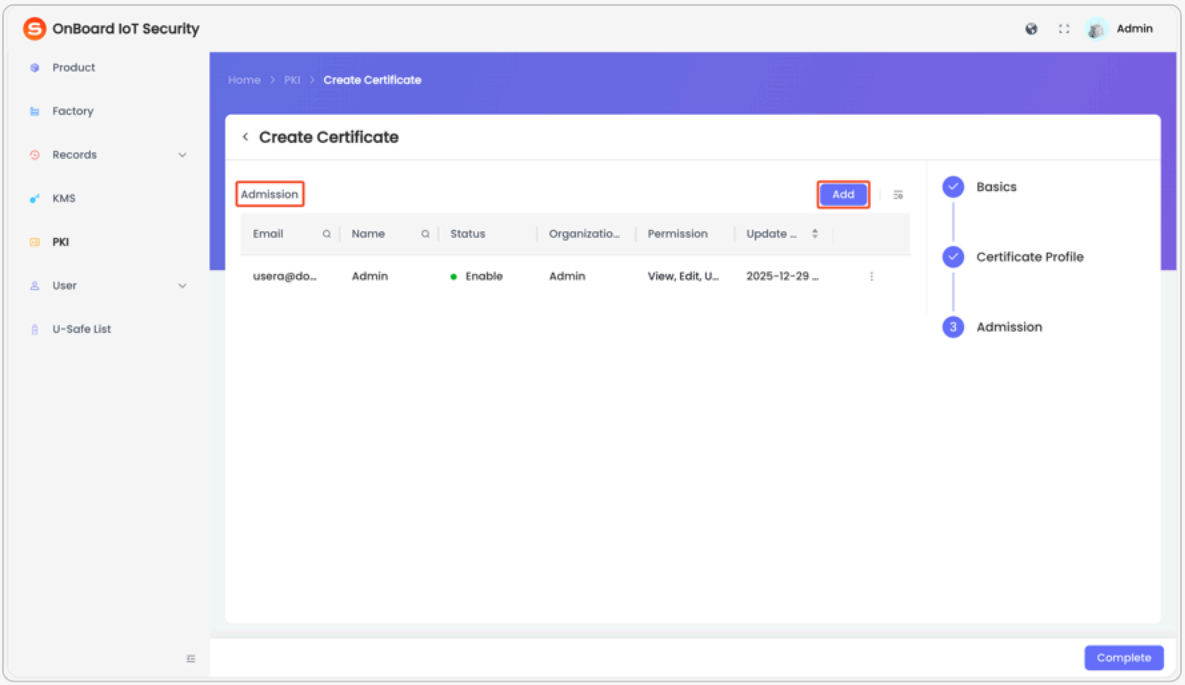
> **TIP**
>
> If the **Critical** checkbox for a certain item is selected, the system must recognize and support that item when processing the certificate; otherwise, the certificate will be rejected.



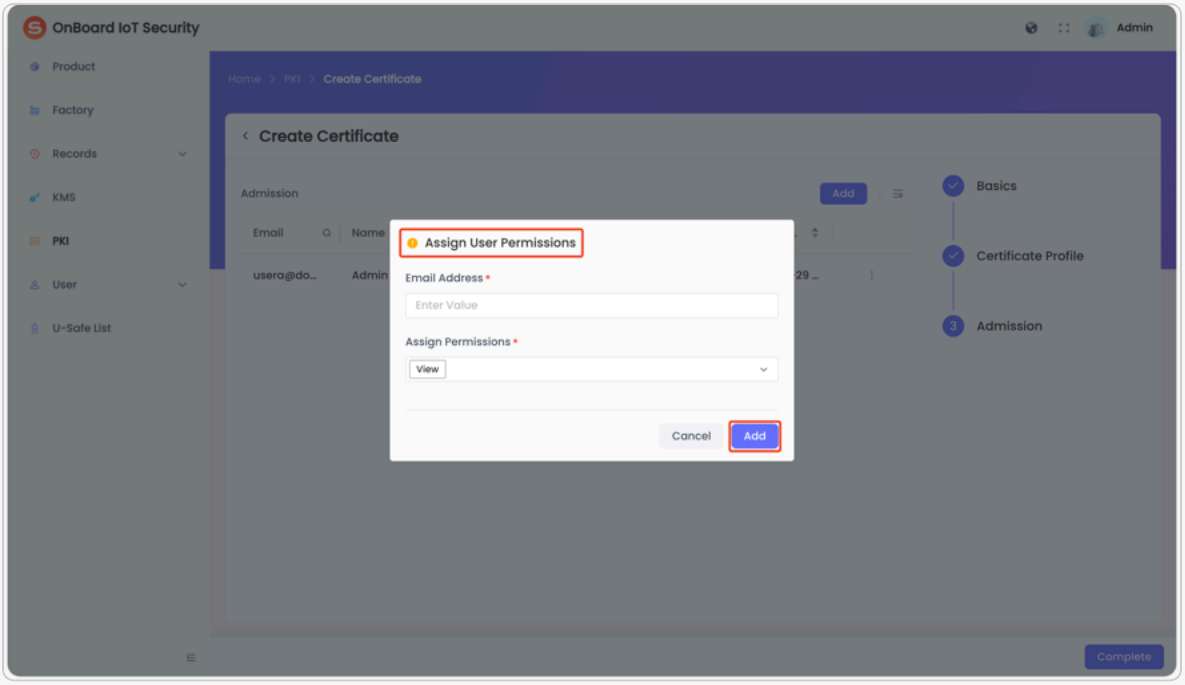
Information	Mandatory	Description
CRL Distribution Points	No	After selecting **Usage**, a CRL Distribution Point URI and a CRL Issuer must be added.

Information	Mandatory	Description
Freshest CRL (a.k.a. Delta CRL DP)	No	After selecting **Usage**, the Freshest CRL URI must be added.
Authority Information Access	No	After selecting **Usage**, an OCSP Service Locator URI must be added, and specific values must be added to the CA issuer URI.

4. After verifying that the information is correct, click **Next**, and then click **Add** on the **Admission** page.



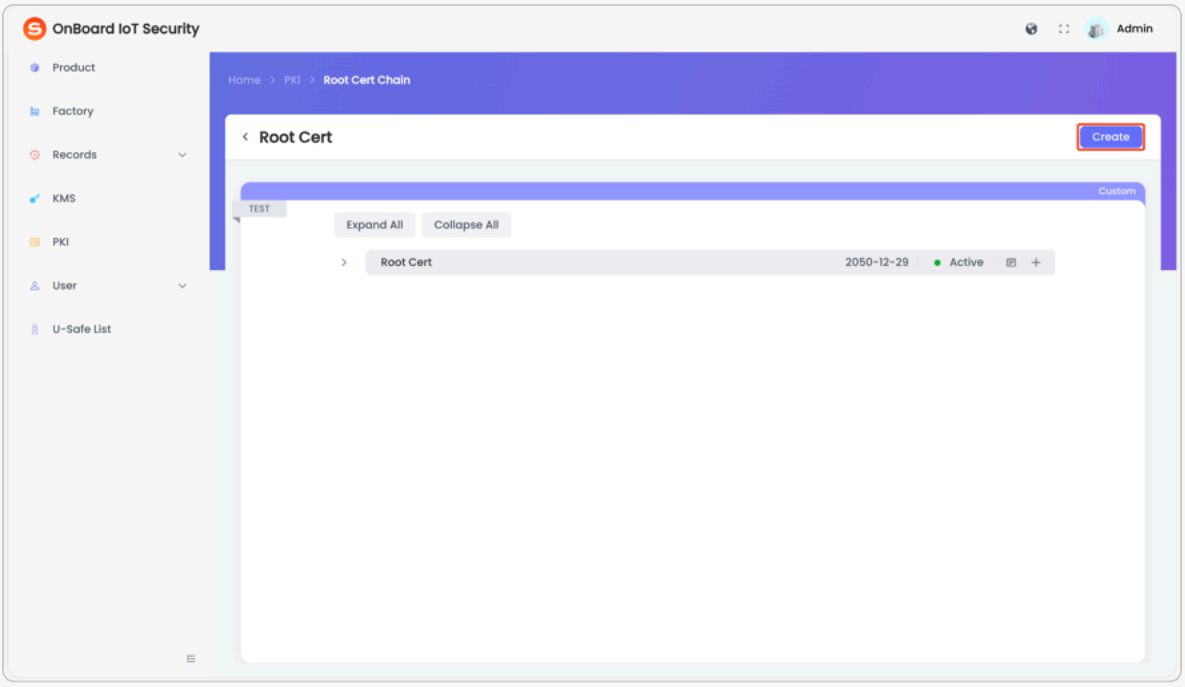
5. In the **Assign User Permissions** window, enter the email address and assign the corresponding permission (View, Edit, Use, Revoke, Grant, and Delete). Then, click **Add**.



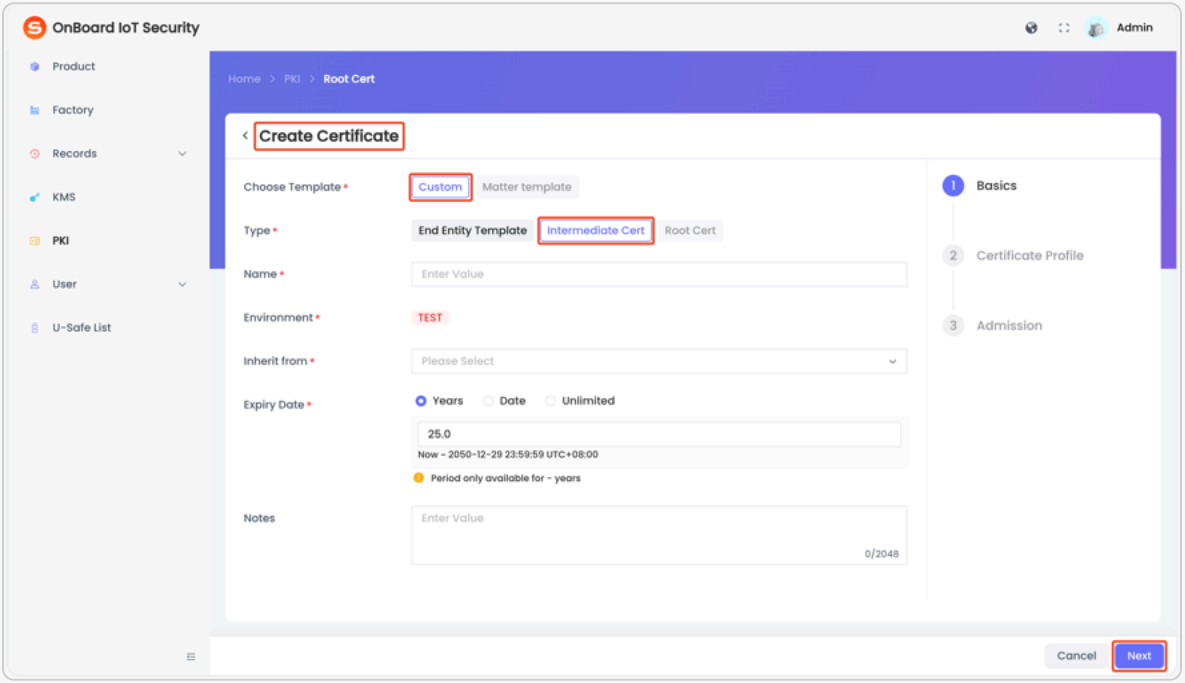
6. Click **Complete** to complete the root certificate creation.

> **INFO**
>
> Before creating an intermediate certificate, please ensure that at least one root certificate has been created so that it can be selected and inherited during configuration.

1. On the **Certificate Chain** page, click the corresponding certificate chain, and then click **Create**.



2. On the **Create Certificate** page, select **Custom** for the format and **Intermediate Cert** for the type, then select and enter the basic information.



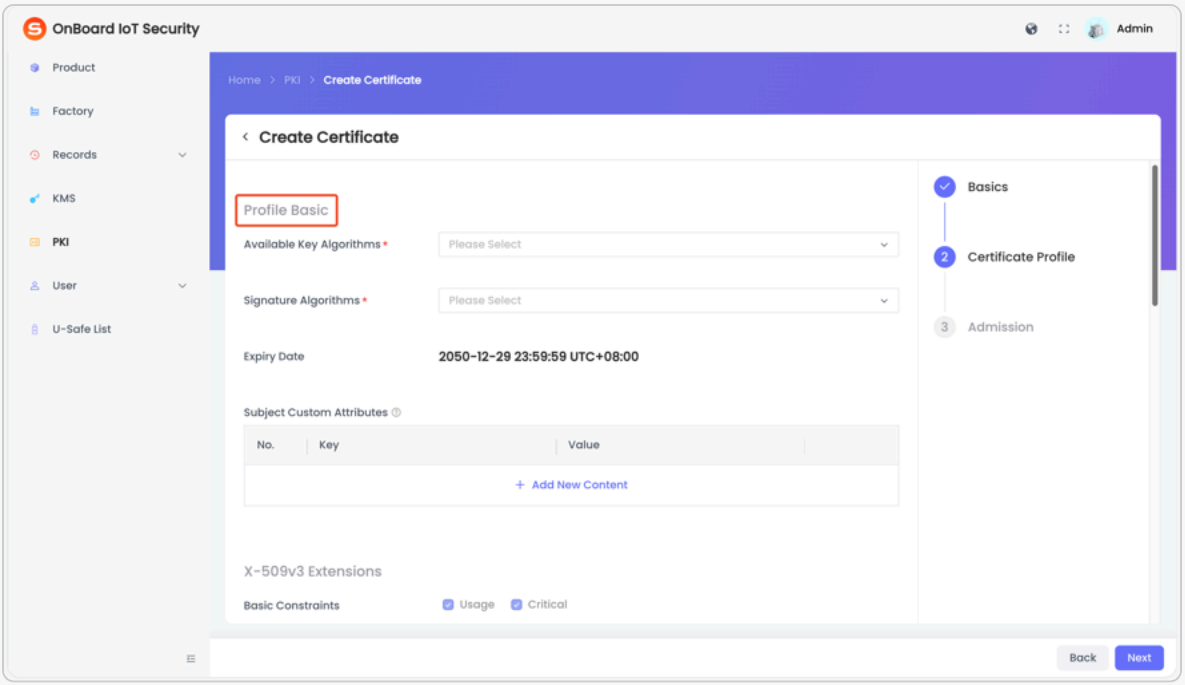
Description of intermediate certificate basic information:

Information	Description
Name	Within the same certificate chain, certificate names at the same hierarchy level must be unique.
Environment	The environment inherited from the root certificate.
Inherit from	Select the root certificate or intermediate certificate to inherit from.
Expiry Date	Set the expiry date of the intermediate certificate. It must not exceed the expiry date of the inherited certificate. **Note**: If the intermediate certificate is inherited from your own root certificate, the expiry date will be inherited from the root certificate.
Notes	Provide additional information related to the intermediate certificate, such as special requirements and important notes.

3. After verifying that the information is correct, click **Next** and configure according to the following information.

> **TIP**
>
> If the intermediate certificate is inherited from your own root certificate, only the **Profile Basic** area is displayed.

Description of Profile Basic information

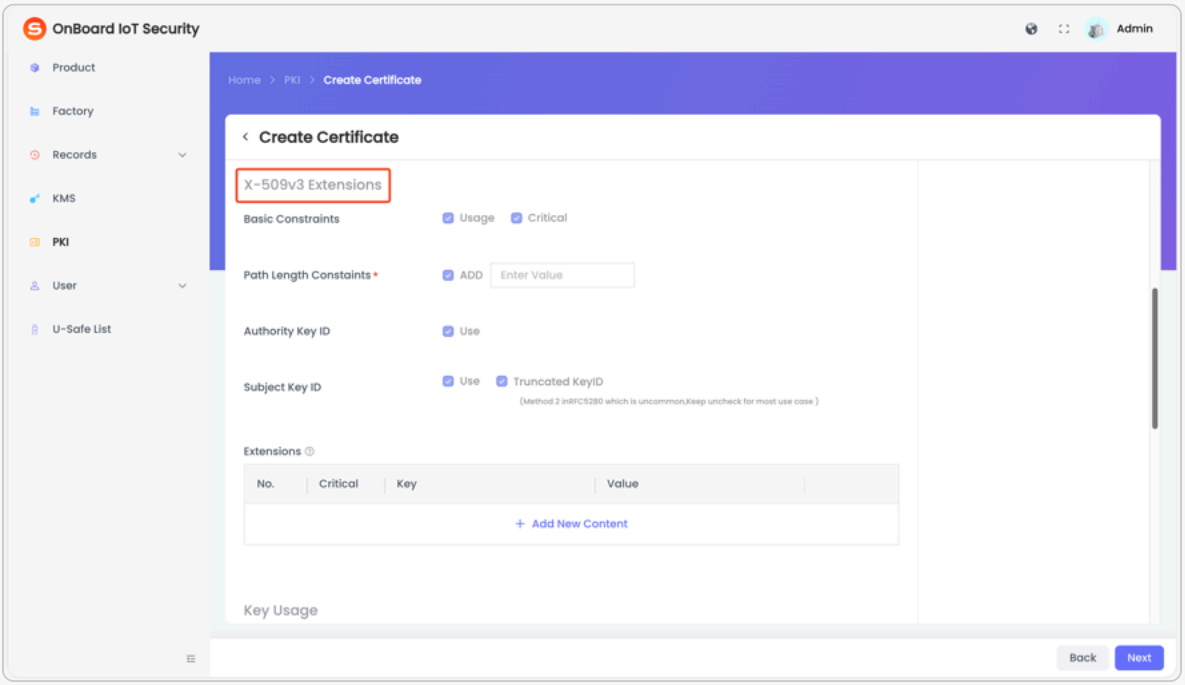


Information	Mandatory	Description
Available Key Algorithms	Yes	Available algorithms for the certificates. Options include: ◦ **RSA_2048** ◦ **RSA_3072** ◦ **RSA_4096** ◦ **EC_secp384r1** ◦ **EC_prime256v1**

Information	Mandatory	Description
Signature Algorithms	Yes	Signature algorithms for the certificates. Options include: ◦ **SHA256WITHRSA** ◦ **SHA384WITHRSA** ◦ **SHA512WITHRSA**
Expiry Date	No	Inherits the Expiry Date set in the previous step and can't be modified.
Subject Custom Attributes	No	Configure subject custom attributes for the certificate by adding specific keys and values.

Description of X-509v3 Extensions information

> **TIP**
>
> If the **Critical** checkbox for a certain item is selected, the system must recognize and support that item when processing the certificate; otherwise, the certificate will be rejected.

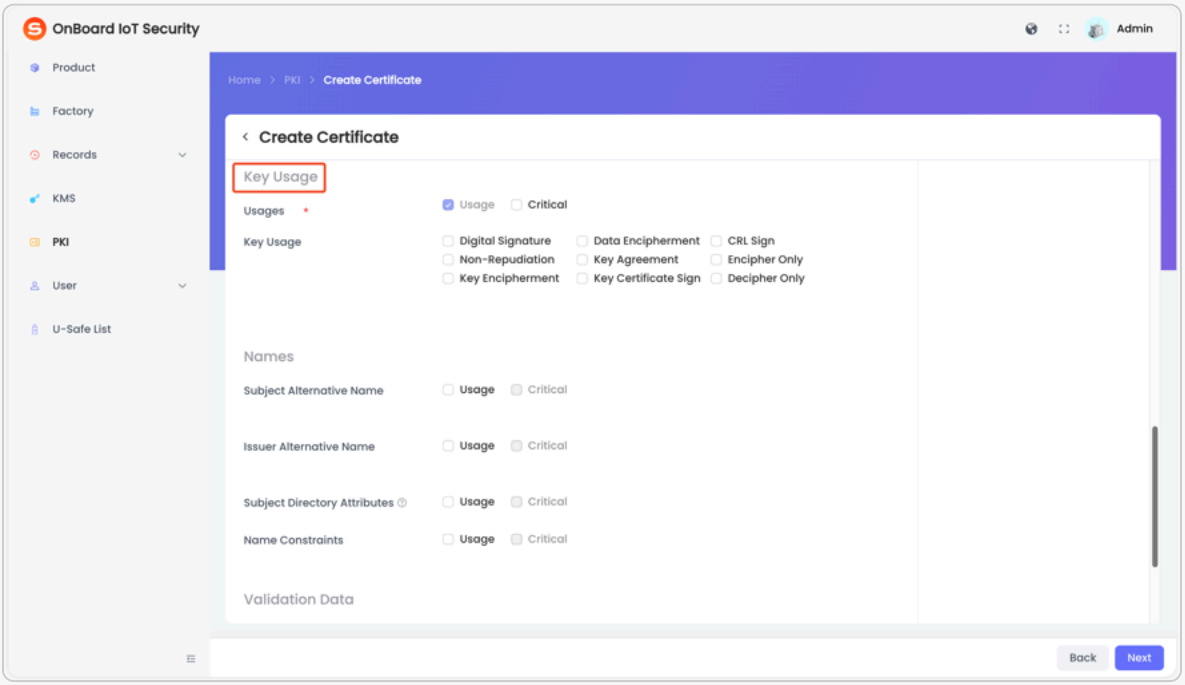


Information	Mandatory	Description
Basic Constraints	No	Automatically parsed from the certificate content and can't be modified.
Path Length Constaints	Yes	Limits the number of sub-certificates.

Information	Mandatory	Description
Authority Key ID	No	Automatically parsed from the certificate content and can't be modified.
Subject Key ID	No	Automatically parsed from the certificate content and can't be modified.
Extensions	No	Configure the X-509v3 extension attributes for the certificate by adding specific keys and values.

Description of Key Usage information

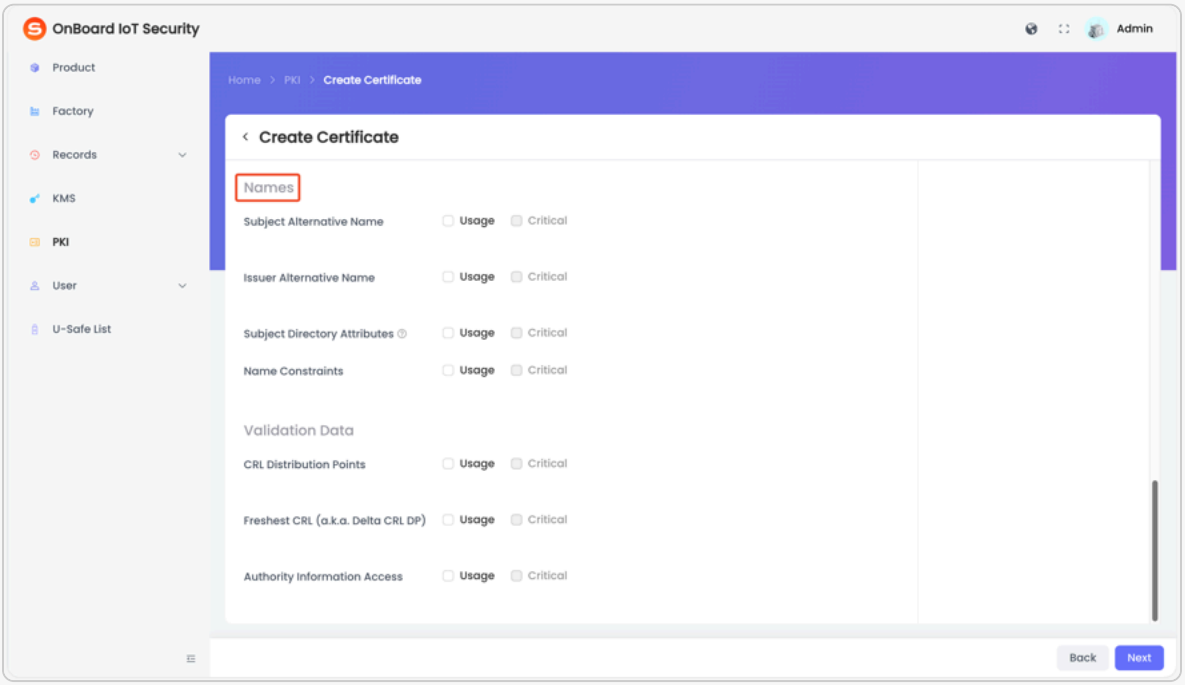
> **TIP**
>
> If the **Critical** checkbox for a certain item is selected, the system must recognize and support that item when processing the certificate; otherwise, the certificate will be rejected.



Information	Mandatory	Description
Usages	Yes	Enabled by default and can't be modified.
Key Usage	Yes	Optional key usages, such as **Digital Signature**, **Data Encipherment**, and **CRL Sign**.

Description of Names information

> **TIP**
>
> If the **Critical** checkbox for a certain item is selected, the system must recognize and support that item when processing the certificate; otherwise, the certificate will be rejected.

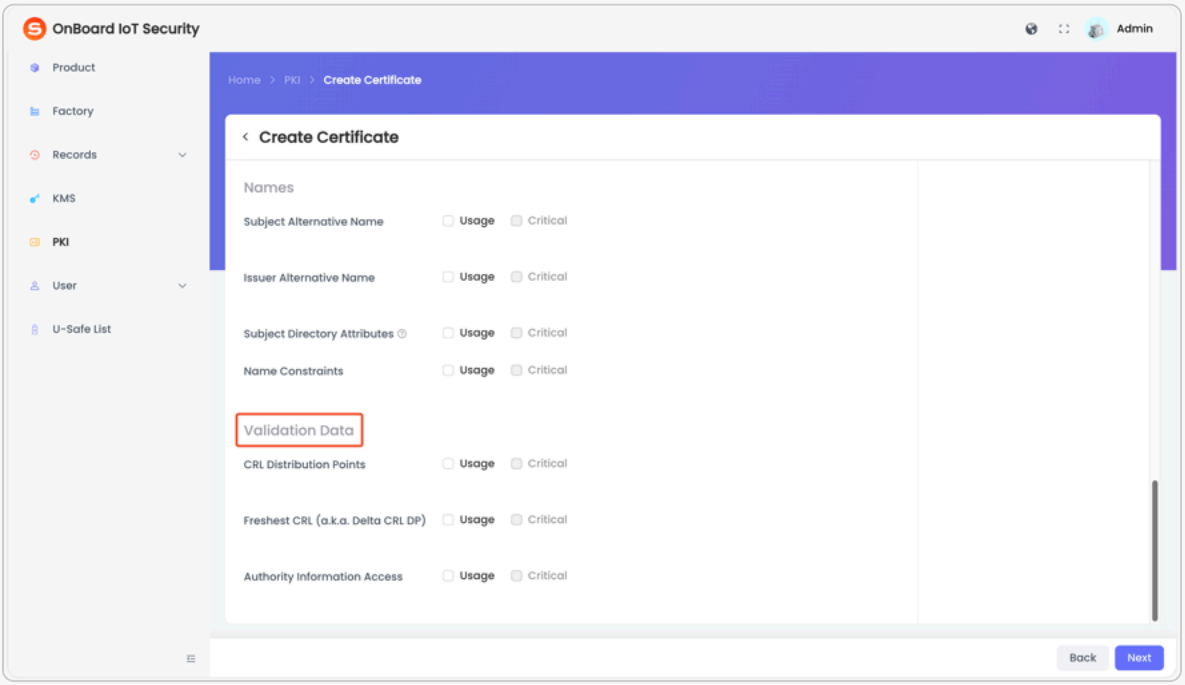


Information	Mandatory	Description
Subject Alternative Name	No	After selecting **Usage**, a subject alternative name must be added.

Information	Mandatory	Description
Issuer Alternative Name	No	After selecting **Usage**, an issuer alternative name must be added.
Subject Directory Attributes	No	After selecting **Usage**, specific keys and values must be added.
Name Constraints	No	After selecting **Usage**, specific values must be added to the Black List.

Description of Validation Data information

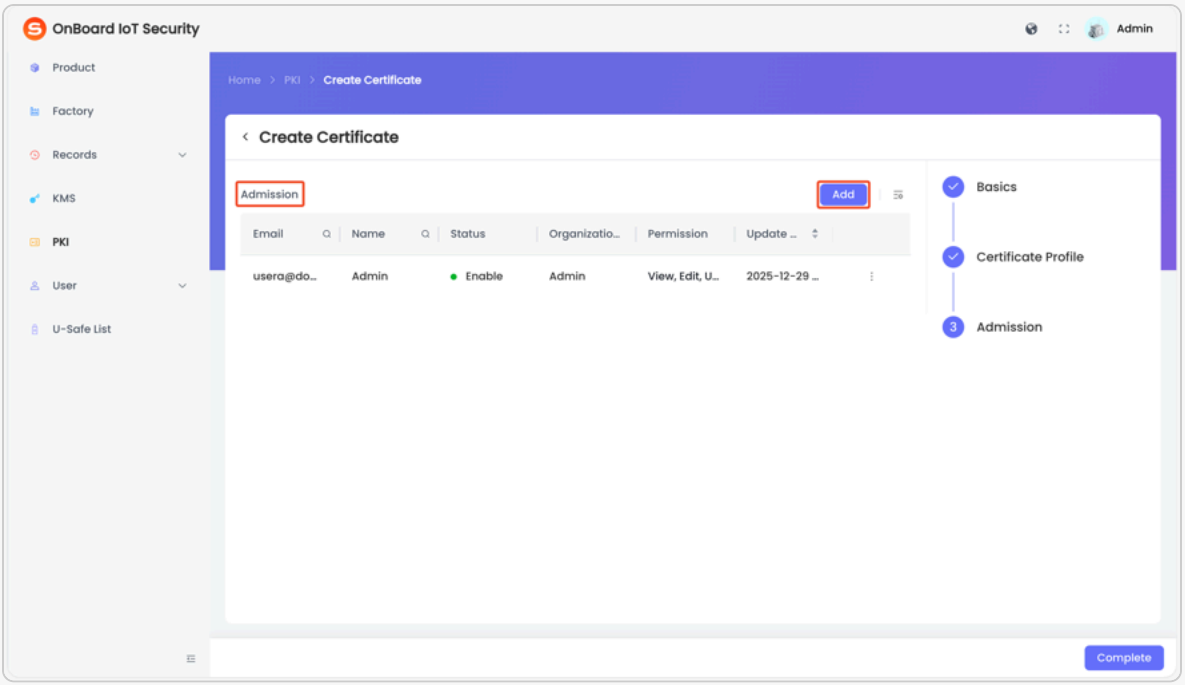
> **TIP**
>
> If the **Critical** checkbox for a certain item is selected, the system must recognize and support that item when processing the certificate; otherwise, the certificate will be rejected.



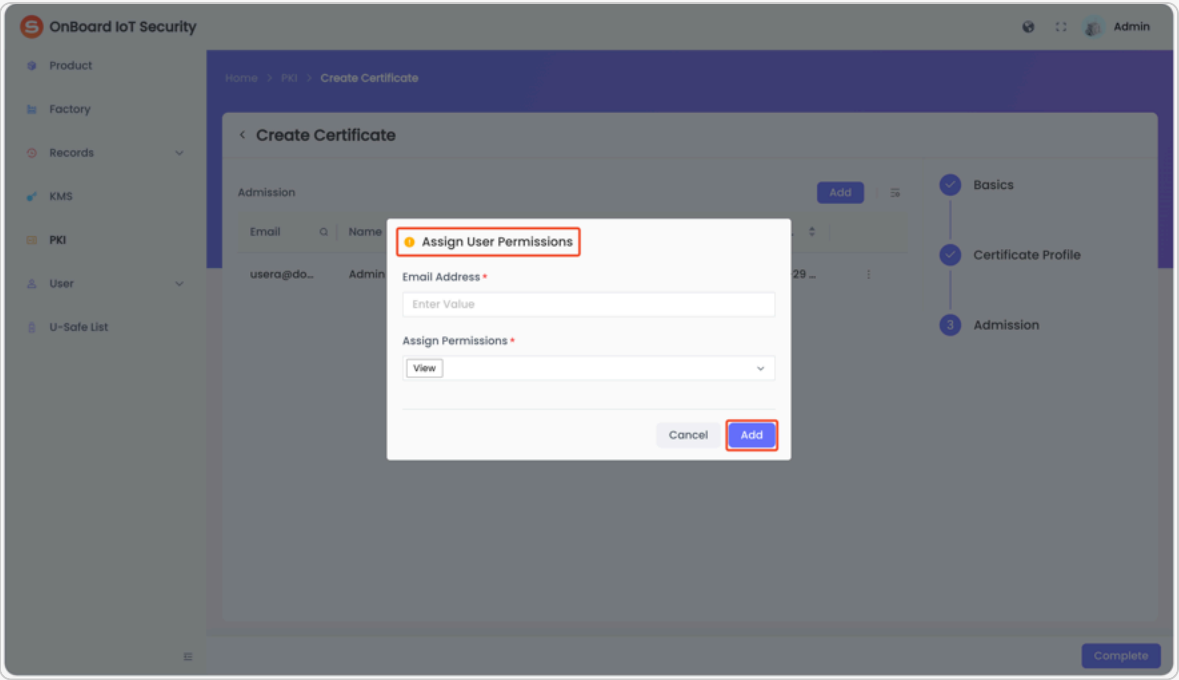
Information	Mandatory	Description
CRL Distribution Points	No	After selecting **Usage**, a CRL Distribution Point URI and a CRL Issuer must be added.

Information	Mandatory	Description
Freshest CRL (a.k.a. Delta CRL DP)	No	After selecting **Usage**, the Freshest CRL URI must be added.
Authority Information Access	No	After selecting **Usage**, an OCSP Service Locator URI must be added, and specific values must be added to the CA issuer URI.

4. After verifying that the information is correct, click **Next**, and then click **Add** on the **Admission** page.



5. In the **Assign User Permissions** window, enter the email address and assign the corresponding permission (View, Edit, Use, Revoke, Grant, and Delete). Then, click **Add**.



6. Click **Complete** to complete the intermediate certificate creation.

> **TIP**
>
> When creating an intermediate certificate, if it is inherited from your own root certificate, you must complete the CSR (Certificate Signing Request) configuration. Before the CSR configuration is completed, the certificate will remain in the **Pending** status and can't be used.

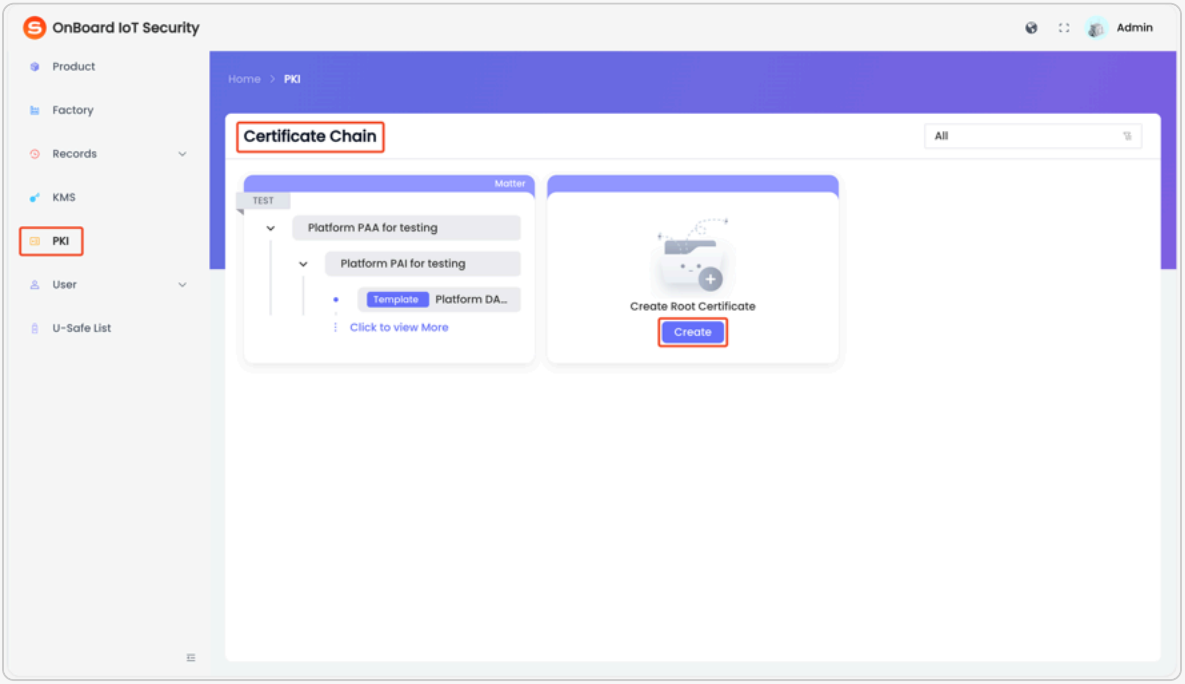
CSR configuration steps for the intermediate certificate

> **TIP**
>
> When creating an intermediate certificate, if it is inherited from your own root certificate, refer to the following steps to complete the CSR configuration.

1. On the **Certificate Chain** page, click the corresponding certificate chain, and then click the corresponding intermediate certificate to navigate to the certificate page.

2. On the certificate page, click the link in the prompt to complete the CSR Configuration.

3. On the **CSR Configuration** page, click the Download icon to download the CSR file. Then, use your Root CA (Certificate Authority) to sign the file, thereby generating an intermediate certificate signed by the Root CA.

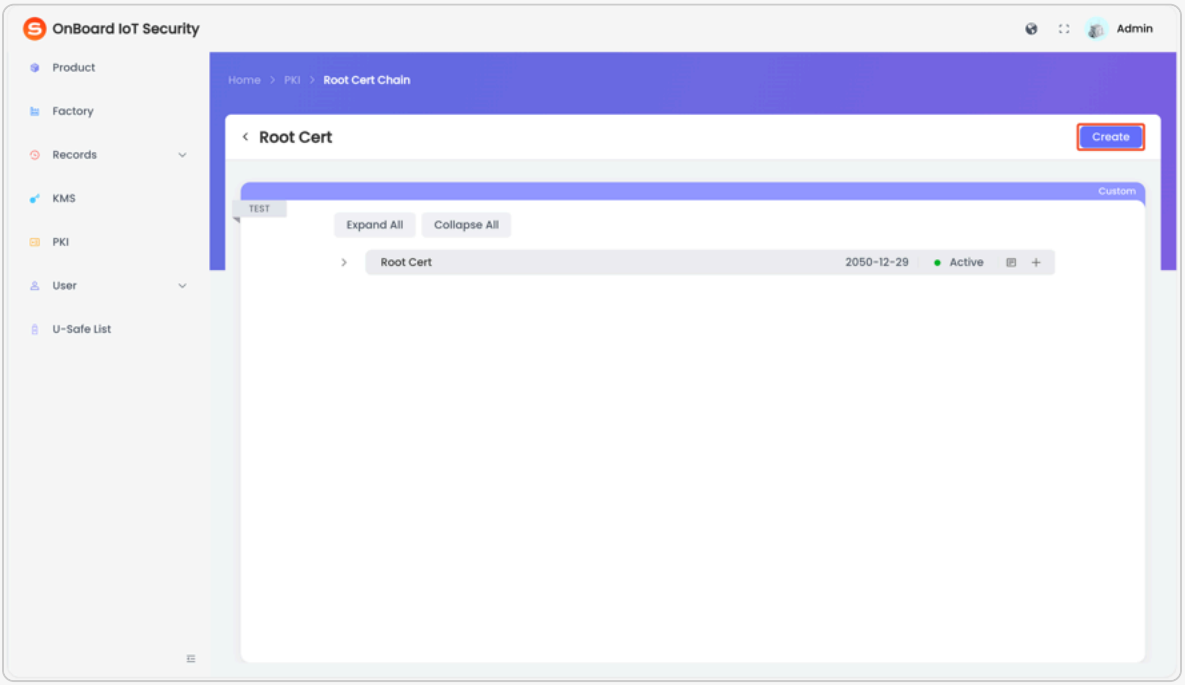


4. After an intermediate certificate is generated, click the Add icon in the upload area, select and upload the intermediate certificate in **.pem** format, or directly drag and drop your intermediate certificate into the upload area.

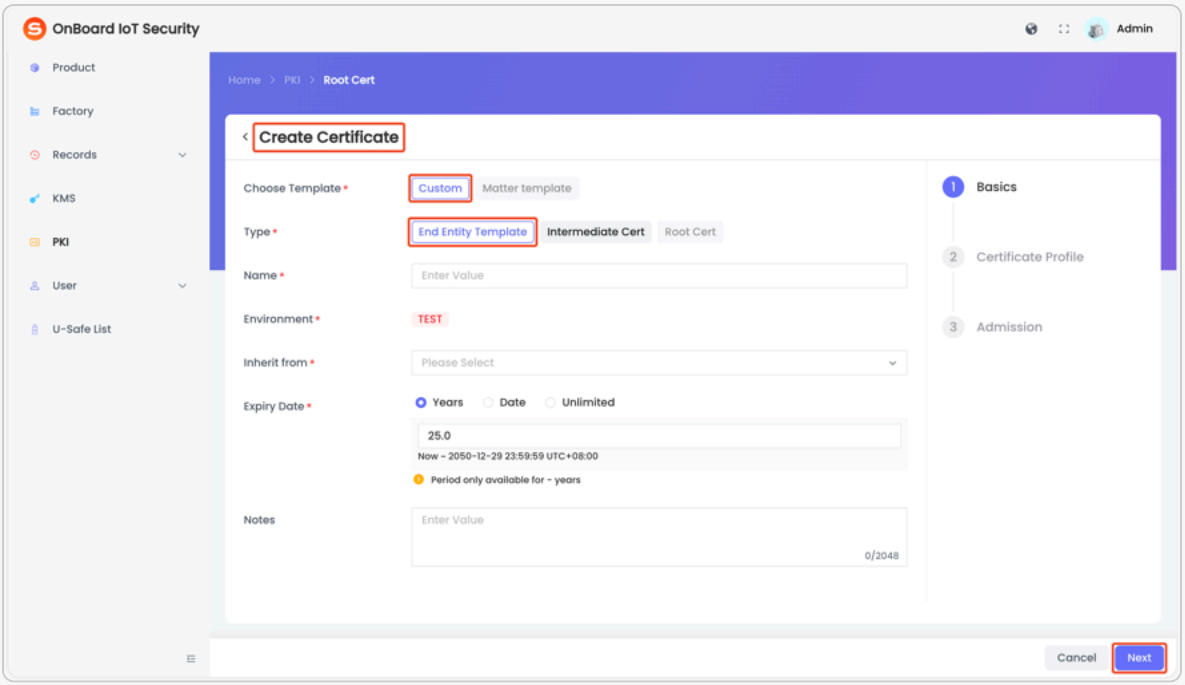
5. Click **Submit**. Once the configuration is completed, the intermediate certificate status will change to **Normal**.

> **INFO**
>
> Before creating an end entity template, please ensure that at least one root certificate or intermediate certificate has been created so that it can be selected and inherited during configuration.

1. On the **Certificate Chain** page, click the corresponding certificate chain, and then click **Create**.



2. On the **Create Certificate** page, select **Custom** for the format and **End Entity Template** for the type, then select and enter the basic information.



Description of end entity template basic information:

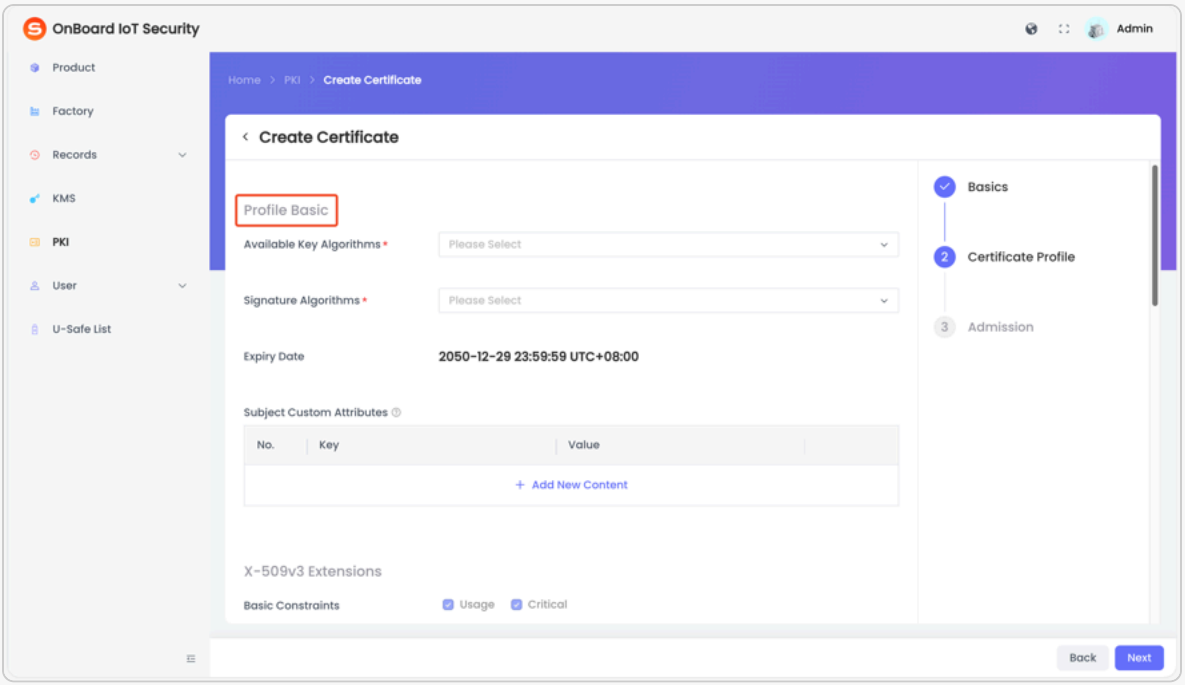
Information	Description
Name	Within the same certificate chain, certificate names at the same hierarchy level must be unique.
Environment	The environment inherited from the root certificate.
Inherit from	Select the root certificate or intermediate certificate to inherit from.

Information	Description
Expiry Date	Set the expiry date of the end entity template. It must not exceed the expiry date of the inherited certificate. **Note**: If the end entity template is inherited from your own root certificate, the expiry date will be inherited from the root certificate.
Notes	Provide additional information related to the end entity template, such as special requirements and important notes.

3. After verifying that the information is correct, click **Next** and configure according to the following information.

> **TIP**
>
> If the end entity template is inherited from your own root certificate, only the **Profile Basic** area is displayed. After configuring the information, you can proceed directly to the next step.

Description of Profile Basic information

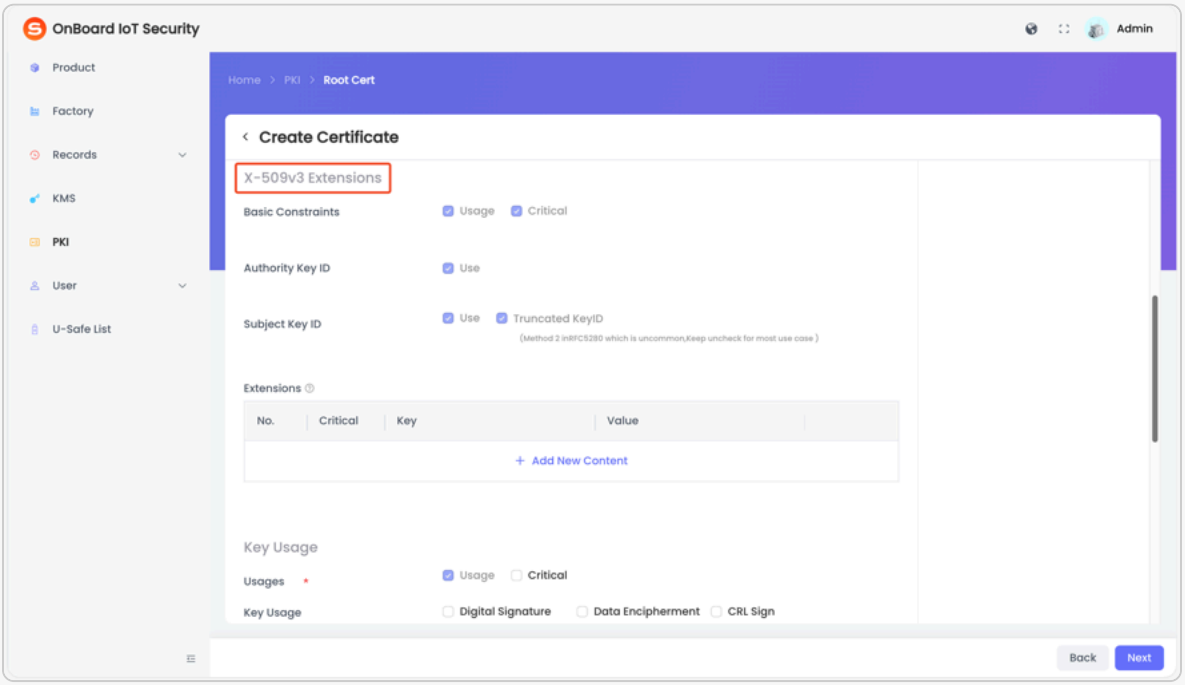


Information	Mandatory	Description
Available Key Algorithms	Yes	Available algorithms for the certificates. Options include: ◦ **RSA_2048** ◦ **RSA_3072** ◦ **RSA_4096** ◦ **EC_secp384r1** ◦ **EC_prime256v1**

Information	Mandatory	Description
Signature Algorithms	Yes	Signature algorithms for the certificates. Options include: ◦ **SHA256WITHRSA** ◦ **SHA384WITHRSA** ◦ **SHA512WITHRSA**
Expiry Date	No	Inherits the Expiry Date set in the previous step and can't be modified.
Subject Custom Attributes	No	Configure subject custom attributes for the certificate by adding specific keys and values.

Description of X-509v3 Extensions information

> **TIP**
>
> If the **Critical** checkbox for a certain item is selected, the system must recognize and support that item when processing the certificate; otherwise, the certificate will be rejected.

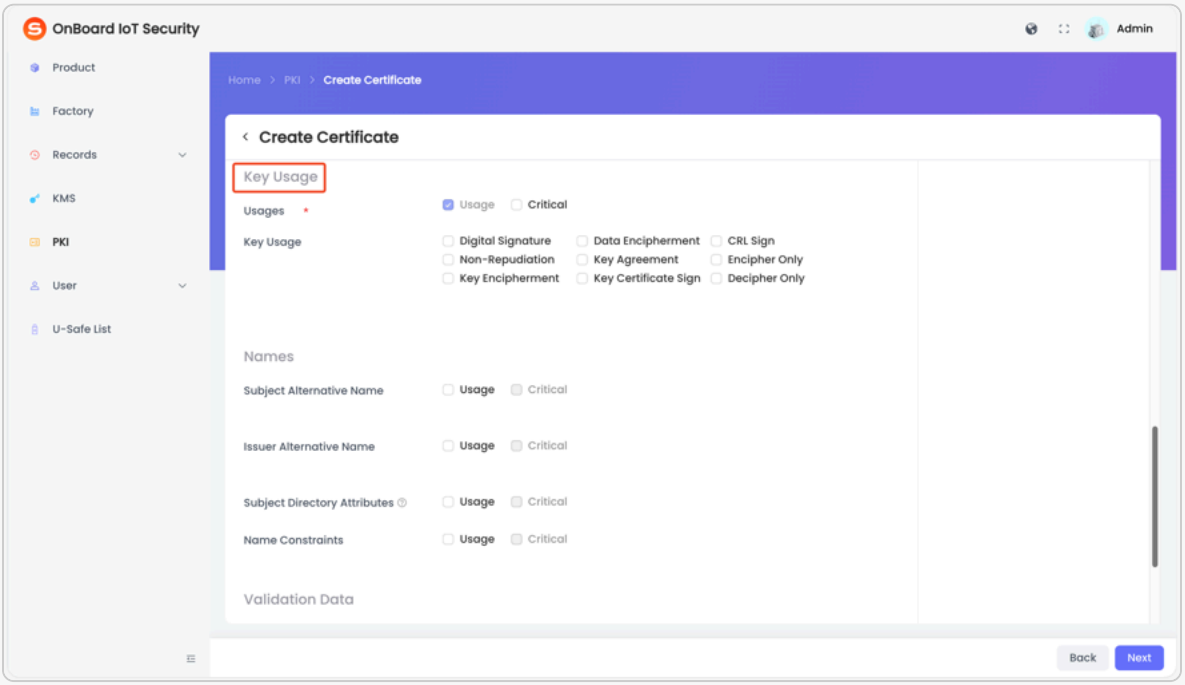


Information	Mandatory	Description
Basic Constraints	No	Automatically parsed from the certificate content and can't be modified.
Authority Key ID	No	Automatically parsed from the certificate content and can't be modified.

Information	Mandatory	Description
Subject Key ID	No	Automatically parsed from the certificate content and can't be modified.
Extensions	No	Configure the X-509v3 extension attributes for the certificate by adding specific keys and values.

Description of Key Usage information

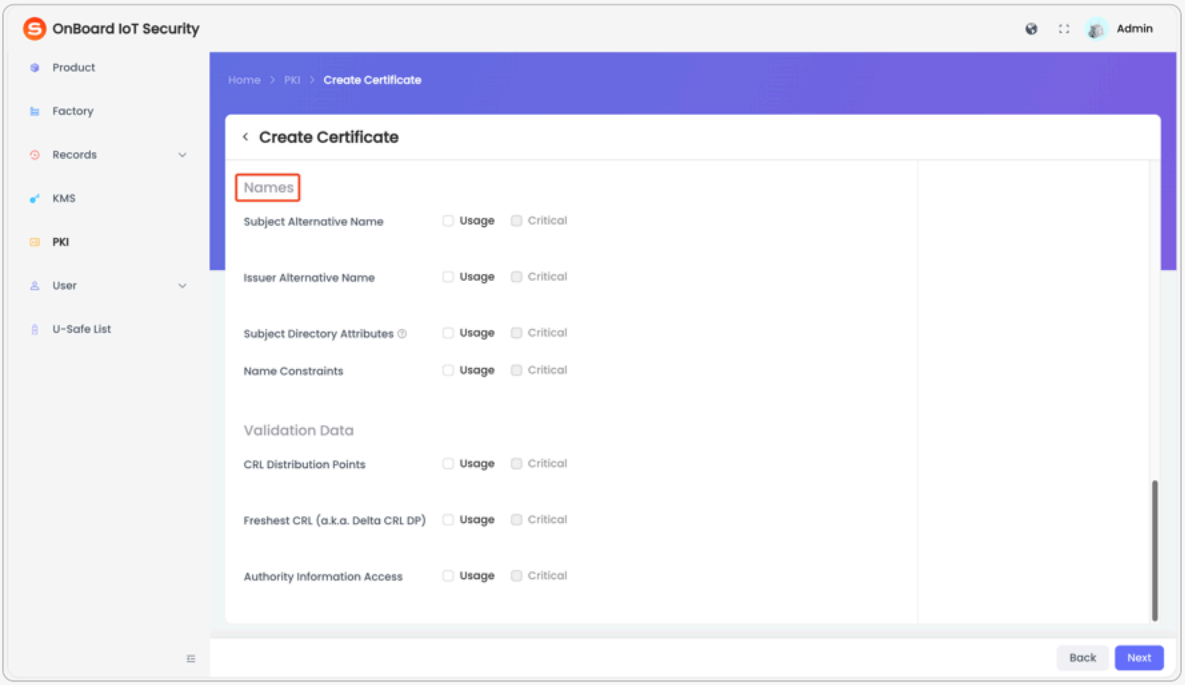
> **TIP**
>
> If the **Critical** checkbox for a certain item is selected, the system must recognize and support that item when processing the certificate; otherwise, the certificate will be rejected.



Information	Mandatory	Description
Usages	Yes	Enabled by default and can't be modified.
Key Usage	Yes	Optional key usages, such as **Digital Signature**, **Data Encipherment**, and **CRL Sign**.

Description of Names information

> **TIP**
>
> If the **Critical** checkbox for a certain item is selected, the system must recognize and support that item when processing the certificate; otherwise, the certificate will be rejected.

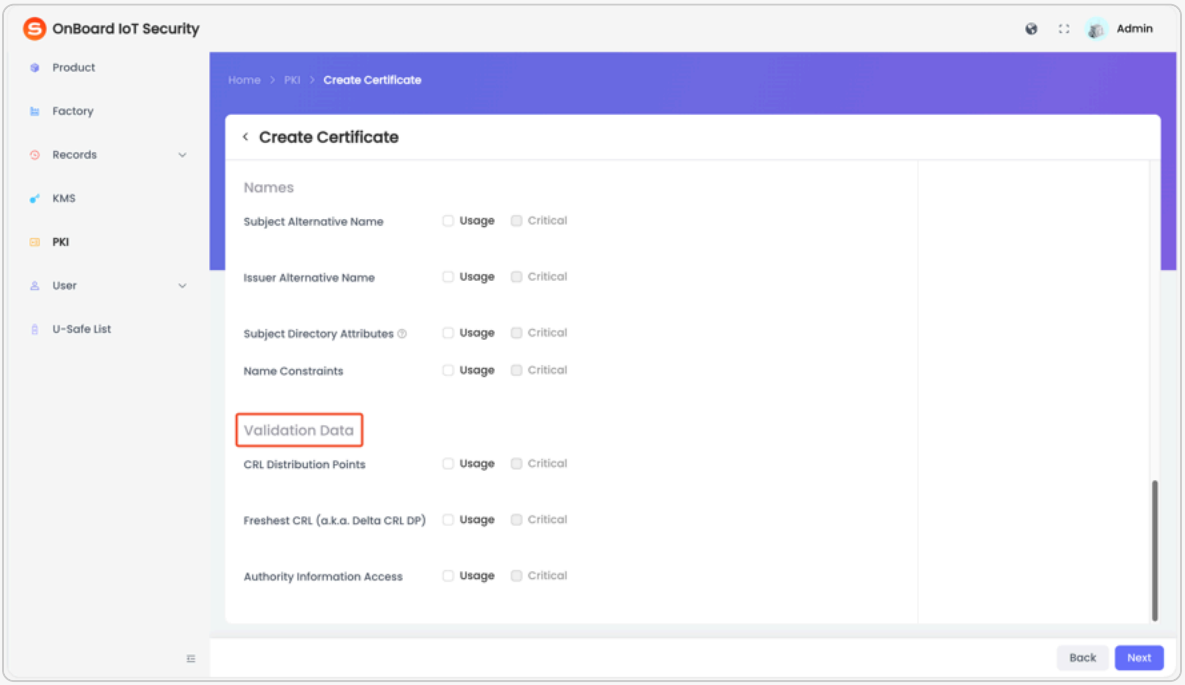


Information	Mandatory	Description
Subject Alternative Name	No	After selecting **Usage**, a subject alternative name must be added.

Information	Mandatory	Description
Issuer Alternative Name	No	After selecting **Usage**, an issuer alternative name must be added.
Subject Directory Attributes	No	After selecting **Usage**, specific keys and values must be added.
Name Constraints	No	After selecting **Usage**, specific values must be added to the Black List.

Description of Validation Data information

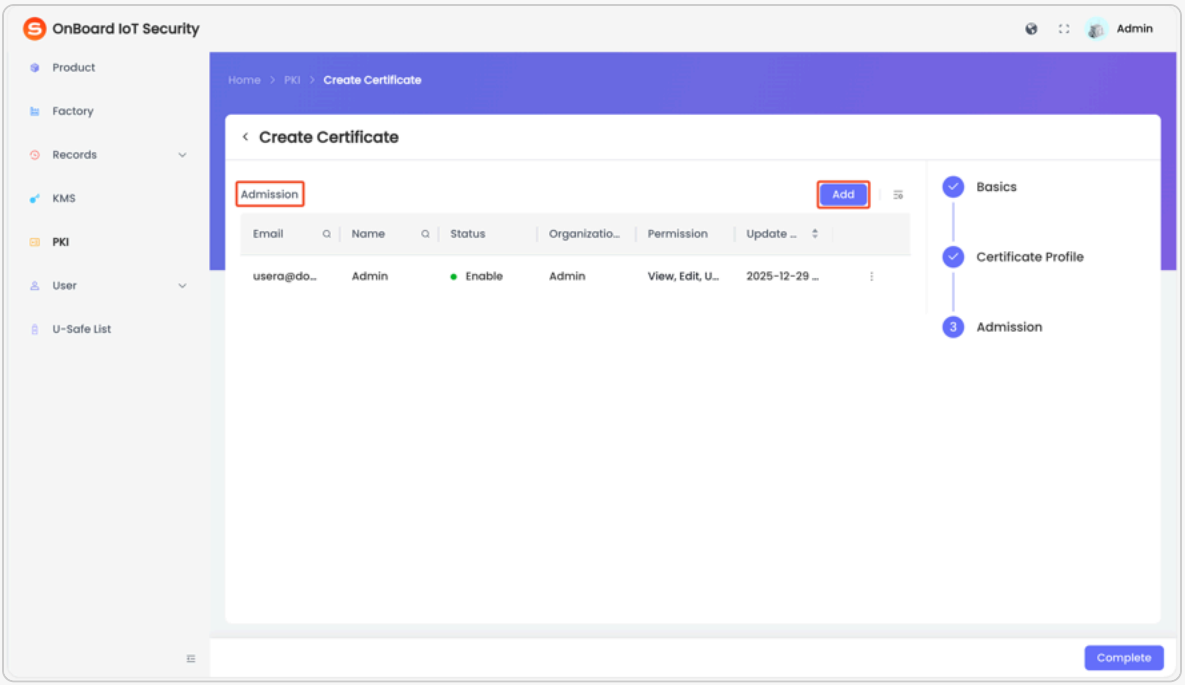
> **TIP**
>
> If the **Critical** checkbox for a certain item is selected, the system must recognize and support that item when processing the certificate; otherwise, the certificate will be rejected.



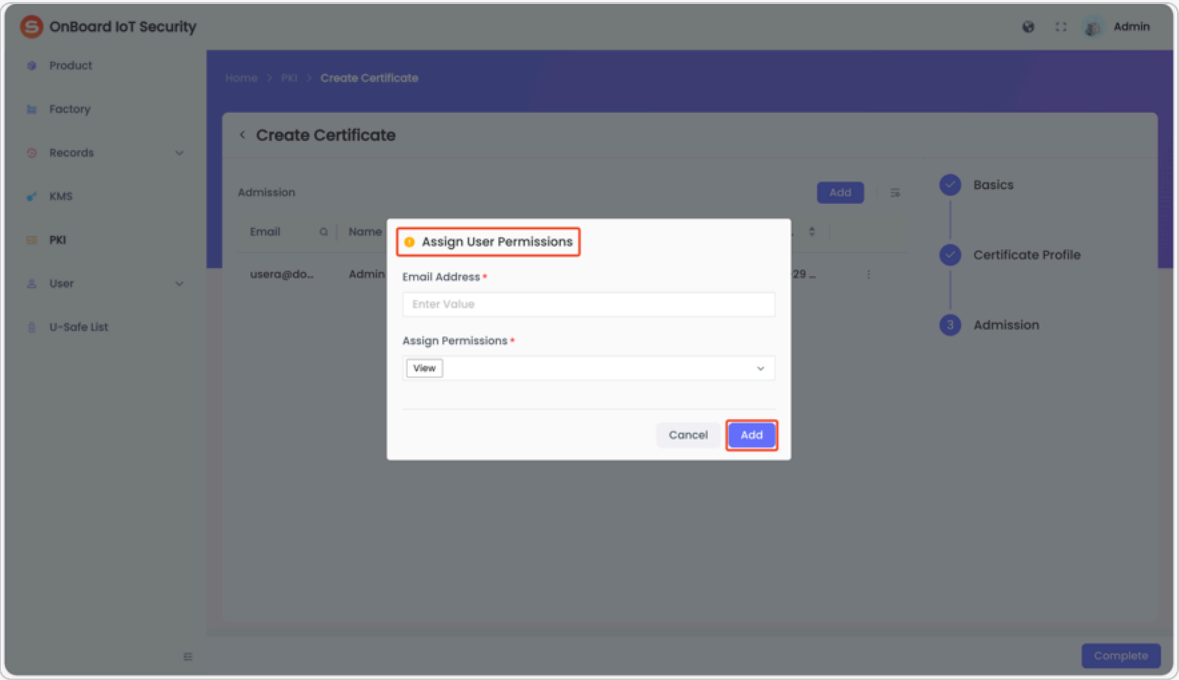
Information	Mandatory	Description
CRL Distribution Points	No	After selecting **Usage**, a CRL Distribution Point URI and a CRL Issuer must be added.

Information	Mandatory	Description
Freshest CRL (a.k.a. Delta CRL DP)	No	After selecting **Usage**, the Freshest CRL URI must be added.
Authority Information Access	No	After selecting **Usage**, an OCSP Service Locator URI must be added, and specific values must be added to the CA issuer URI.

4. After verifying that the information is correct, click **Next**, and then click **Add** on the **Admission** page.



5. In the **Assign User Permissions** window, enter the email address and assign the corresponding permission (View, Edit, Use, Grant, and Delete). Then, click **Add**.



6. Click **Complete** to complete the end entity template creation.

> **TIP**
>
> When creating an end entity template certificate, if it is inherited from your own root certificate, you must complete the CSR (Certificate Signing Request) configuration. Before the CSR configuration is completed, the template will remain in the **Pending** status and can't be used.

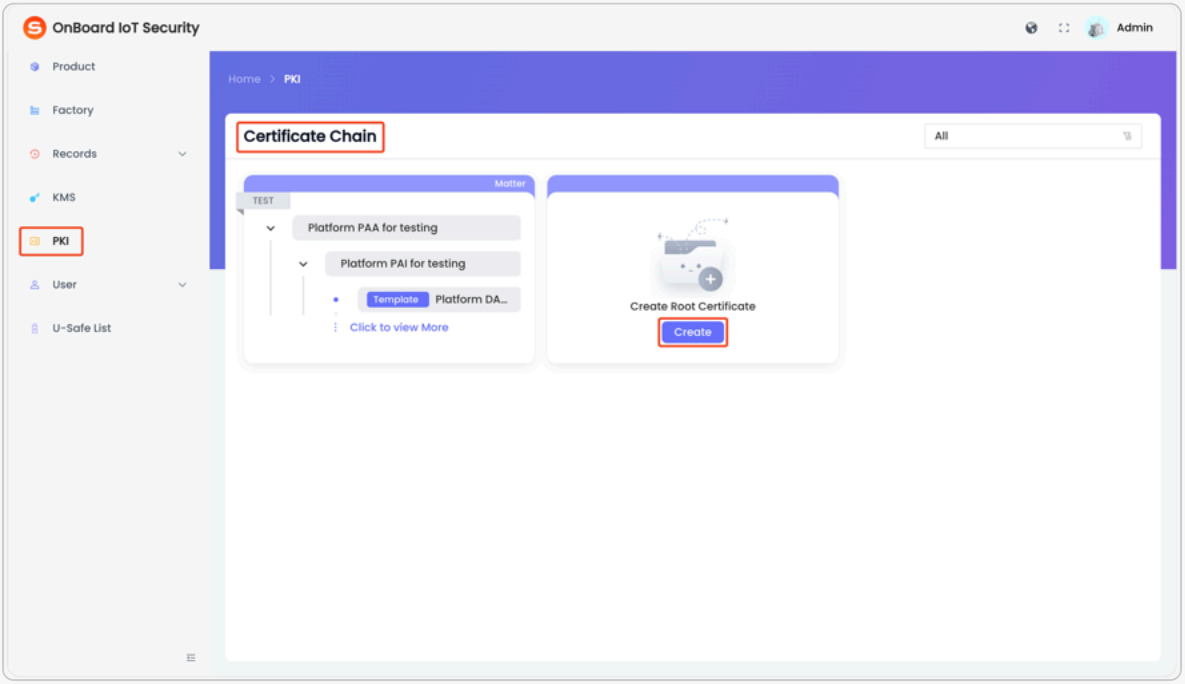
CSR configuration steps for the end entity template

> **TIP**
>
> When creating an end entity template, if it is inherited from your own root certificate, refer to the following steps to complete the CSR configuration.

1. On the **Certificate Chain** page, click the corresponding certificate chain, and then click the corresponding end entity template to navigate to the certificate page.

2. On the certificate page, click the link in the prompt to complete the CSR Configuration.

3. On the **CSR Configuration** page, click the Download icon to download the CSR file. Then, use your Root CA (Certificate Authority) to sign the file, thereby generating an end entity template signed by the Root CA.



4. After an end entity template is generated, click the Add ➕ icon in the upload area, select and upload the end entity template in **.pem** format, or directly drag and drop your end entity template into the upload area.

5. Click **Submit**. Once the configuration is completed, the end entity template status will change to **Normal**.

Create a certificate based on the Matter template

> **TIP**
>
> After a certificate or certificate template based on the Matter template is created, it can be selected when configuring Certificate for a product.

Create a PAA certificate based on the Matter template

Create a PAI certificate based on the Matter template

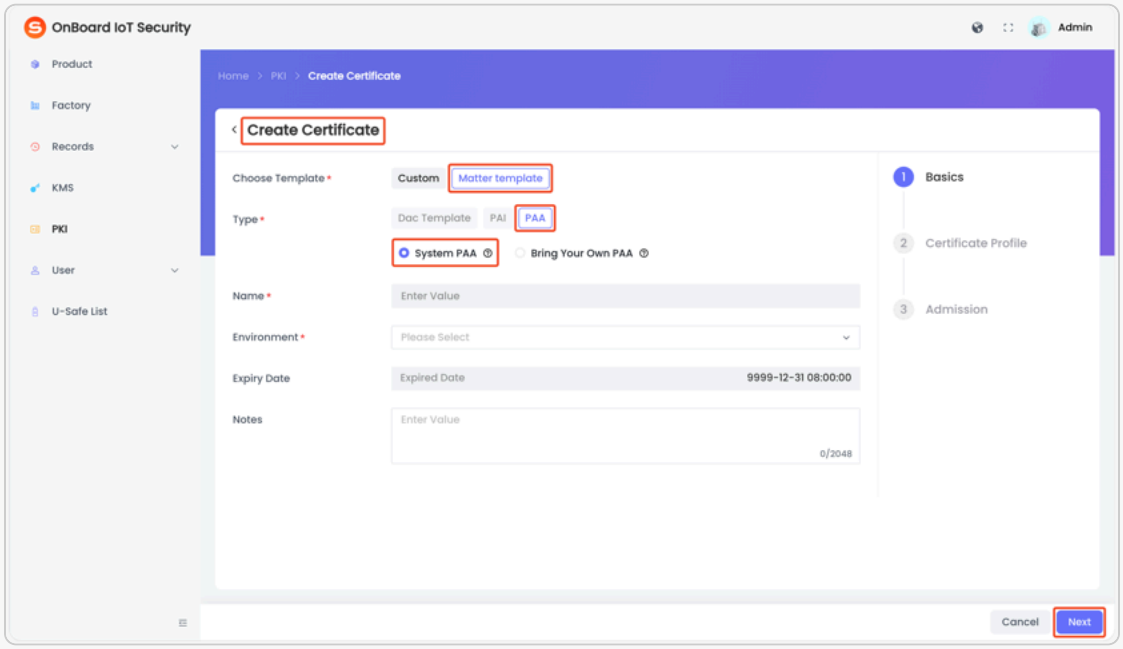
Create a DAC template based on the Matter template

1. On the **Create Certificate** page, select **Matter template** for the format and **PAA** for the type.

2. Select **System PAA** or **Bring Your Own PAA** as needed.

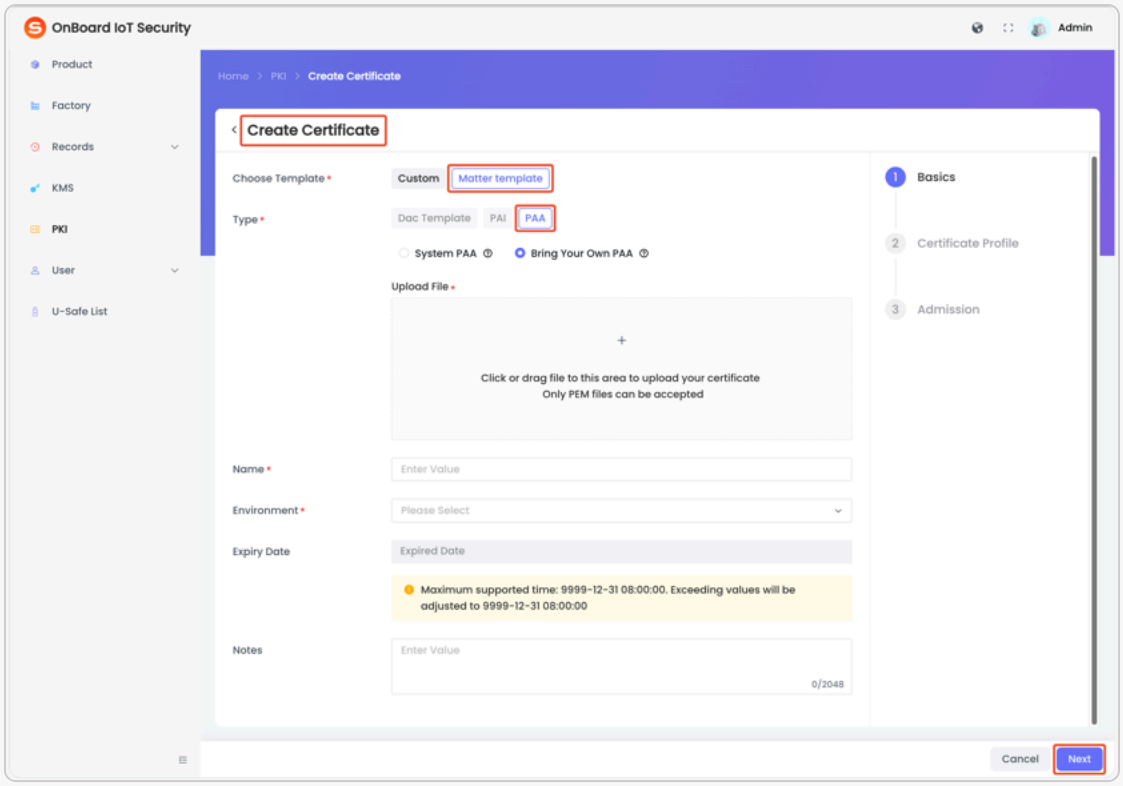
    - If you are not using your own PAA certificate based on the Matter template, select **System PAA**, then select and enter the basic information.

> **TIP**
>
> Use the PAA provided by Snowball.



- If you are using your own PAA certificate based on the Matter template, select **Bring Your Own PAA**, then select and enter the basic information.

> **TIP**
>
> Manually upload your own PAA certificate (with public key only) generated outside the system for verifying signature.



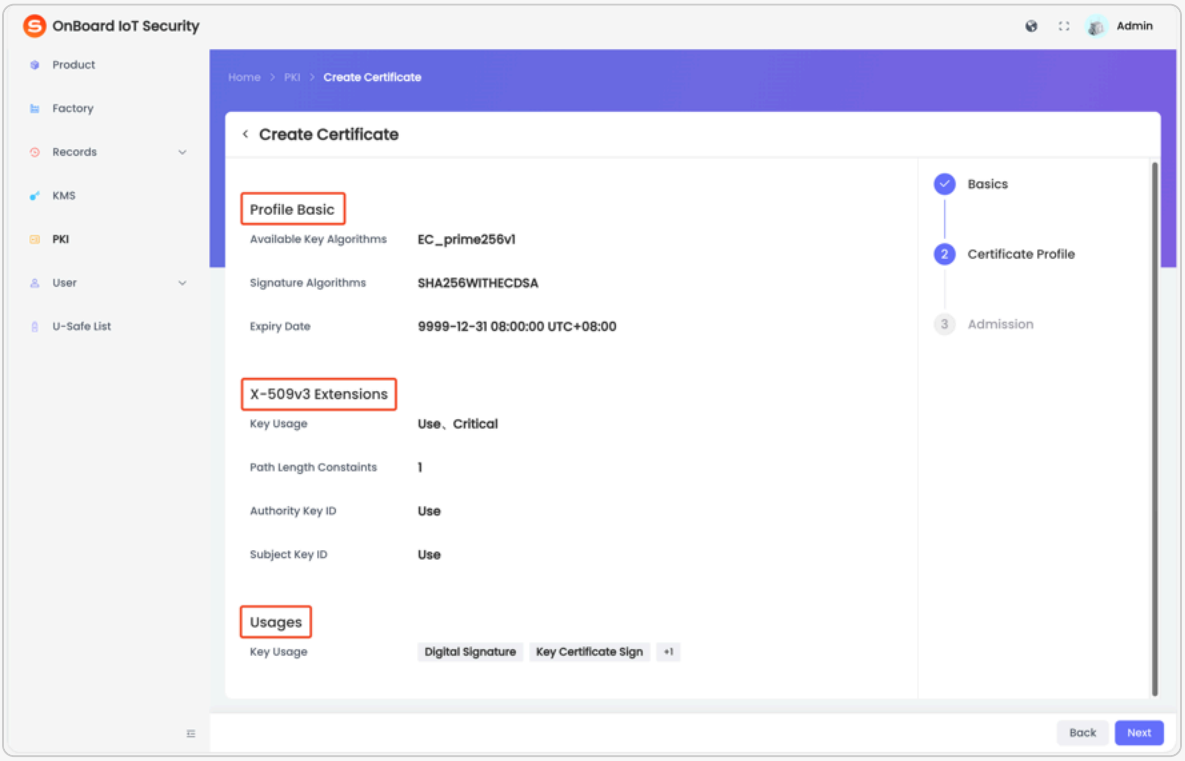
Description of PAA certificate basic information:

Information	System PAA	Bring Your Own PAA
Upload file	No file upload is required.	in the **Upload File** area, click the Add ➕ icon to select and upload a PAA certificate in **.pem** format, or directly drag your own PAA certificate into the upload area.

Information	System PAA	Bring Your Own PAA
Name	The name is fixed and can't be modified. ▪ TEST environment: Snowball Matter UAT PAA 01 ▪ PROD environment: Snowball Matter PROD PAA 01	The PAA certificate name must be unique in the same environment.
Environment	Two environments can be selected: **TEST** and **PROD**.	
Expiry Date	The expiry date of the System PAA certificate and can't be modified.	Automatically parsed from the certificate content and can't be modified.
Notes	Provide additional information related to the PAA certificate, such as special requirements and important notes.	

3. After verifying that the information is correct, click **Next** and configure according to the following information.

> **TIP**
>
> If using a private PAA certificate, the page only displays the **Profile Basic** area.

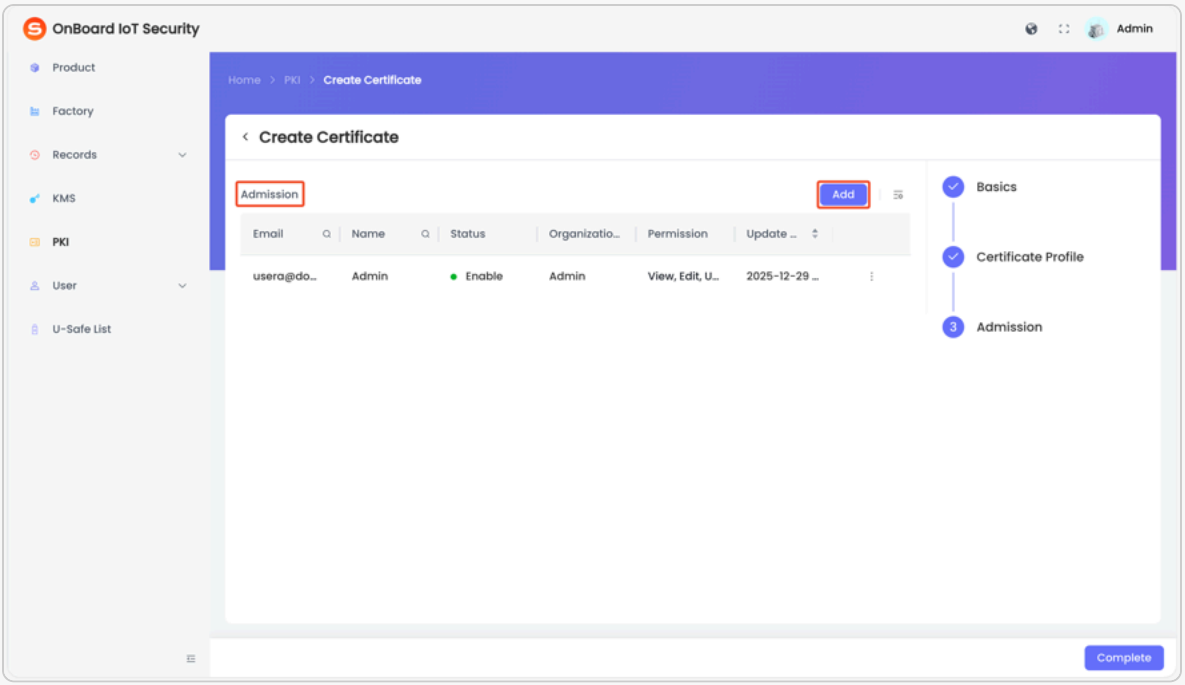


Description of PAA certificate profile information:

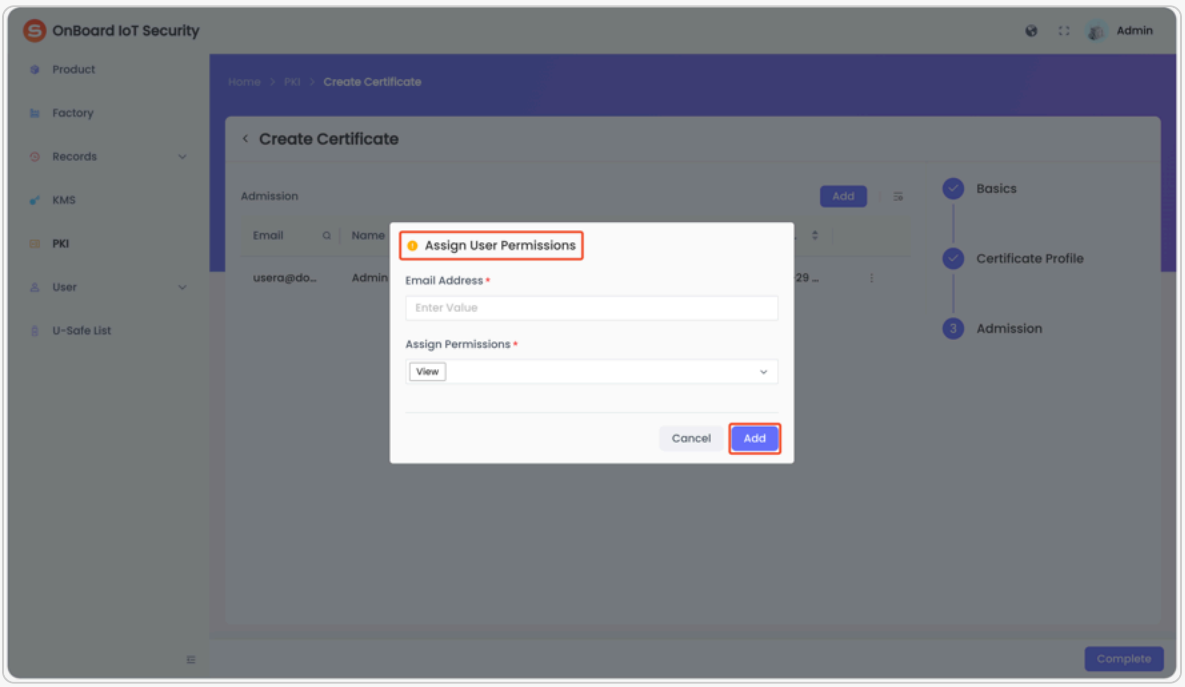
Area	Information	Description
Profile Basic	Available Key Algorithms	Automatically parsed from the certificate content and can't be modified.

Area	Information	Description
	Signature Algorithms	Automatically parsed from the certificate content and can't be modified.
	Expiry Date	Automatically parsed from the certificate content and can't be modified.
X-509v3 Extensions	Key Usage	Automatically parsed from the certificate content and can't be modified.
	Path Length Constaints	Automatically parsed from the certificate content and can't be modified.
	Authority Key ID	Automatically parsed from the certificate content and can't be modified.
	Subject Key ID	Automatically parsed from the certificate content and can't be modified.
Usages	Key Usage	Optional key usages, such as **Digital Signature**, **Key Certificate Sign**, and **CRL Sign**.

4. After verifying that the information is correct, click **Next**, and then click **Add** on the **Admission** page.



5. In the **Assign User Permissions** window, enter the email address and assign corresponding permission (View, Edit, Use, Revoke, Grant, and Delete). Then, click **Add**.

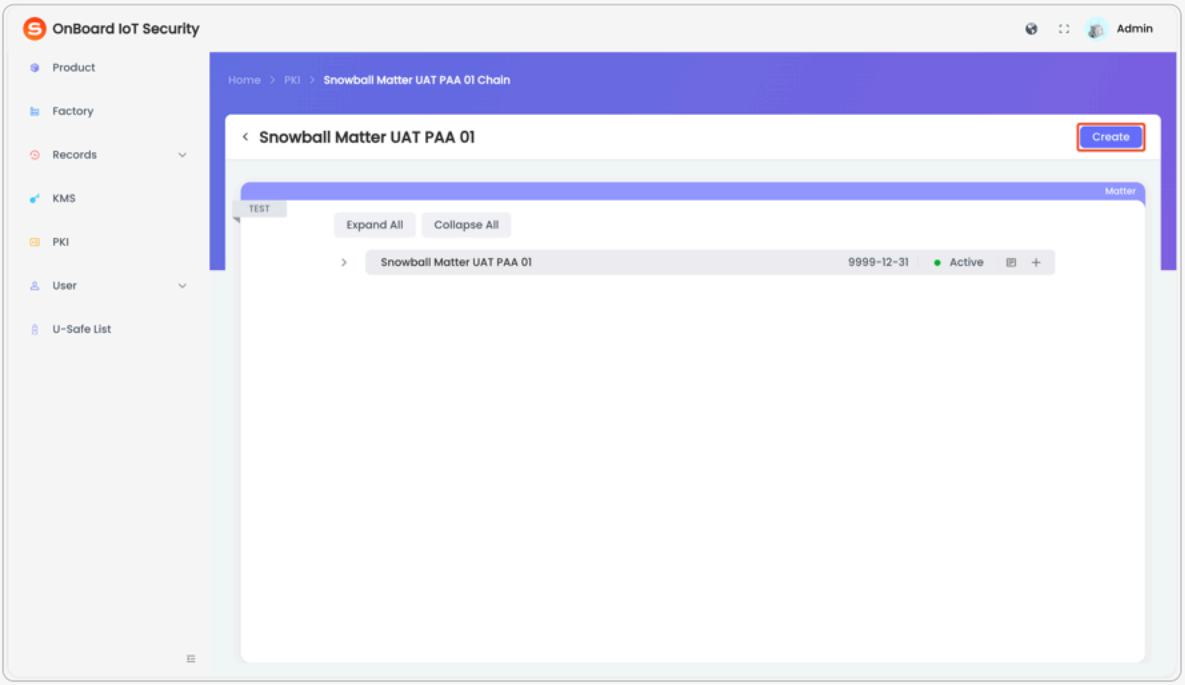


6. Click **Complete** to complete the PAA certificate creation.

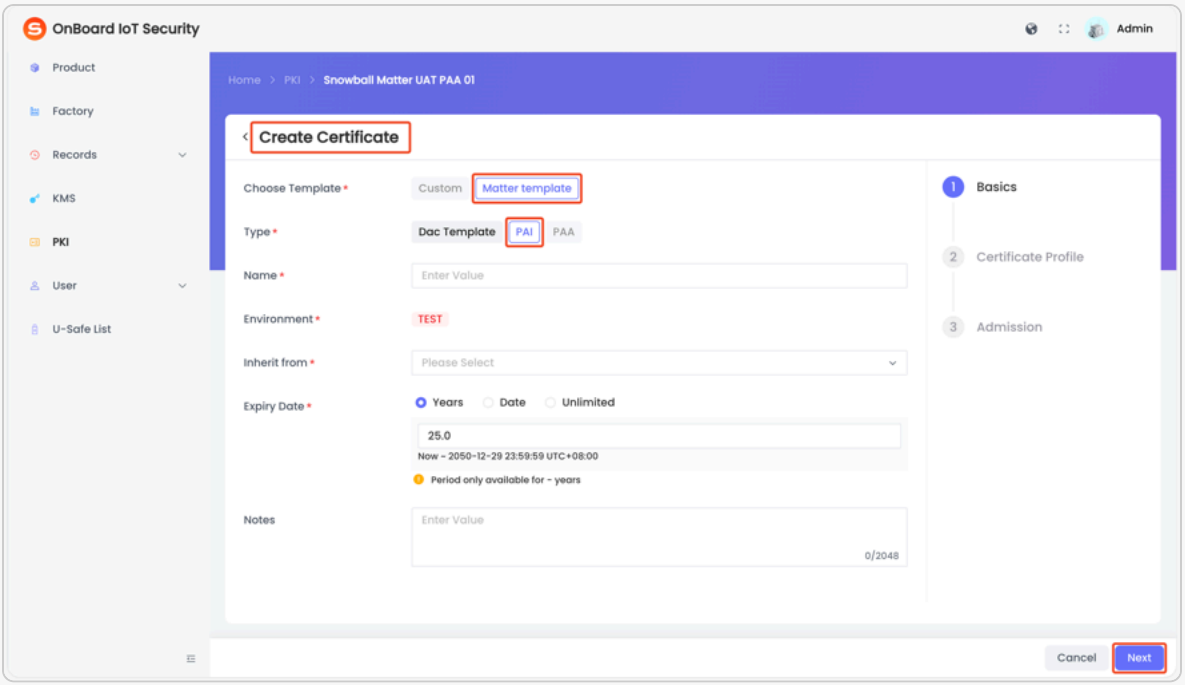
INFO

Before creating a PAI certificate based on the Matter template, please ensure that at least one PAA certificate based on the Matter template has been created so that it can be selected and inherited during configuration.

1. On the **Certificate Chain** page, click the corresponding certificate chain, and then click **Create**.



2. On the **Create Certificate** page, select **Matter template** for the format and **PAI** for the type, then select and enter the basic information.



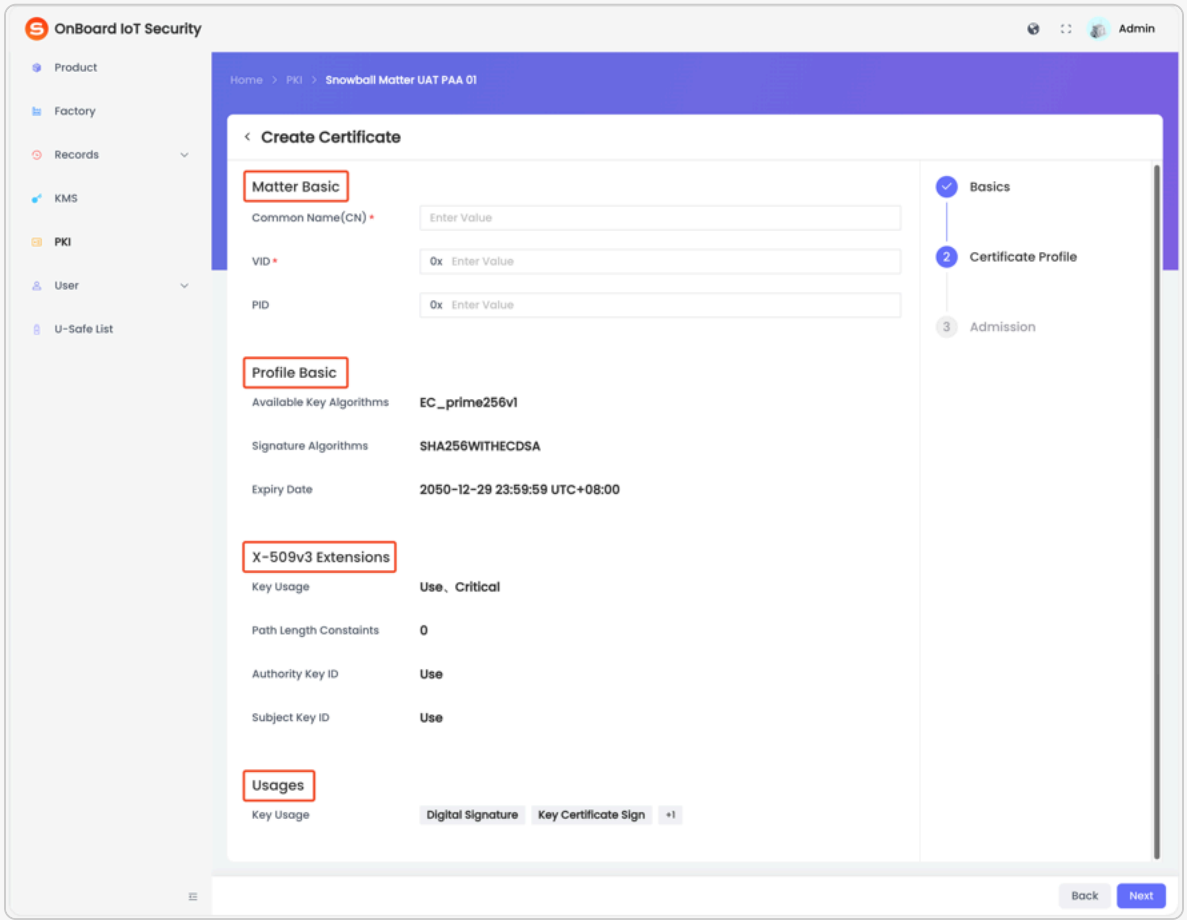
Description of PAI certificate basic information:

Information	Description
Name	Within the same certificate chain, certificate names at the same hierarchy level must be unique.
Environment	The environment inherited from the PAA certificate.
Inherit from	Select the PAA certificate to inherit from.
Expiry Date	Set the expiry date of the PAI certificate. It must not exceed the expiry date of the inherited PAA certificate. **Note**: If the PAI certificate is inherited from your own PAA certificate, the expiry date will be inherited from the PAA certificate.

Information	Description
Notes	Provide additional information related to the PAI certificate, such as special requirements and important notes.

3. After verifying that the information is correct, click **Next** and configure according to the following information.

> **TIP**
>
> If the PAI certificate is inherited from your own PAA certificate, only the **Matter Basic** and **Profile Basic** areas are dispalyed.

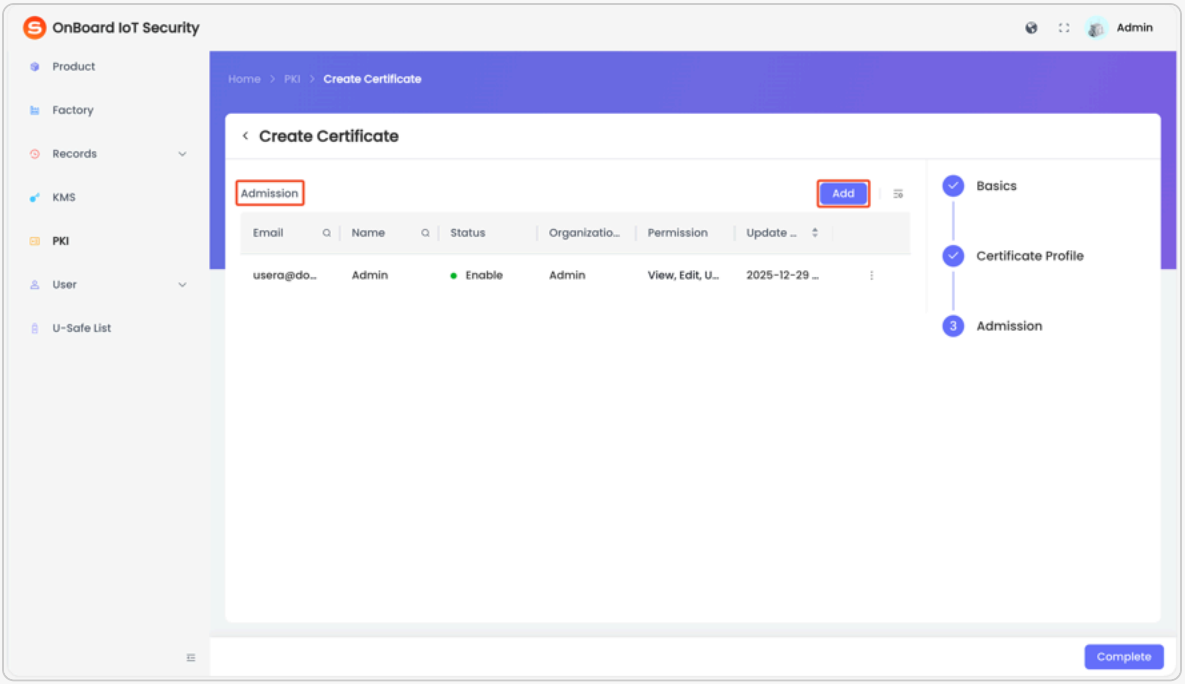


Description of PAI certificate profile information:

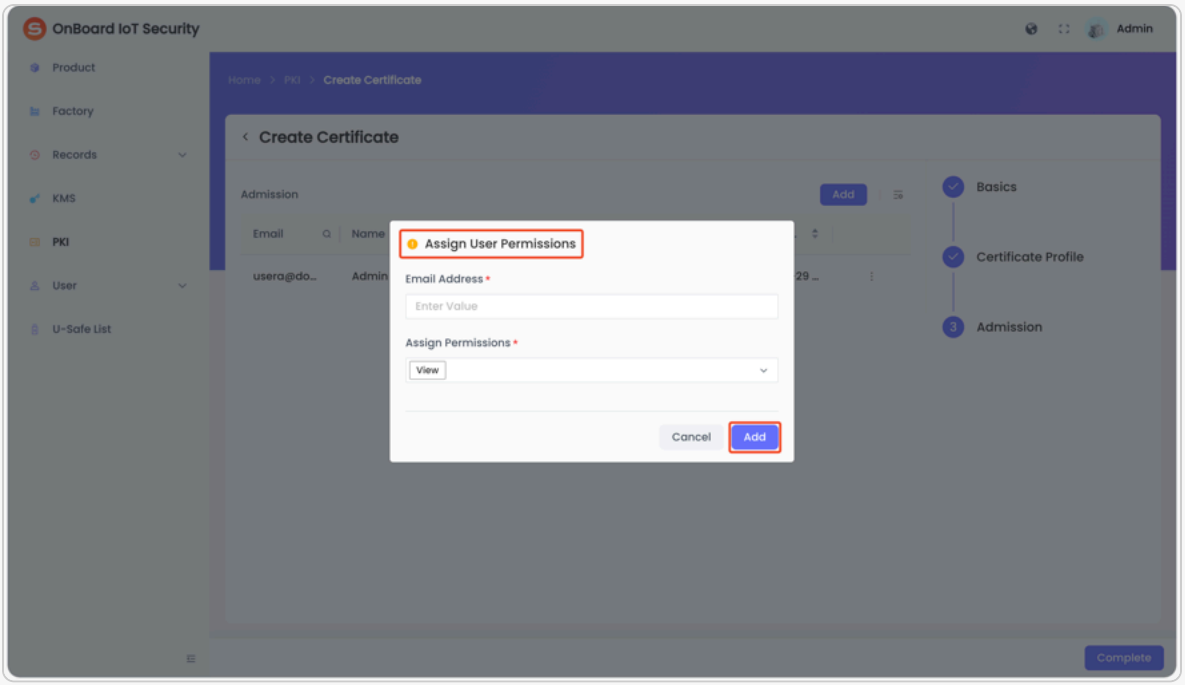
Area	Information	Description
Matter Basic	Common Name (CN) (Required)	The common name of the certificate.
	VID (Required)	Vendor ID.
	PID (optional)	Product ID.
Profile Basic	Available Key Algorithms	Automatically parsed from the certificate content and can't be modified.
	Signature Algorithms	Automatically parsed from the certificate content and can't be modified.
	Expiry Date	Inherits the Expiry Date set in the previous step and can't be modified.
X-509v3 Extensions	Key Usage	Automatically parsed from the certificate content and can't be modified.
	Path Length Constaints	Automatically parsed from the certificate content and can't be modified.
	Authority Key ID	Automatically parsed from the certificate content and can't be modified.

Area	Information	Description
	Subject Key ID	Automatically parsed from the certificate content and can't be modified.
Usages	Key Usage	Optional key usages, such as **Digital Signature**, **Key Certificate Sign**, and **CRL Sign**.

4. After verifying that the information is correct, click **Next**, and then click **Add** on the **Admission** page.



5. In the **Assign User Permissions** window, enter the email address and assign the corresponding permission (View, Edit, Use, Revoke, Grant, and Delete). Then, click **Add**.



6. Click **Complete** to complete the PAI certificate creation.

> **TIP**
>
> When creating a PAI certificate, if it is inherited from your own PAA certificate, you must complete the CSR (Certificate Signing Request) configuration. Before the CSR configuration is completed, the certificate will remain in the Pending status and can't be used.

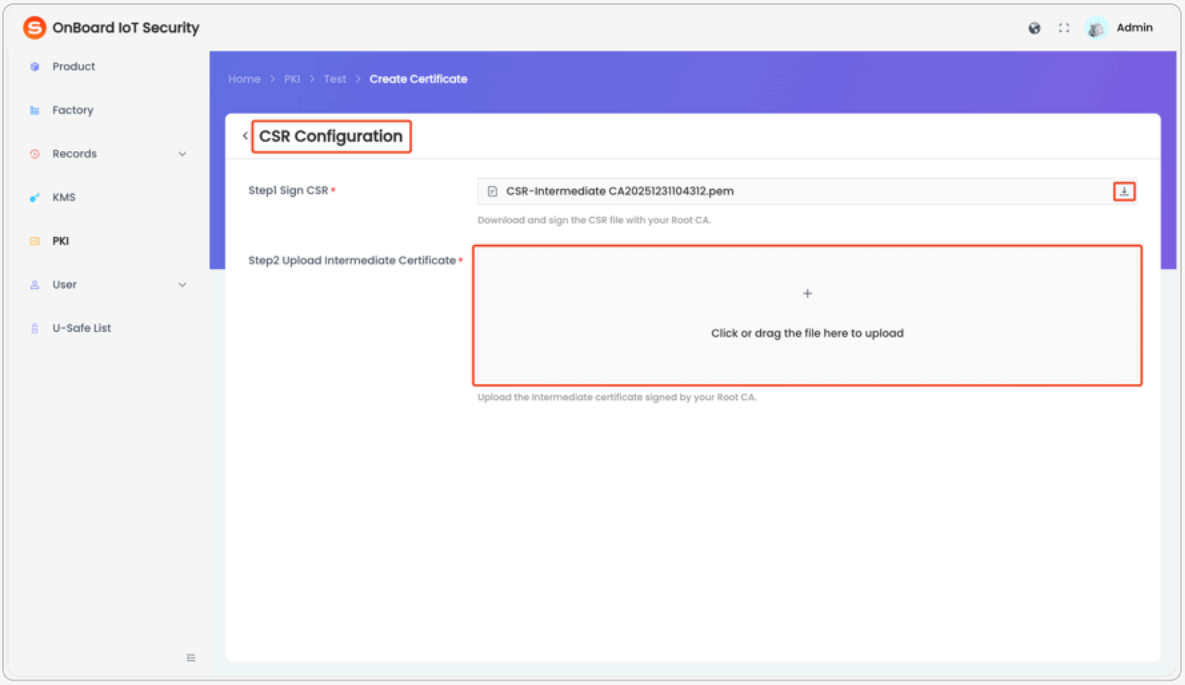
CSR configuration steps for the PAI certificate

> **TIP**
>
> When creating a PAI certificate, if it is inherited from your own PAA certificate, refer to the following steps to complete the CSR configuration.

1. On the **Certificate Chain** page, click the corresponding certificate chain, and then click the corresponding PAI certificate to navigate to the certificate page.

2. On the certificate page, click the link in the prompt to complete the CSR Configuration.

3. On the **CSR Configuration** page, click the Download icon to download the CSR file. Then, use your Root CA to sign the file, thereby generate a PAI certificate signed by the Root CA.



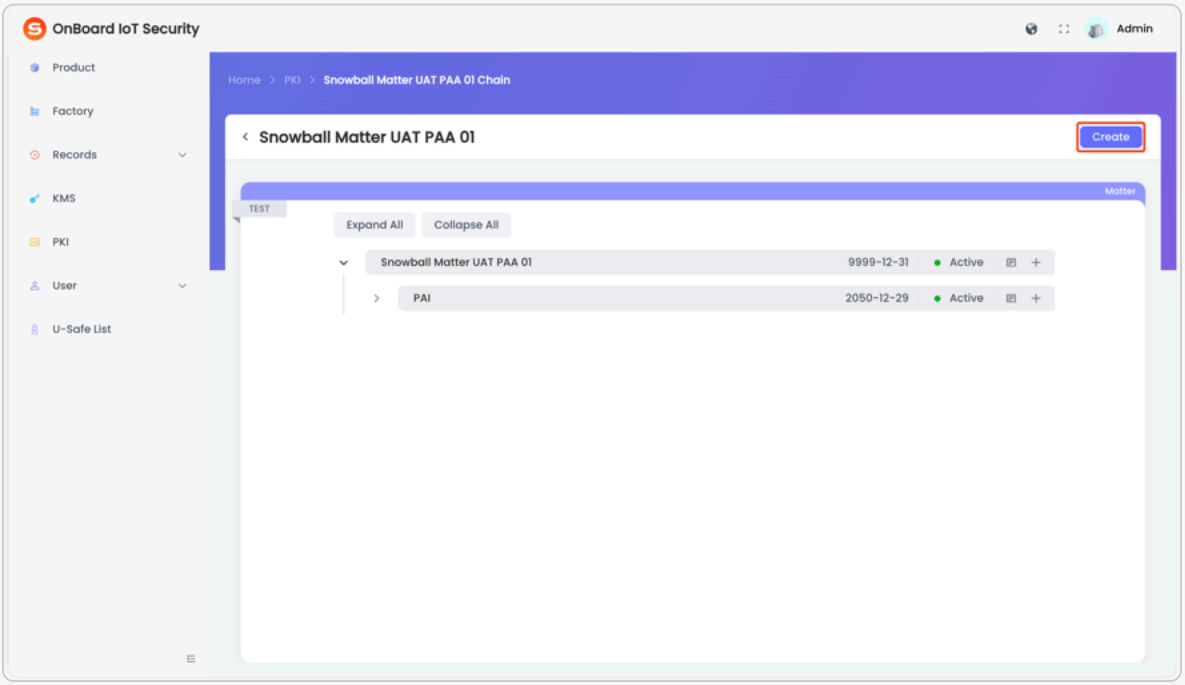
4. After a PAI certificate is generated, click the Add icon in the upload area, select and upload the PAI certificate in **.pem** format, or directly drag and drop your PAI certificate into the upload area.

5. Click **Submit**. Once the configuration is completed, the PAI certificate status will change to **Normal**.

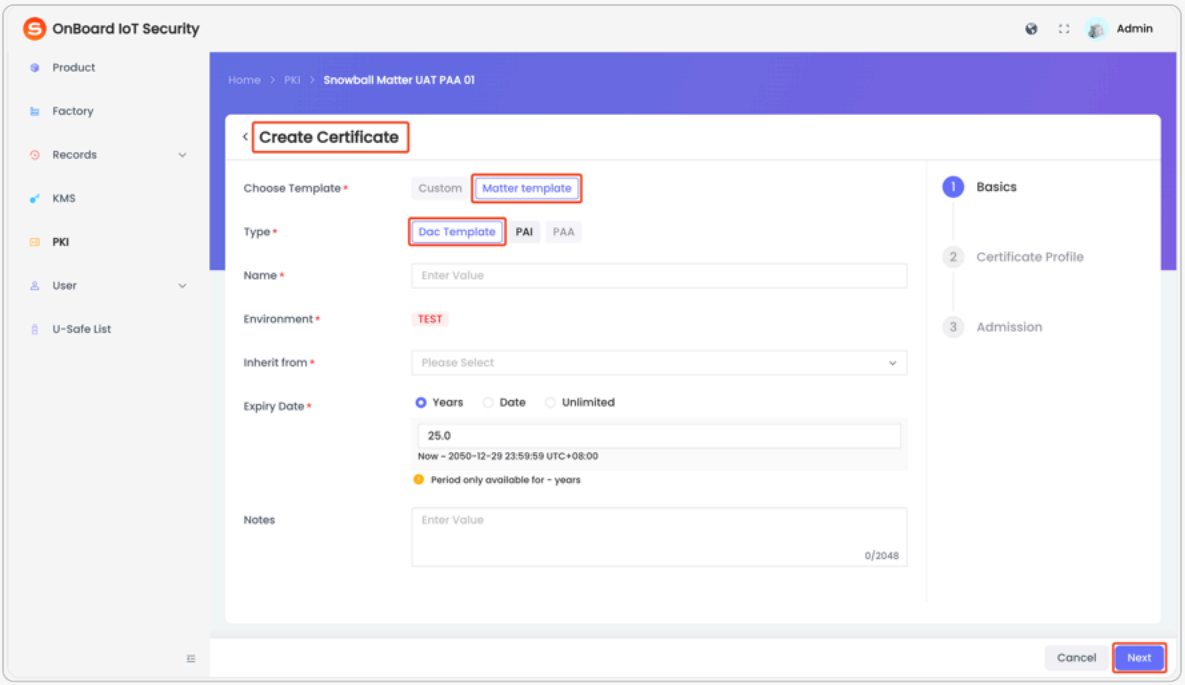
INFO

Before creating a DAC template, please ensure that at least one PAI certificate based on the Matter template has been created so that it can be selected and inherited during configuration.

1. On the **Create Certificate** page, click the corresponding certificate chain, and then click **Create**.



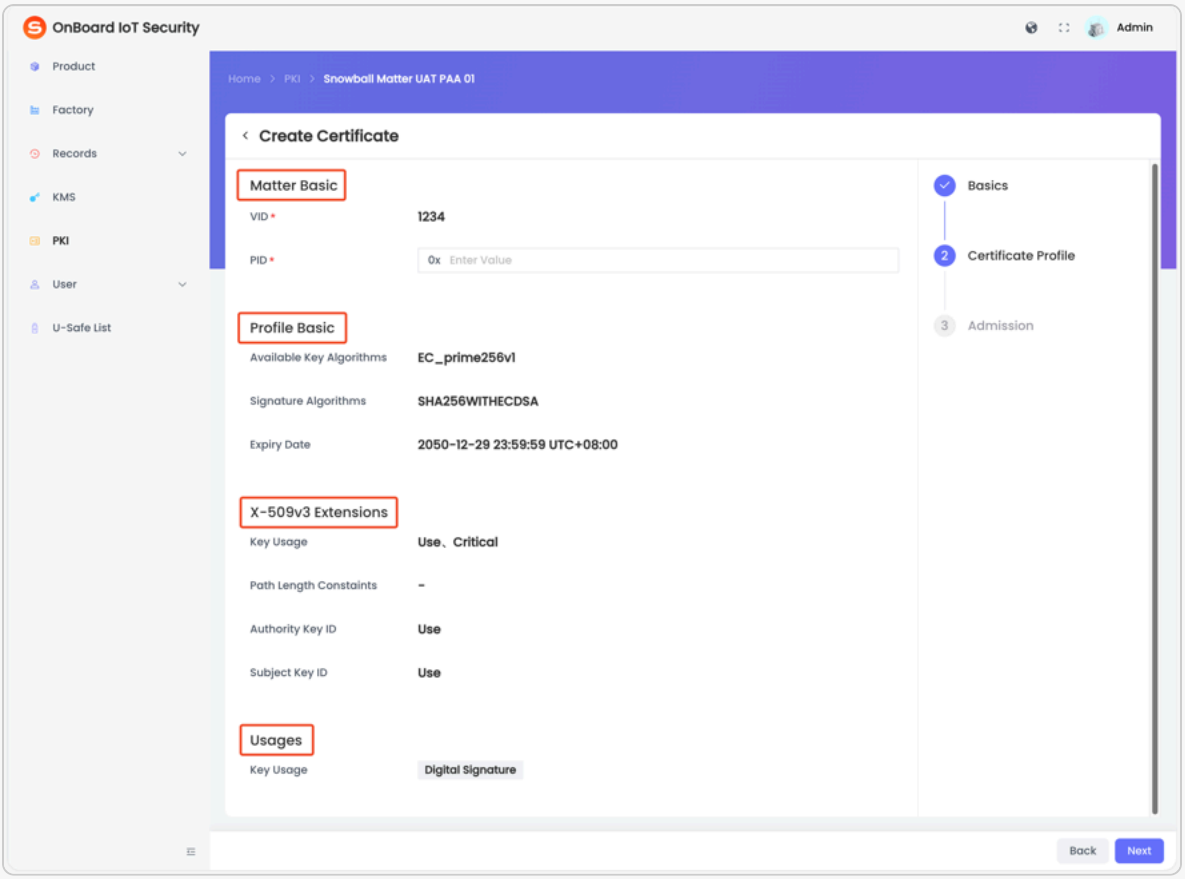
2. On the **Create Certificate** page, select **Matter template** for the format and **Dac Template** for the type, then select and enter the basic information.



Description of DAC template basic information:

Information	Description
Name	Within the same certificate chain, certificate names at the same hierarchy level must be unique.
Environment	The environment inherited from the PAA certificate.
Inherit from	Select the PAI certificate to inherit from.
Expiry Date	Set the expiry date of the DAC template. It must not exceed the expiry date of the inherited PAI certificate.
Notes	Provide additional information related to the DAC template, such as special requirements and important notes.

3. After verifying that the information is correct, click **Next** and configure according to the following information.



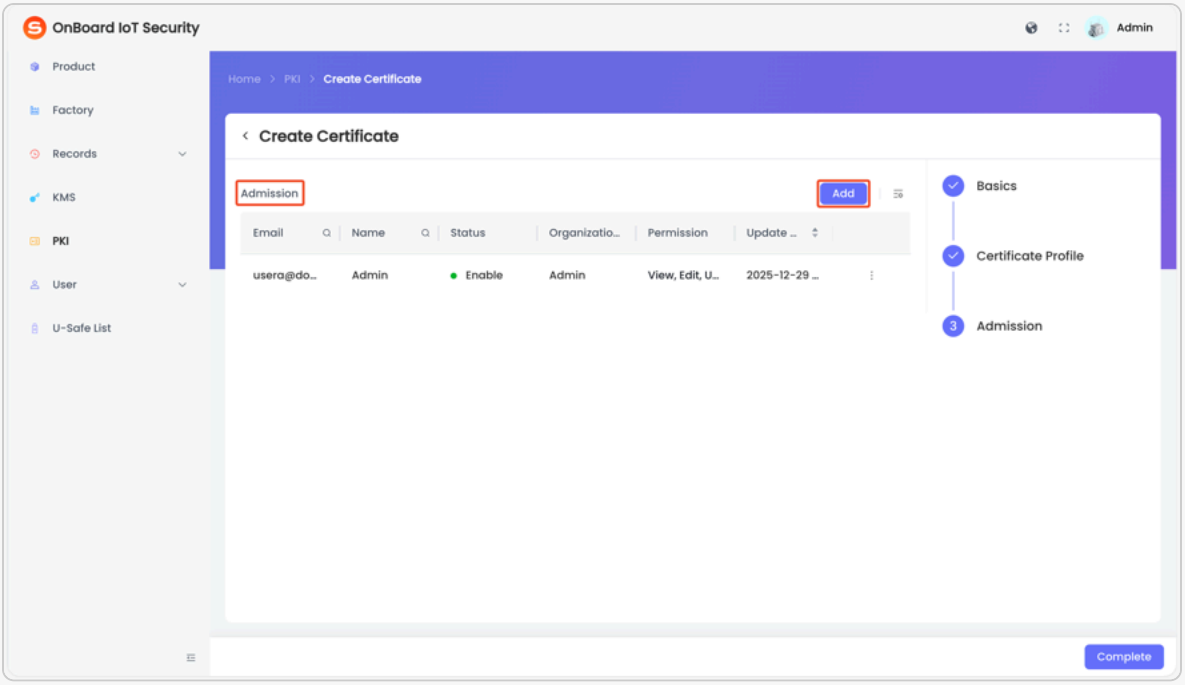
Description of DAC template profile information:

Area	Information	Description
Matter Basic	VID (Required)	Vendor ID inherited from the PAI certificate.

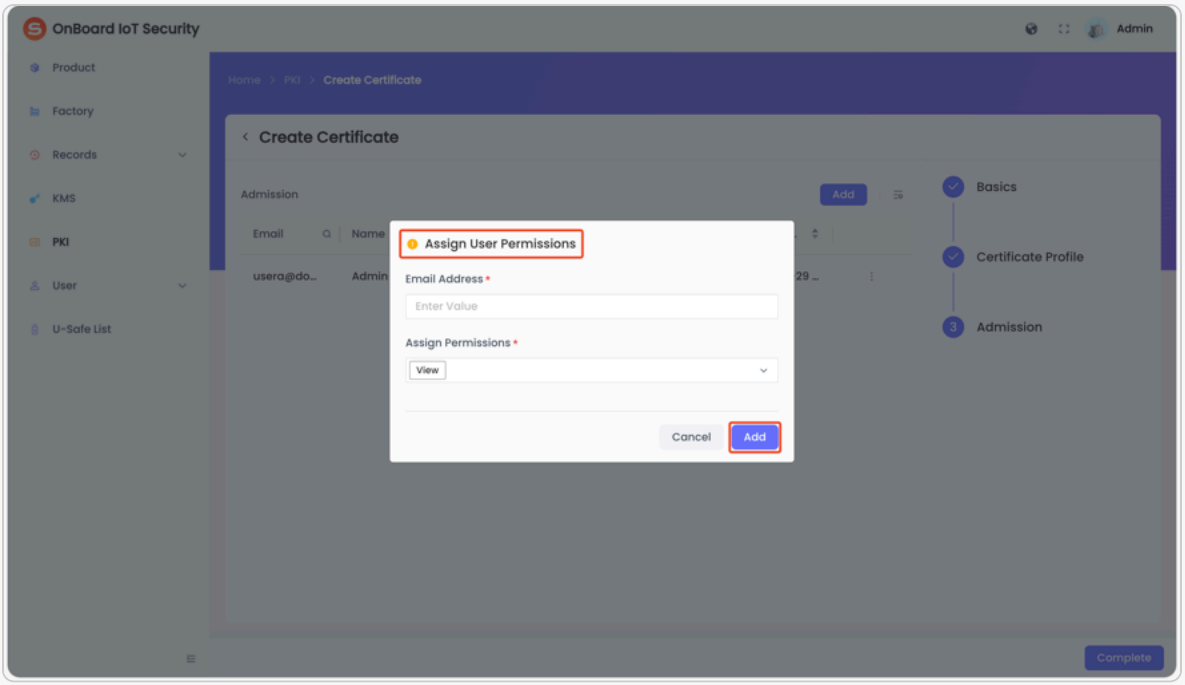
Area	Information	Description
	PID (Required)	Product ID. **Note**: When a DAC template is inherited from a PAI certificate with a configured PID, the PID is automatically inherited and can't be modified.
Profile Basic	Available Key Algorithms	Automatically parsed from the certificate content and can't be modified.
	Signature Algorithms	Automatically parsed from the certificate content and can't be modified.
	Expiry Date	Inherits the Expiry Date set in the previous step and can't be modified.
X-509v3 Extensions	Key Usage	Automatically parsed from the certificate content and can't be modified.
	Path Length Constaints	Automatically parsed from the certificate content and can't be modified.
	Authority Key ID	Automatically parsed from the certificate content and can't be modified.
	Subject Key ID	Automatically parsed from the certificate content and can't be modified.

Area	Information	Description
Usages	Key Usage	Optional key usages, such as **Digital Signature**.

4. After verifying that the information is correct, click **Next**, and then click **Add** on the **Admission** page.

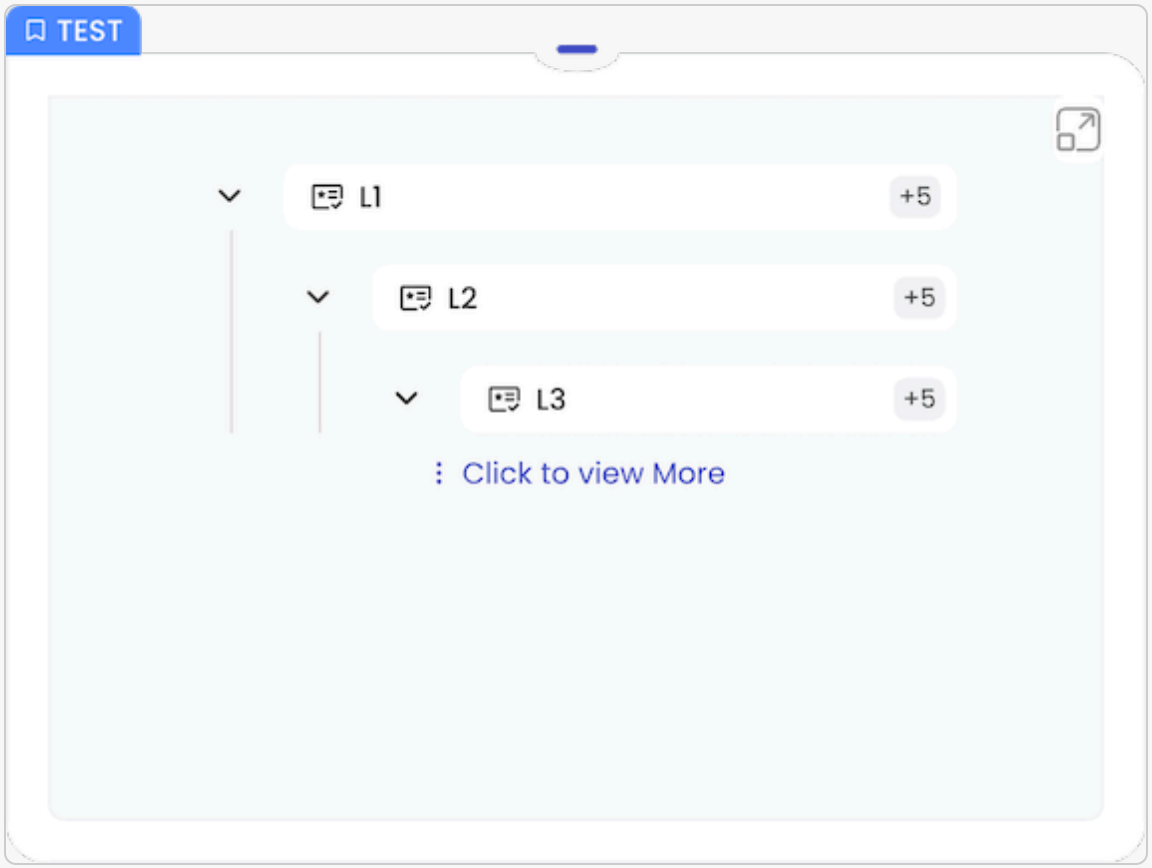


5. In the **Assign User Permissions** window, enter the email address and assign the corresponding permission (View, Edit, Use, Grant, and Delete), and then click **Add**.



6. Click **Complete** to complete the DAC template creation.

Example of certificate chain hierarchy



Level	Description
L1	Root Certificate Authority
L2	Intermediate Certificate Authority
L3	End Entity Certificate

Version: v3.2.1

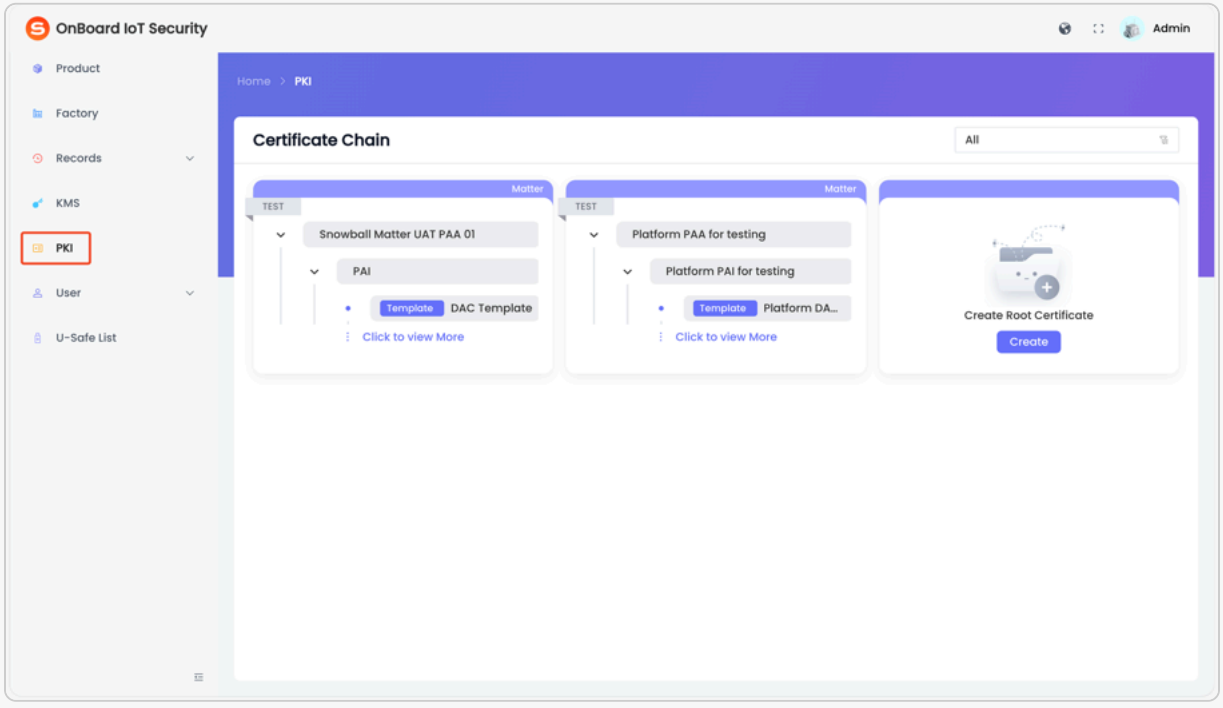
How to check certificates in OnBoard™ IoT Security

This article will guide you on how to check certificate basic information, certificate profile, and logs in OnBoard™ IoT Security (OBIS).

Check basic information

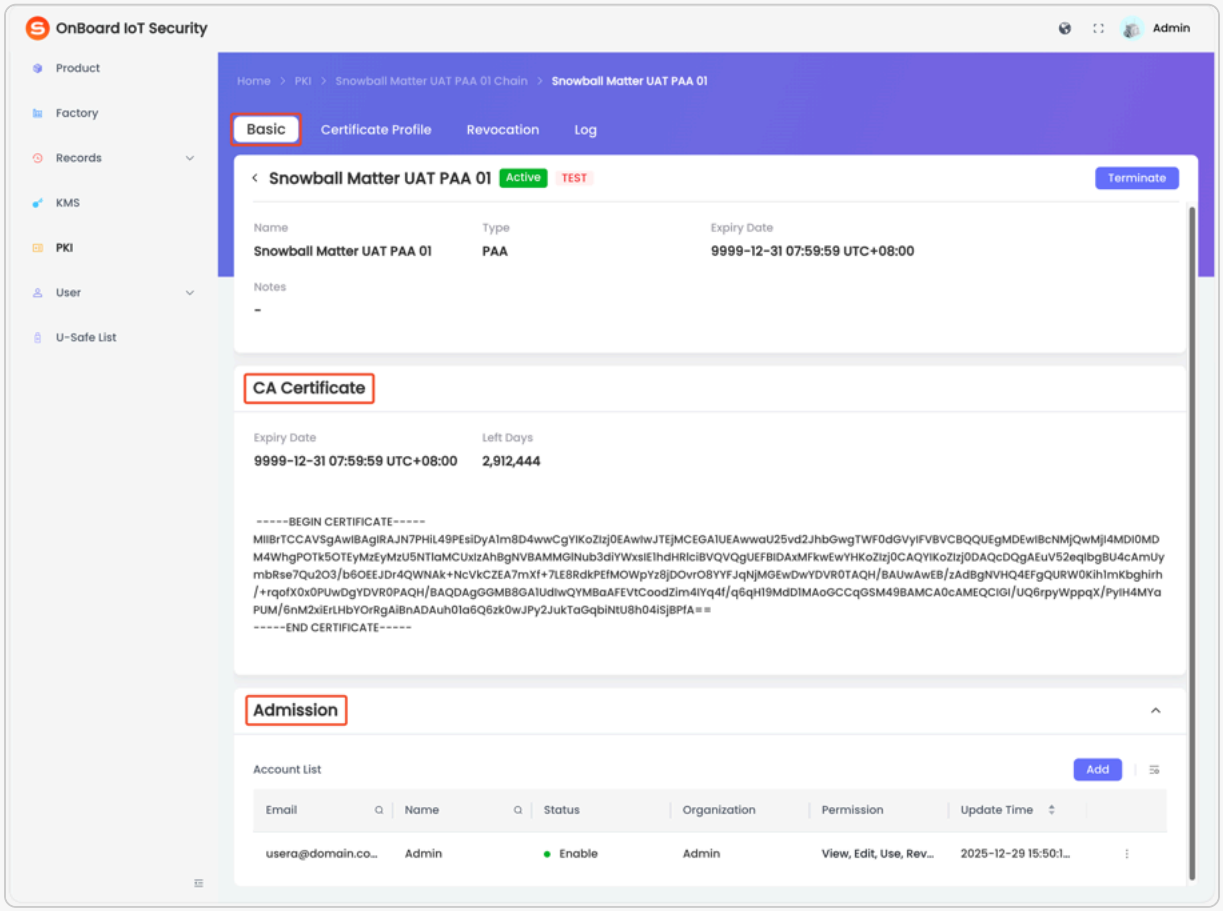
1. Sign in to OBIS.

2. In the left menu, click **PKI**, and then select the corresponding certificate chain.



3. Select the corresponding certificate name. On the **Basic** tab of the certificate page, you can check the certificate-related information (such as **Name**, **Type**, and **Expiry Date**).

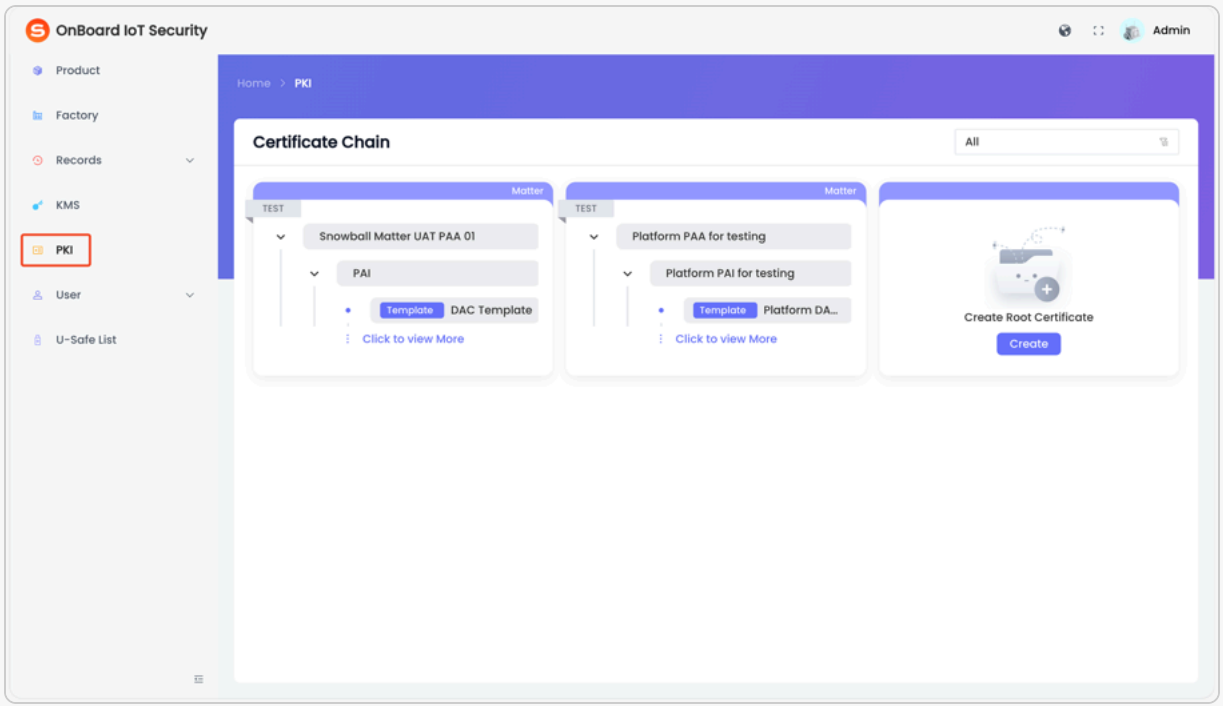
> **TIP**
>
> In the **CA Certificate** area, you can extract certificate attributes such as the issuer information, validity period, and public key by parsing the encoding of the CA certificate.
>
> In the **Admission** area, you can manage the certificate permissions.



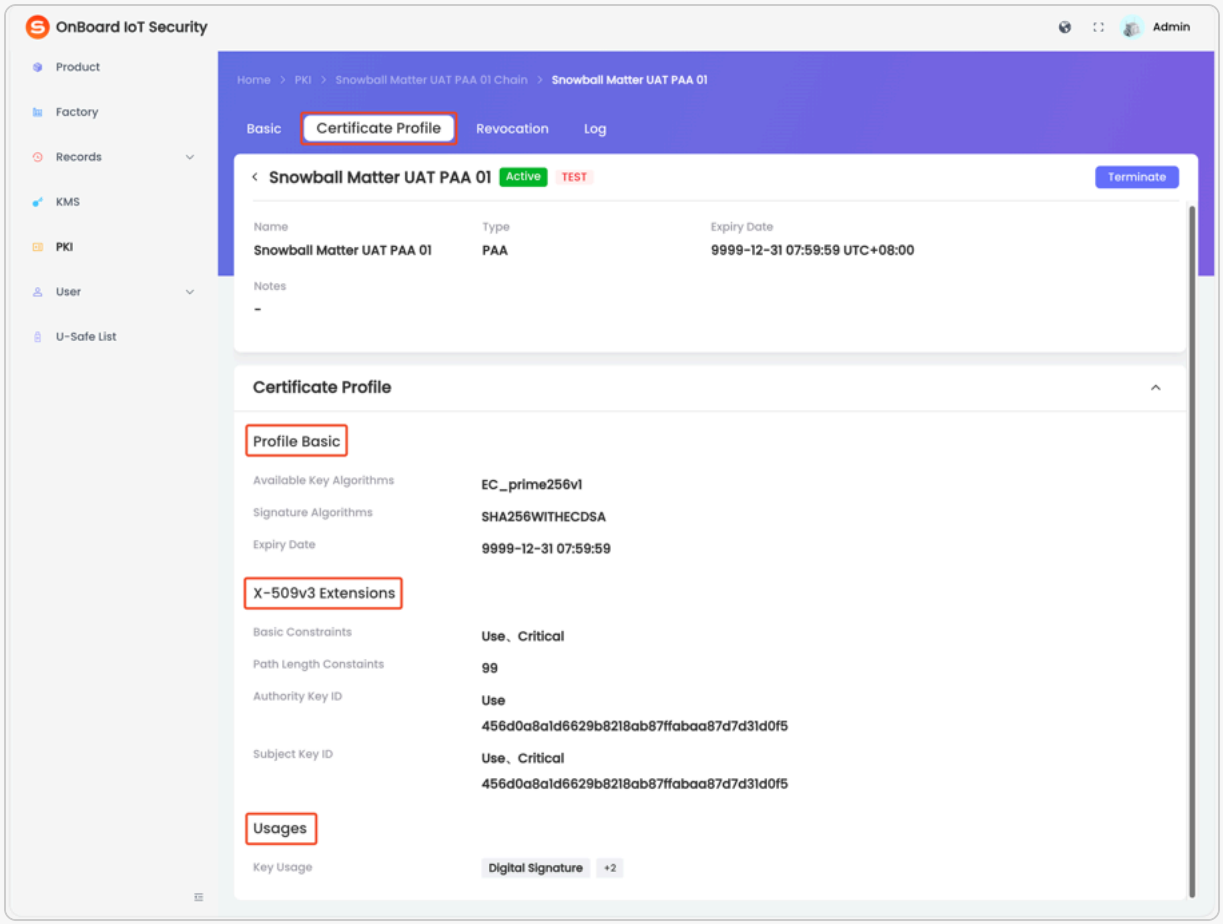
Check certificate profile

1. Sign in to OBIS.

2. In the left menu, click **PKI**, and then select the corresponding certificate chain.



3. Select the corresponding certificate name. On the **Certificate Profile** tab of the certificate page, you can check the certificate configuration (such as **Profile Basic**, **X-509v3 Extensions**, and **Usages**).

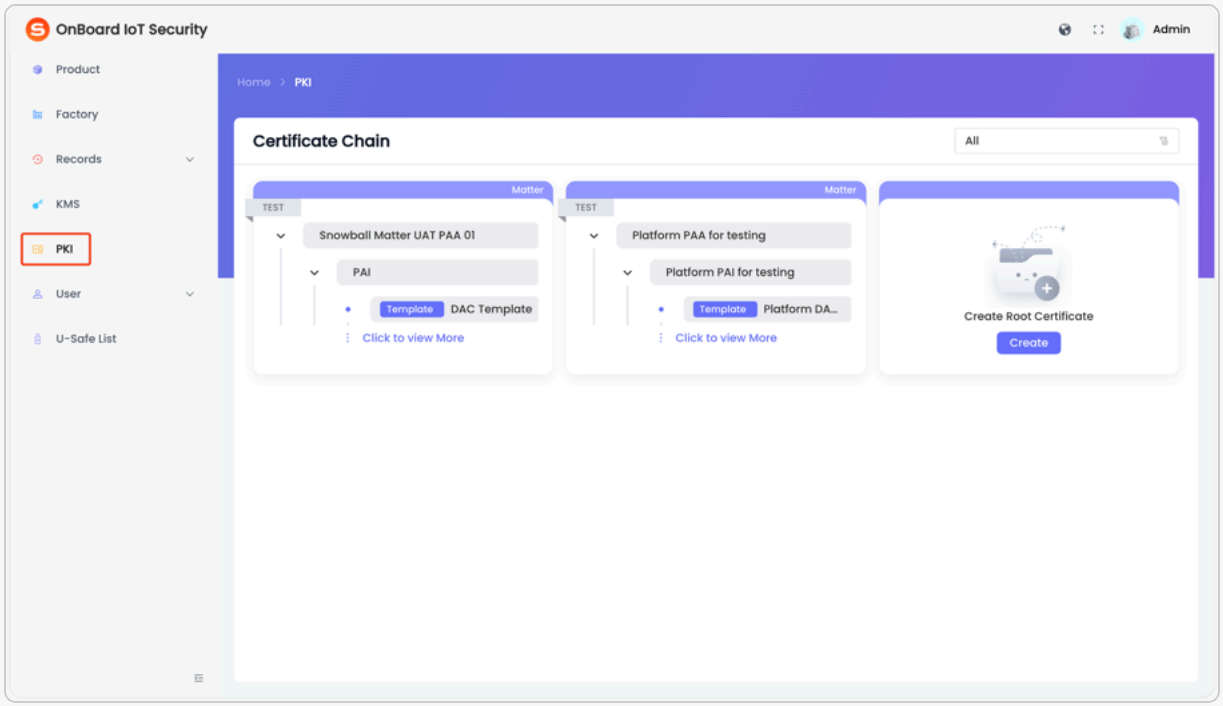


Check logs

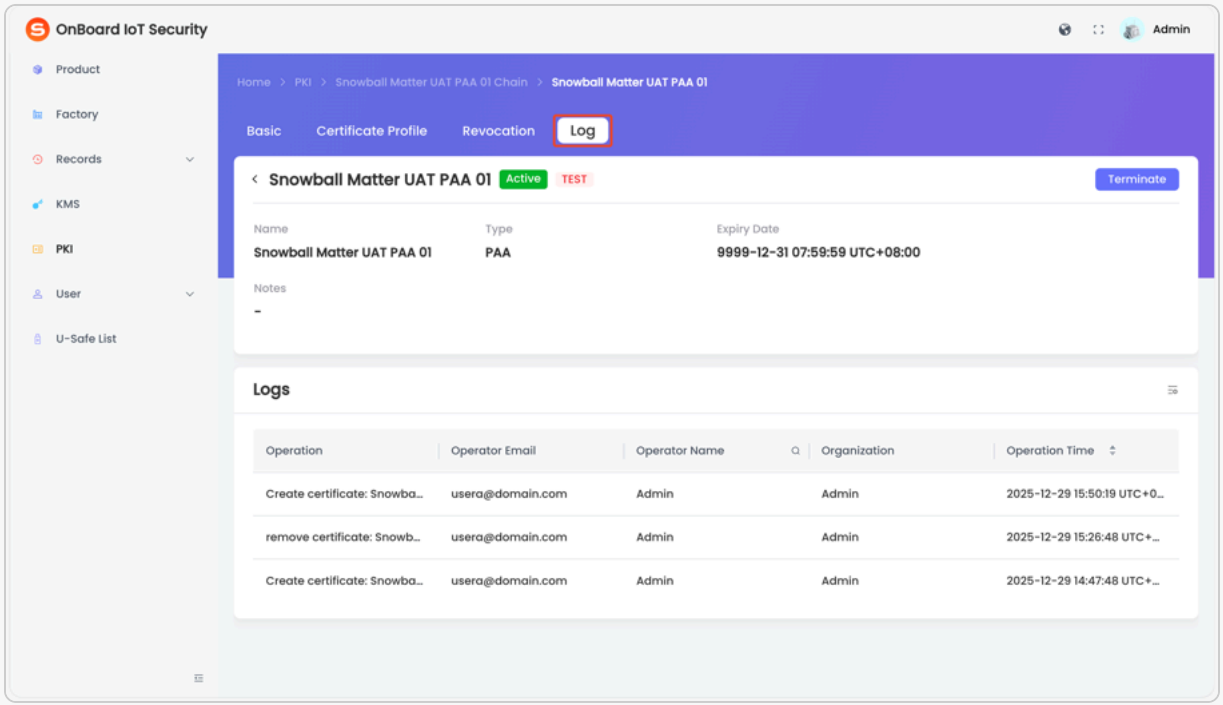
> **TIP**
>
> Through the certificate logs, you can check the historical operations of the certificate.

1. Sign in to OBIS.

2. In the left menu, click **PKI**, and then select the corresponding certificate chain.



3. Select the corresponding certificate name. On the **Log** tab of the certificate page, you can check the certificate history (such as Operation and Operator Email).



Version: v3.2.1

How to manage certificates in OnBoard™ IoT Security

This article will guide you through the following operations in OnBoard™ IoT Security (OBIS):

- Manage certificate permissions
- Revoke a certificate
- Delete a certificate

Prerequisites

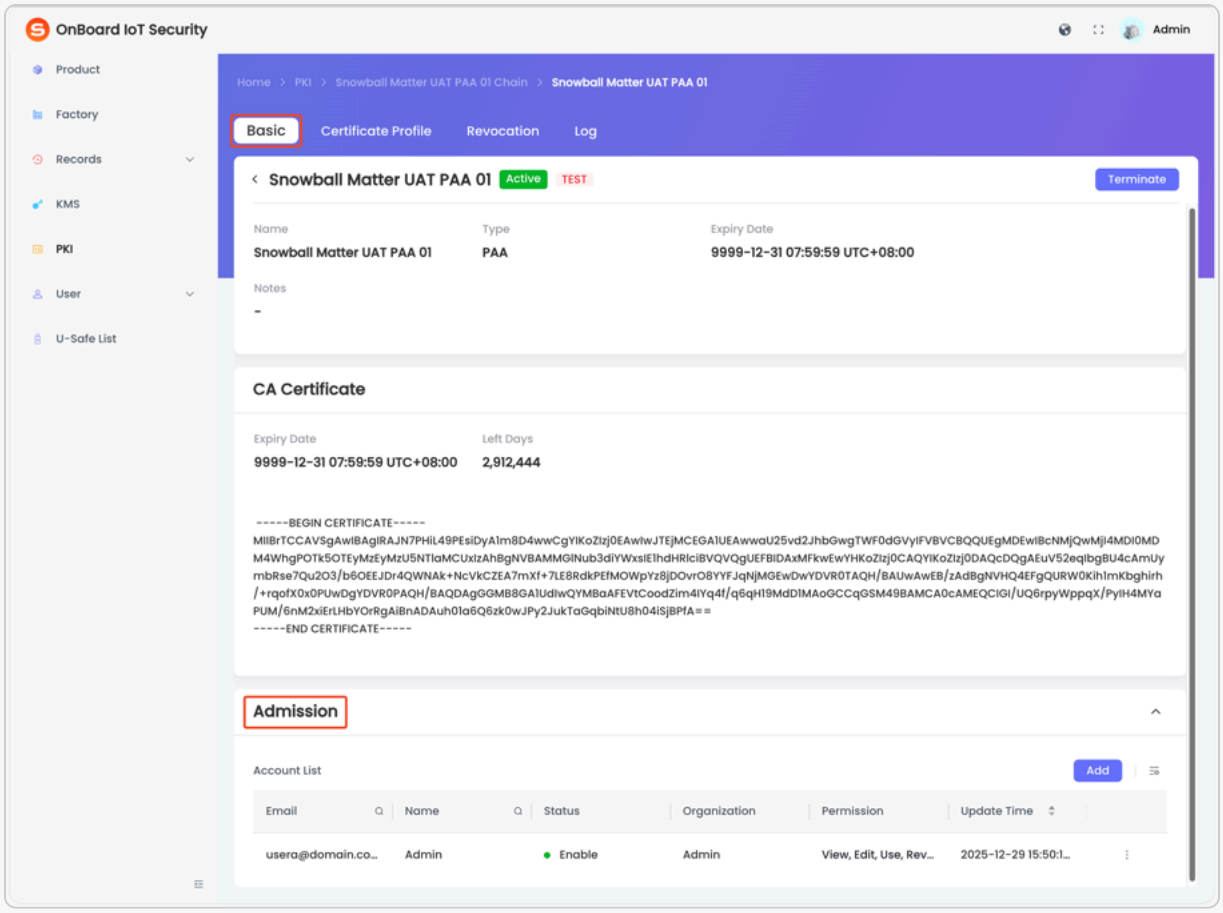
Before getting started, please ensure that you have OBIS role permissions as a System Manager or Resource Manager, or have operational permissions for the corresponding certificate.

Manage certificate permissions

You can adjust and set user permissions (View, Edit, Use, Revoke, Grant, and Delete) for certificates by following these steps:

1. Sign in to OBIS.
2. In the left menu, click **PKI**, and then select the corresponding certificate chain.

3. Select the corresponding certificate name. On the certificate page, you can manage the certificate permissions in the **Admission** area.



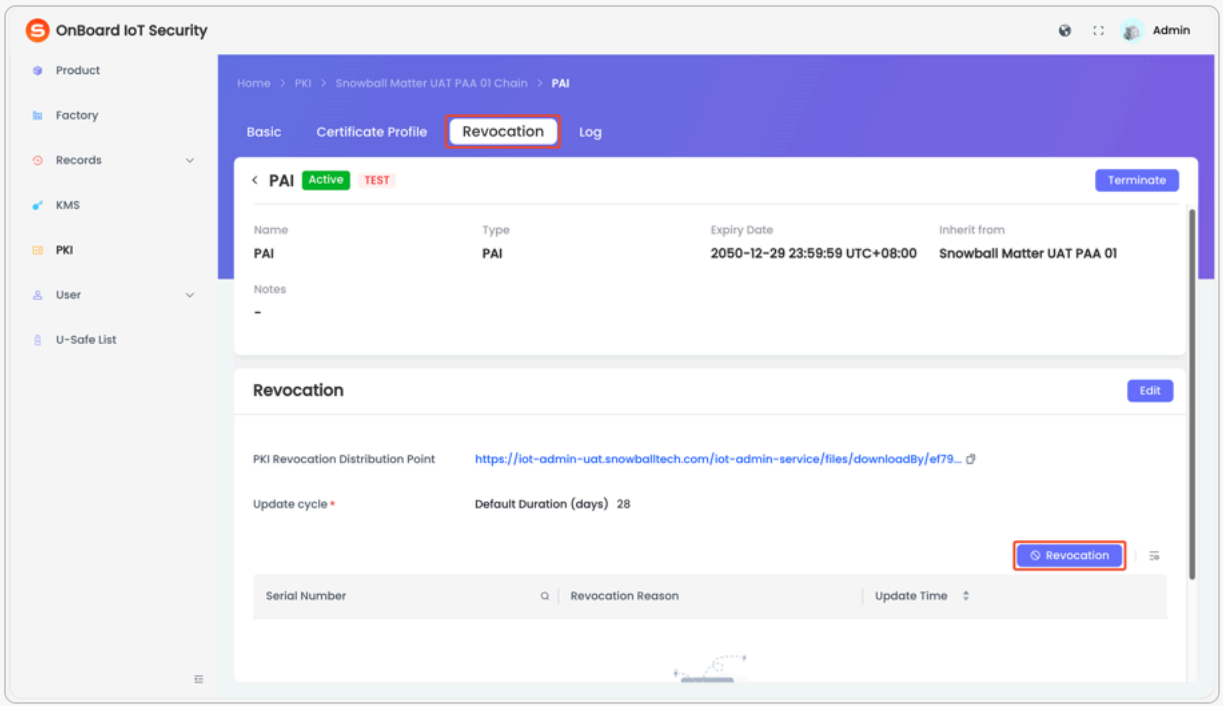
Revoke a certificate

> **WARNING**
>
> Once a certificate is revoked, the corresponding device can no longer perform commissioning operations.

1. Sign in to OBIS.

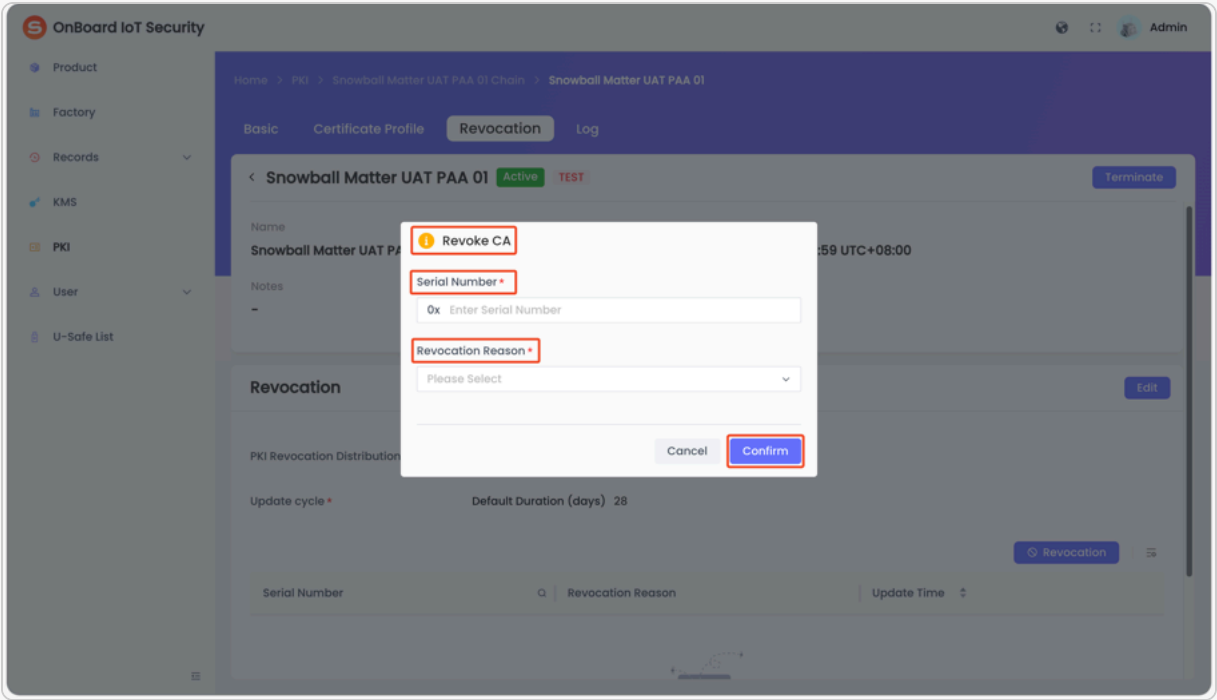
2. In the left menu, click **PKI**, and then select the corresponding certificate chain.

3. Select the corresponding certificate name. On the **Revocation** tab of the certificate page, click **Revocation**.



4. In the **Revoke CA** window, enter the **Serial Number** and select the **Revocation Reason**, and then click Confirm to revoke the certificate.

> **TIP**
>
> You can check the serial number on the device page.



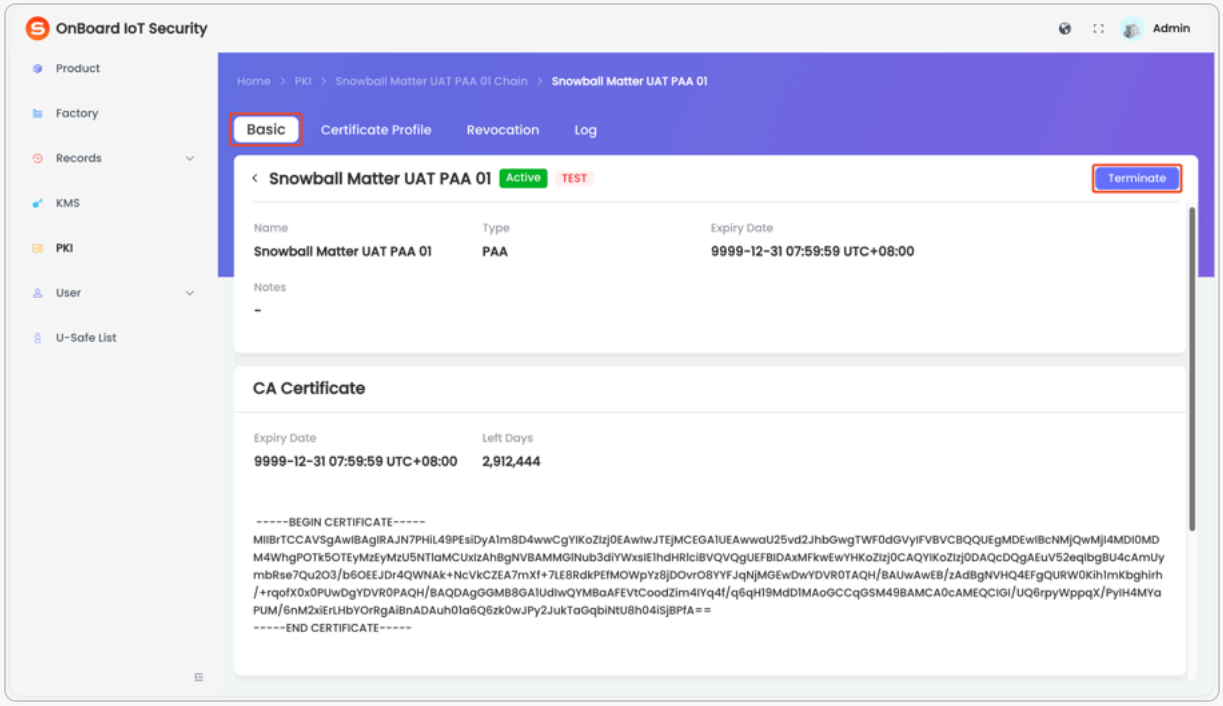
Delete a certificate

INFO

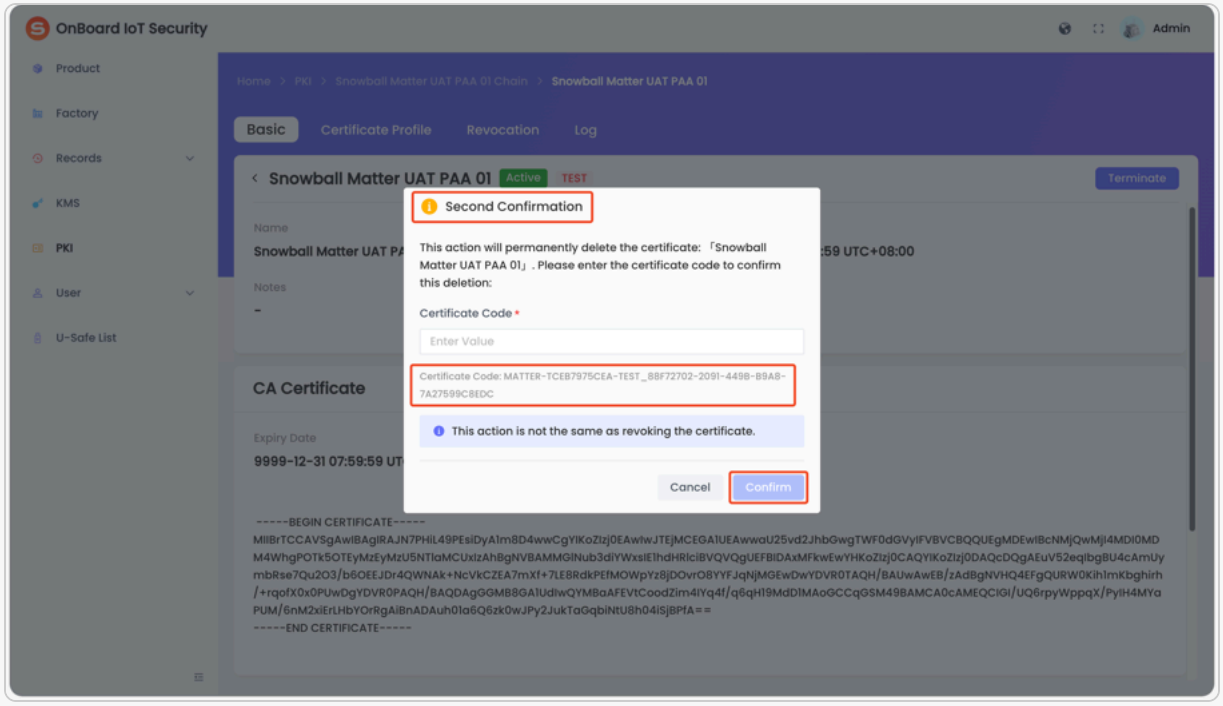
Before deleting a certificate, please ensure:

- All sub-certificates under the certificate have been deleted. If the certificate is the parent certificate and the sub-certificates have not been deleted, you will see the message "Certificate cannot be deleted. It has the following sub certificates" along with a list of the sub-certificates.
- The certificate is not referenced by any resources. If it is referenced, you will see the message "Certificate cannot be deleted, the certificate is currently used by the following list" along with a list of the referencing resources.

1. Sign in to OBIS.
2. In the left menu, click **PKI**, and then select the corresponding certificate chain.
3. Select the corresponding certificate name. On the certificate page, click **Terminate**.



4. In the **Second Confirmation** window, enter the certificate code prompted at the bottom of the window, and then click **Confirm** to delete the certificate.


OnBoard IoT Security
Admin
Product
Factory
Records
KMS
PKI
User
U-Safe List
Home > PKI > Snowball Matter UAT PAA 01 Chain > Snowball Matter UAT PAA 01
Basic
Certificate Profile
Revocation
Log
Snowball Matter UAT PAA 01
Active
TEST
Terminate
Name
Snowball Matter UAT PA
:59 UTC+08:00
Notes
-
CA Certificate
Expiry Date
9999-12-31 07:59:59 UT
-----BEGIN CERTIFICATE-----
MIIBrTCCAVSgAwIBAgIRAJN7PHiL49PEsiDyA1m8D4wwCgYIKoZIzj0EAwIwJTEjMCEGA1UEAwwaU25vd2JhbGwgTWF0dGVyIFVBVCBQQUEgMDEwIBcNMjQwMjI4MDI0MD
M4WhgPOTk5OTEyMzEyMzU5NTlaMCUxIzAhBgNVBAMMGlNub3diYWxsIE1hdHRlciBVQVQgUEFBIDAxMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEuV52eqIbgBU4cAmUy
mbRse7Qu2O3/b6OEEJDr4QWNAk+NcVkCZEA7mXf+7LE8RdkPEfMOWpYz8jDOvrO8YYFJqNjMGEwDwYDVR0TAQH/BAUwAwEB/zAdBgNVHQ4EFgQURW0Kih1mKbghirh
/+rqofX0x0PUwDgYDVR0PAQH/BAQDAgGGMB8GA1UdIwQYMBaAFEVtCoodZim4iYq4f/q6qH19MdD1MAoGCCqGSM49BAMCA0cAMEQCIGI/UQ6rpyWppqX/PyIH4MYa
PUM/6nM2xiErLHbYOrRgAiBnADAuh01a6Q6zk0wJPy2JukTaGqbiNtU8h04iSjBPfA==
-----END CERTIFICATE-----
Second Confirmation
This action will permanently delete the certificate: 「Snowball Matter UAT PAA 01」. Please enter the certificate code to confirm this deletion:
Certificate Code *
Enter Value
Certificate Code: MATTER-TCEB7975CEA-TEST_88F72702-2091-449B-B9A8-7A27599C8EDC
This action is not the same as revoking the certificate.
Cancel
Confirm

Version: v3.2.1

How to create an account in OnBoard™ IoT Security

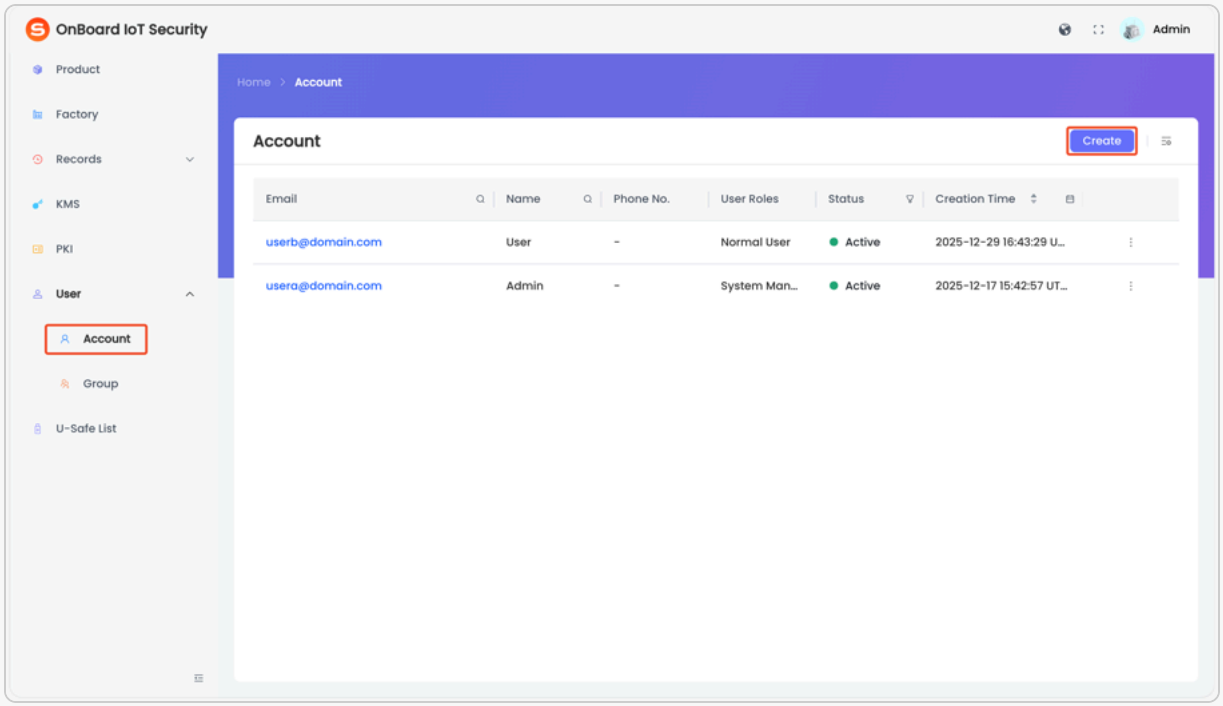
This article will guide you on how to create an account in OnBoard™ IoT Security (OBIS) so that you can add users to access and operate OBIS.

Prerequisites

Before getting started, please ensure that you have OBIS role permissions as a System Manager.

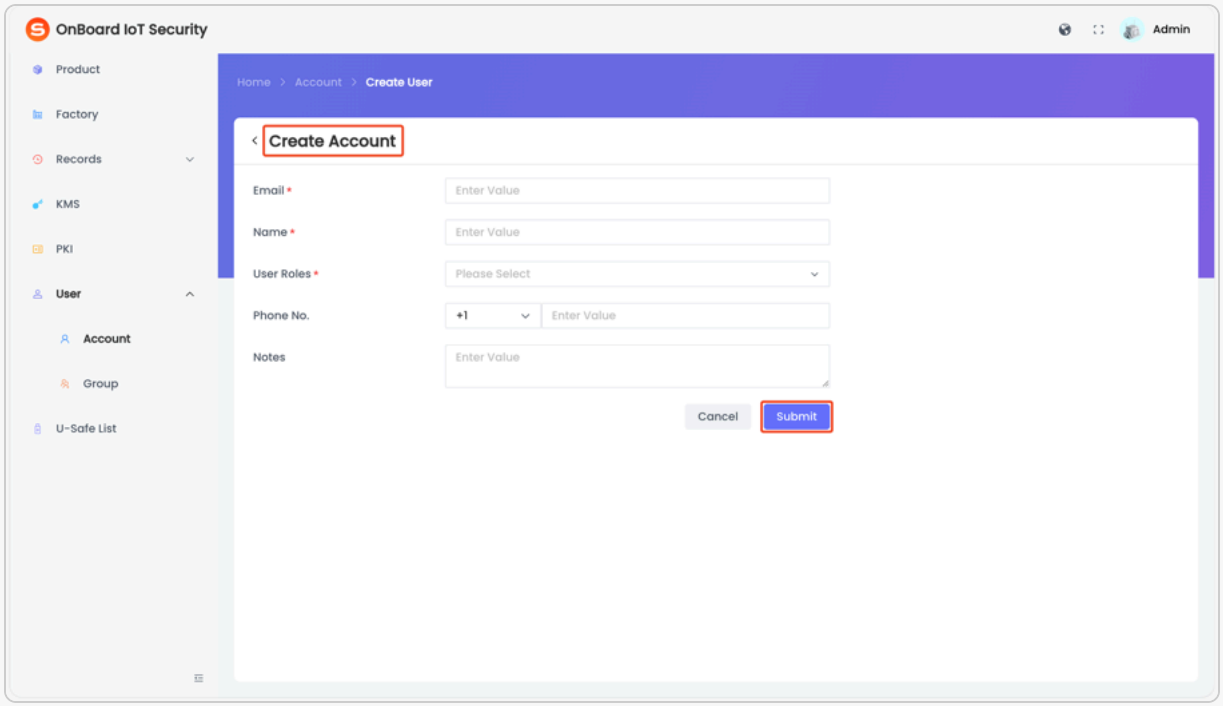
Instructions

1. Sign in to OBIS.
2. In the left menu, click **User** > **Account**, and then click **Create**.



3. On the **Create Account** page, select and enter the user-related information.

> **TIP**
>
> For more information, refer to the Role classification and permission description.



Description of user information:

Information	Mandatory	Description
Email	Yes	User email, used to receive notifications and verification information; it must be real, valid, and unique. **Note**: After the account is created, the email can't be changed.
Name	Yes	User name, which does not need to be unique.

Information	Mandatory	Description
User Role	Yes	A user can be assigned multiple roles. User roles determine the user's access and operation permissions. For more information, refer to the Role classification and permission description.
Phone No.	No	Provide an additional contact method besides email for easy communication.
Notes	No	Provide additional information related to the user, such as special requirements and important notes.

4. Click **Submit** to complete the account creation.

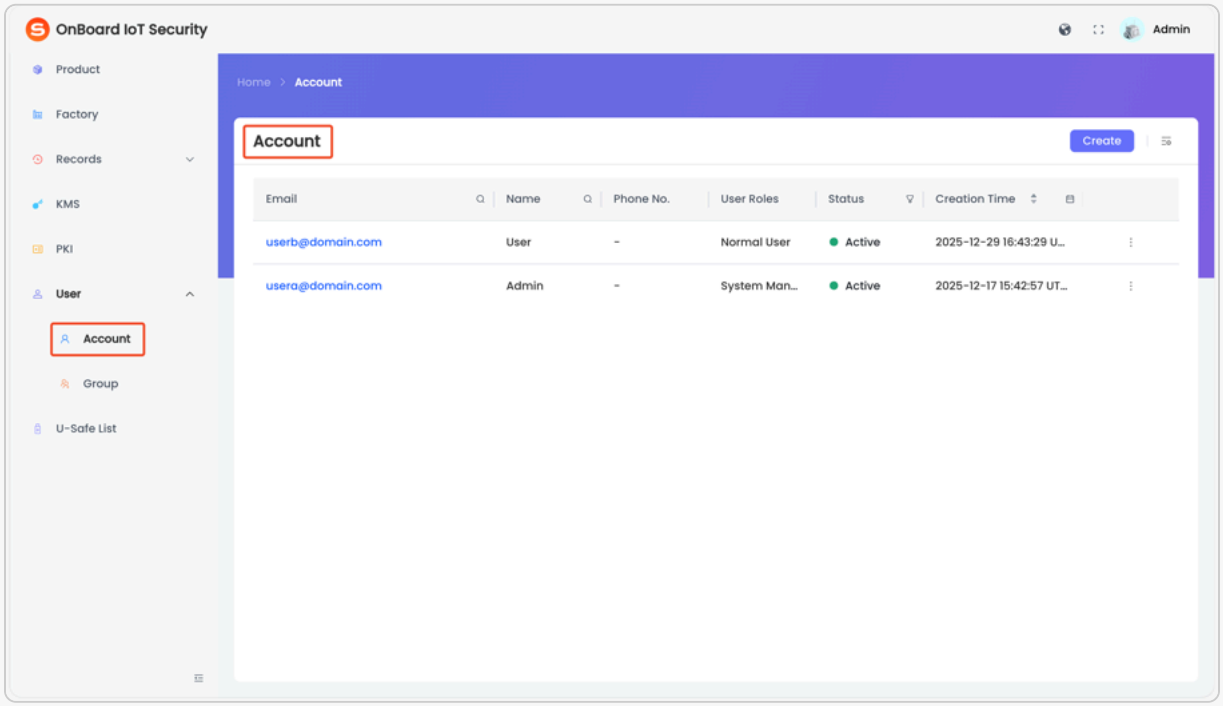
Version: v3.2.1

How to check accounts in OnBoard™ IoT Security

This article will guide you on how to check account information, AK (Access Key), and SK (Secret Key) in OnBoard™ IoT Security (OBIS).

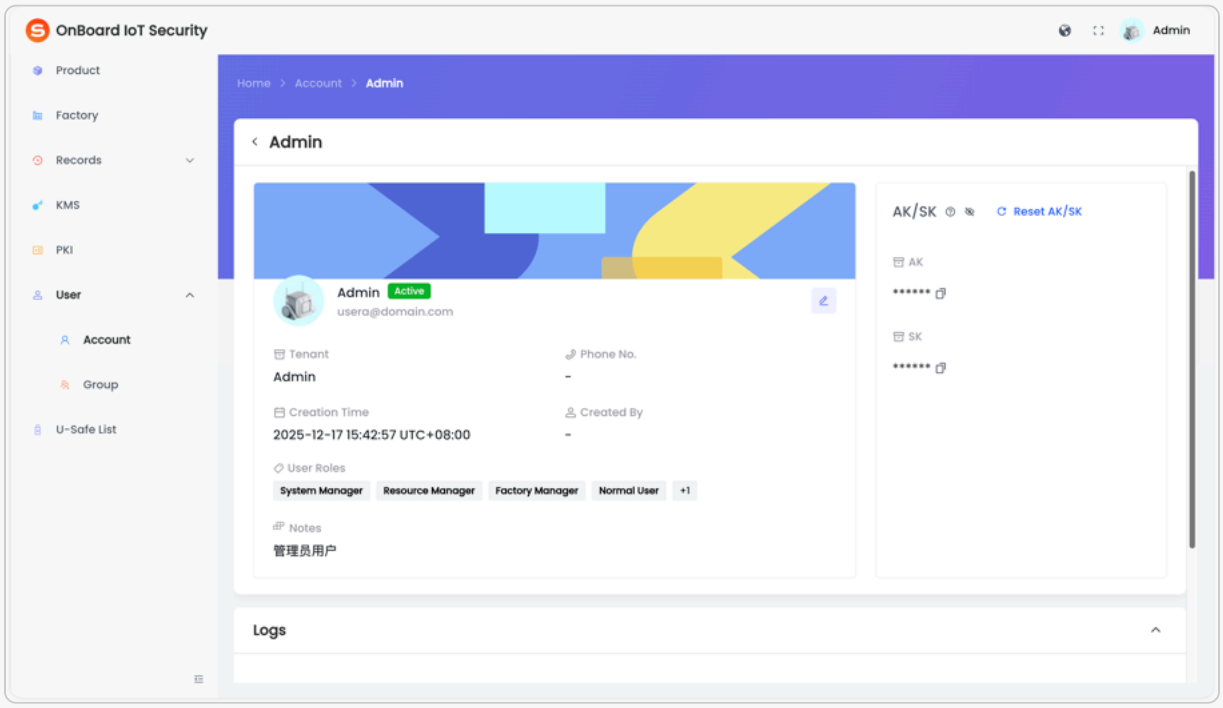
Instructions

1. Sign in to OBIS.
2. In the left menu, click **User** > **Account** to navigate to the **Account** page.
3. On the **Account** page, select the corresponding email.



4. On the user page, you can check the account-related information (such as **Tenant**, **Creation Time**, **Created By**, and **Logs**).

> **TIP**
>
> In the **AK/SK** area, you can view or reset the AK or SK. AK and SK can be used for authentication and API calls.



Version: v3.2.1

How to manage accounts in OnBoard™ IoT Security

This article will guide you through the following operations in OnBoard™ IoT Security (OBIS) to manage user access and permissions for OBIS:

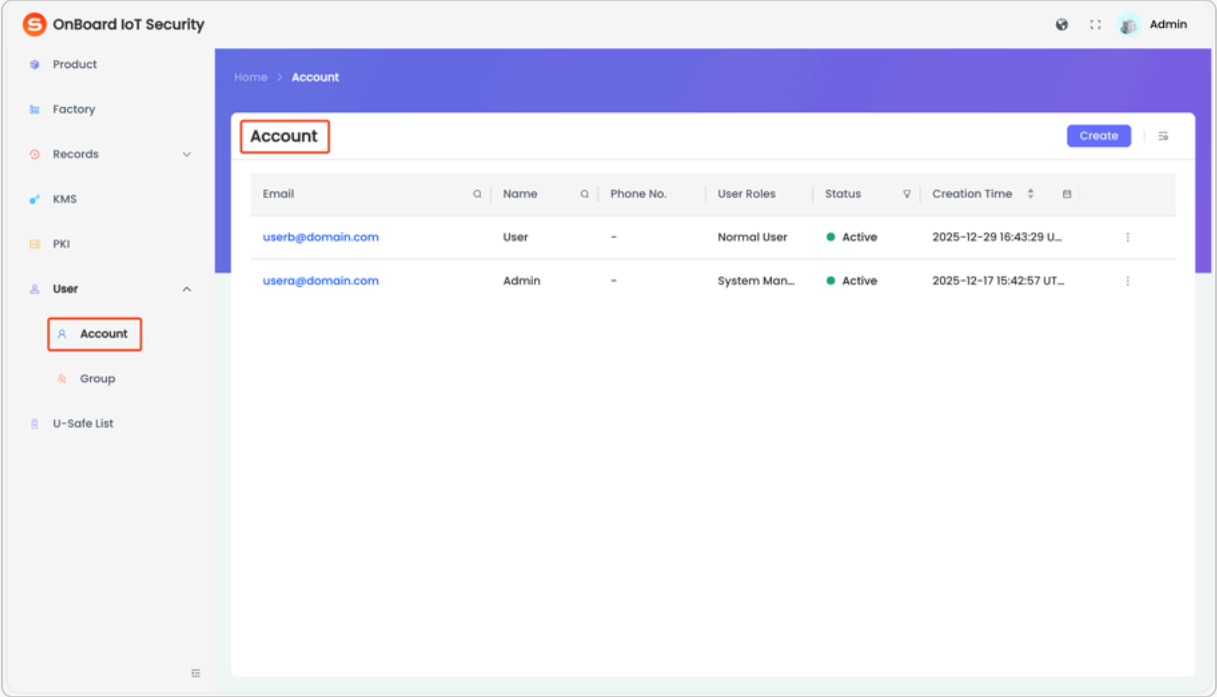
- Edit an account
- Lock or unlock an account
- Delete an account

Prerequisites

Before getting started, please ensure that you have OBIS role permissions as a System Manager.

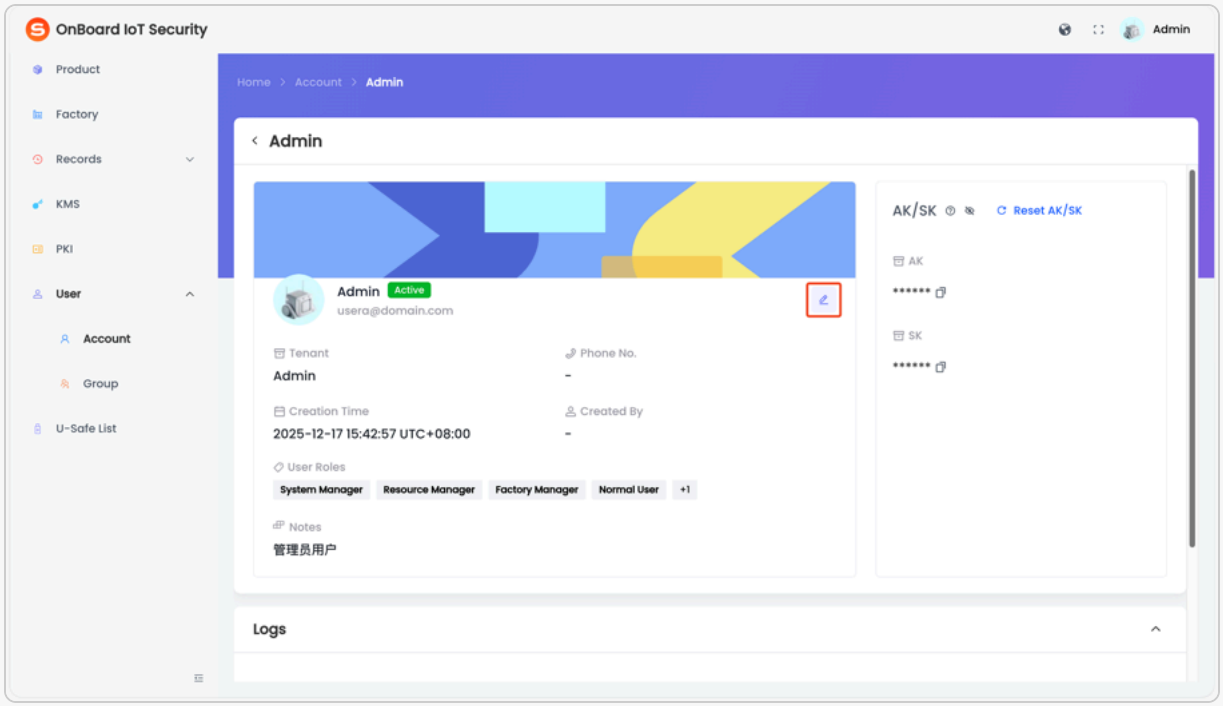
Edit an account

1. Sign in to OBIS.
2. In the left menu, click **User** > **Account** to navigate to the **Account** page.
3. On the **Account** page, select the corresponding email.



4. On the user page, click the Edit icon to change the user information (**Name**, **User Roles**, **Phone No.**, and **Notes**).

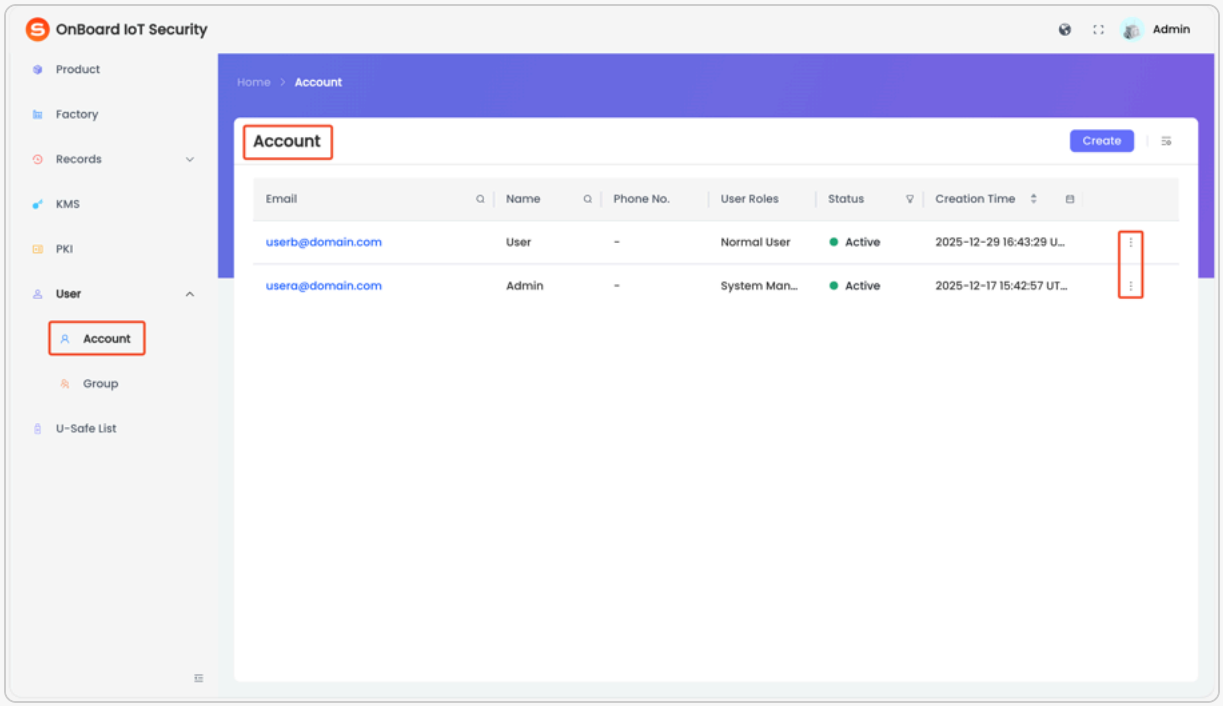
> **TIP**
>
> A user can be assigned multiple roles. User roles determine the user's access and operation permissions. For more information, refer to the Role classification and permission description.



Lock or unlock an account

> **TIP**
>
> Lock the account to prevent its sign-in and use. Once locked, the account status will change to **Disabled**.

1. Sign in to OBIS.
2. In the left menu, click **User** > **Account** to navigate to the **Account** page.
3. On the **Account** page, click the Options ⋮ icon for the corresponding account and select **Lock** or **Unlock**.

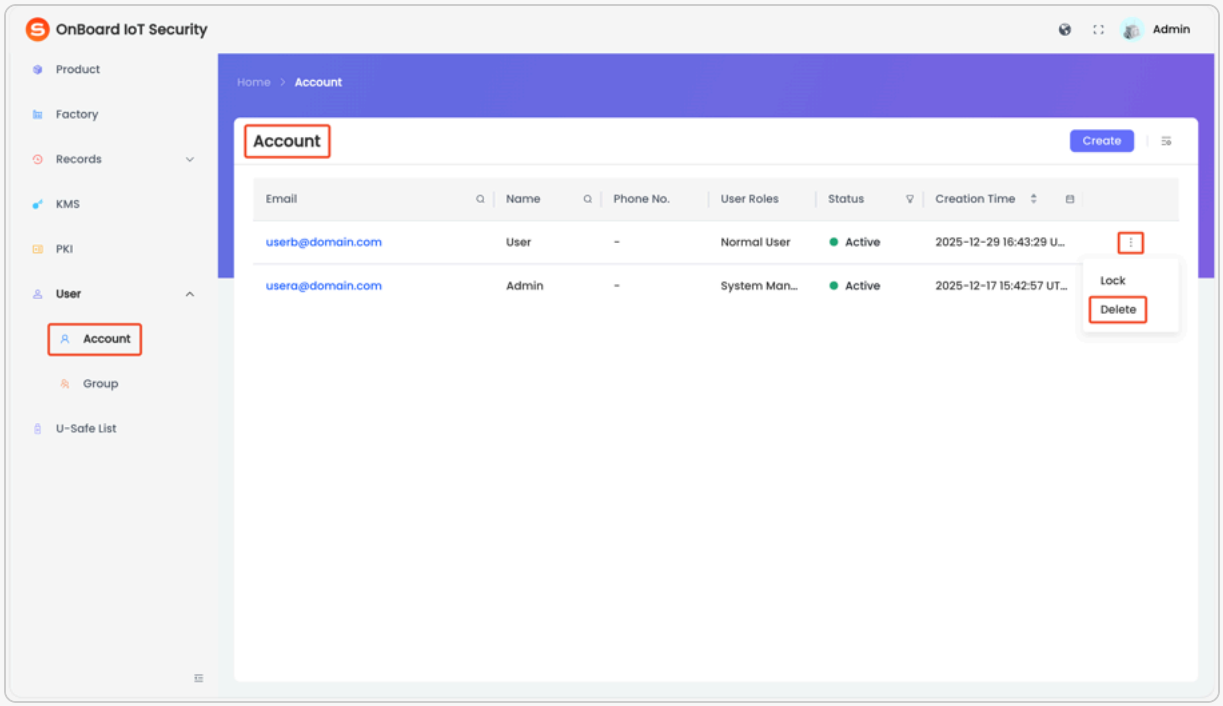


4. In the pop-up window, click **Confirm** to lock or unlock the account.

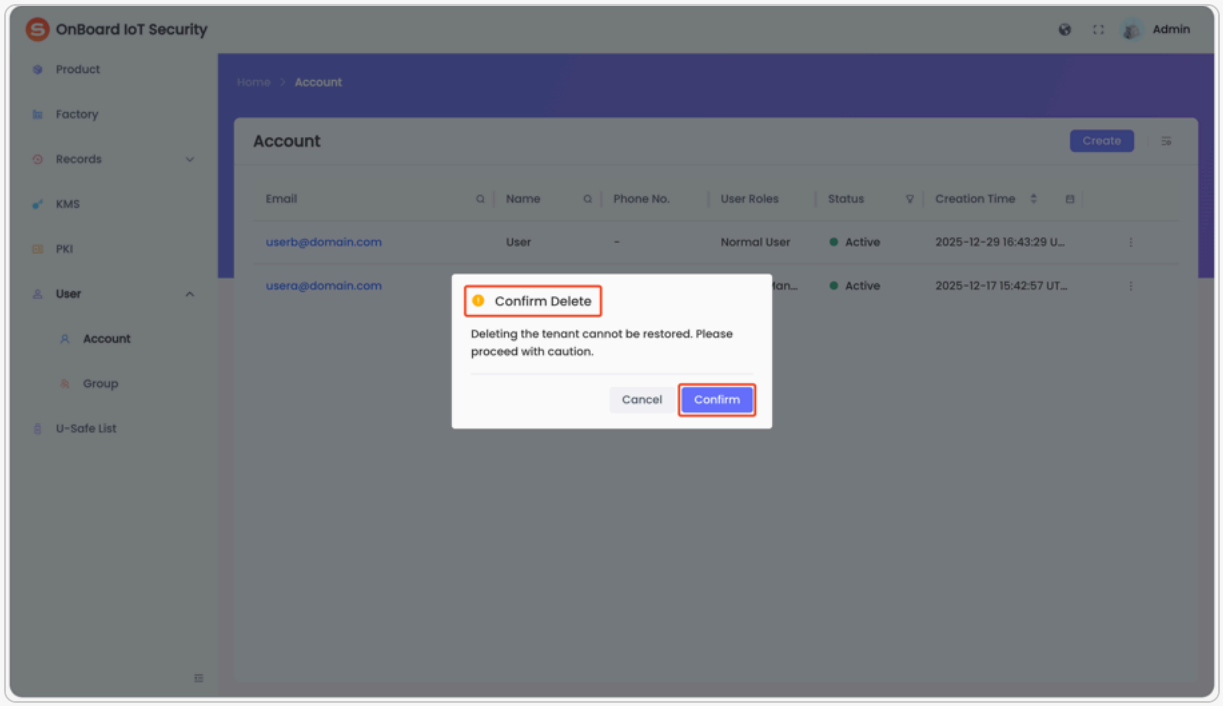
Delete an account

> **WARNING**
>
> Deleting an account cannot be restored. Please proceed with caution.

1. Sign in to OBIS.
2. In the left menu, click **User** > **Account** to navigate to the **Account** page.
3. On the **Account** page, click the Options ⋮ icon for the corresponding account and select **Delete**.



4. In the **Confirm Delete** window, click **Confirm** to delete the account.



Version: v3.2.1

How to create a group in OnBoard™ IoT Security

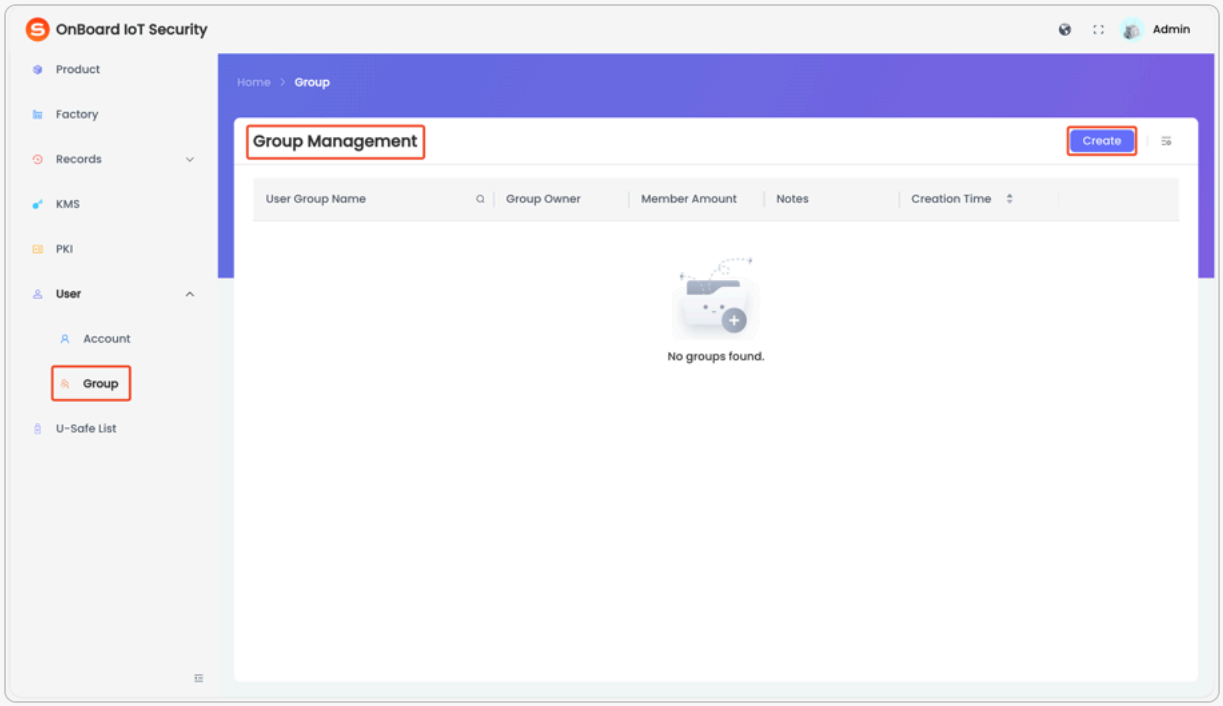
This article will guide you on how to create a user group in OnBoard™ IoT Security (OBIS) for centralized user management.

Prerequisites

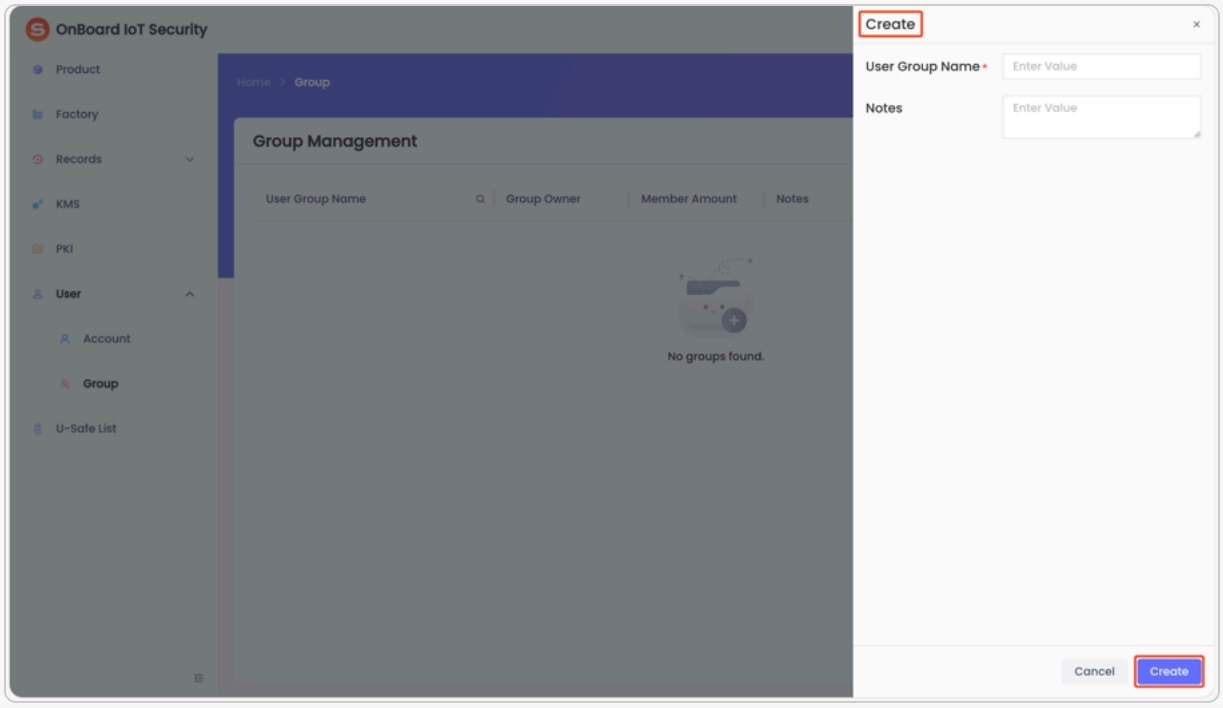
Before getting started, please ensure that you have OBIS role permissions as a System Manager.

Instructions

1. Sign in to OBIS.
2. In the left menu, click **User** > **Group**, and then click **Create**.



3. On the **Create** panel, enter the user group related information.



Description of user group information:

Information	Mandatory	Description
User Group Name	Yes	The user group name must be unique.
Notes	No	Provide additional information related to the user group, such as special requirements and important notes.

4. Click **Create** to complete the user group creation.

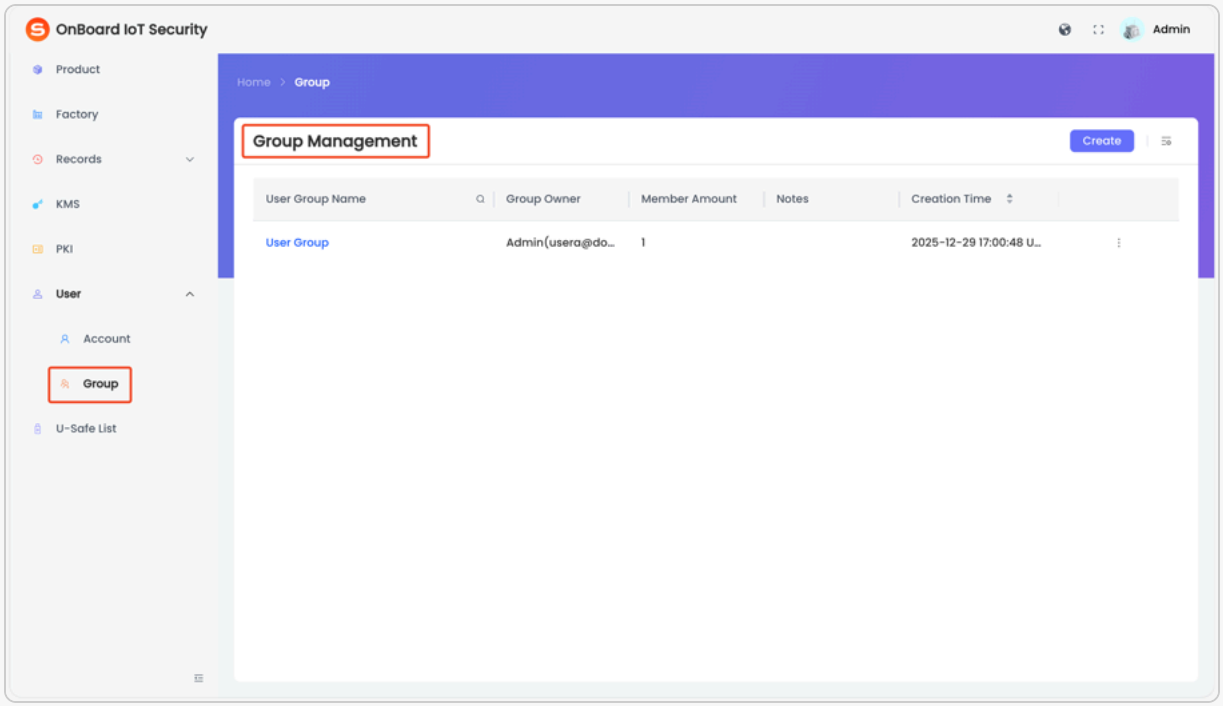
Version: v3.2.1

How to check groups in OnBoard™ IoT Security

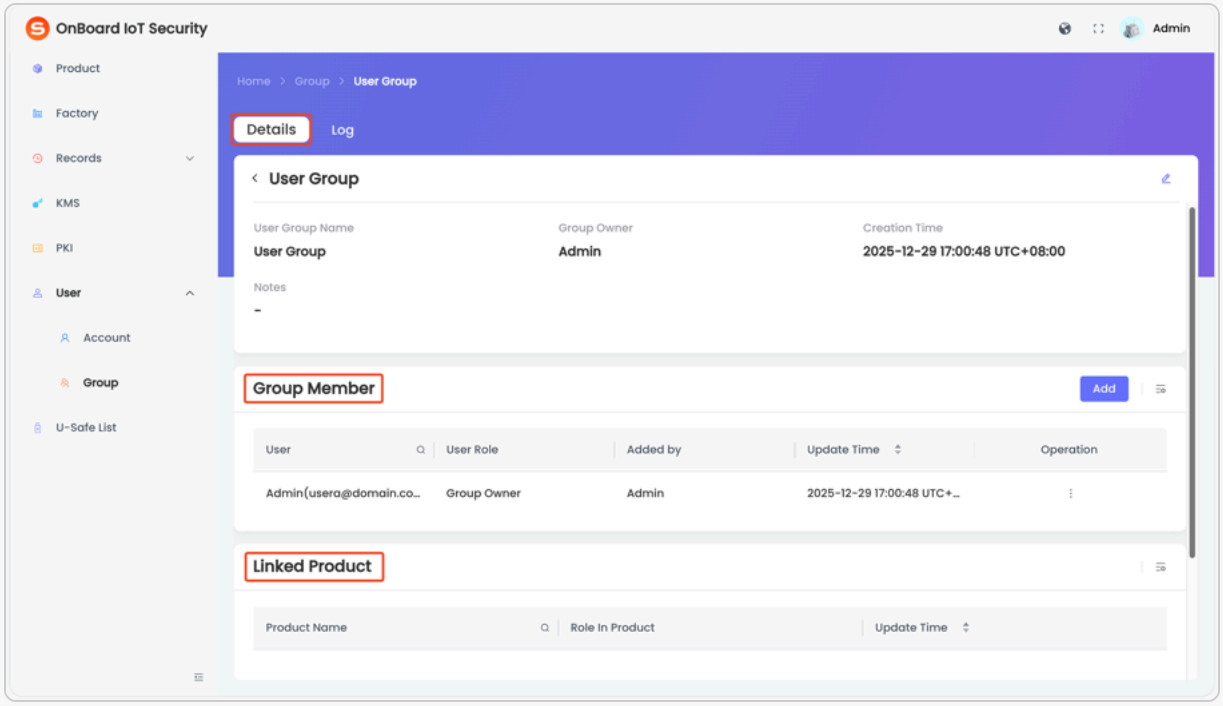
This article will guide you on how to check group details and logs in OnBoard™ IoT Security (OBIS).

Check details

1. Sign in to OBIS.
2. In the left menu, click **User** > **Group** to navigate to the **Group Management** page.
3. On the **Group Management** page, select the corresponding group name.



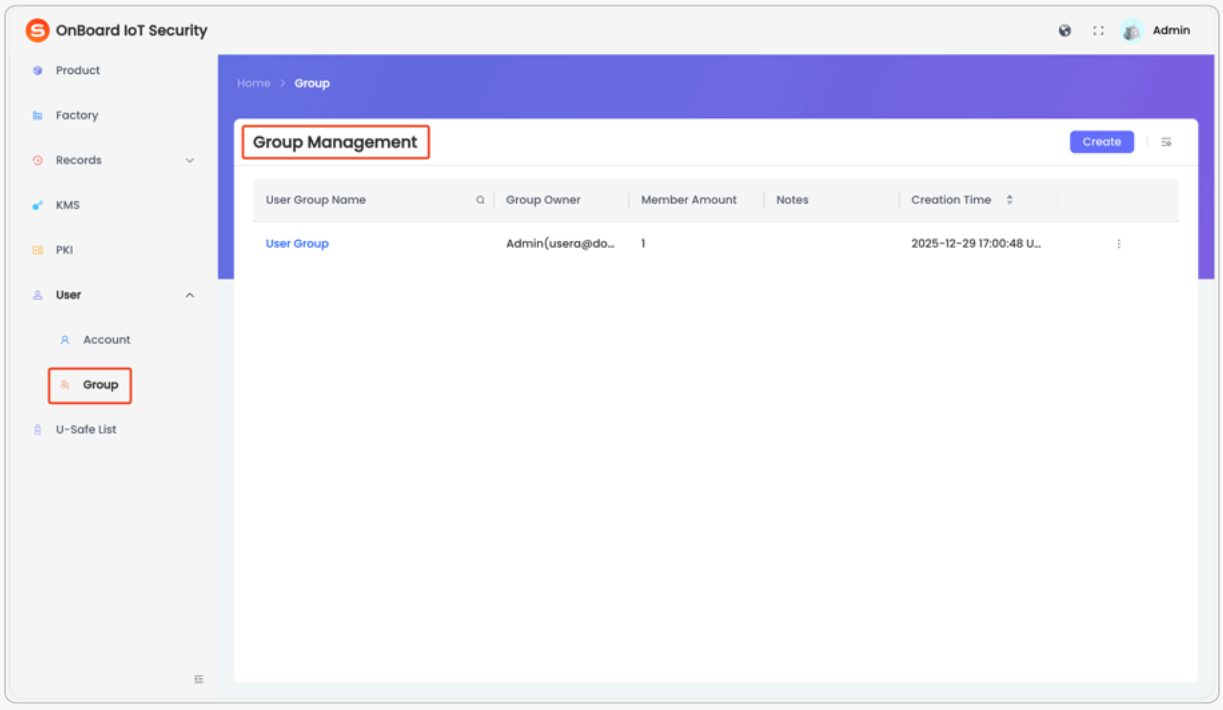
4. On the **Details** tab, you can check the group related information (such as **User Group Name**, **Group Owner**, and **Linked Product**).



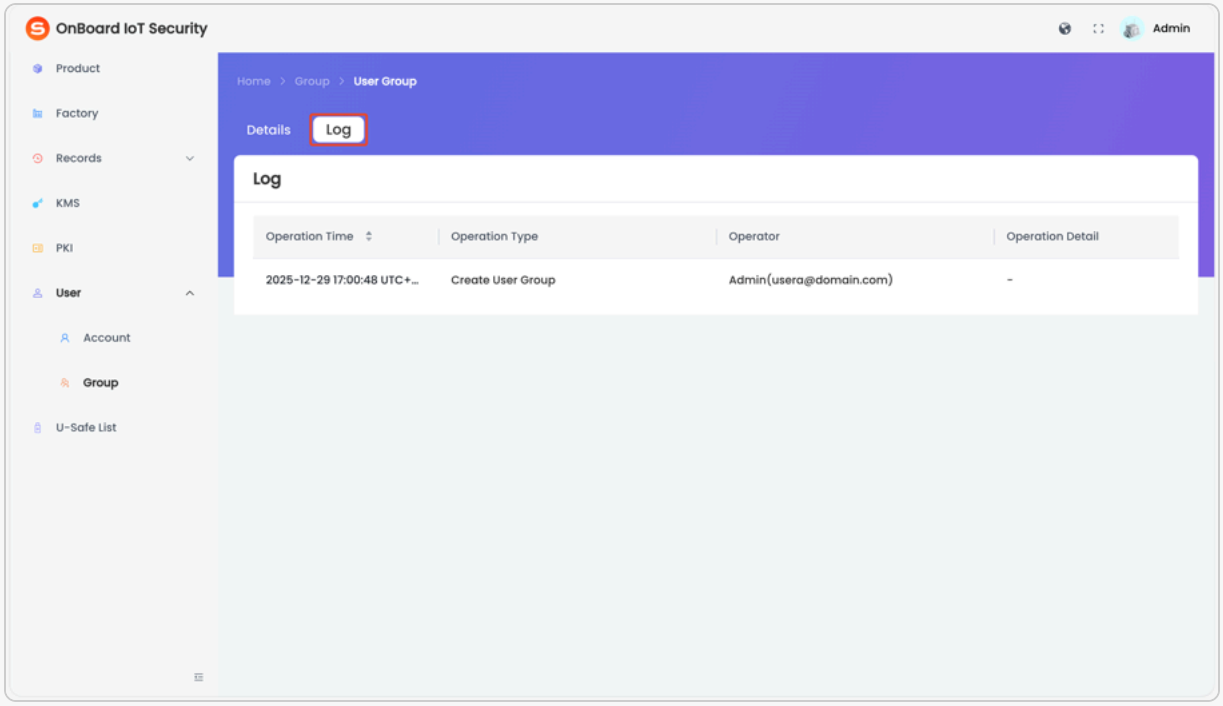
Check logs

> **TIP**
>
> Through the group logs, you can check the historical operations of the group.

1. Sign in to OBIS.
2. In the left menu, click **User** > **Group** to navigate to the **Group Management** page.
3. On the **Group Management** page, select the corresponding group name.



4. On the **Log** tab, you can check the historical operations of the group (such as **Operation Time** and **Operation Type**).


OnBoard IoT Security
Admin
Product
Factory
Records
KMS
PKI
User
Account
Group
U-Safe List
Home > Group > User Group
Details
Log
Log
Operation Time
Operation Type
Operator
Operation Detail
2025-12-29 17:00:48 UTC+...
Create User Group
Admin(usera@domain.com)
-

Version: v3.2.1

How to manage groups in OnBoard™ IoT Security

This article will guide you through the following operations in OnBoard™ IoT Security (OBIS) to manage the access and permissions of user group members for OBIS:

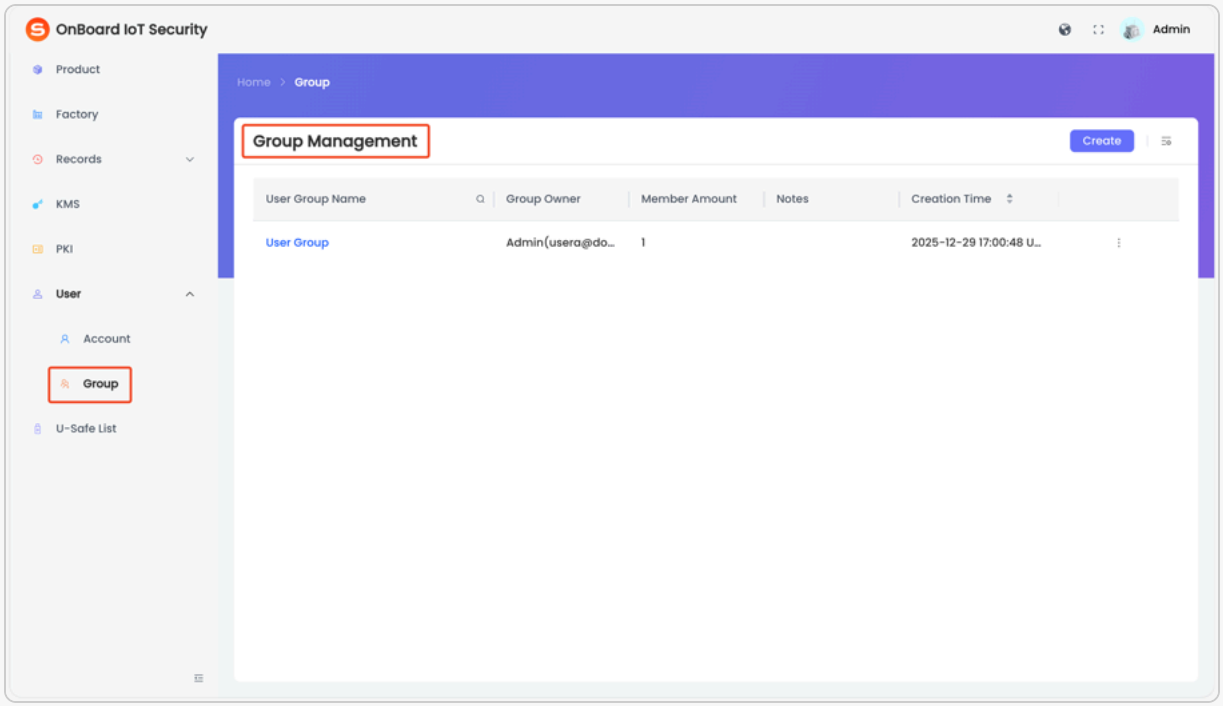
- Edit a user group
- Add a user group member
- Update the user group member role
- Remove a user group member
- Delete a user group

Prerequisites

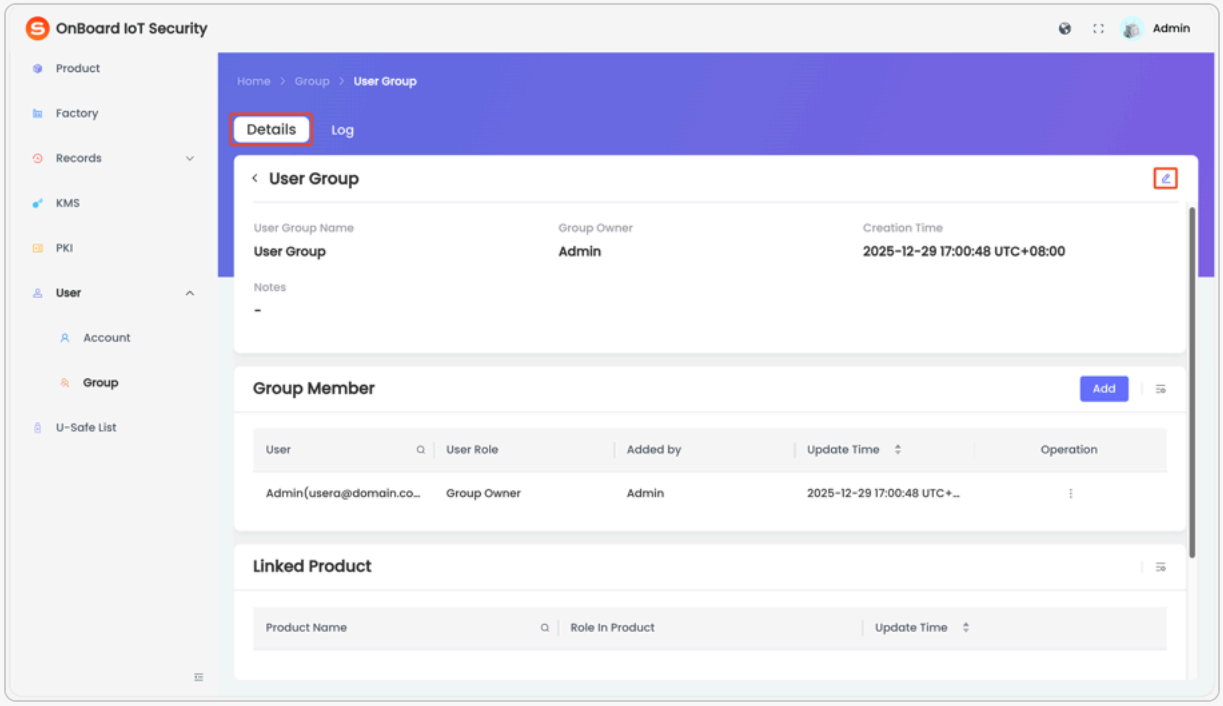
Before getting started, please ensure that you have OBIS role permissions as a System Manager.

Edit a user group

1. Sign in to OBIS.
2. In the left menu, click **User** > **Group** to navigate to the **Group Management** page.
3. On the **Group Management** page, select the corresponding user group name.

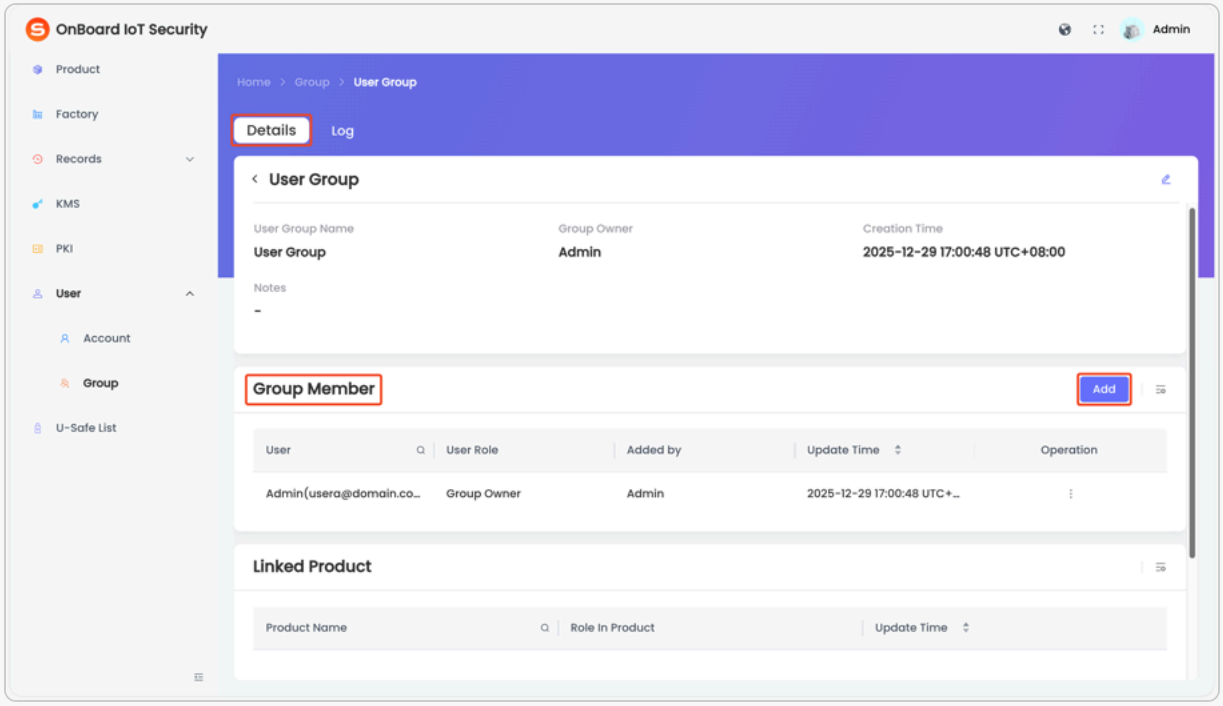


4. On the **Details** tab, click the Edit icon to change the user group information (**User Group Name**, **Group Owner**, and **Notes**).

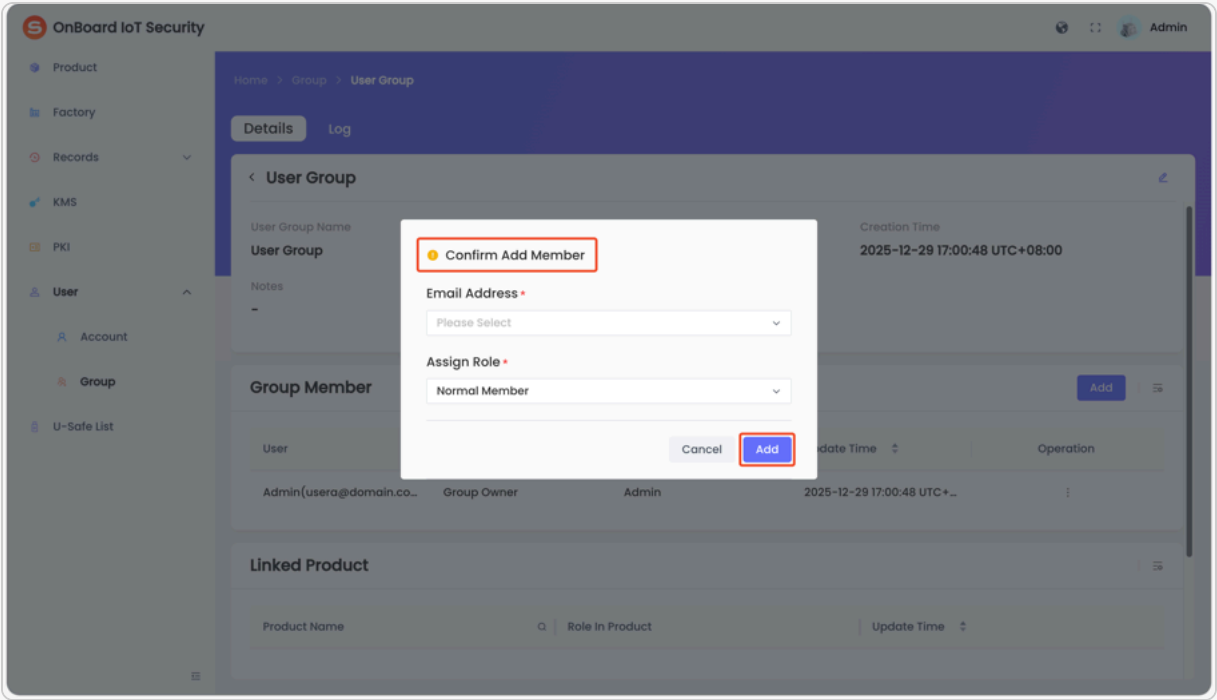


Add a user group member

1. Sign in to OBIS.
2. In the left menu, click **User** > **Group** to navigate to the **Group Management** page.
3. On the **Group Management** page, click the corresponding user group name.
4. On the **Details** tab, click **Add** in the **Group Member** area.

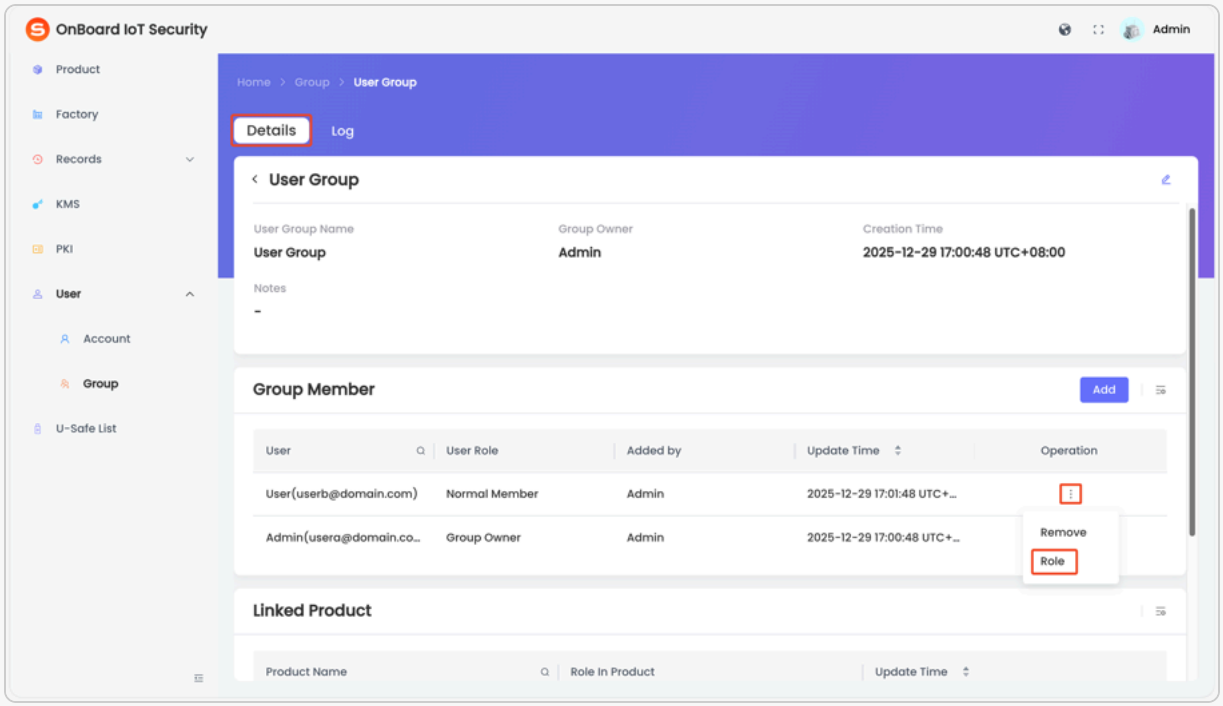


5. In the **Confirm Add Member** window, enter the email address and assign the corresponding roles. Then, click **Add** to complete adding the user group member.

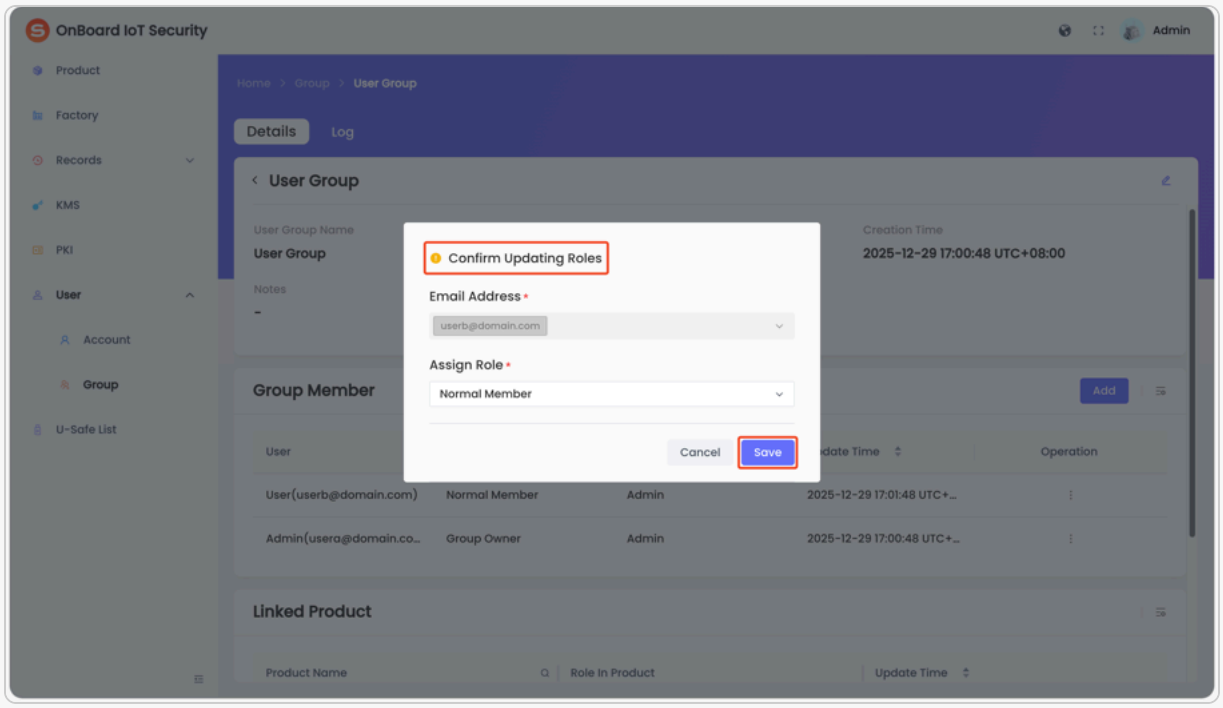


Update a user group member role

1. Sign in to OBIS.
2. In the left menu, click **User** > **Group** to navigate to the **Group Management** page.
3. On the **Group Management** page, click the corresponding user group name.
4. On the **Details** tab, locate the **Group Member** area and then click the Options ⋮ icon for the corresponding member and select **Role**.

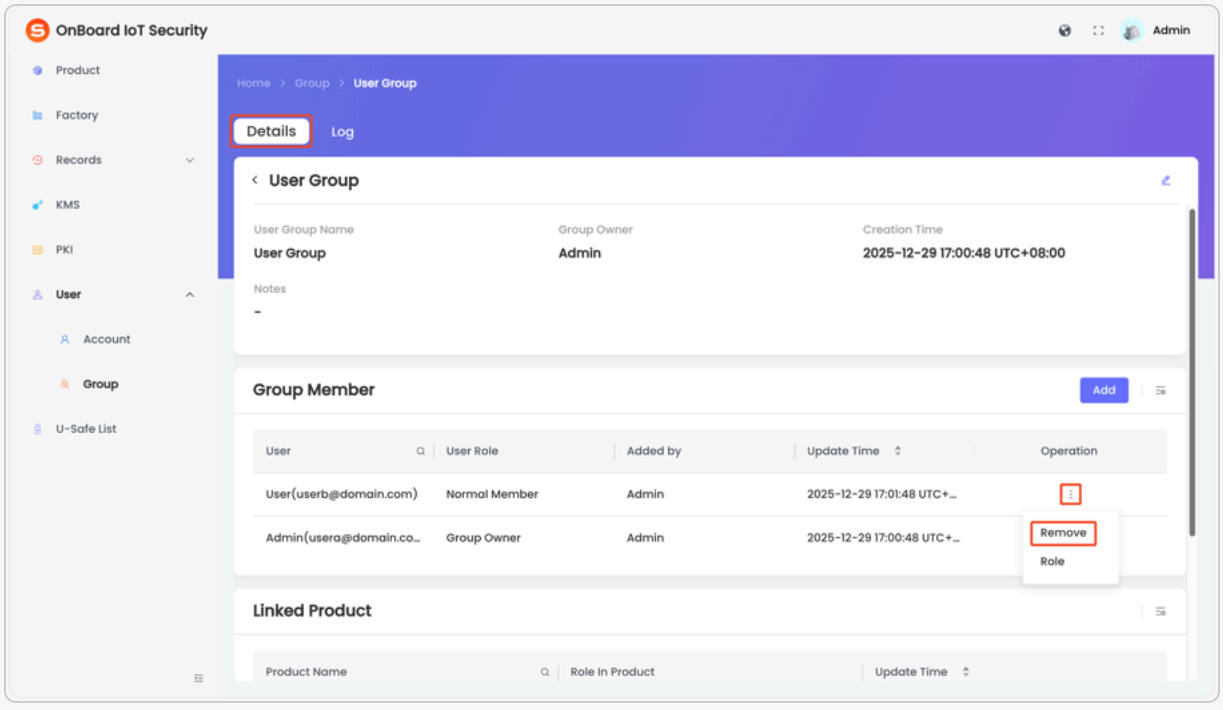


5. In the **Confirm Updating Roles** window, select the corresponding roles. Then, click **Save** to update the user group member role.

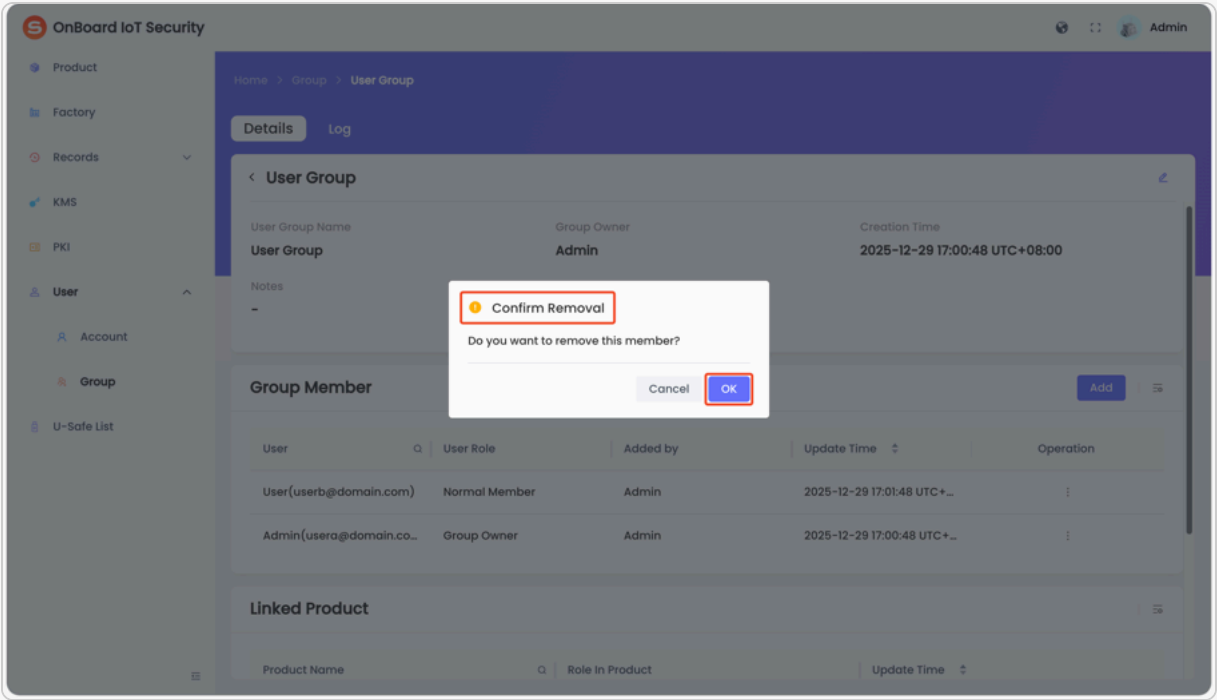


Remove a user group member

1. Sign in to OBIS.
2. In the left menu, click **User** > **Group** to navigate to the **Group Management** page.
3. On the **Group Management** page, select the corresponding user group name.
4. On the **Details** tab, locate the **Group Member** area and then click the Options icon for the corresponding member and select **Remove**.



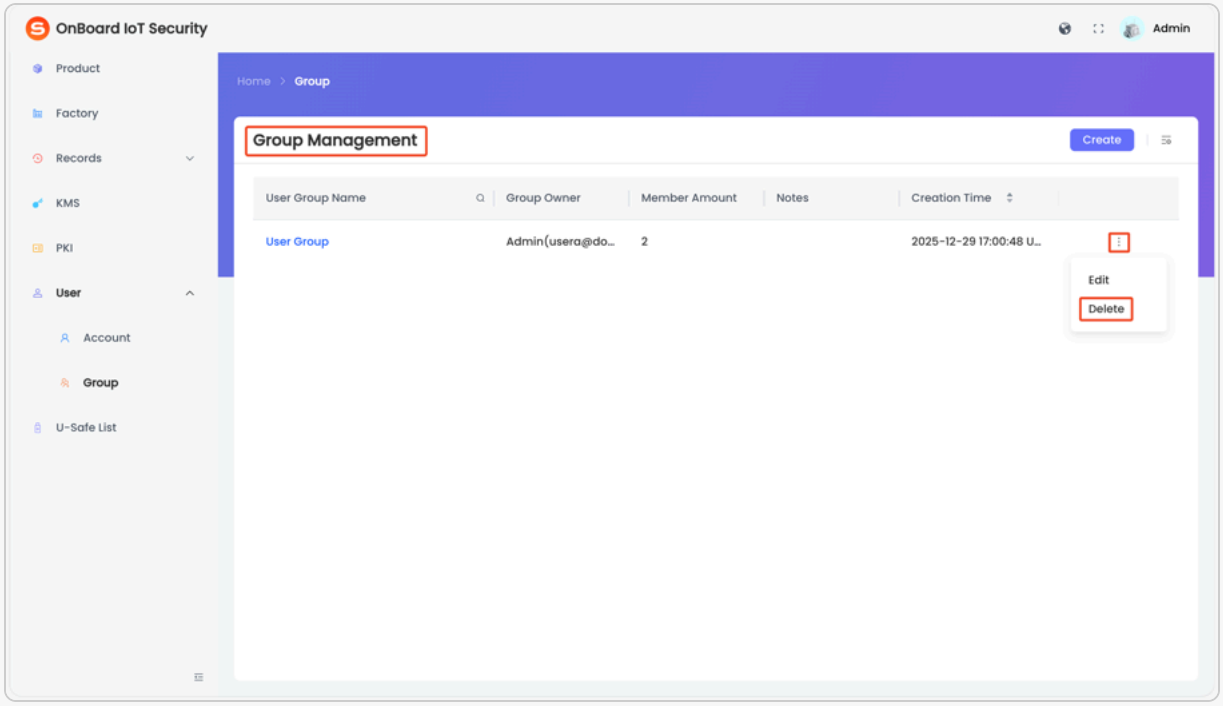
5. In the **Confirm Removal** window, click **OK** to remove the user group member.



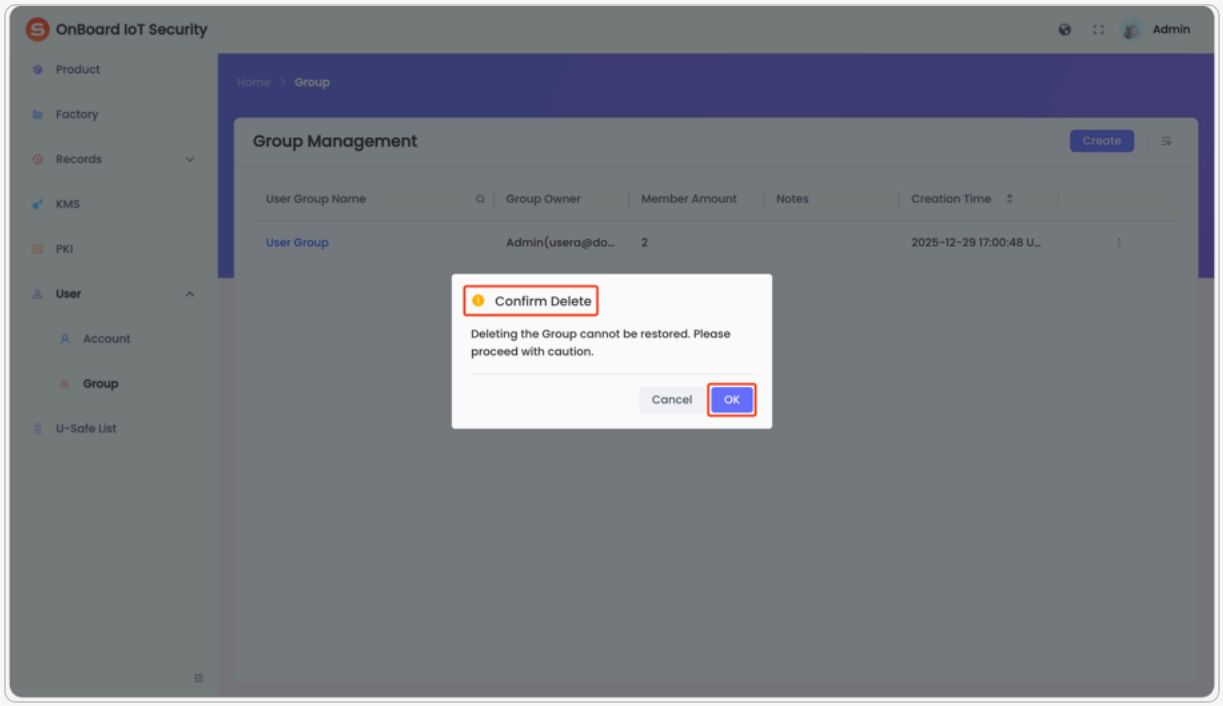
Delete a user group

> **WARNING**
>
> Deleting a group cannot be restored. Please proceed with caution.

1. Sign in to OBIS.
2. In the left menu, click **User** > **Group** to navigate to the **Group Management** page.
3. On the **Group Management** page, click the Options ⋮ icon for the corresponding user group name and select **Delete**.



4. In the **Confirm Delete** window, click **OK** to delete the user group.



Version: v3.2.1

How to check U-Safes in OnBoard™ IoT Security

This article will guide you on how to check the parameters and status of the U-Safe bound to the factory in OnBoard™ IoT Security (OBIS).

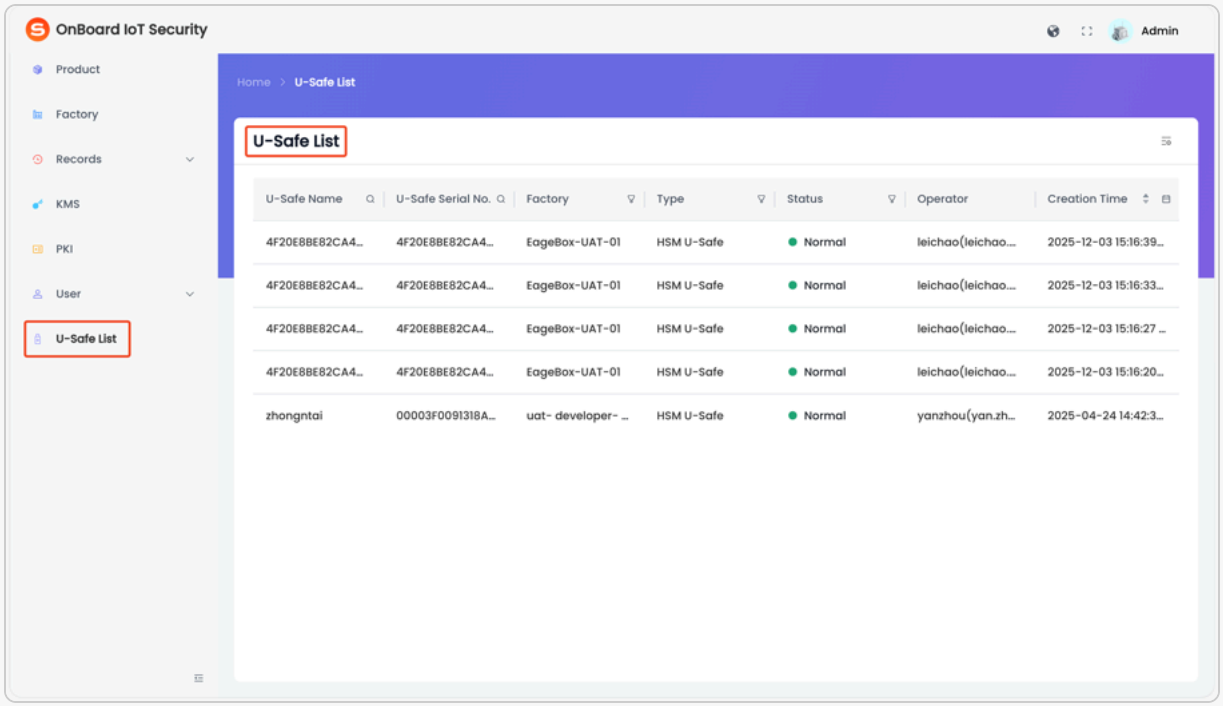
Prerequisites

Before getting started, please ensure that:

- The factory has been created and that the HSM U-Safe and Identity U-Safe have been bound.
- You have OBIS role permissions as a System Manager.

Instructions

1. Sign in to OBIS.
2. In the left menu, click **U-Safe List**. On the **U-Safe List** page, you can check all the U-Safe information bound to factories within the tenant.



Description of U-Safe status:

Status	Description
Normal	The U-Safe status after adding the HSM U-Safe and Identity U-Safe during factory configuration, and the U-Safe can be used normally.
Locked	The U-Safe has been locked, and can't be used normally.

Version: v3.2.1

Overview of OnBoard Edge Service

> **INFO**
>
> Operations in OnBoard Edge Service are applicable only to local factories, while operations for cloud factories can be performed directly in OnBoard™ IoT Security (OBIS). Please contact Snowball for assistance with the deployment.

OnBoard Edge Service manages Production Batches at the factory level, enforces authorization constraints locally, and enables intranet-connected Programming Stations to perform production execution. It provides an intranet-accessible web portal through which factory managers can monitor production status and perform administrative operations. OnBoard Edge Service provides the following core capabilities:

Production Batch Management

It supports concurrent Production Batch operations from one or more OEMs. Factory managers monitor batch status, allocate batches to Programming Stations, and manage batch lifecycle through the web portal.

Programming Station Management

Factory managers can manage Programming Stations through the web portal, including:

- Create and configure Programming Stations, including secure authentication method for connections to OnBoard Edge Service (Identity U-Safe, password, or no password)
- Allocate batches to Programming Stations from OnBoard Edge Service (alternatively, operators can specify a Batch ID or Production Order ID directly from Programming Station Software)
- Deliver the associated Programming Station Software package to Programming Stations

Production Device Record Management

OnBoard Edge Service generates per-device production records from Programming Stations and submits them to the OBIS cloud platform. It can also interoperate with existing factory systems—such as MES and factory testing systems—to exchange production records. OnBoard Edge Service does not have access to product key materials at any point. All cryptographic key material processing occurs exclusively within the EdgeHSM hardware boundary.

Version: v3.2.1

How to sign in to OnBoard Edge Service

> **INFO**
>
> Operations in OnBoard Edge Service are applicable only to local factories, while operations for cloud factories can be performed directly in OnBoard™ IoT Security (OBIS).

This article will guide you on how to sign in to OnBoard Edge Service using either the "Email + Password" or "Email + Verification code" method.

Prerequisites

Before getting started, please ensure that:

- The factory has been created and configured. For detailed instructions, refer to the steps for creating a factory, configuring a factory, and configuring a programming station.
- You have factory role permissions as a Factory Manager.
- Have contacted Snowball for assistance in deploying of OnBoard Edge Service.
- You have obtained the HSM U-Safe for identity authentication from Snowball, and inserted the corresponding HSM U-Safe (added during factory configuration) into the local factory server.
- Your sign-in device is within the factory service intranet environment.

> **TIP**
>
> The factory server requires at least 1 HSM U-Safe. Inserting multiple HSM U-Safes can improve server performance, and the quantity can be purchased according to the needs of the production line.

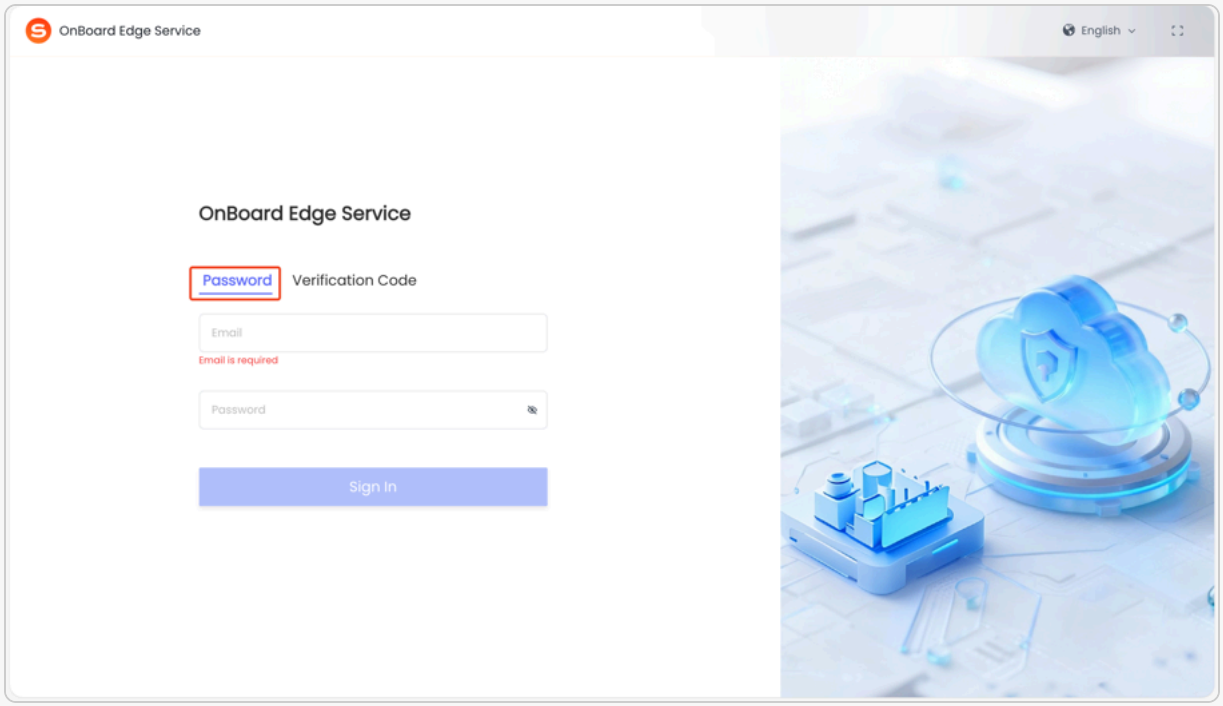
Method one: Password

> **INFO**
>
> - When the Internet is disconnected, OnBoard Edge Service only supports identity verification using the "Email + Password" method.
> - After a Factory Manager is added, the system automatically generates an initial password and sends it to the corresponding email address. Currently, this randomly generated password can be reset, but custom modification is not yet supported.

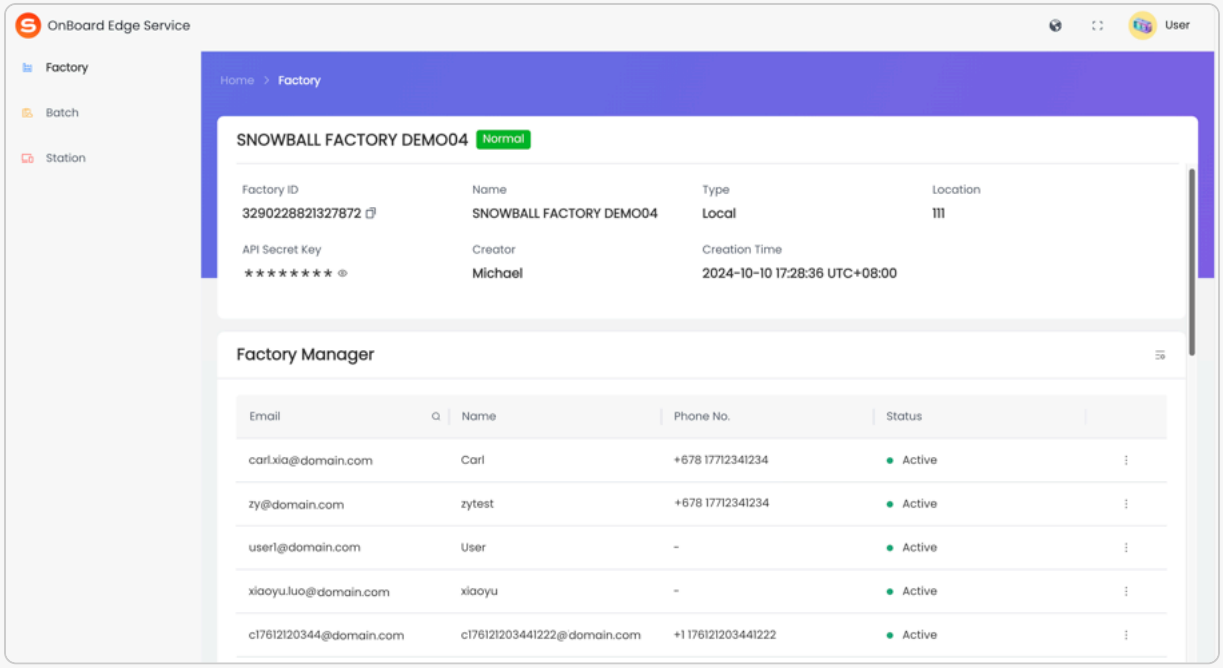
1. Enter the following address in the browser address bar: **http://<Service Address>:32334/login**.

   > **TIP**
   >
   > Replace **<Service Address>** with the actual service IP. For more information, refer to how to check service address.

2. On the OnBoard Edge Service sign-in page, select **Password**, enter your email and password, and then click **Sign In**.



3. After a successful sign-in, the page will navigate to the home page of OnBoard Edge Service.

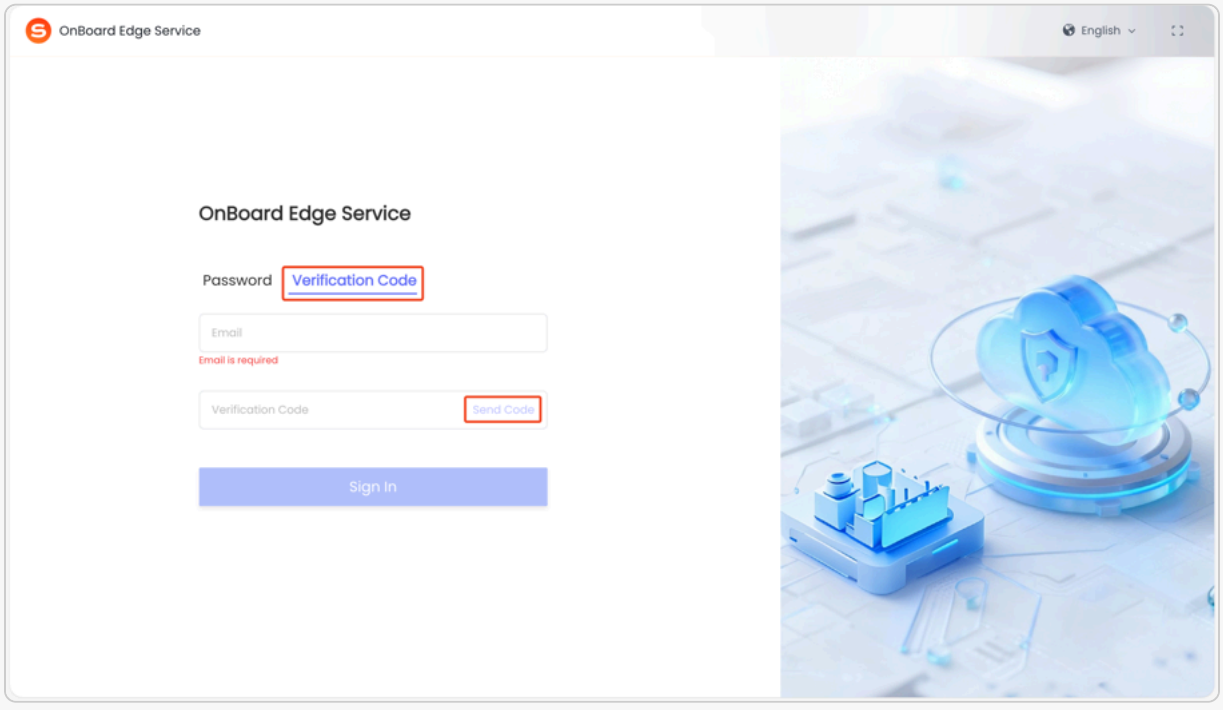


Method two: Verification code

1. Enter the following address in the browser address bar: **http://<Service Address>:32334/login**.

   > **TIP**
   >
   > Replace **<Service Address>** with the actual service IP. For more information, refer to how to check service address.

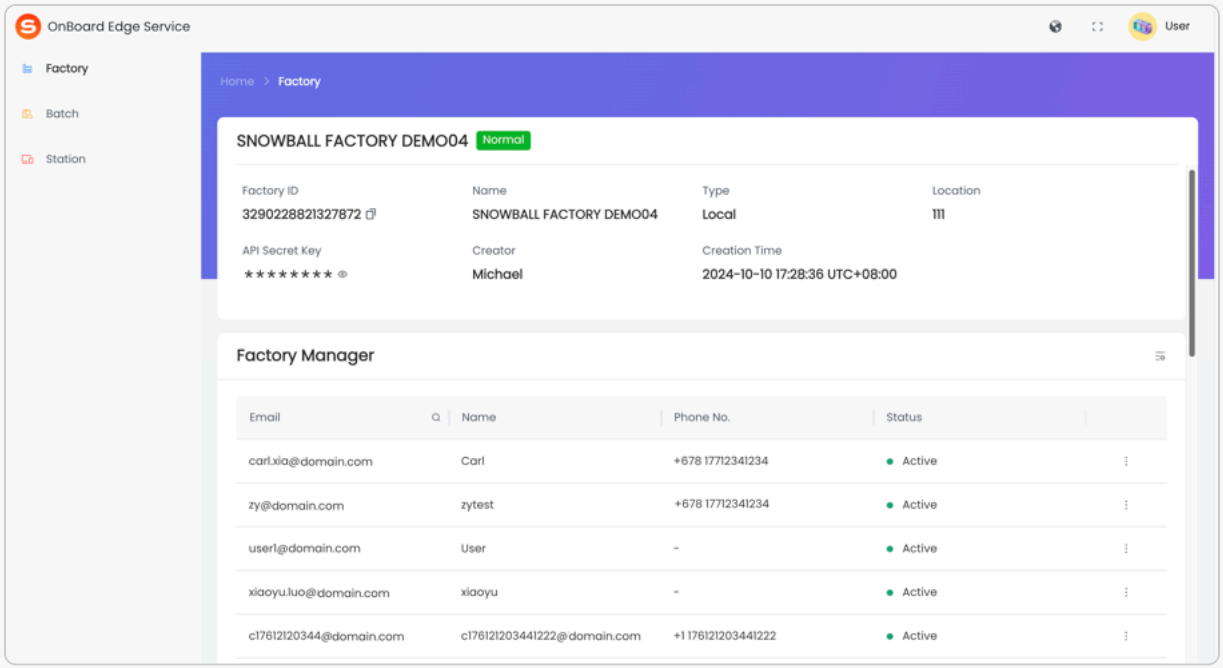
2. On the OnBoard Edge Service sign-in page, select **Verification Code**, enter your email, and then click **Send Code**.



3. Sign in to your email, check the verification email from OBIS which contains a six-digit verification code.

4. On the OBIS sign-in page, enter the six-digit verification code you received in your email, and then click **Sign In**.

5. After a successful sign-in, the page will navigate to the home page of OnBoard Edge Service.



Version: v3.2.1

How to manage factories in OnBoard Edge Service

> **INFO**
>
> Operations in OnBoard Edge Service are applicable only to local factories, while operations for cloud factories can be performed directly in OnBoard™ IoT Security (OBIS).

This article will guide you through the following operations in OnBoard Edge Service:

- Reset the Factory Manager password
- Delete a Factory Manager
- Lock or unlock an HSM U-Safe
- Lock or unlock an Identity U-Safe

Prerequisites

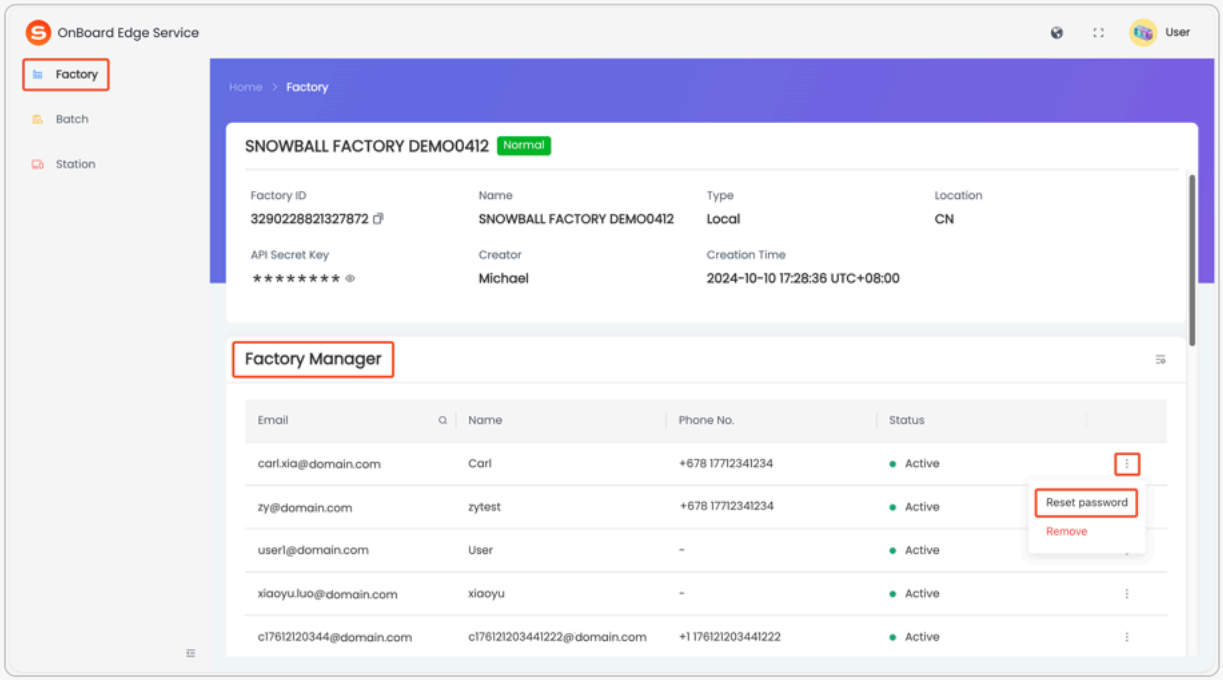
Before getting started, please ensure that you can sign in to OnBoard Edge Service.

Reset the Factory Manager password

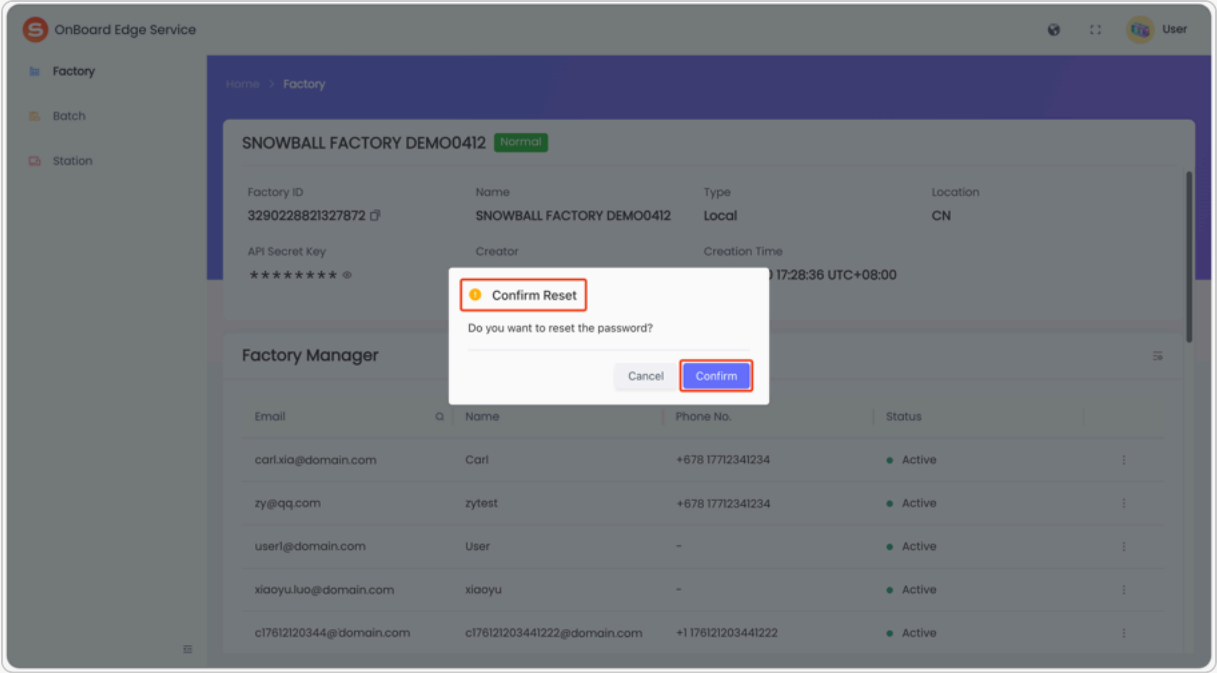
> **TIP**
>
> After the password is reset, the system automatically generates a random password and sends it to the corresponding email address.

1. Sign in to OnBoard Edge Service.

2. In the left menu, click **Factory**, locate the **Factory Manager** area, and then click the Options ⋮ icon for the corresponding Factory Manager and select **Reset password**.



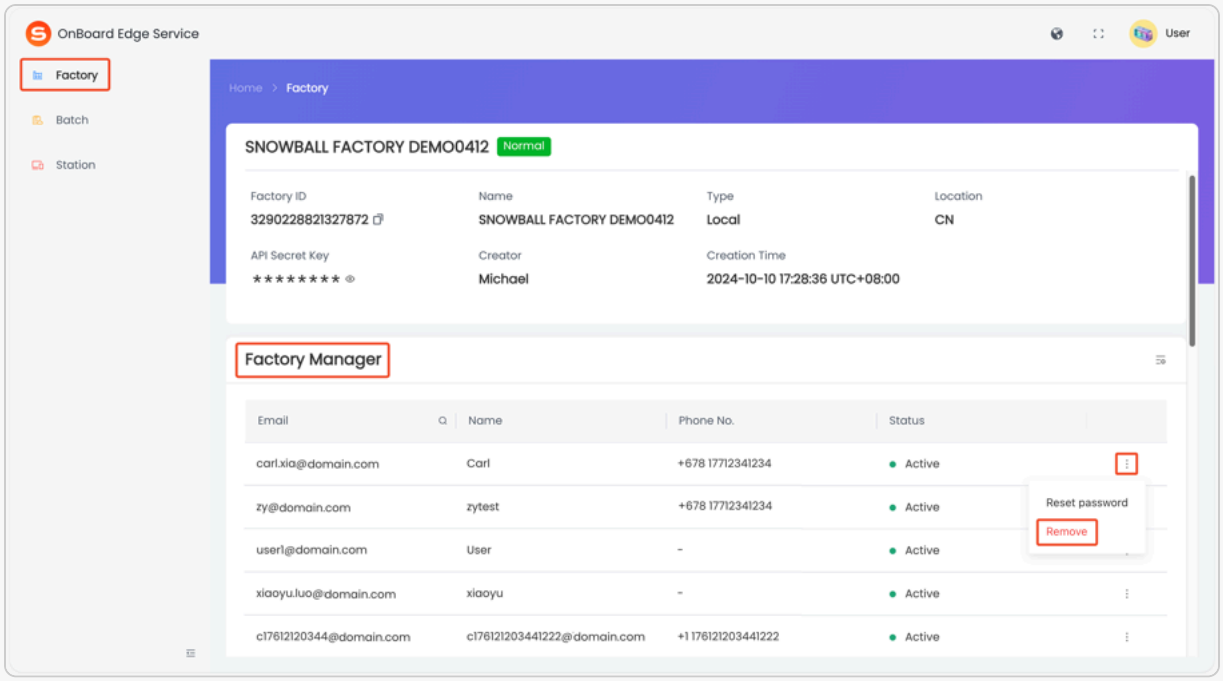
3. In the **Confirm Reset** window, click **Confirm** to reset the password.



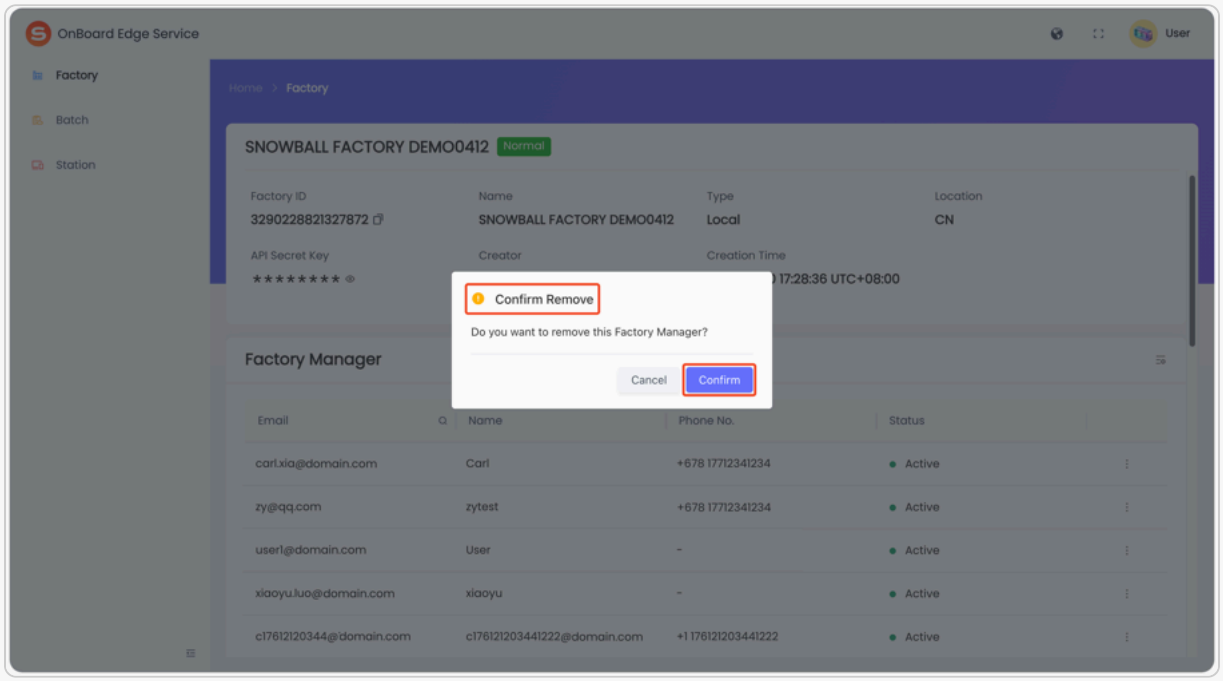
Delete a Factory Manager

1. Sign in to OnBoard Edge Service.

2. In the left menu, click **Factory**, locate the **Factory Manager** area, and then click the Options ⋮ icon for the corresponding Factory Manager and select **Remove**.



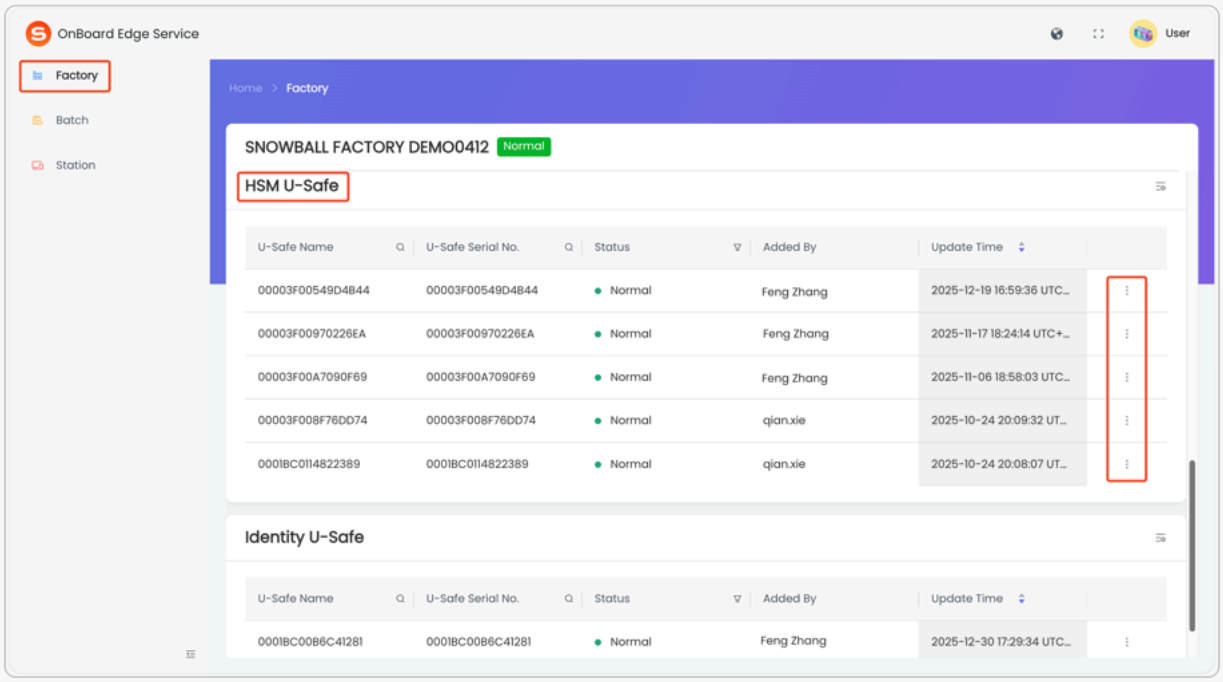
3. In the **Confirm Remove** window, click **Confirm** to delete the Factory Manager.



Lock or unlock an HSM U-Safe

> **TIP**
>
> Lock the HSM U-Safe to prevent its use and unbinding. Once locked, its status will change to **Locked**.

1. Sign in to OnBoard Edge Service.
2. In the left menu, click **Factory**, locate the **HSM U-Safe** area, and then click the Options ⋮ icon for the corresponding HSM U-Safe and select **Lock** or **Unlock**.

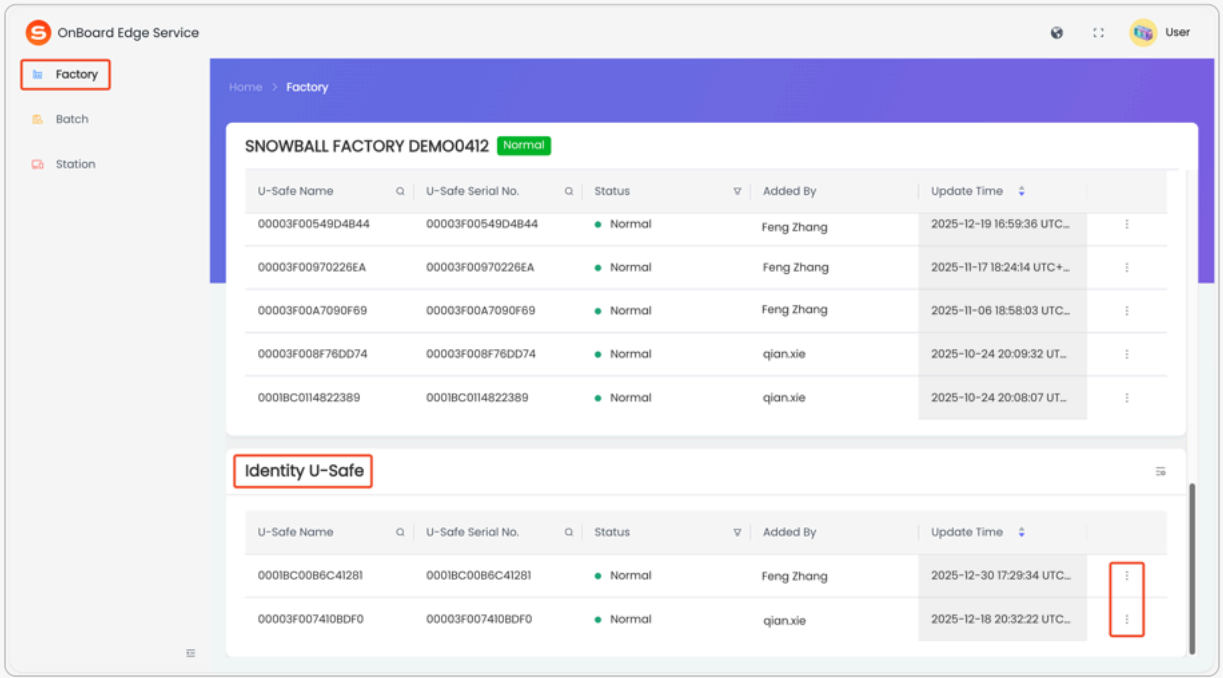


3. In the pop-up window, click **Confirm** to lock or unlock the HSM U-Safe.

Lock or unlock an Identity U-Safe

> **TIP**
>
> Lock the Identity U-Safe to prevent its use and unbinding. Once locked, its status will change to **Locked**.

1. Sign in to OnBoard Edge Service.
2. In the left menu, click **Factory**, locate the **Identity U-Safe** area, and then click the Options ⋮ icon for the corresponding Identity U-Safe and select **Lock** or **Unlock**.



3. In the pop-up window, click **Confirm** to lock or unlock the Identity U-Safe.

Version: v3.2.1

How to manage a batch and assign a programming station in OnBoard Edge Service

> **INFO**
>
> Operations in OnBoard Edge Service are applicable only to local factories, while operations for cloud factories can be performed directly in OnBoard™ IoT Security (OBIS).

This article will guide you through the following operations in OnBoard Edge Service:

- Check batches
- Assign a programming station to a batch
- Unassign a programming station from a batch
- Turn off or delete a batch

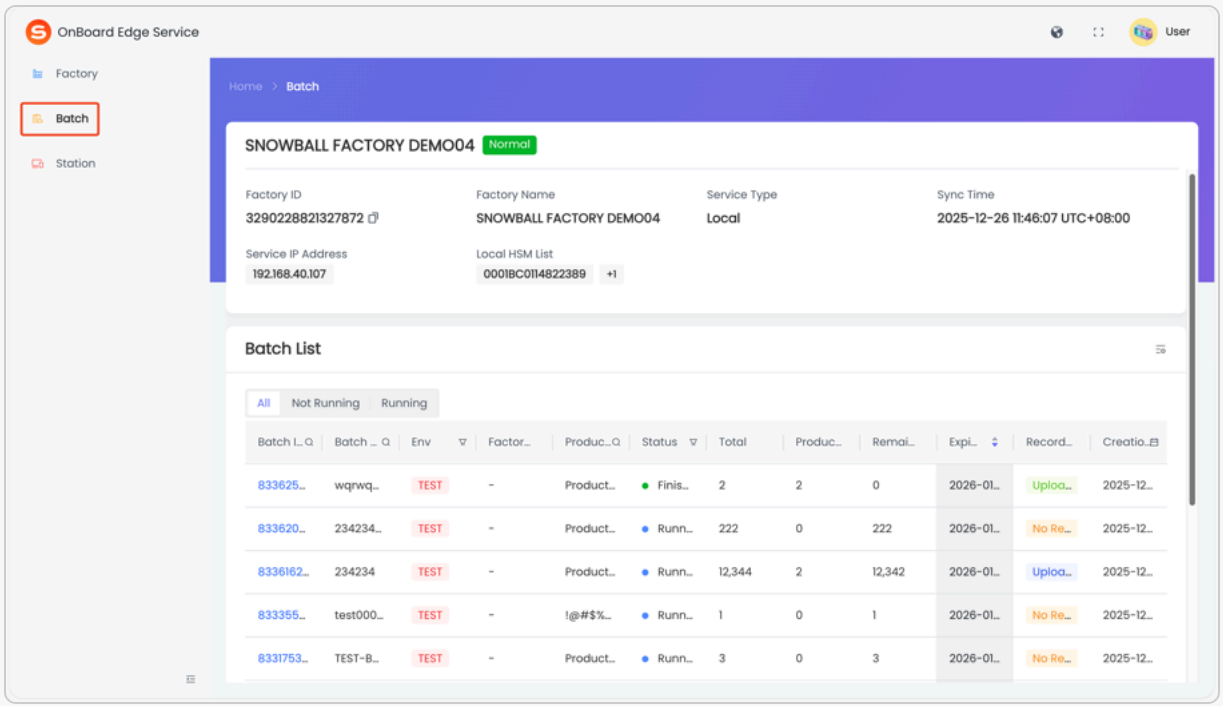
Prerequisites

Before getting started, please ensure that you can sign in to OnBoard Edge Service.

Check batches

1. Sign in to OnBoard Edge Service.

2. In the left menu, click **Batch** to navigate to the **Batch** page.

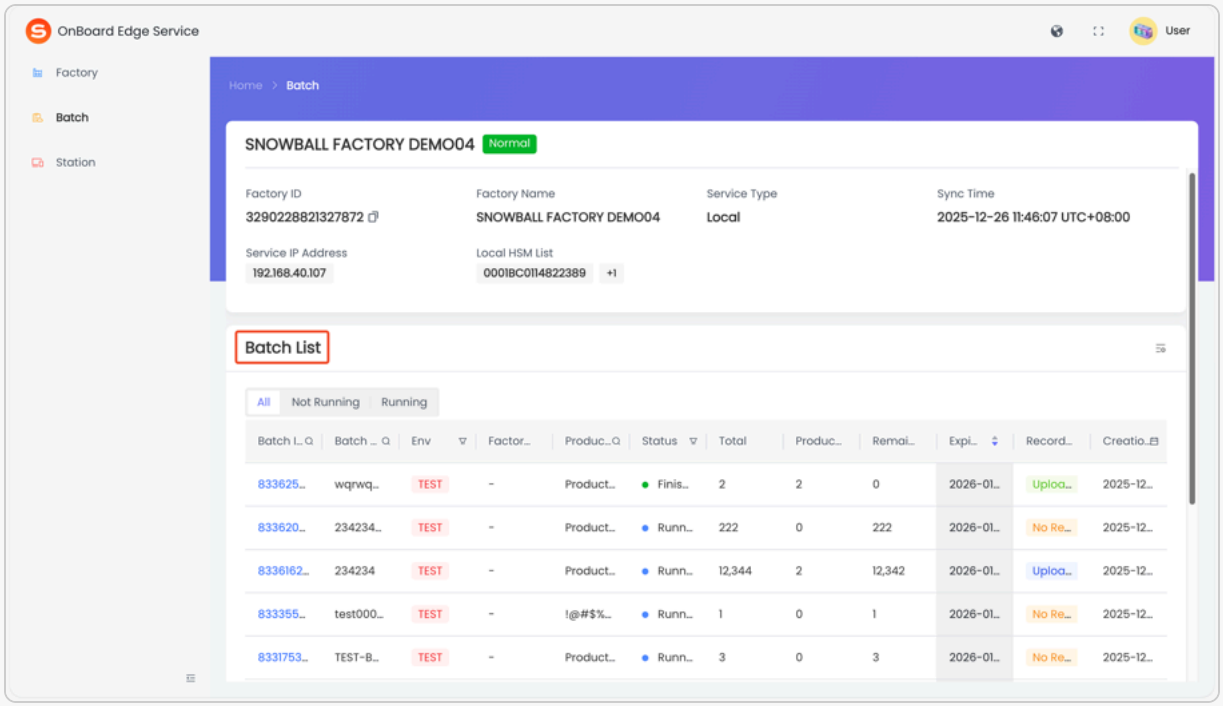


Description of batch status:

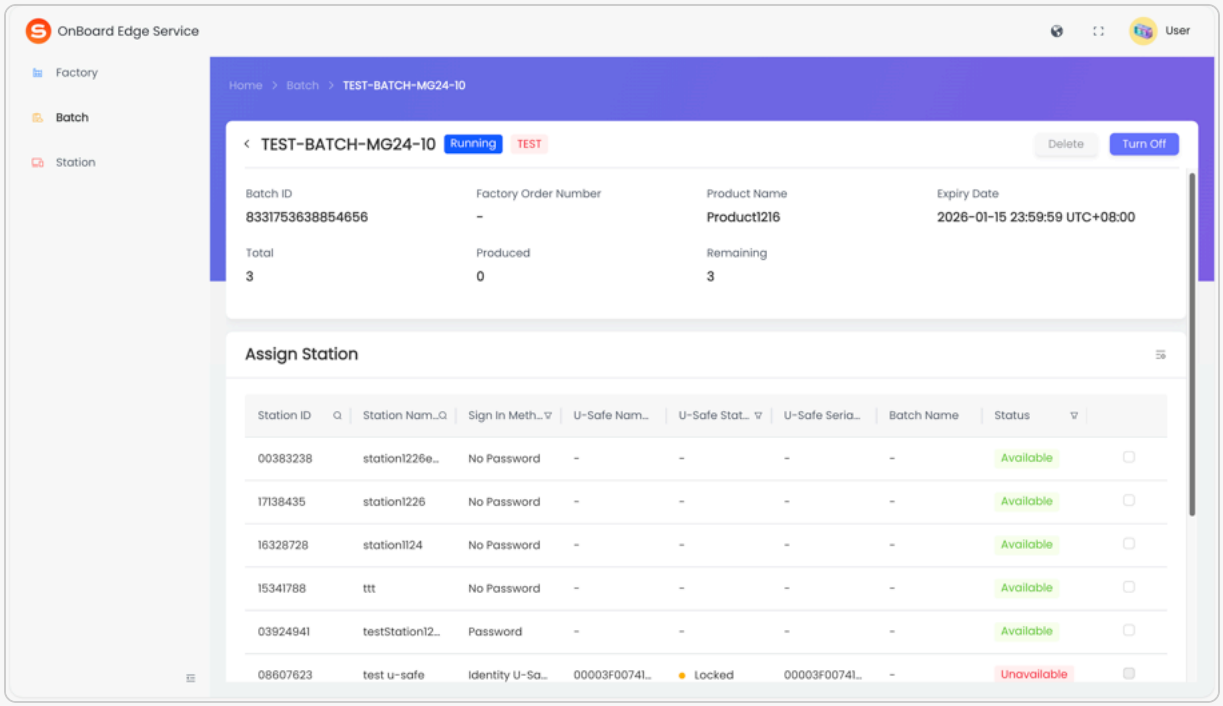
Status	Can be turned on	Description
Running	-	The default state after creating a batch. You can bind it to a programming station.

Status	Can be turned on	Description
Stopped	Yes	The batch is currently turned off. You can turn it on to bind it to a programming station.
Finished	No	The status after batch programming is completed.
Expired	No	The batch has expired before the programming was completed. **Note**: After a batch expires, most functions related to that batch in OnBoard Edge Service will no longer operate normally, including programming in the Programming Station client. It is recommended to delete the expired batches.

1. On the **Batch** page, locate the **Batch List** area, and then select the corresponding batch ID to navigate to the batch page.



2. On the batch page, you can check the batch-related information.

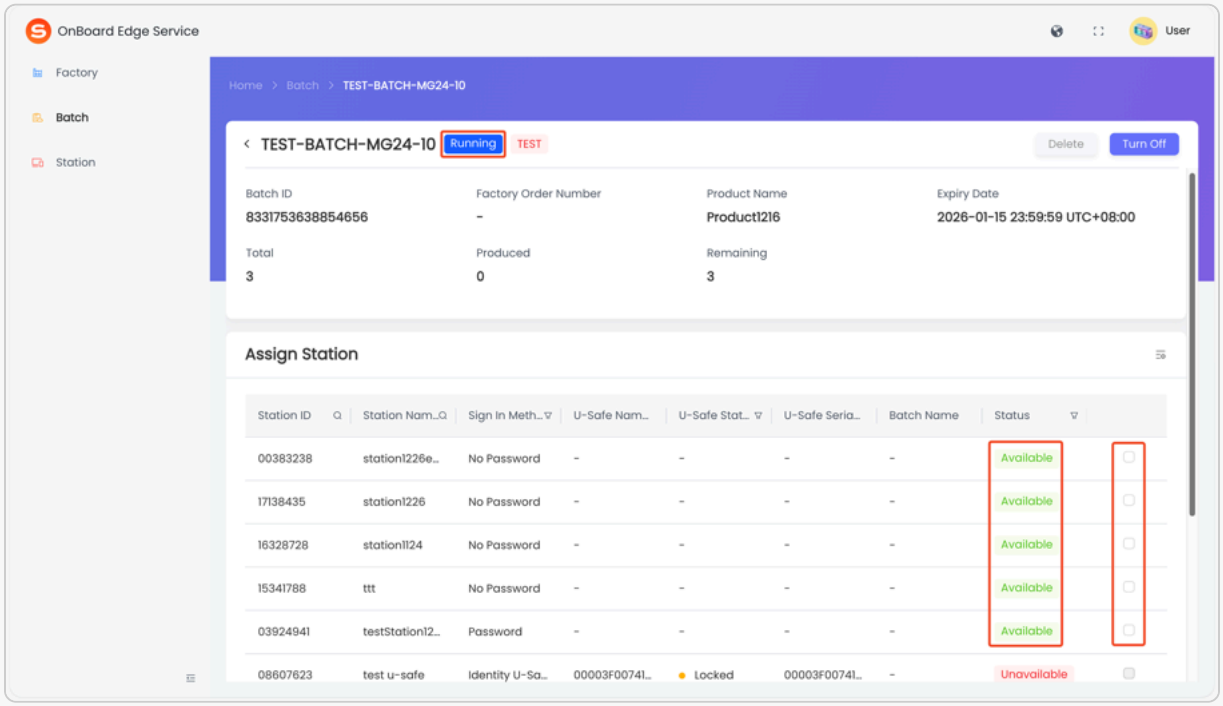


Assign a programming station to a batch

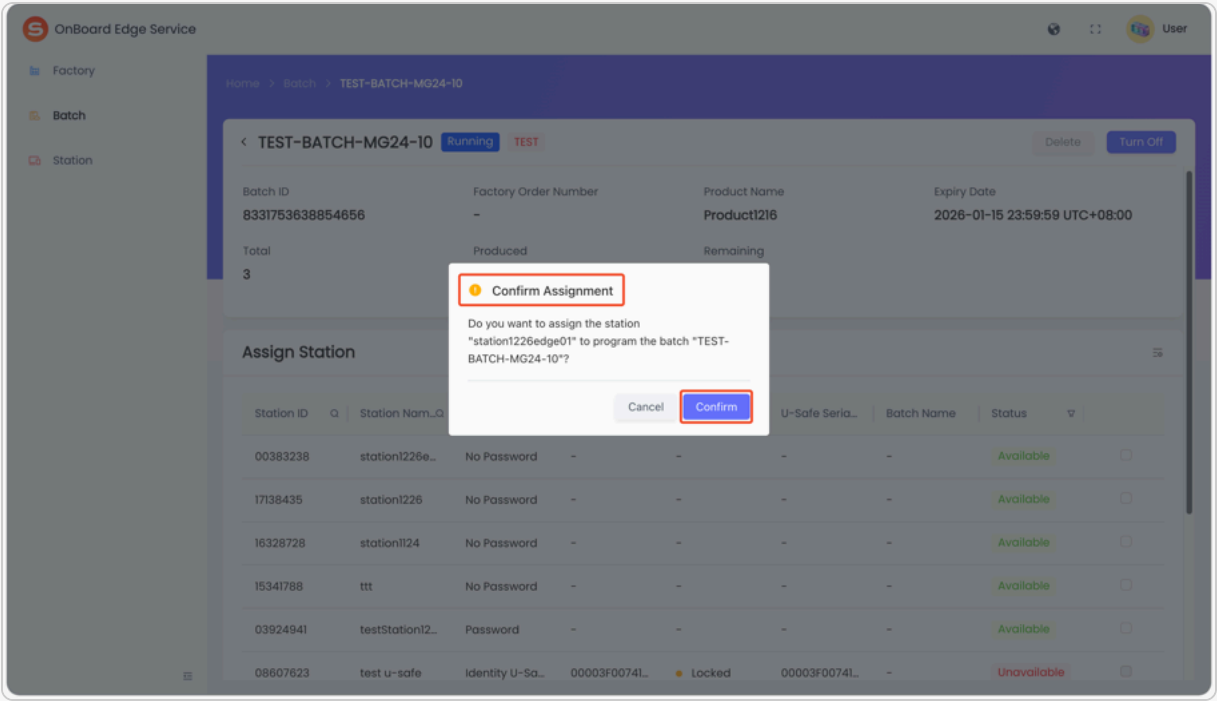
> **TIP**
>
> A batch can be assigned to multiple programming stations, while a programming station can only program one batch at the same time.

Assign a programming station with the **Available** status to a batch for programming:

1. Sign in to OnBoard Edge Service.
2. In the left menu, click **Batch**, and then select the corresponding batch ID to navigate to the batch page.
3. On the batch page, confirm that the batch status is **Running**, and then select the programming station with the **Available** status for assignment.



4. In the **Confirm Assignment** window, click **Confirm** to complete the assignment of the programming station. After the assignment, the programming station status will change to **Busy**.



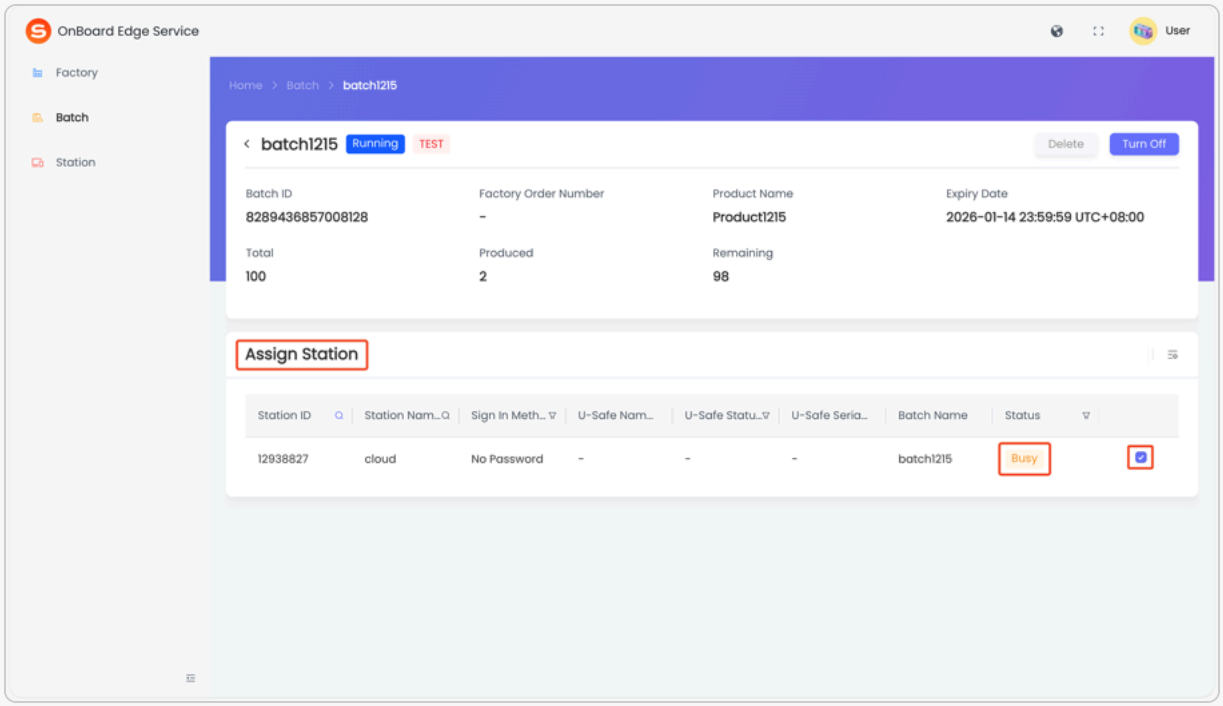
Unassign a programming station from a batch

> **WARNING**
>
> After unassignment, the corresponding programming station can no longer be used for programming this batch.

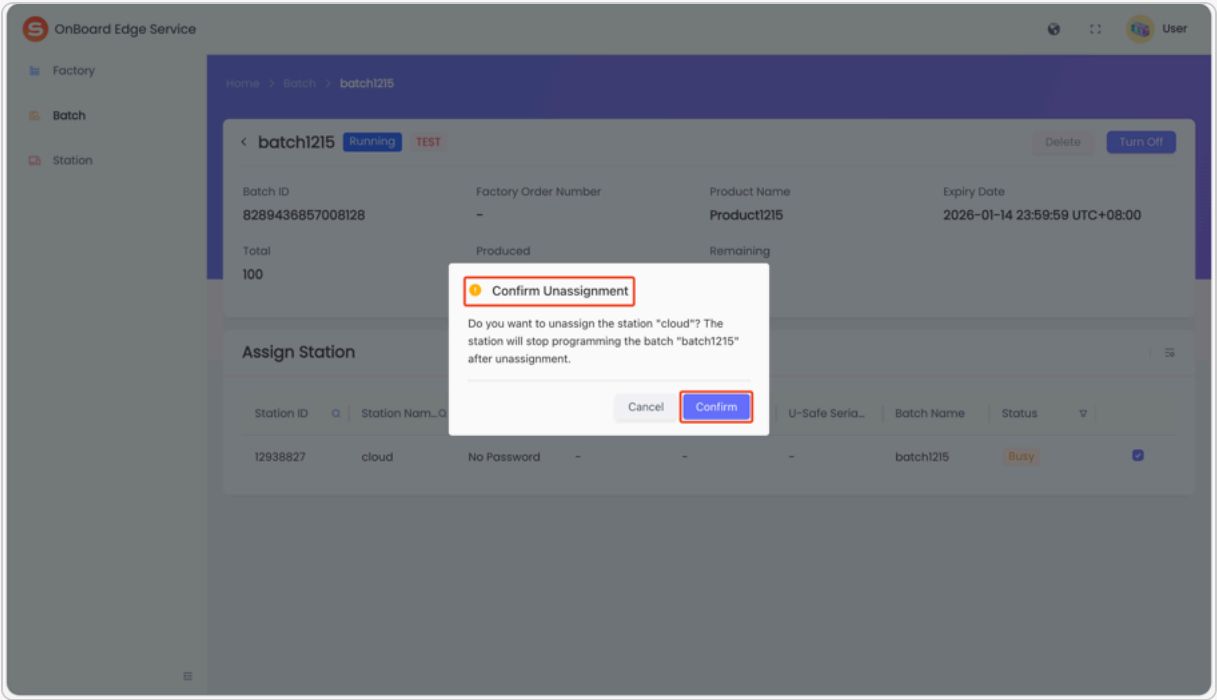
1. Sign in to OnBoard Edge Service.

2. In the left menu, click **Batch**, and then select the corresponding batch ID to navigate to the batch page.

3. In the **Assign Station** area of the batch page, find the corresponding batch and uncheck the programming station with the **Busy** status.



4. In the **Confirm Unassignment** window, click **Confirm** to complete the unassignment of the programming station. After unassignment, the programming station status will change to **Available**.

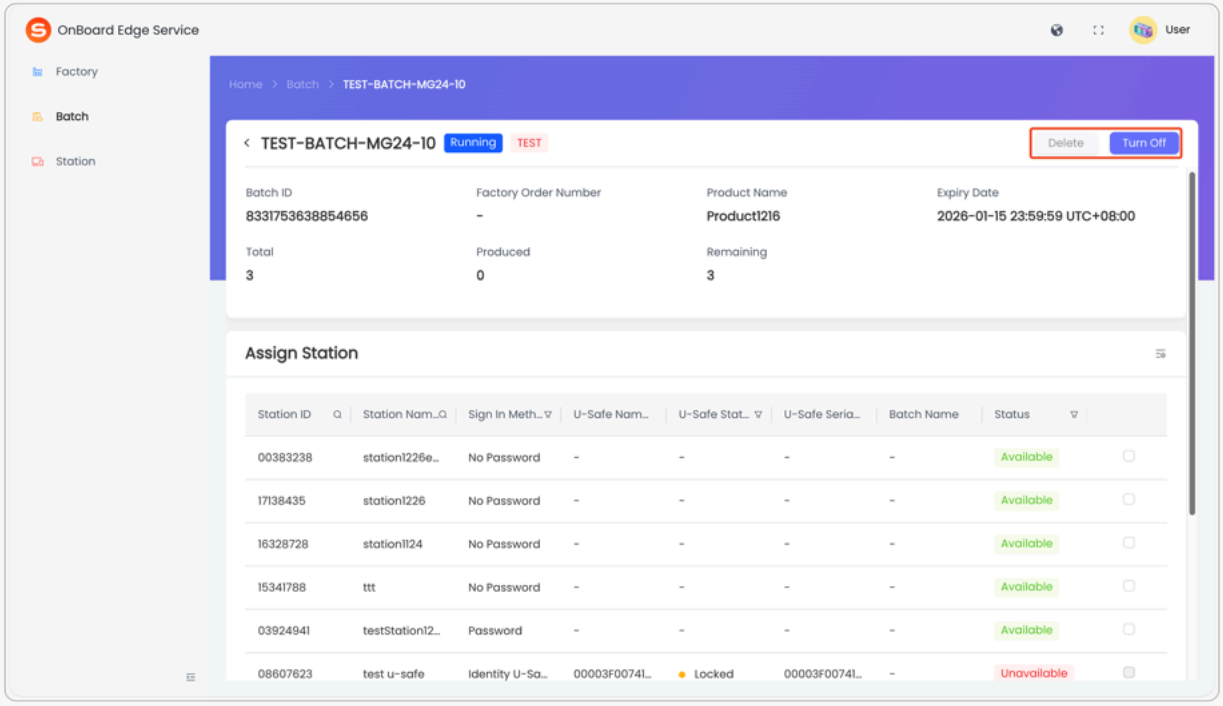


Turn off or delete a batch

> **INFO**
>
> After a batch is deleted, any programming stations assigned to that batch will be automatically unassigned.

1. Sign in to OnBoard Edge Service.
2. In the left menu, click **Batch**, and then select the corresponding batch ID to navigate to the batch page.
3. On the batch page, click **Turn Off** or **Delete**.

> **TIP**
>
> If the **Delete** button appears grayed out, it is currently unavailable. Please click **Turn Off** to turn off the batch. Once the batch is in a non-running status, it can be deleted.



Version: v3.2.1

How to configure and manage programming stations in OnBoard Edge Service

> **INFO**
>
> Operations in OnBoard Edge Service are applicable only to local factories, while operations for cloud factories can be performed directly in OnBoard™ IoT Security (OBIS).

> **TIP**
>
> You can configure and manage the programming station in OnBoard Edge Service or OBIS as needed.

This article will guide you through the following operations in OnBoard Edge Service:

- Create a programming station
- Bind an Identity U-Safe to a programming station
- Unbind an Identity U-Safe from a programming station
- Delete a programming station
- Check the programming station information
- Check the programming station password

Prerequisites

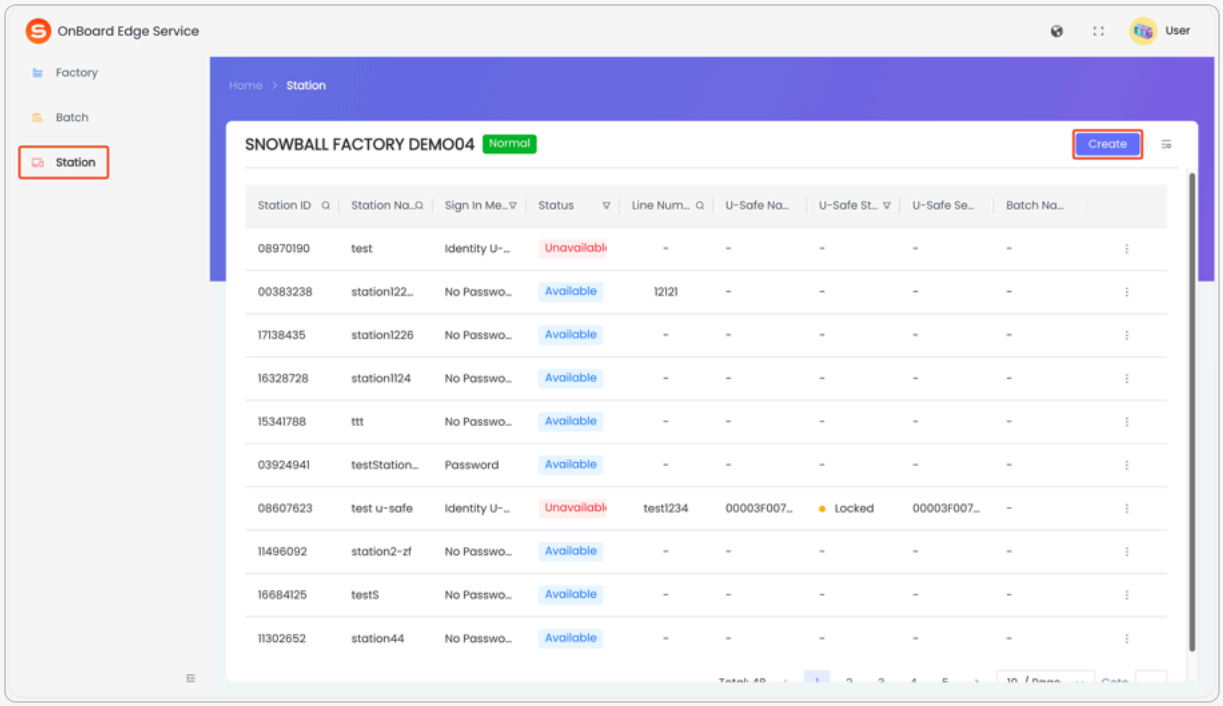
Before getting started, please ensure that:

- You can sign in to OnBoard Edge Service.
- You have obtained the Identity U-Safe for identity authentication from Snowball.

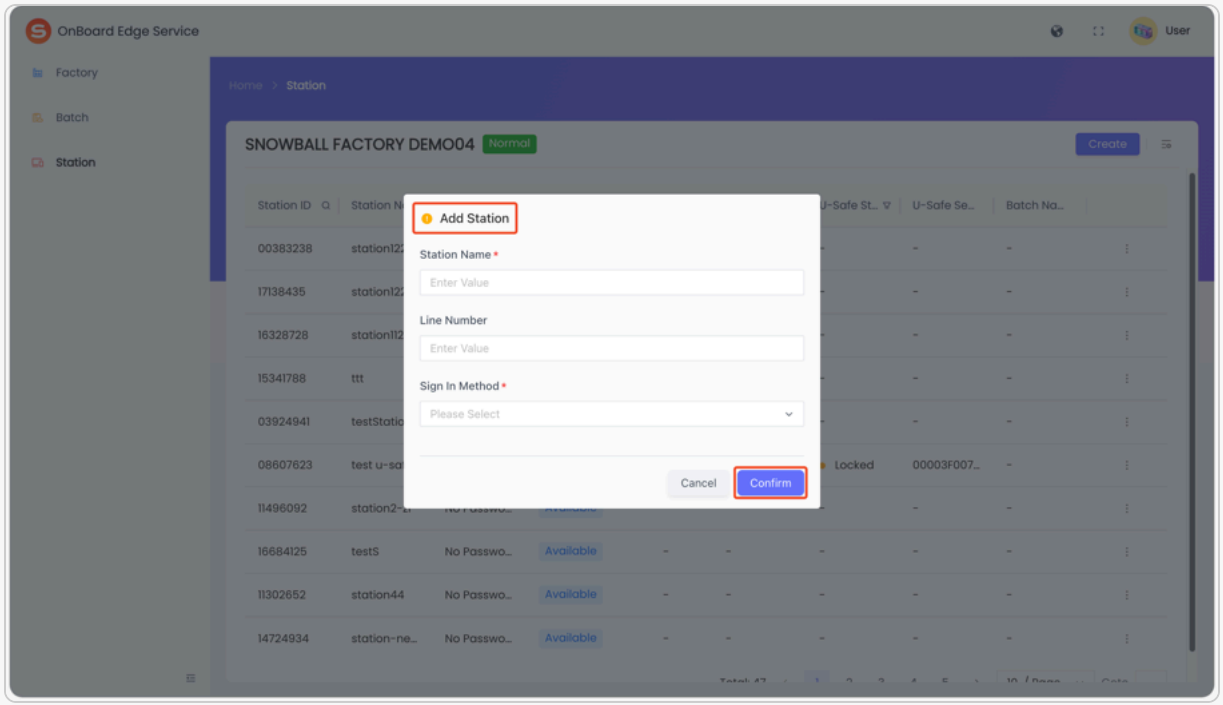
Create a programming station

> **TIP**
>
> After creating a programming station for the factory, you can assign specific programming stations to batches in OnBoard Edge Service as needed. Before programming a device, it is necessary to query and retrieve the corresponding batch information and programming data based on the programming station ID.
>
> Two programming methods are supported:
>
> - Using the Programming Station client
> - Using the OBIS Programming CLI

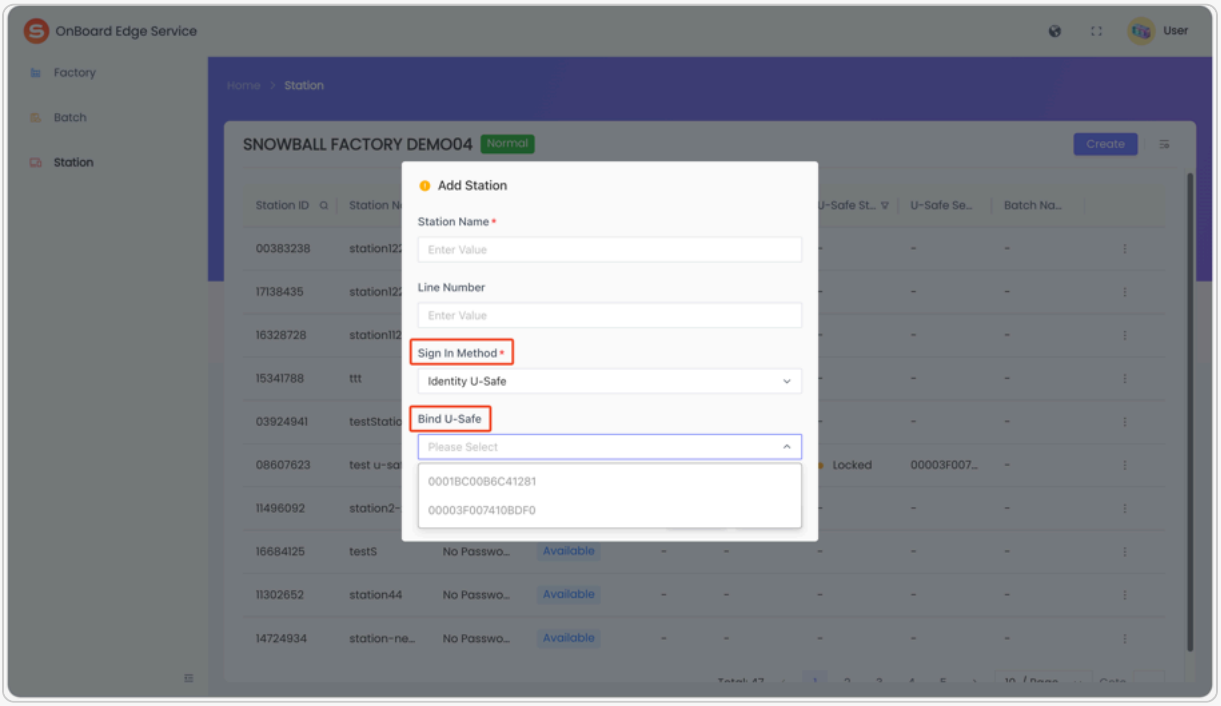
1. Sign in to OnBoard Edge Service.
2. In the left menu, click **Station**, and then click **Create** on the station page.



3. In the **Add** window, enter the **Station Name** and **Line Number**, and then select the **Sign In Method**. The methods include: **No Password**, **Password**, or **Identity U-Safe**.



> **TIP**
>
> If the sign-in method is **Identity U-Safe**, you can select an already added Identity U-Safe from the **Bind U-Safe** drop-down list to bind it, or you can bind it after adding the programming station.

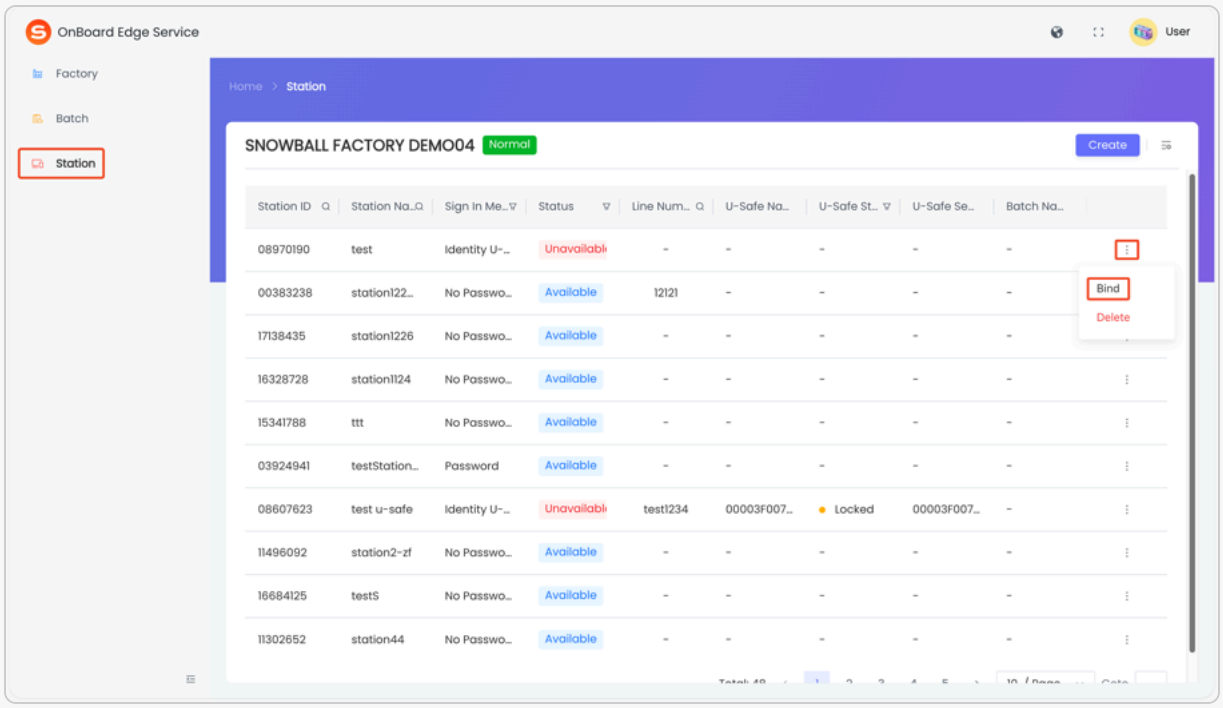


4. Click **Confirm** to create the programming station.

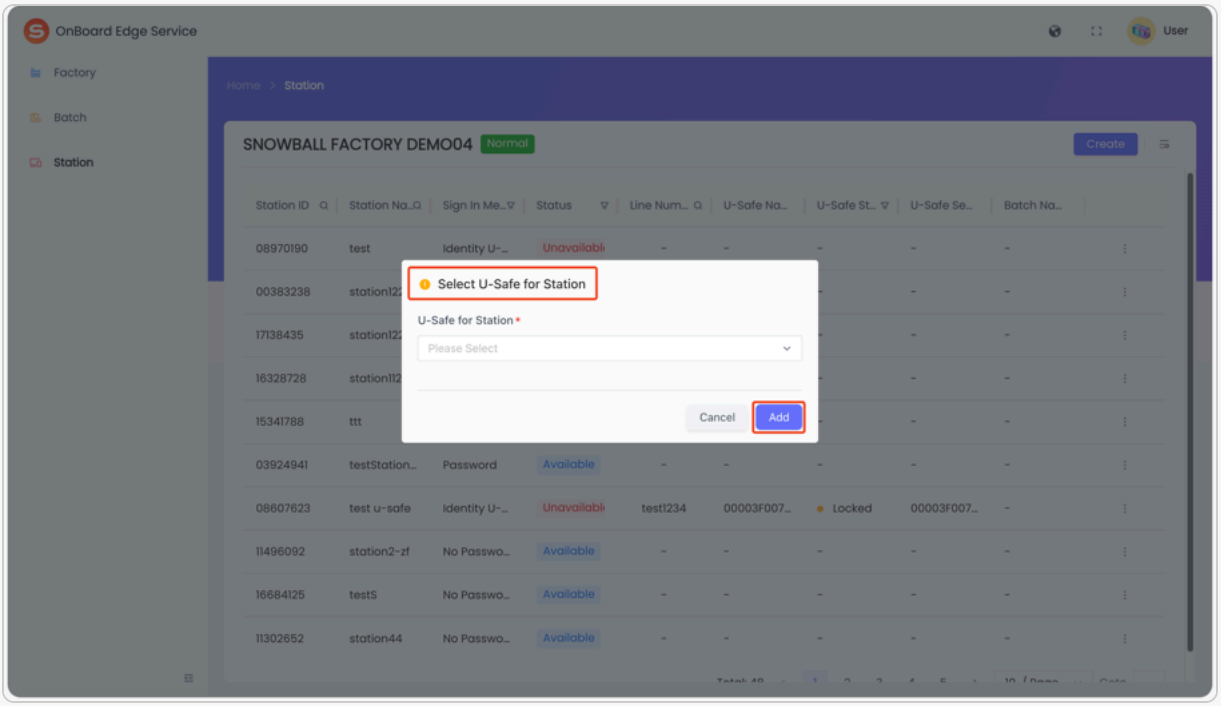
Bind an Identity U-Safe to a programming station

> **TIP**
>
> You can use the Identity U-Safe to sign in to the Programming Station client after binding.

1. Sign in to OnBoard Edge Service.
2. In the left menu, click **Station**. On the station page, click the Options ⋮ icon for the corresponding programming station and select **Bind**.



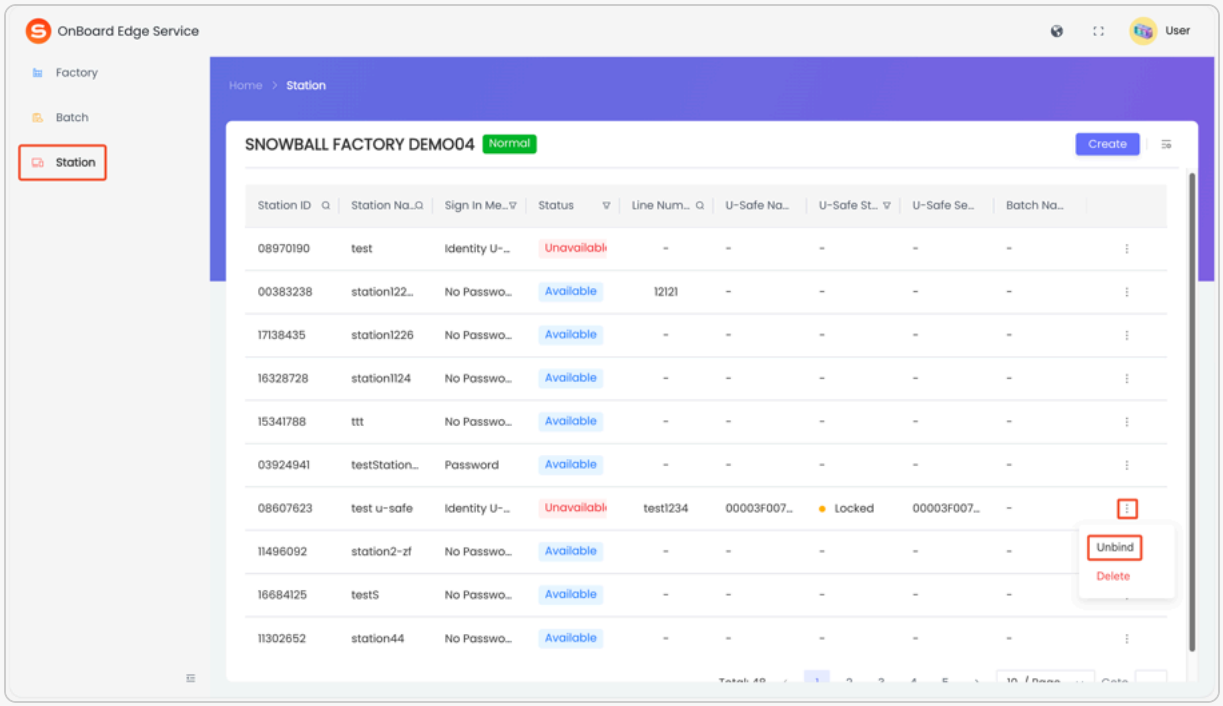
3. In the **Select U-Safe for Station** window, select the Identity U-Safe you want to bind and then click **Add** to bind it to the programming station.



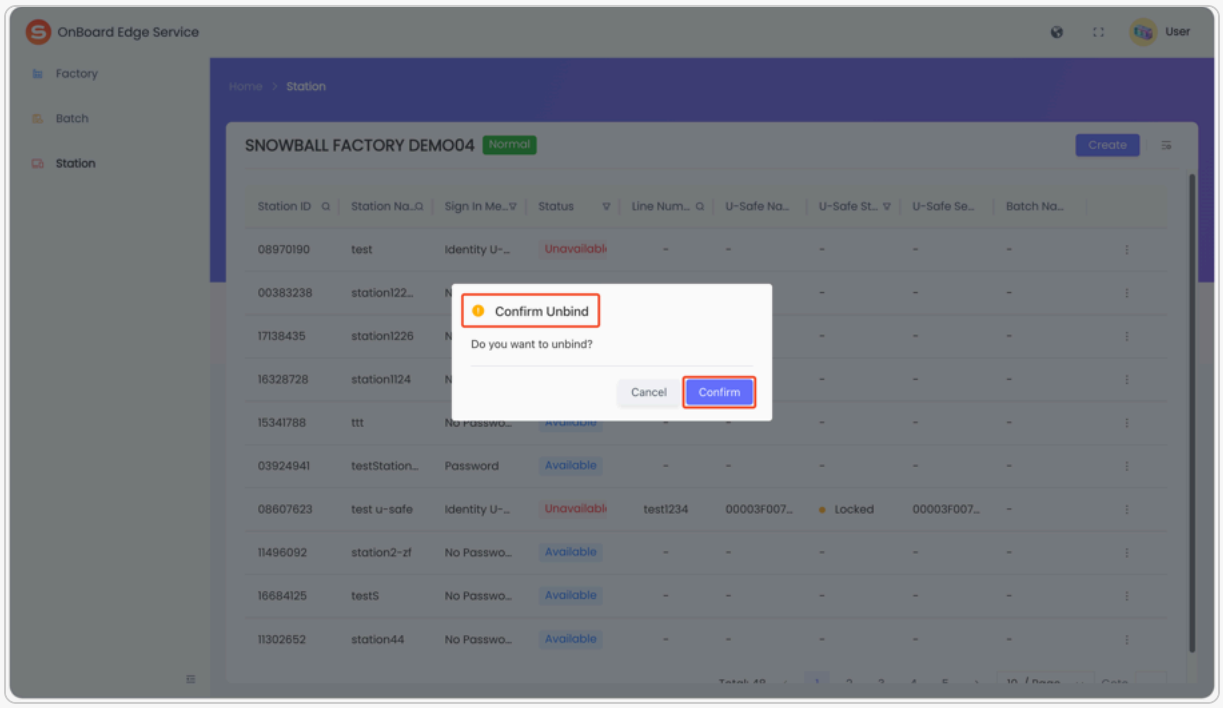
Unbind an Identity U-Safe from a programming station

> **WARNING**
>
> After unbinding, the Identity U-Safe can no longer be used to sign in to the Programming Station client.

1. Sign in to OnBoard Edge Service.
2. In the left menu, click **Station**. On the station page, click the Options ⋮ icon for the corresponding programming station and click **Unbind**.



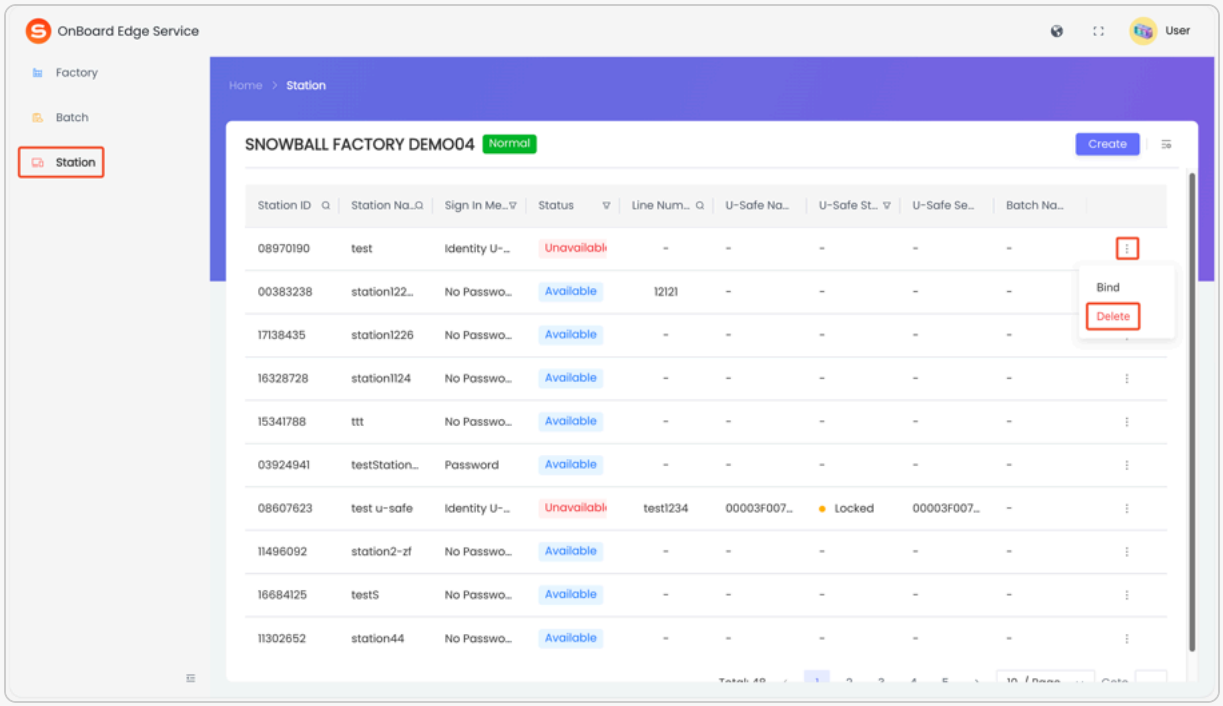
3. In the **Confirm Unbind** window, click **Confirm** to unbind the Identity U-Safe for the programming station.



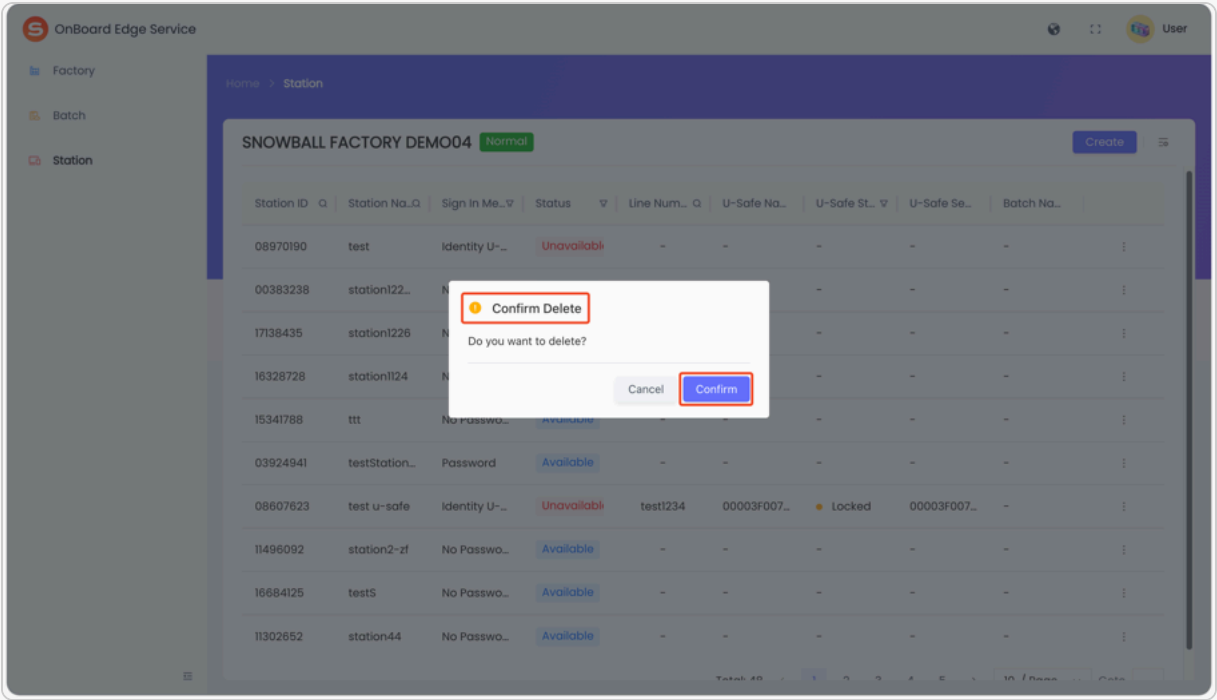
Delete a programming station

1. Sign in to OnBoard Edge Service.

2. In the left menu, click **Station**. On the station page, click the Options ⋮ icon for the corresponding programming station and select **Delete**.



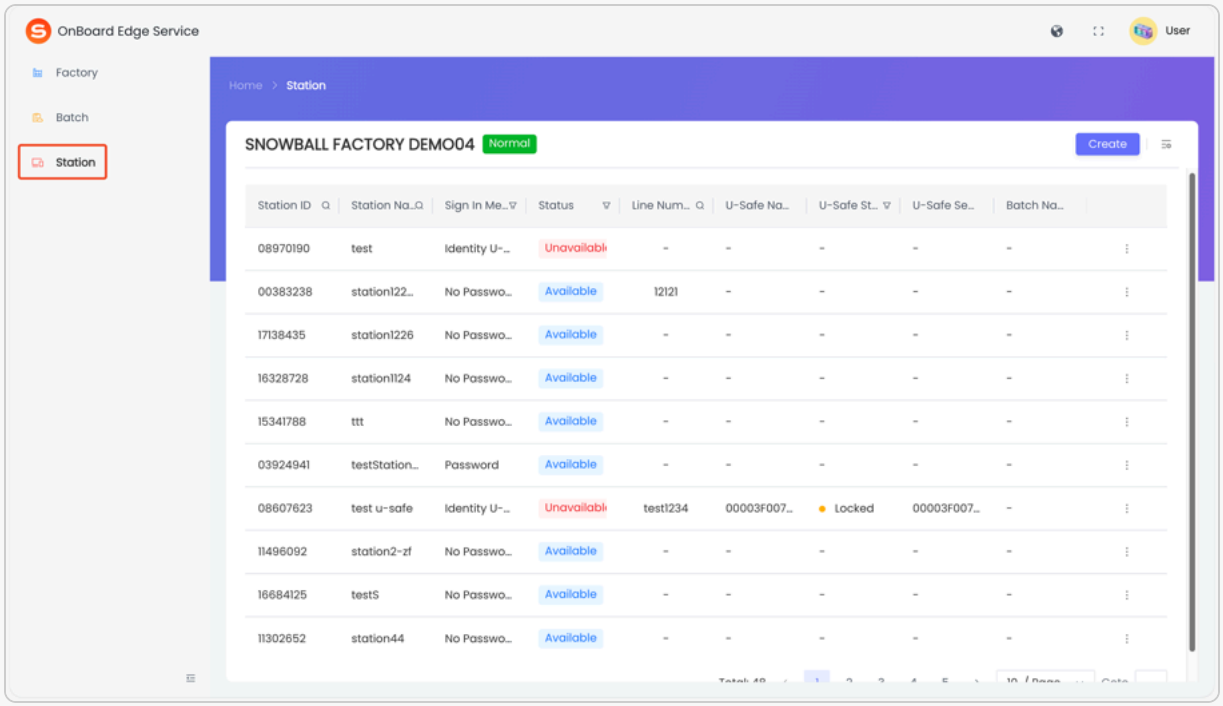
3. In the **Confirm Delete** window, click **Confirm** to delete the programming station.



Check the programming station information

1. Sign in to OnBoard Edge Service.

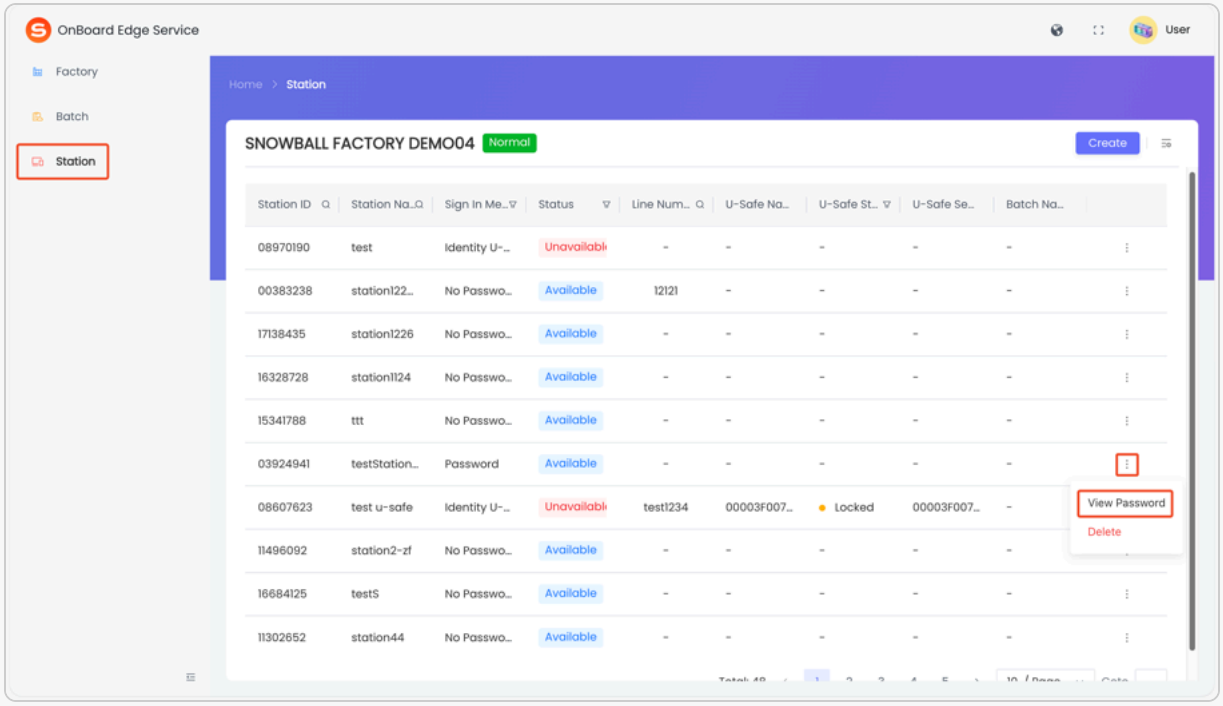
2. In the left menu, click **Station**. On the station page, you can check the programming station information (such as **Station ID**, **Station Name**, and **Sign In Method**).



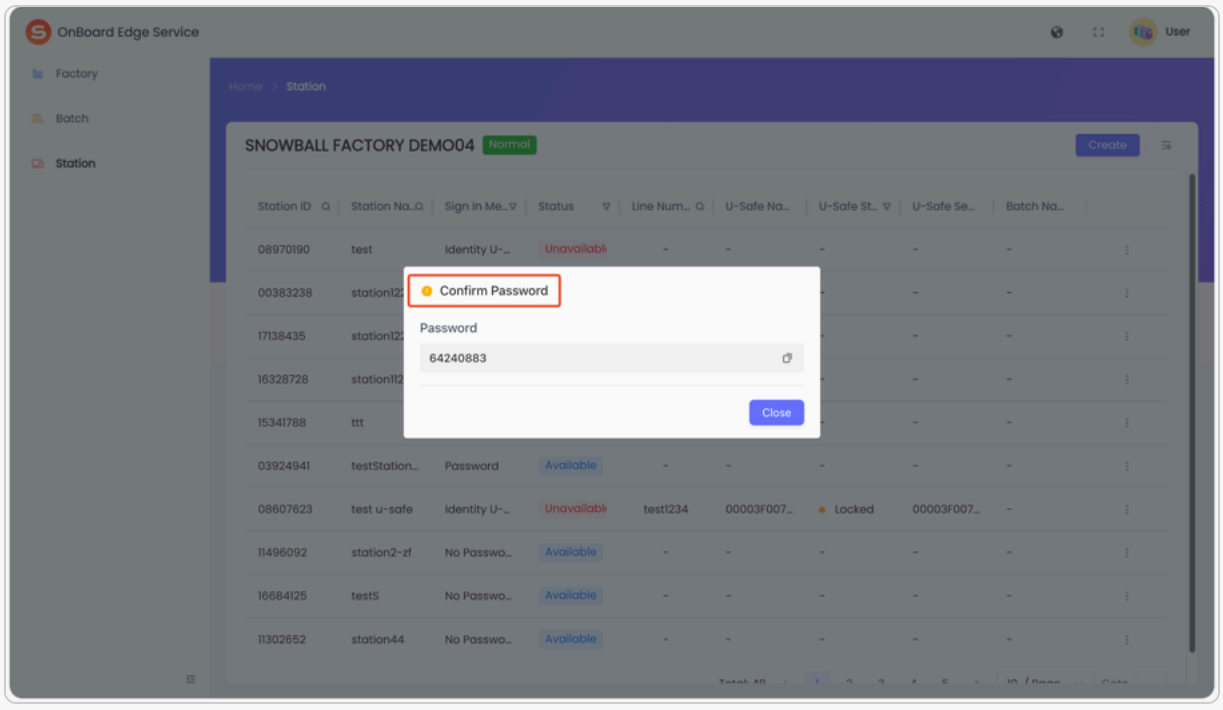
Check the programming station password

> **TIP**
>
> When the sign-in method for the programming station is **Password**, the system will generate a random password, which can be used to sign in to the Programming Station client. Currently, the randomly generated password can't be changed.

1. Sign in to OnBoard Edge Service.
2. In the left menu, click **Station**. On the station page, click the Options ⋮ icon for the corresponding programming station and select **View Password**.



3. In the **Confirm Password** window, you can check and copy the sign-in password for the programming station.



Version: v3.2.1

Overview of Programming Station

> **INFO**
>
> Currently, the Programming Station client is available only on Windows.

The Programming Station is the last-mile execution endpoint where devices are physically programmed and provisioned. Programming Station Software is the tooling software used to connect to the devices to be programmed and execute various chip programming tasks. As a key part of the chip programming process that directly interfaces with the operator, it offers a clean and user-friendly operating environment, allowing operators to complete all chip programming tasks easily without in-depth knowledge of complex underlying technologies.

Programming Station Software wraps and governs existing chip vendor and programmer vendor tooling through a layered component architecture, enabling flexibility across different chips and product requirements, as summarized in the following table:

Component	Provider	Role
Programming Station Application	OBIS	User interface for programming engineers • Execute an assigned Production Batch, or manually enter either a Production Batch identifier or a Production Order identifier to download the associated Programming Station Software package, which combines the Execution Process Control Scripts CLI and Chip Adaptation CLI tools • Configure channel settings and USB port-to-channel mapping
Execution Process Control Scripts CLI	OEM	OEM-developed scripts, orchestrate programming and provisioning flows
Chip Adaptation CLI	Chip vendor	Chip vendor provided tooling software, supports chip-specific programming and provisioning interfaces
Programmer CLI	Programmer vendor	(If applicable) Programmer vendor provided drivers and tools (e.g., J-Link), interfaces directly with physical programming hardware

This layered architecture allows OBIS to support a wide range of chip platforms and programmer hardware while maintaining a consistent, centralized governance model for all provisioning operations.

Version: v3.2.1

How to install Programming Station

> **INFO**
>
> Currently, the Programming Station client is available only on Windows.

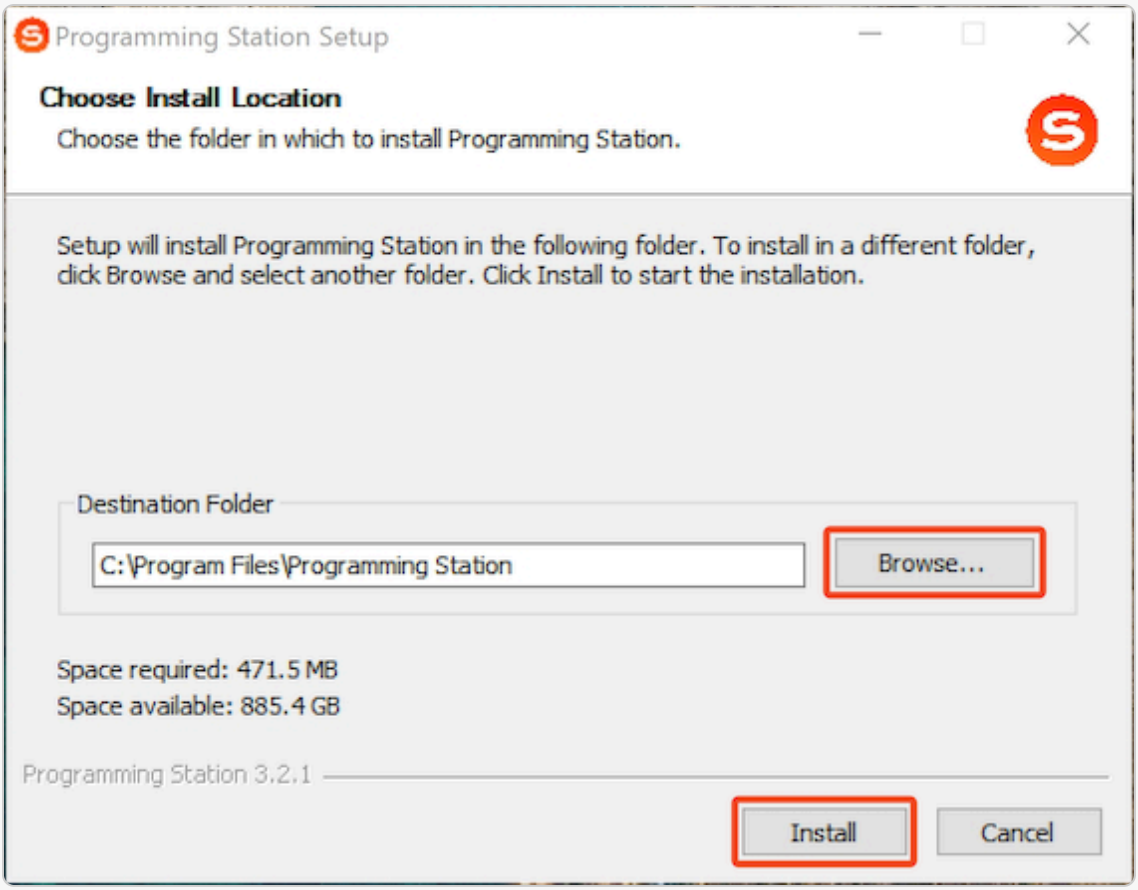
This article will guide you on how to install the Programming Station client.

Prerequisites

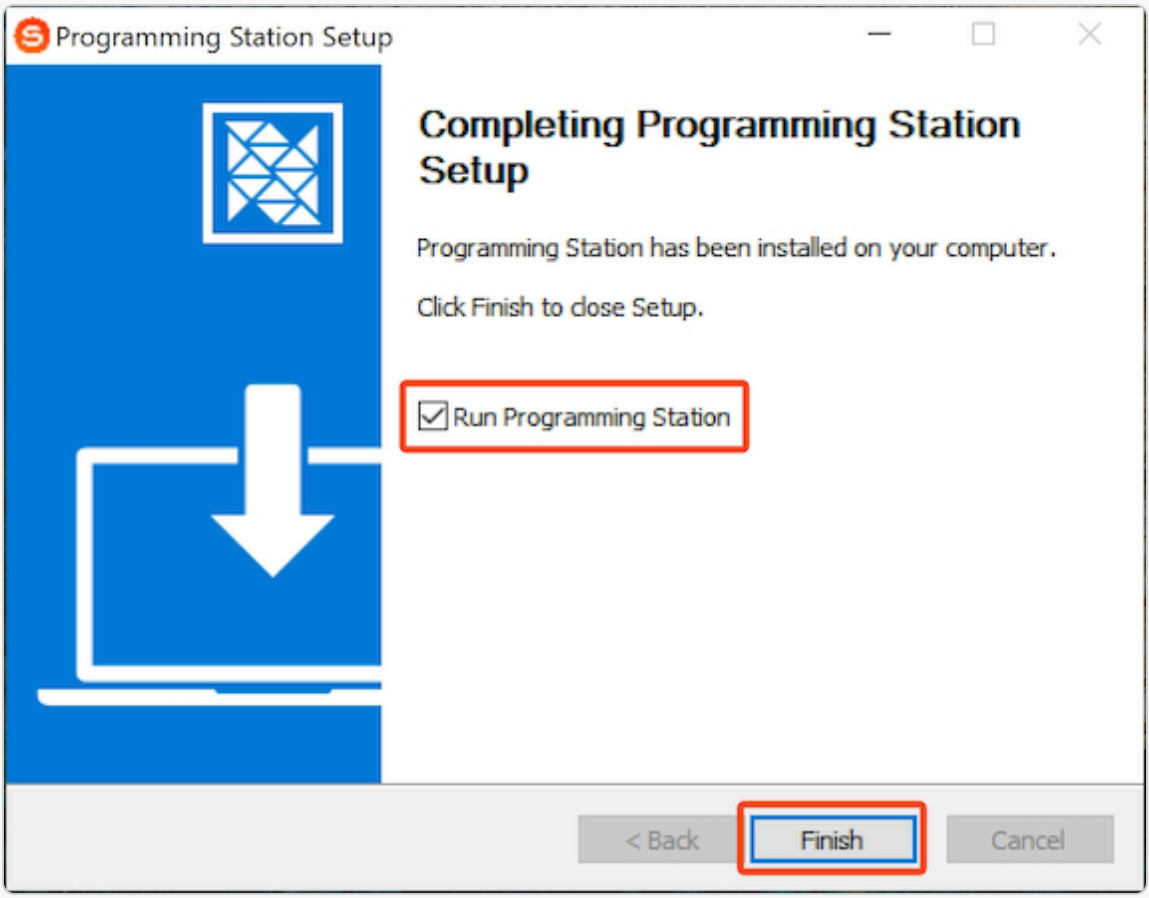
Before getting started, please ensure that you have obtained the installation package for the Programming Station client from Snowball.

Instructions

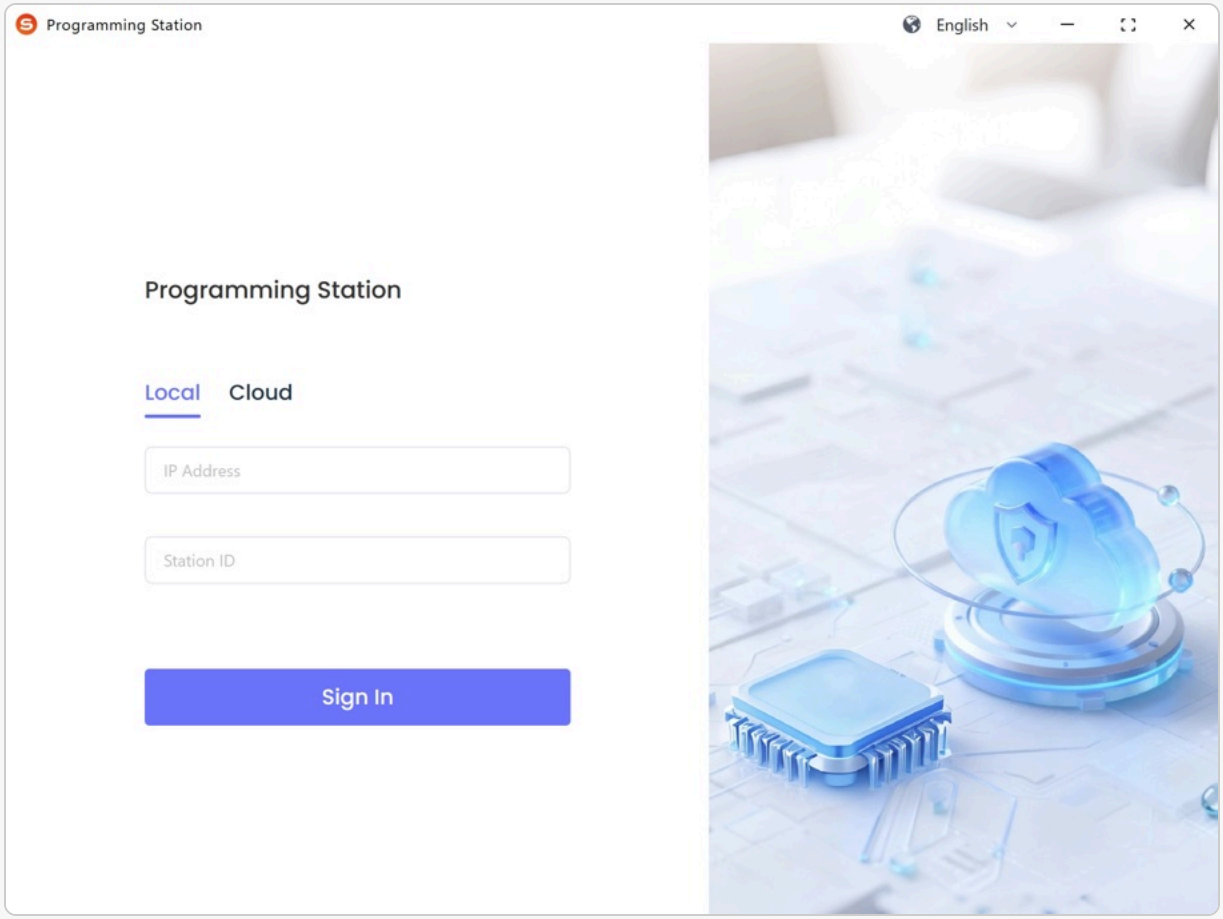
1. Run the installation package of the Programming Station client to start the installation.
2. In the **Choose Install Location** window, click **Browse** to specify an installation location (or use the default path) for the client, and then click **Install**.



3. In the pop-up window, select the **Run Programming Station** check box as needed, and then click **Finish**.



4. If the **Run Programming Station** check box is selected, the Programming Station client will start automatically.


Programming Station
English
Programming Station
Local
Cloud
IP Address
Station ID
Sign In

Version: v3.2.1

How to sign in to Programming Station

> **INFO**
>
> Currently, the Programming Station client is available only on Windows.

This article will guide you on how to sign in to the Programming Station client using either the "No Password", "Password", or "Identity U-Safe" method.

Prerequisites

Before getting started, please ensure that:

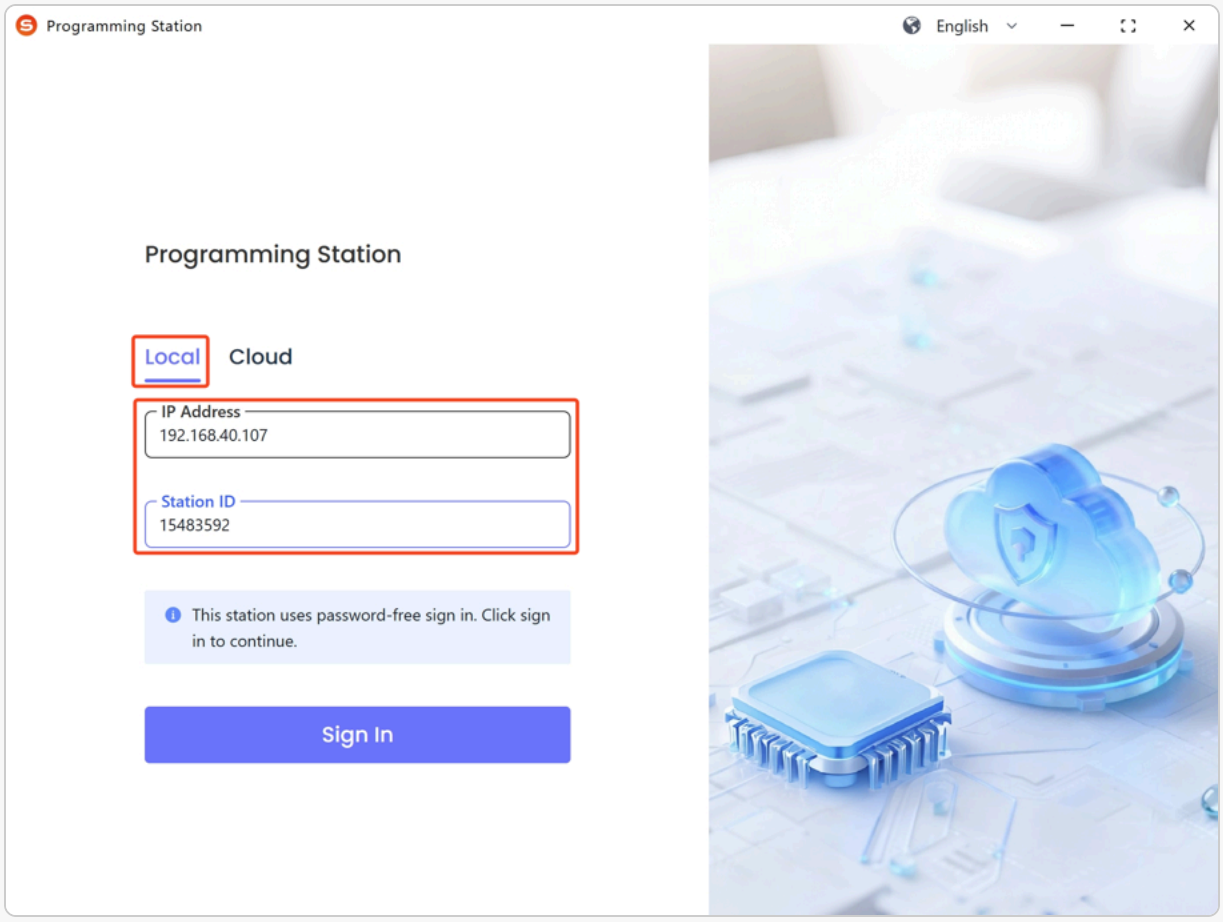
- A programming station has been added in OnBoard™ IoT Security (OBIS) (local factory & cloud factory) or OnBoard Edge Service.
- The Programming Station client has been installed.

Local factory

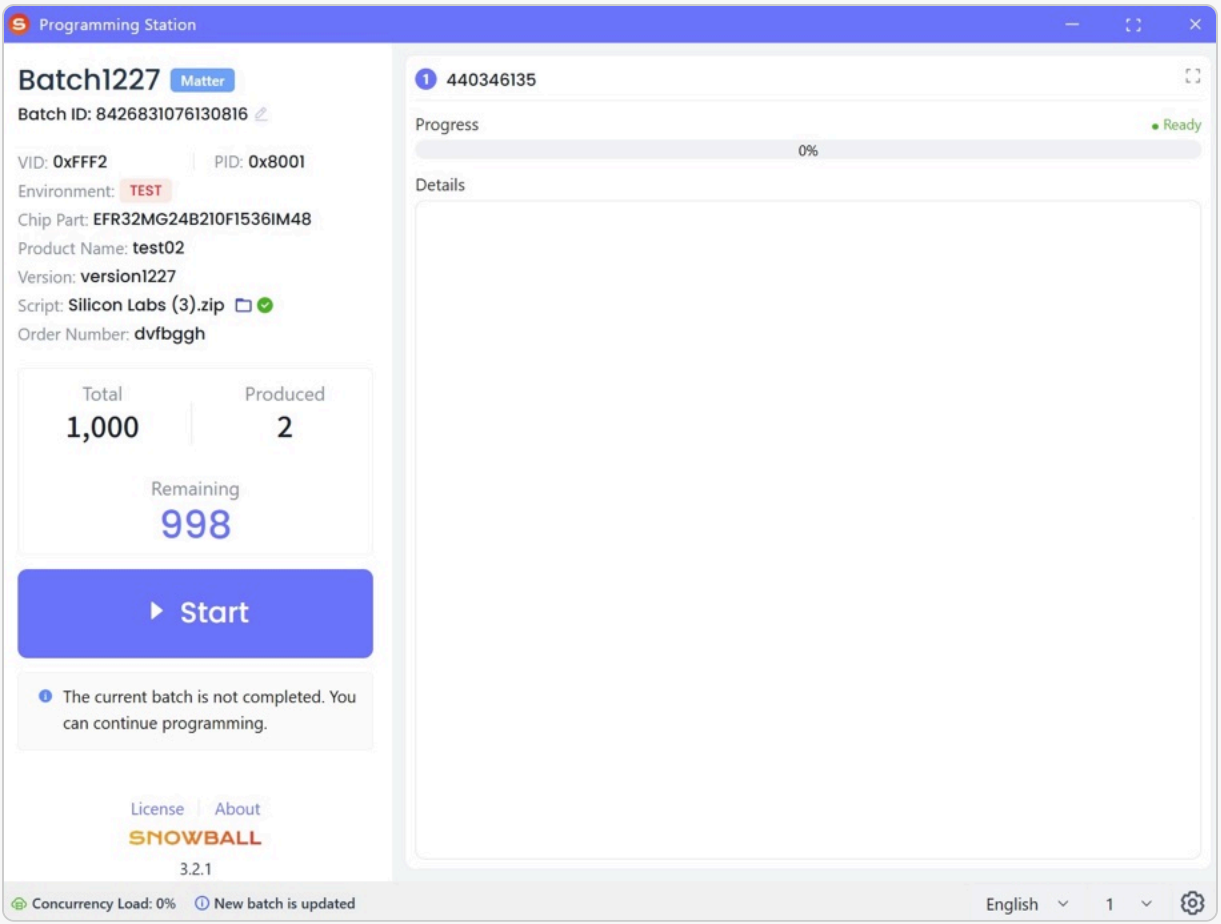
> **TIP**
>
> Please ensure that you have contacted Snowball for assistance with the deployment of OnBoard Edge Service, and that your sign-in device is within the factory service intranet environment.

Method one: No Password

1. Start the Programming Station client, and then select **Local**.
2. Enter the IP address (for example, 192.168.0.1) in the **IP Address** field. For more information, refer to how to check service address.



3. Enter the corresponding programming station ID and then click **Sign In**. Once signed in, the page will navigate to the main window of the Programming Station client.

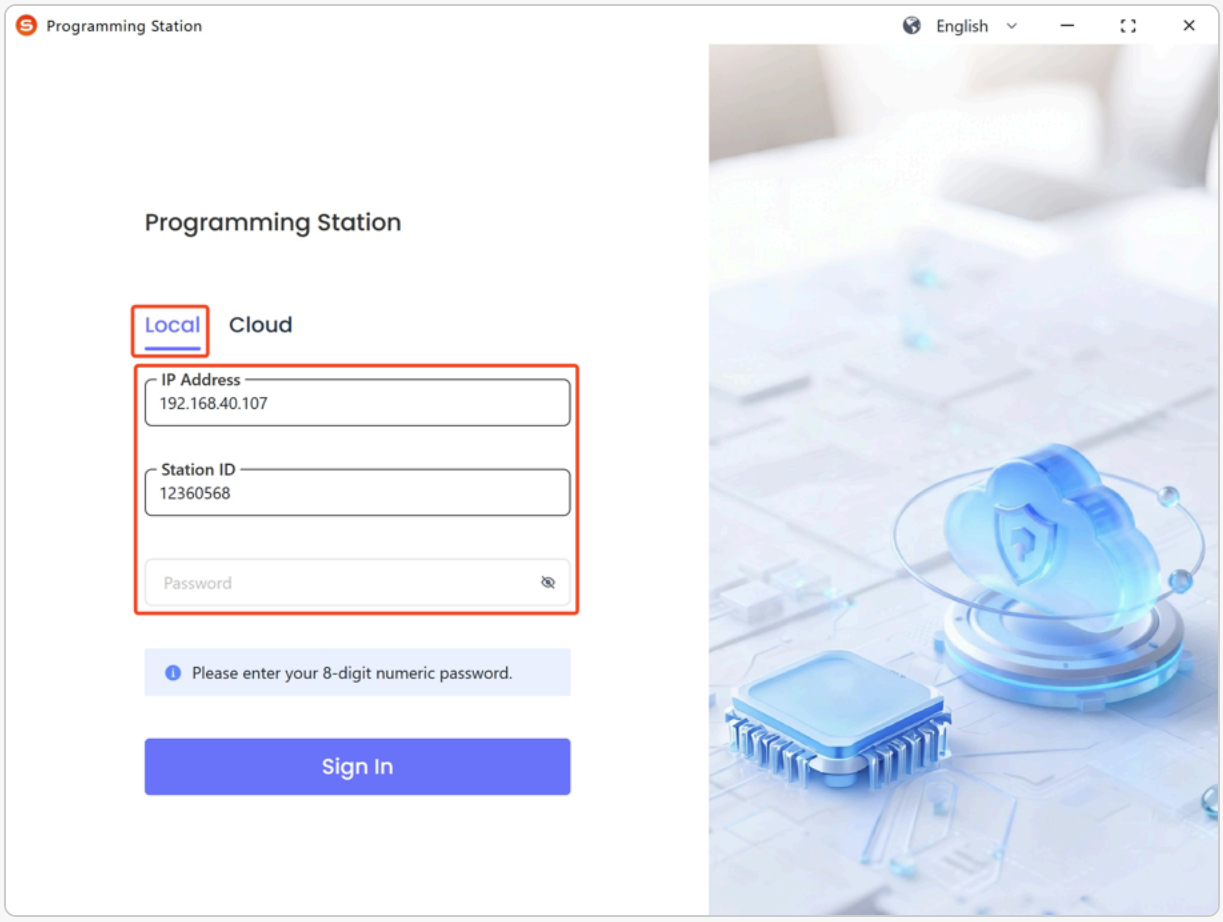


Method two: Password

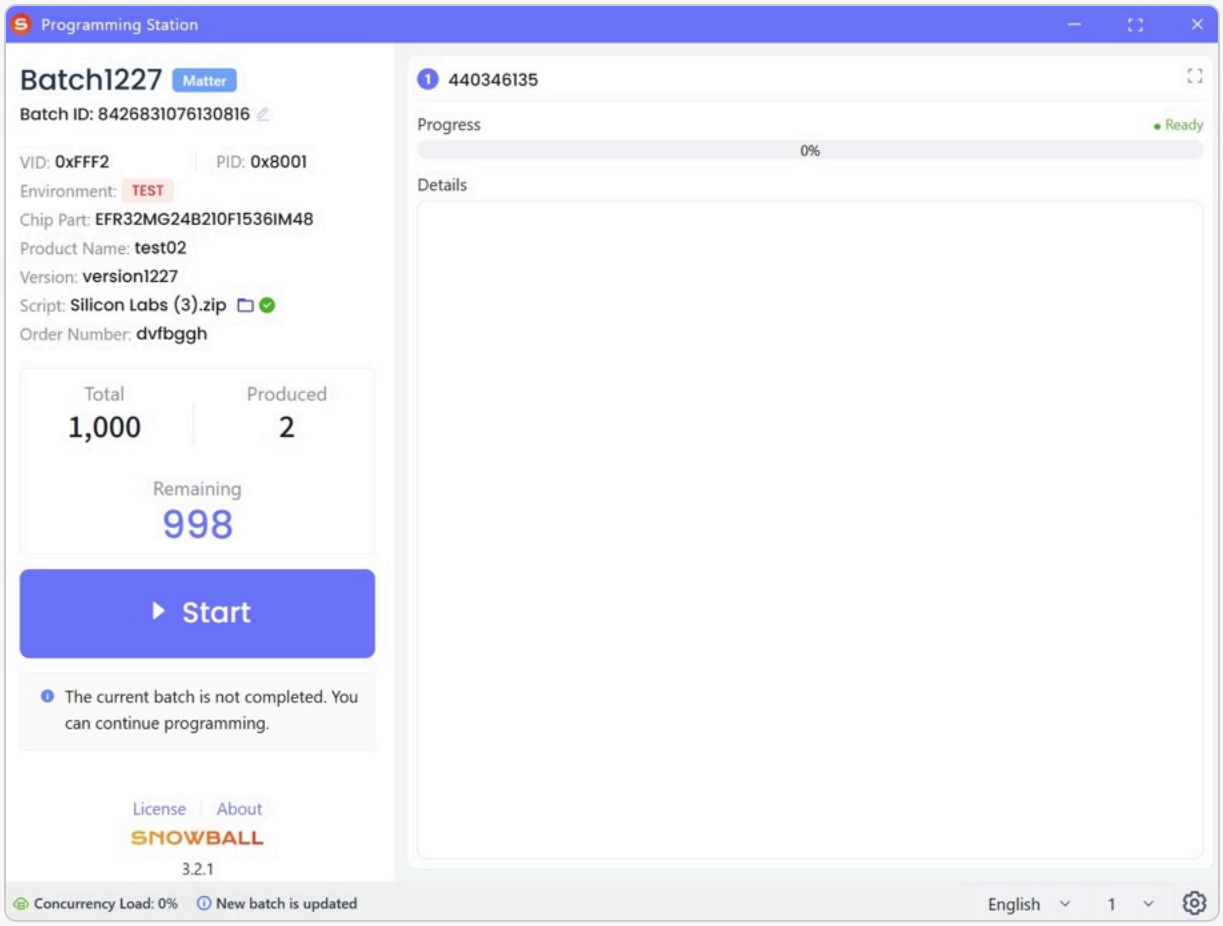
> **TIP**
>
> The initial sign-in password is randomly generated by the system after a programming station is added to the factory. You can check the sign-in password in OBIS or OnBoard Edge Service. Currently, the randomly generated password can't be changed.

1. Start the Programming Station client, and then select **Local**.

2. Enter the IP address (for example, 192.168.0.1) in the **IP Address** field. For more information, refer to how to check service address.



3. Enter the corresponding programming station ID and password, and then click **Sign In**. Once signed in, the page will navigate to the main window of the Programming Station client.

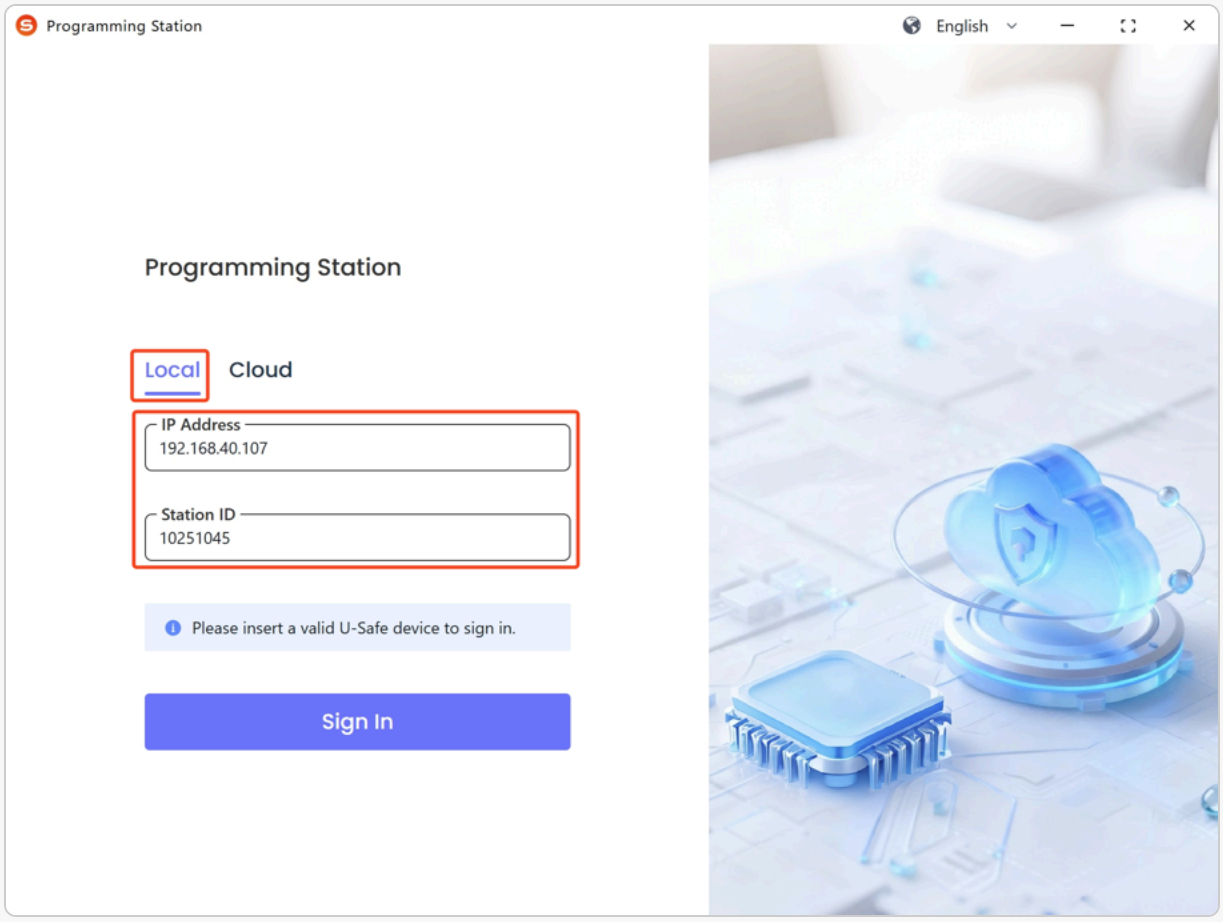


Method three: Identity U-Safe

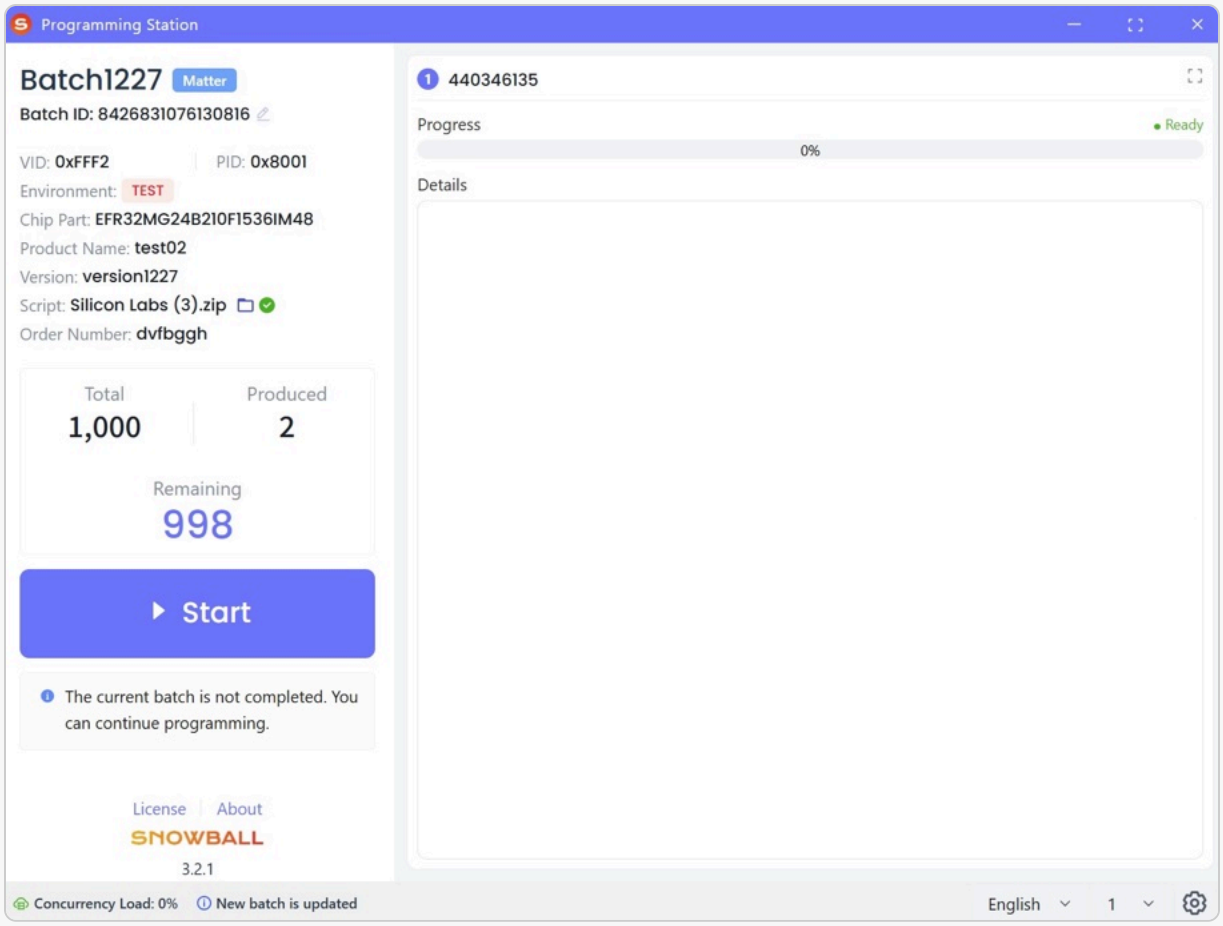
> **INFO**
>
> When signing in using the Identity U-Safe method, you need to obtain the Identity U-Safe used for identity authentication from Snowball, and insert the corresponding Identity U-Safe (added during factory configuration) into the device that has installed the Programming Station client.

1. Start the Programming Station client, and then select **Local**.

2. Enter the IP address (for example, 192.168.0.1) in the **IP Address** field. For more information, refer to how to check service address.



3. Enter the corresponding programming station ID and click **Sign In**. Once signed in, the page will navigate to the main window of the Programming Station client.

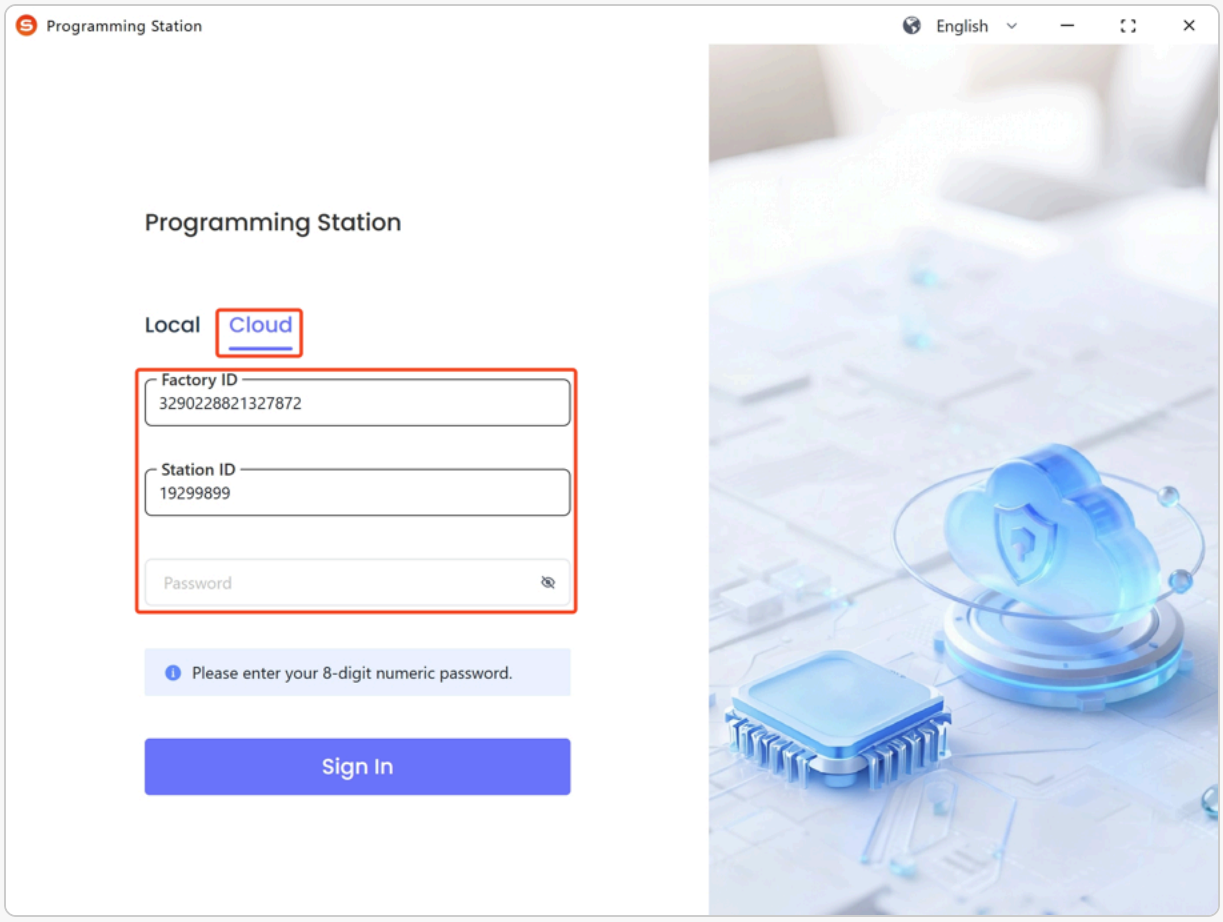


Cloud factory

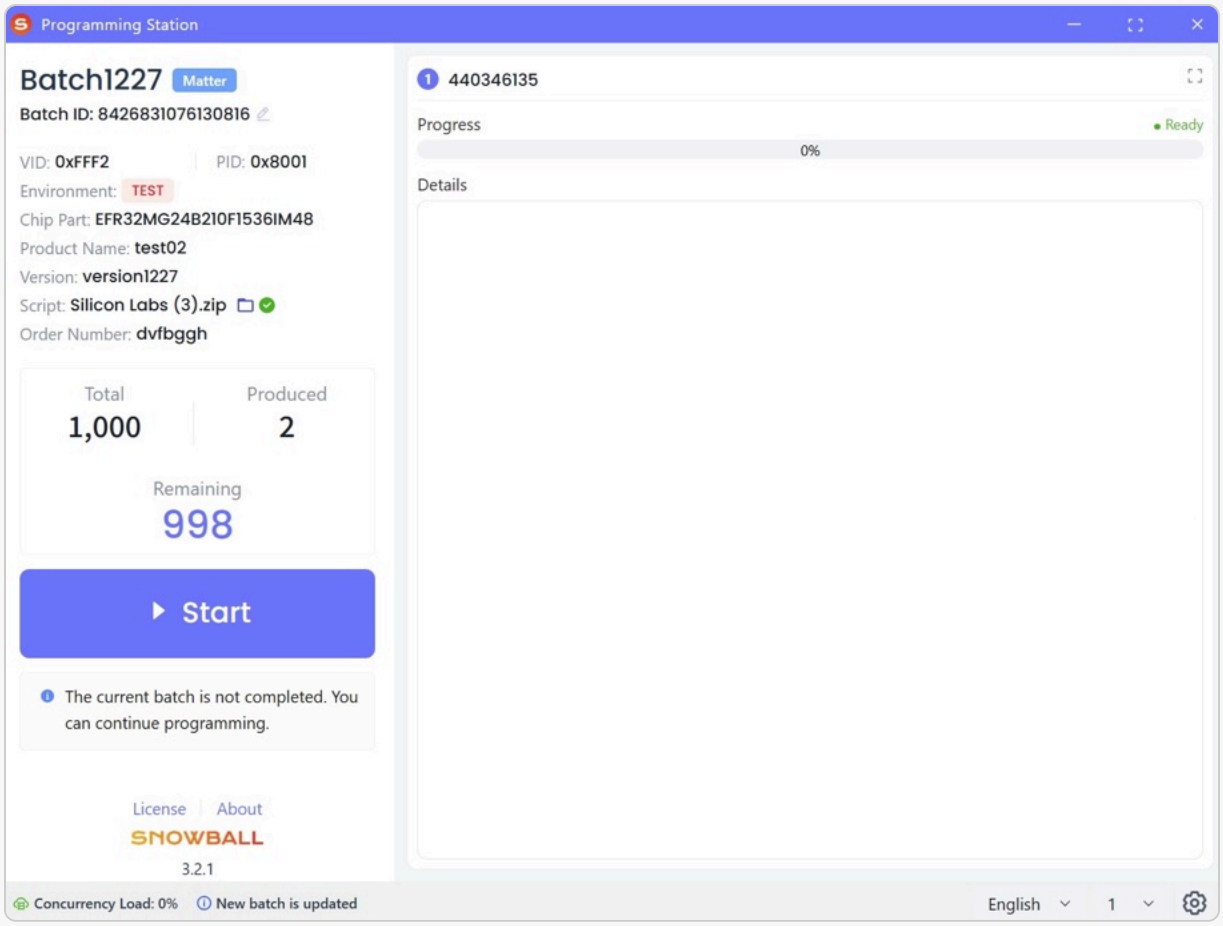
Method one: Password

> **TIP**
>
> The initial sign-in password is randomly generated by the system after a programming station is added to the factory. You can check the sign-in password in OBIS. Currently, the randomly generated password can't be changed.

1. Start the Programming Station client, select **Cloud**, and then enter the corresponding factory ID.



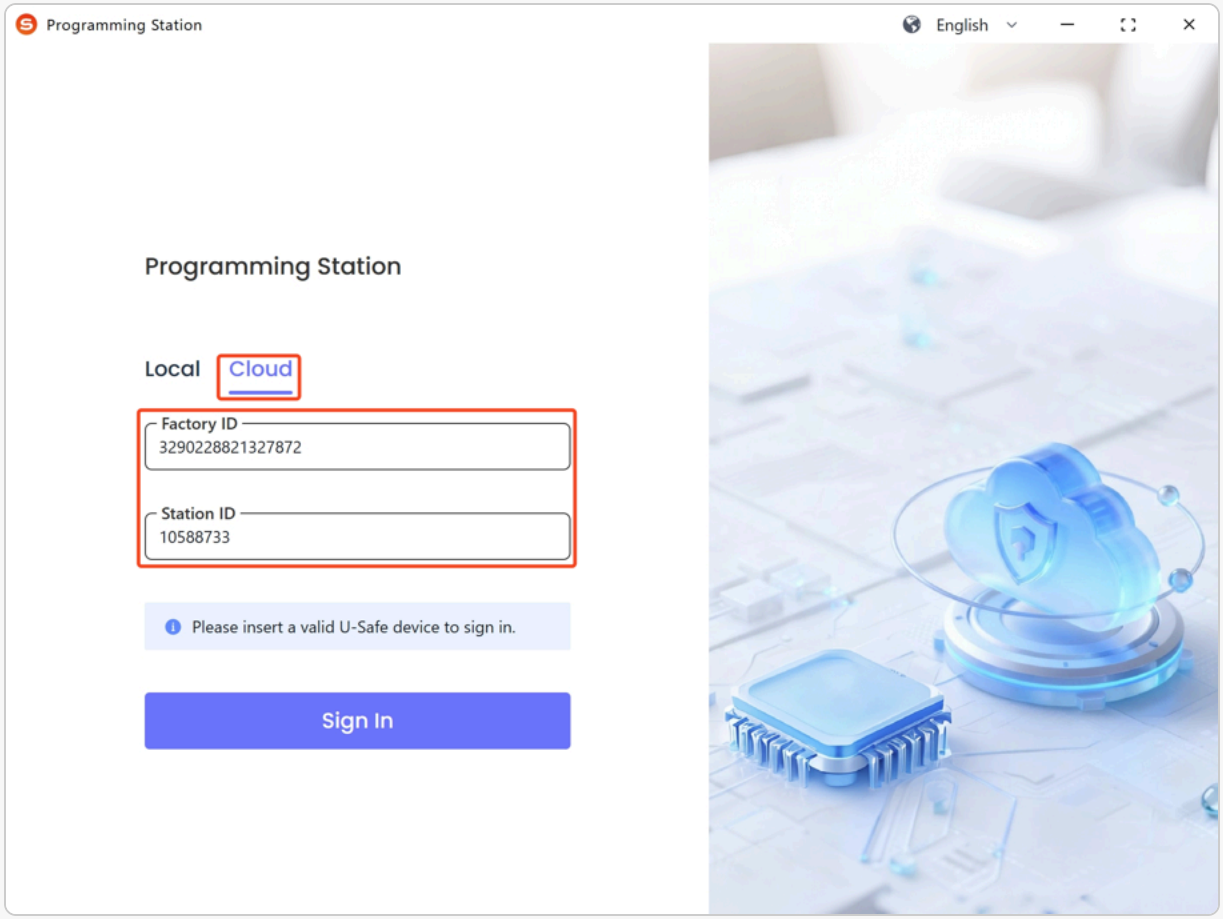
2. Enter the corresponding programming station ID and password, and then click **Sign In**. Once signed in, the page will navigate to the main window of the Programming Station client.



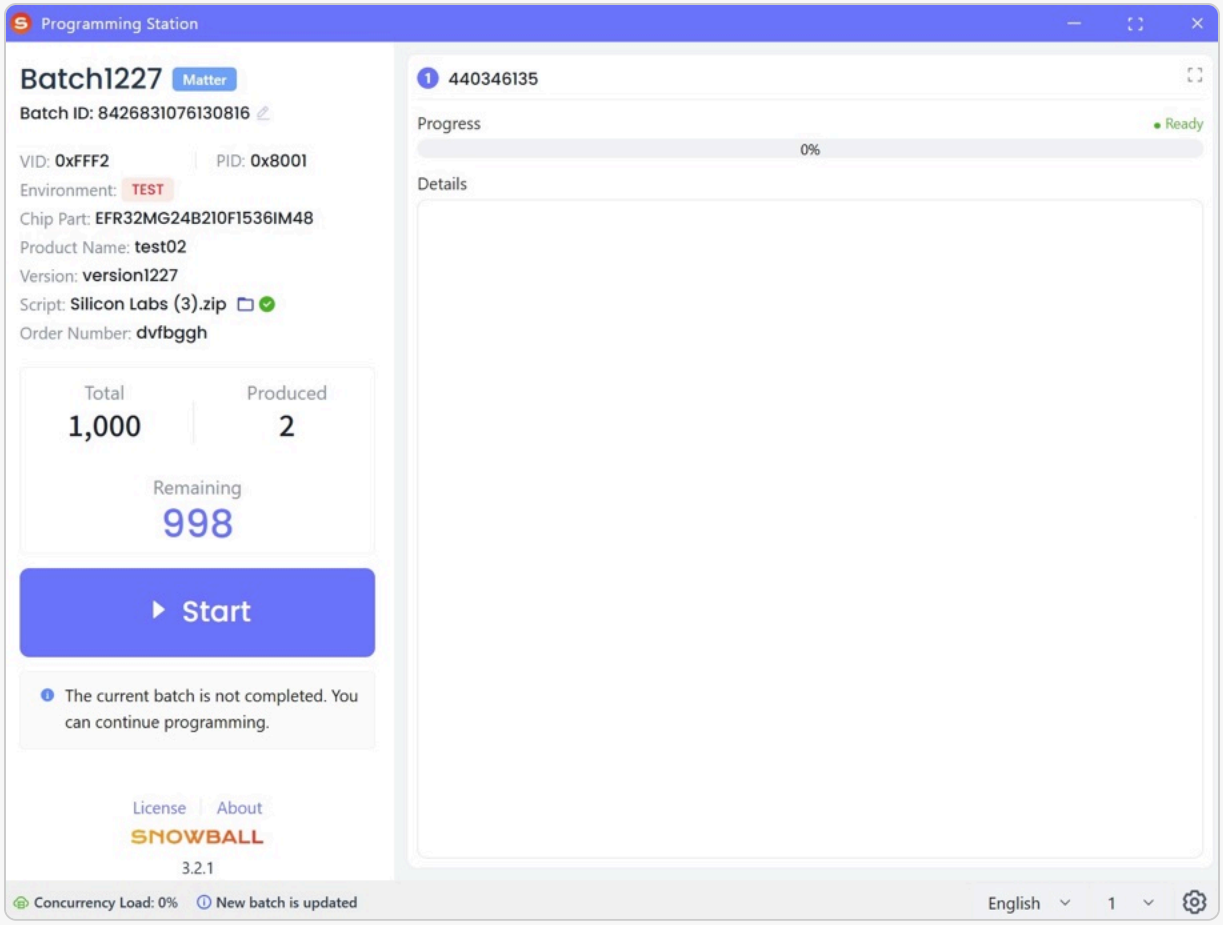
Method two: Identity U-Safe

> **INFO**
>
> When signing in using the Identity U-Safe method, you need to obtain the Identity U-Safe used for identity authentication from Snowball, and insert the corresponding Identity U-Safe (added during factory configuration) into the device that has installed the Programming Station client.

1. Start the Programming Station client, select **Cloud**, and then enter the corresponding factory ID.



2. Enter the corresponding programming station ID and click **Sign In**. Once signed in, the page will navigate to the main window of the Programming Station client.


Programming Station
Batch1227
Matter
Batch ID: 8426831076130816
VID: 0xFFF2
PID: 0x8001
Environment: TEST
Chip Part: EFR32MG24B210F1536IM48
Product Name: test02
Version: version1227
Script: Silicon Labs (3).zip
Order Number: dvfbggh
Total
1,000
Produced
2
Remaining
998
Start
The current batch is not completed. You can continue programming.
License
About
SNOWBALL
3.2.1
1
440346135
Progress
Ready
0%
Details
Concurrency Load: 0%
New batch is updated
English
1

Version: v3.2.1

How to execute device programming in Programming Station

> **INFO**
>
> Currently, the Programming Station client is available only on Windows.

This article will guide you on how to use the Programming Station client to execute device programming.

Prerequisites

Before getting started, please ensure that you can sign in to the Programming Station client.

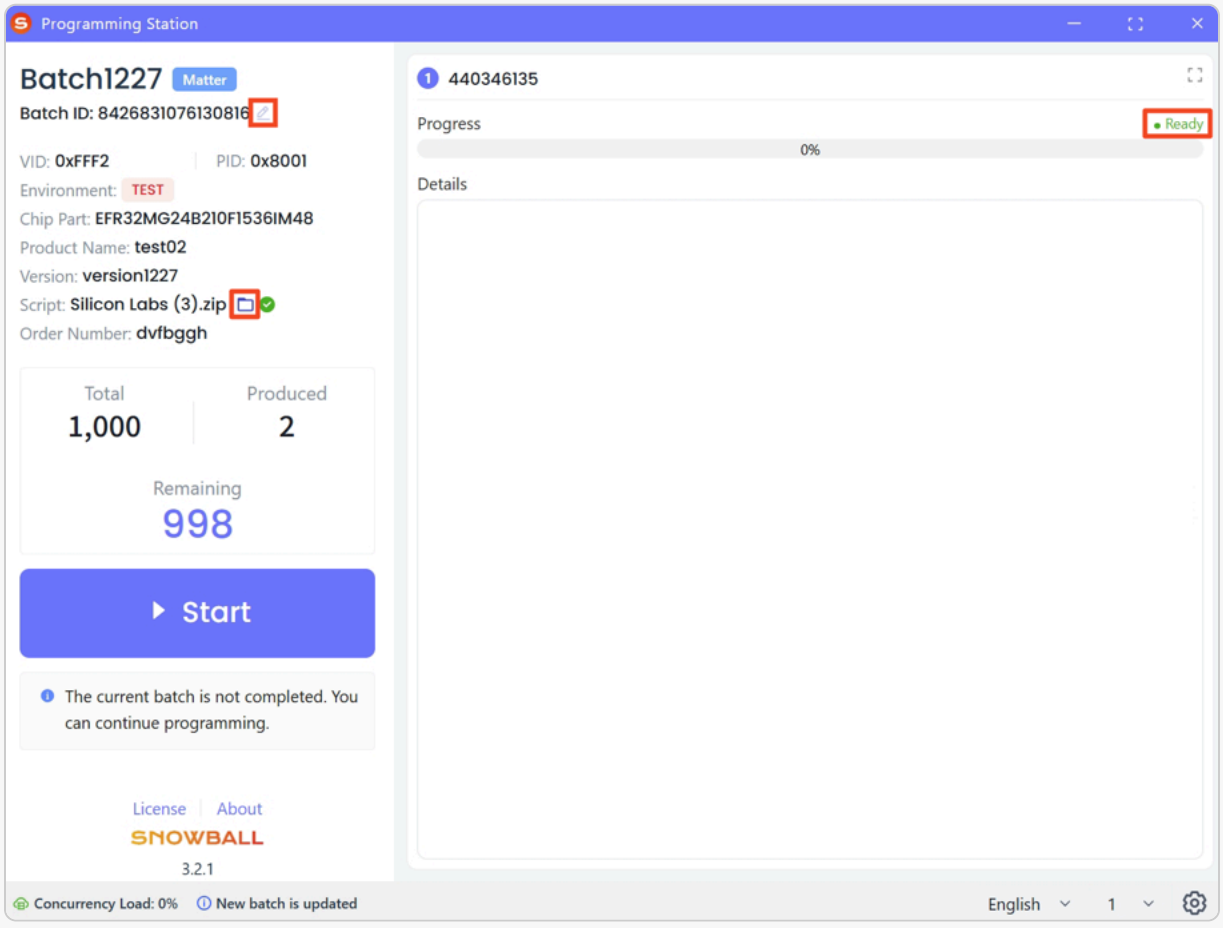
Instructions

1. Sign in to the Programming Station client, and then verify that the device status is **Ready**.

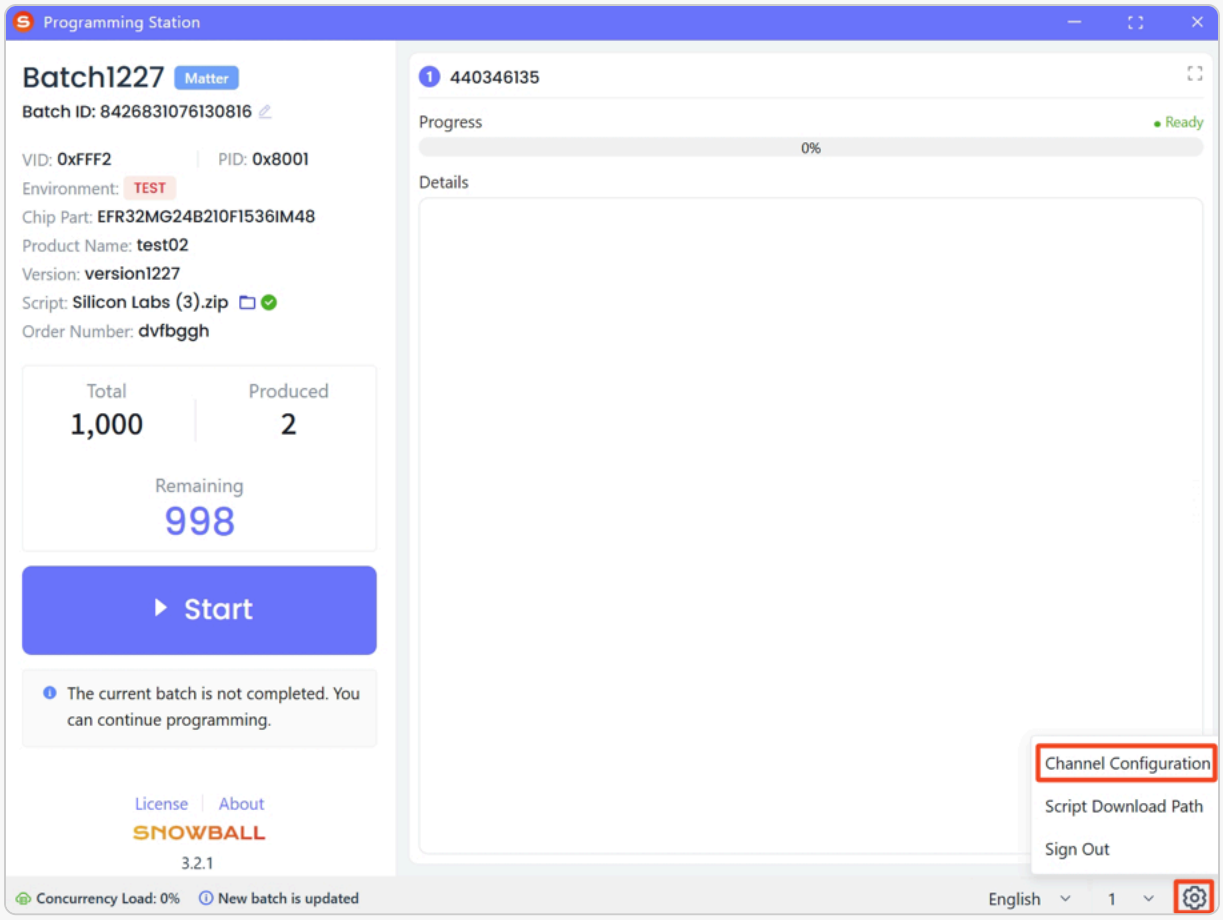
TIP

If the programming script is not selected when creating a batch, or if you want to select a different programming script, you can click the Folder icon to select the required programming script (any script format is supported). Additionally, you can click the Settings icon in the bottom-right corner and select **Script Download Path** to customize the default script download path.

If the programming station is not yet assigned to a batch, you can click the Edit icon, and then enter the batch ID to query and retrieve the corresponding batch information.

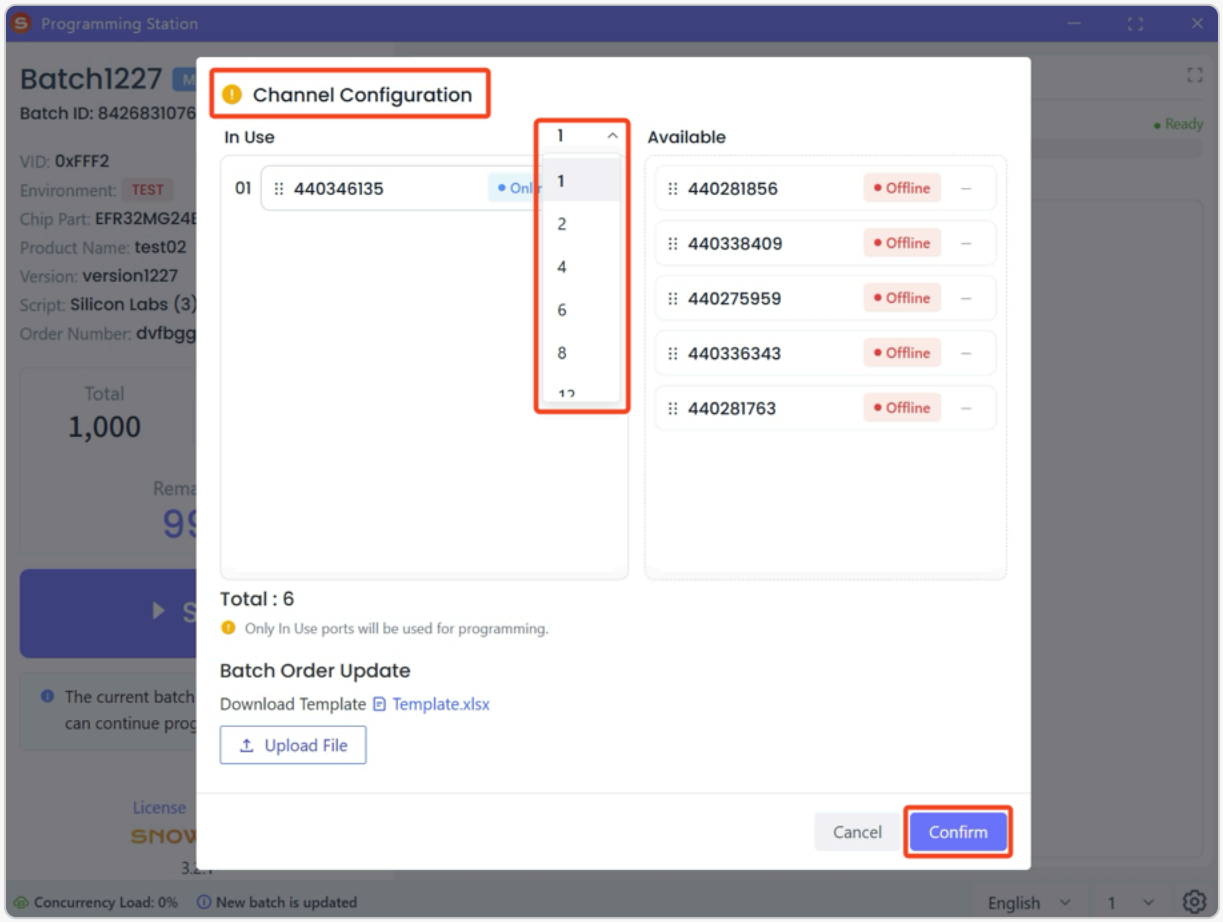


2. In the main window, click the Settings icon in the bottom-right corner and then select **Channel Configuration**.



3. In the **Channel Configuration** window, set the number of connected programmers.

> **TIP**
>
> Set the number of connected programmers to adjust the number of devices programmed simultaneously. Currently, the system supports setting the number of connected programmers as **1**, **2**, **4**, **6**, **8**, **12**, and **16**, which you can select as needed.
>
> Meanwhile, the **In Use** and **Available** ports can be dragged and reordered.

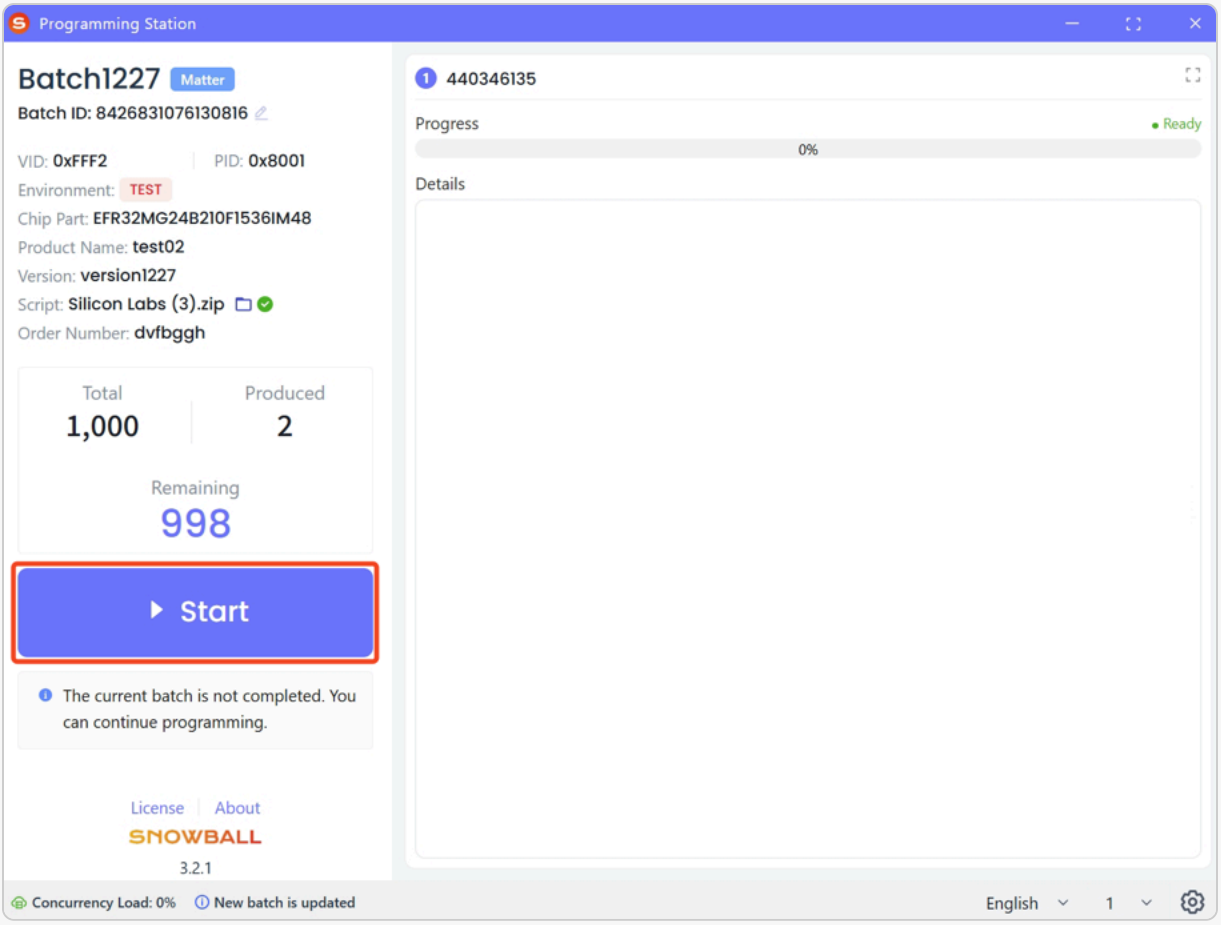


4. After verifying that the information is correct, click **Start** in the main window to execute device programming.

TIP

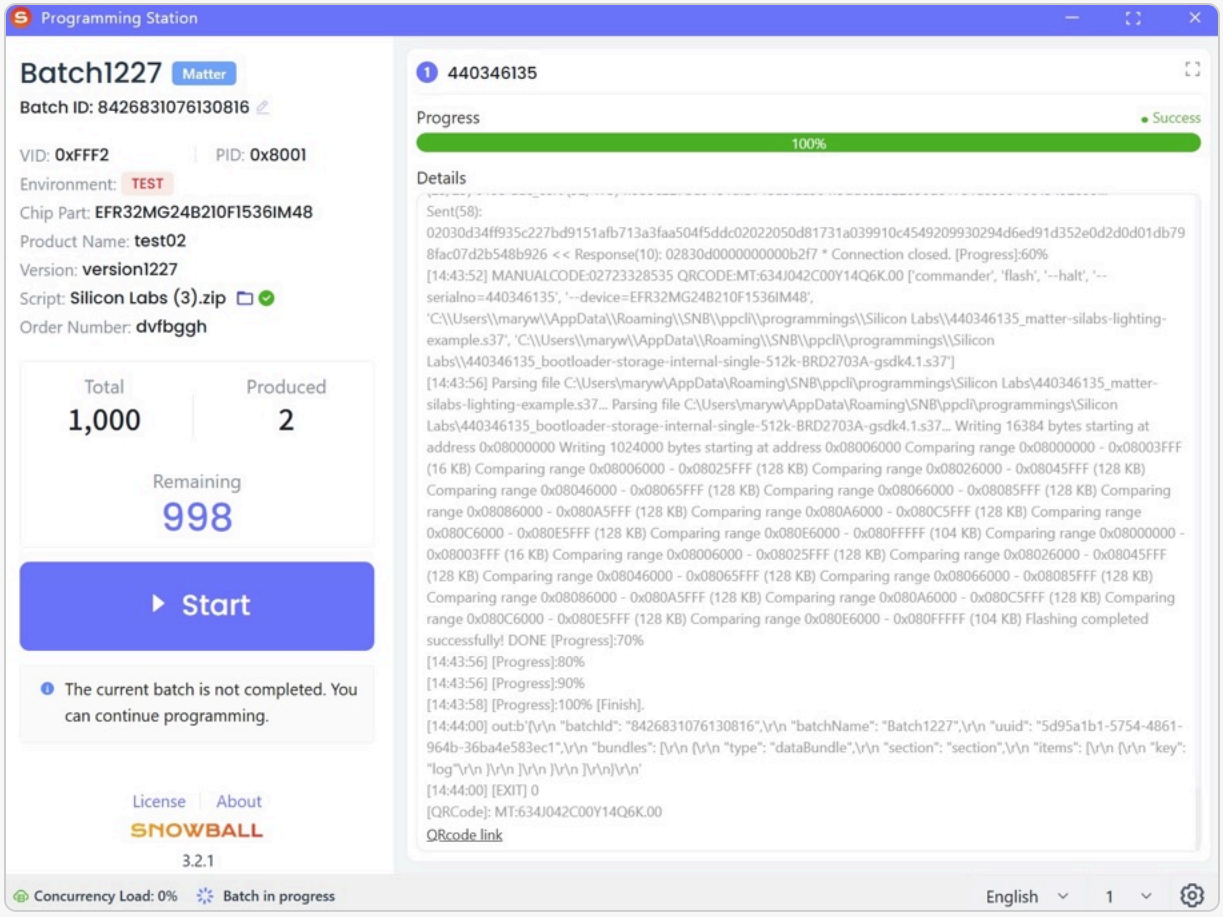
The **Concurrency Load** in the bottom-left corner of the client indicates the server load when multiple programming stations are simultaneously programming:

- If the load ≥ 90%, you will see the message "Concurrency limit nearly reached. Please contact the factory administrator to check the server", and you can continue programming at this point.
- If the load = 100%, you will see the message "Concurrency limit reached. Please contact the factory administrator to check the server", and you can still continue programming.



5. After programming is completed, the programming details will be displayed.

> **TIP**
>
> You can also check the programming details in OnBoard™ IoT Security (OBIS).


Programming Station
Batch1227
Matter
Batch ID: 8426831076130816
VID: 0xFFF2
PID: 0x8001
Environment: TEST
Chip Part: EFR32MG24B210F1536IM48
Product Name: test02
Version: version1227
Script: Silicon Labs (3).zip
Order Number: dvfbggh
Total
1,000
Produced
2
Remaining
998
Start
The current batch is not completed. You can continue programming.
License
About
SNOWBALL
3.2.1
440346135
Progress
Success
100%
Details
Sent(58):
02030d34ff935c227bd9151afb713a3faa504f5ddc02022050d81731a039910c4549209930294d6ed91d352e0d2d0d01db798fac07d2b548b926 << Response(10): 02830d0000000000b2f7 * Connection closed. [Progress]:60%
[14:43:52] MANUALCODE:02723328535 QRCODE:MT:634J042C00Y14Q6K.00 ['commander', 'flash', '--halt', '--serialno=440346135', '--device=EFR32MG24B210F1536IM48', 'C:\\Users\\maryw\\AppData\\Roaming\\SNB\\ppcli\\programmings\\Silicon Labs\\440346135_matter-silabs-lighting-example.s37', 'C:\\Users\\maryw\\AppData\\Roaming\\SNB\\ppcli\\programmings\\Silicon Labs\\440346135_bootloader-storage-internal-single-512k-BRD2703A-gsdk4.1.s37']
[14:43:56] Parsing file C:\Users\maryw\AppData\Roaming\SNB\ppcli\programmings\Silicon Labs\440346135_matter-silabs-lighting-example.s37... Parsing file C:\Users\maryw\AppData\Roaming\SNB\ppcli\programmings\Silicon Labs\440346135_bootloader-storage-internal-single-512k-BRD2703A-gsdk4.1.s37... Writing 16384 bytes starting at address 0x08000000 Writing 1024000 bytes starting at address 0x08006000 Comparing range 0x08000000 - 0x08003FFF (16 KB) Comparing range 0x08006000 - 0x08025FFF (128 KB) Comparing range 0x08026000 - 0x08045FFF (128 KB) Comparing range 0x08046000 - 0x08065FFF (128 KB) Comparing range 0x08066000 - 0x08085FFF (128 KB) Comparing range 0x08086000 - 0x080A5FFF (128 KB) Comparing range 0x080A6000 - 0x080C5FFF (128 KB) Comparing range 0x080C6000 - 0x080E5FFF (128 KB) Comparing range 0x080E6000 - 0x080FFFFF (104 KB) Comparing range 0x08000000 - 0x08003FFF (16 KB) Comparing range 0x08006000 - 0x08025FFF (128 KB) Comparing range 0x08026000 - 0x08045FFF (128 KB) Comparing range 0x08046000 - 0x08065FFF (128 KB) Comparing range 0x08066000 - 0x08085FFF (128 KB) Comparing range 0x08086000 - 0x080A5FFF (128 KB) Comparing range 0x080A6000 - 0x080C5FFF (128 KB) Comparing range 0x080C6000 - 0x080E5FFF (128 KB) Comparing range 0x080E6000 - 0x080FFFFF (104 KB) Flashing completed successfully! DONE [Progress]:70%
[14:43:56] [Progress]:80%
[14:43:56] [Progress]:90%
[14:43:58] [Progress]:100% [Finish].
[14:44:00] out:b'{\r\n "batchId": "8426831076130816",\r\n "batchName": "Batch1227",\r\n "uuid": "5d95a1b1-5754-4861-964b-36ba4e583ec1",\r\n "bundles": [\r\n {\r\n "type": "dataBundle",\r\n "section": "section",\r\n "items": [\r\n {\r\n "key": "log"\r\n }\r\n]\r\n }\r\n]\r\n}\r\n'
[14:44:00] [EXIT] 0
[QRCode]: MT:634J042C00Y14Q6K.00
QRcode link
Concurrency Load: 0%
Batch in progress
English
1

Version: v3.2.1

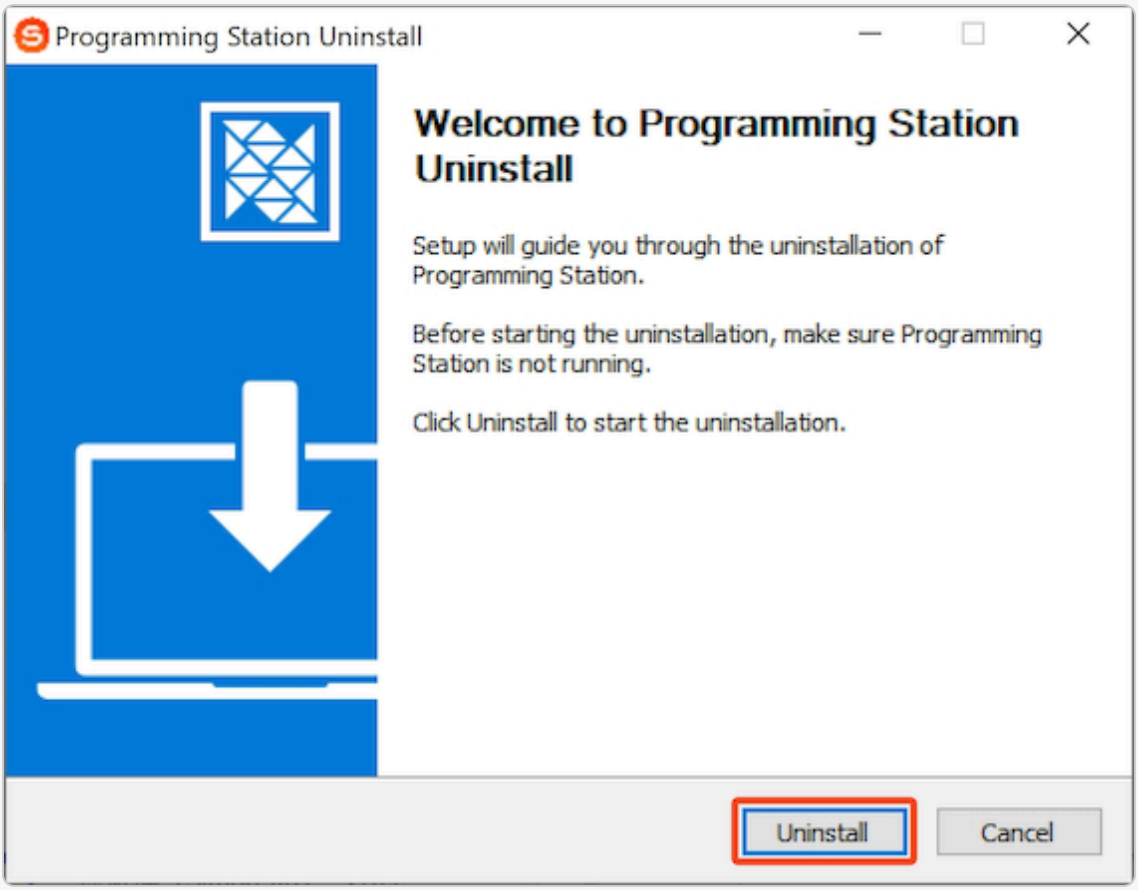
How to uninstall Programming Station

> **INFO**
>
> Currently, the Programming Station client is available only on Windows.

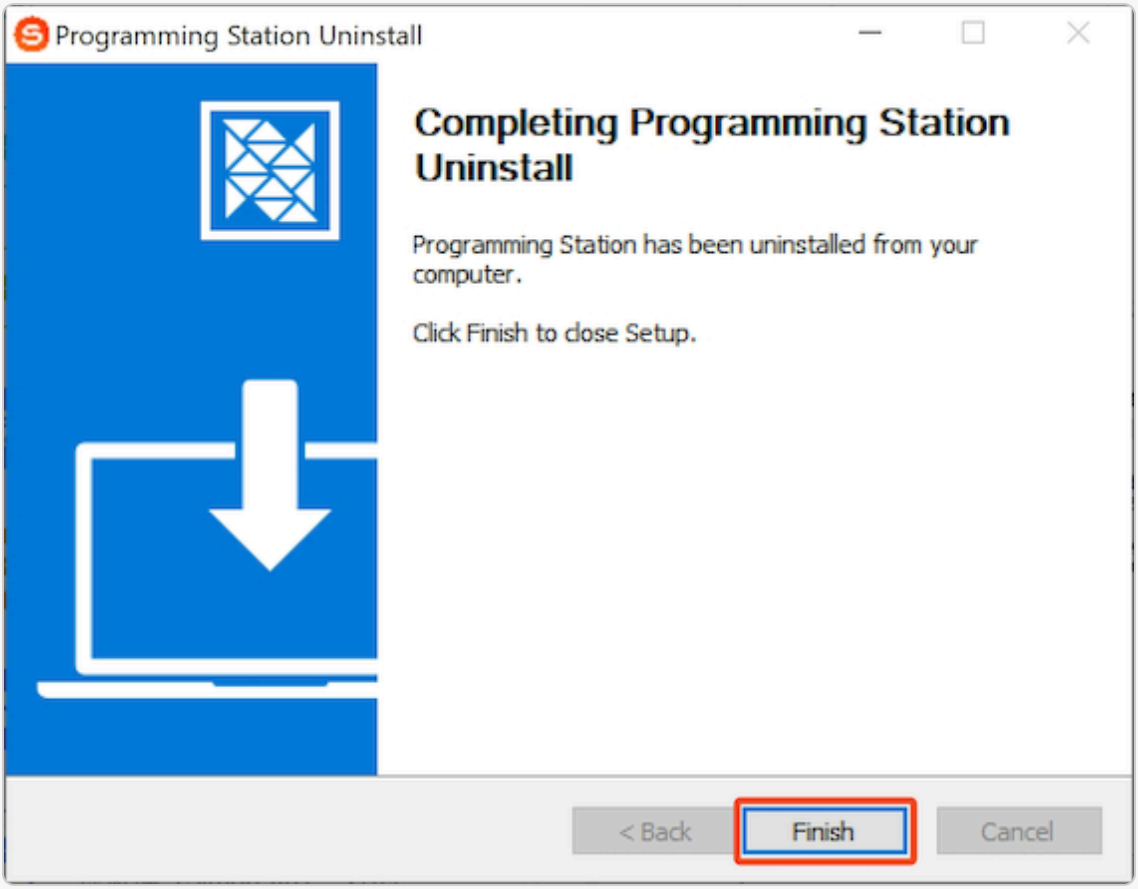
This article will guide you on how to uninstall the Programming Station client.

Instructions

1. Right-click the client icon on the desktop or the Start menu, and then click **Open file location**.
2. In the installation location, find the file named **Uninstall Programming Station** and double-click to run it.
3. In the pop-up window, click **Uninstall** to start the uninstallation.



4. After the client is uninstalled, click **Finish** to complete the uninstallation.


Programming Station Uninstall
Completing Programming Station Uninstall
Programming Station has been uninstalled from your computer.
Click Finish to close Setup.
< Back
Finish
Cancel

Version: v3.2.1

Factory Edge Service API - v1.5

1. Revision Record

Version	Modified Date	Notes
v1.0	2025/03/26	Initial release
v1.1	2025/12/17	Added "Query Device Details" API, which can be used to retrieve programming status
v1.2	2025/12/22	• Updated the required parameter validation and parameter descriptions of the "Query Device Details" API (Supports using the Matter QR code as a query condition to retrieve device information) • Added API exception status code definitions • Added an error code list
v1.3	2026/01/26	• Deprecated "Query Device Data" and "Update Device Info" APIs • Added "Update Devices Profile" API
v1.4	2026/09/10	Added the "Upload Device Production Data" API.
v1.5	2026/10/16	• Updated the "Query Device Details", "Update Devices Profile", and "Upload Device Production Data" APIs. • Added "Query Device Information", "Recover Matter QR Code", "Upload Device Test File", "Connection Heartbeat", "Query Login Options", "Query Programming Station Login Method", "U-Safe Pre-login", "Programming Station Login", "Programming Station Logout", "Query Batches by Order Number", "Assign Programming Station Batch", "Query Programming Station Batch Information", "Get Dynamic Programming Data", "Get Batch Metadata", "Get Full Programming Data" (compatibility API), "Submit Intermediate Programming Request Data", "Report Programming Result", "SE APDU Stepwise Programming", "Get Programming Script URL", "Download File", "Upload Device Event Logs", "Generate Matter Verifier", "Generate Matter Onboarding Payload", and "Generate Matter Unique ID" APIs. • Added compatibility notes for the deprecated "Query Device Data" and "Update Device Info" APIs. • vAdded common call conventions and error code descriptions.

2. Overview

Factory Edge Service API supports Programming Station login, production batch management, device programming, production data upload, device data queries, and device profile updates. These APIs use HTTP for data transmission, and all requests and responses are encoded in UTF-8.

This document describes the API conventions and common request headers, and provides request parameters, response parameters, and call examples for each API.

3. Common Request Headers

These APIs use the following keys (all lowercase):

Key	Mandatory	Description
content-type	Yes	Content type, unless otherwise specified, should always be `application/json;charset=utf-8`
x-snbps-factory-id	No	Factory ID
x-snbps-timestamp	No	Request time. For example, 2021-09-17T16:47:05.132+0800
x-snbps-lang	No	Language code, default is English: en-US, supports Simplified Chinese: zh-CN
x-snbps-request-id	No	Request `traceID`

3.1 API Call Conventions and Authentication

3.1.1 Service Address and Request Format

The service prefix is `{BASE_URL}/api/iot-factory-edge-service`. `{BASE_URL}` is the actual Edge address, including protocol, IP address or domain name, and port; it must not already include the service prefix. Paths below are relative to this prefix. JSON requests use UTF-8 encoding. Pass GET parameters in the URL query string; the client generates the boundary for `multipart` file uploads.

3.1.2 Programming Station Session and API Validation

After a Programming Station (PS) calls `/login`, retrieve the session token from `respData.token` and send it in the `x-snbps-session` header of subsequent requests. The station session token is distinct from a programming `traceId` or `uuid`. `x-snbps-request-id` is only for request tracing and cannot replace the business `traceId`.

The session lifetime is deployment-dependent and is refreshed when a protected API is called.

`UkeyPermissionFilter` enforces session validation for the following APIs:

```
/assignBatch
/queryPPCBatchInfo
/v1/queryBatchByOrderNumber
/getPPData
/getProgrammingData
/uploadRequestData
/programmingResult
/executeApdu
/v1/eventLog/upload
/v1/matter/onboarding-payloads/recover
/v1/device/information/query
/v1/testFile/upload
/v1/deviceData/upload
```

If validation data is missing or invalid, the service returns `HTTP 200` with business code `8050`.

3.1.3 Platform Authentication and Factory Context

Platform authentication is controlled separately by deployment configuration. In the current repository, `application.yml` does not exempt the following API paths from platform authentication:

- `/v1/device/detail`
- `/v1/updateDevicesProfile`
- `/v1/deviceData/upload`
- `/executeApdu`
- `/v1/eventLog/upload`
- `/v1/queryBatchByOrderNumber`
- The three `/generate*` API paths

When platform authentication is enabled in the target environment, these calls also require `x-snbps-auth-token`. Check the target environment's authentication configuration before integration testing; a valid station session token alone does not establish that all authentication requirements are met.

`x-snbps-factory-id` carries common factory information. The current factory context is primarily resolved from platform login information or a valid station session; this header alone cannot switch factories.

3.2 Unified Response and Error Handling

Factory Edge Service API generally uses a unified JSON response format, with the following fields:

Field	Type	Mandatory / condition	Description
success	boolean	Response	`true` indicates business success.
respCode	string	Response	`8000` indicates success; see the appendix for other business error codes.
respMsg	string	Response	Status description, affected by the selected language.
respData	object / array / null	Response	Business data for the API; usually `null` when no data is returned or the business operation fails.
respTime	integer (int64)	Response	Unix timestamp in seconds, not a string.

3.2.1 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": null,
  "respTime": 1792108800
}
```

3.2.2 Error Handling

Business exceptions (`ClientApiException`) and expired station sessions are returned with HTTP 200. Callers must check `success` and `respCode` rather than relying on the HTTP status alone.

Parameter binding errors, Bean Validation failures, platform authentication failures, and gateway errors are not guaranteed to return HTTP 200. Handle the actual HTTP status and response content.

3.2.3 Special Responses

On success, `/download` returns a file stream directly rather than the unified JSON format.

4. API Index (29 APIs)

This section lists the current 29 Factory Edge Service APIs, including method, relative path, purpose, and caller identity requirements. See API Call Conventions and Authentication for authentication rules.

No.	Request Method	Relative Path	API Name	Caller Identity
5.1	POST	/v1/device/detail	Query Device Details	Platform authentication depends on deployment configuration.
5.2	POST	/v1/updateDevicesProfile	Update Devices Profile	Platform authentication depends on deployment configuration.
5.3	POST	/v1/deviceData/upload	Upload Device Production Data	Station session; see common conventions for platform authentication.
5.4	POST	/v1/device/information/query	Query Device Information	Station session; see common conventions for platform authentication.
5.5	POST	/v1/matter/onboarding-payloads/recover	Recover Matter QR Code	Station session; see common conventions for platform authentication.
5.6	POST	/v1/testFile/upload	Upload Device Test File	Station session; see common conventions for platform authentication.
5.7	GET	/heartBeat	Connection Heartbeat	Before login / initialization.
5.8	GET	/queryLoginOptions	Query Login Options	Before login / initialization.
5.9	POST	/queryLoginType	Query Programming Station Login Method	Before login / initialization.
5.10	GET	/preLogin	U-Safe Pre-login	Before login / initialization.
5.11	POST	/login	Programming Station Login	Before login / initialization.
5.12	POST	/logout	Programming Station Logout	Provide station context as required.

No.	Request Method	Relative Path	API Name	Caller Identity
5.13	GET	/v1/ queryBatchByOrderNumber	Query Batches by Order Number	Station session; see common conventions for platform authentication.
5.14	POST	/assignBatch	Assign Programming Station Batch	Station session; see common conventions for platform authentication.
5.15	GET	/queryPPCBatchInfo	Query Programming Station Batch Information	Station session; see common conventions for platform authentication.
5.16	GET	/getProgrammingData	Get Dynamic Programming Data	Station session; see common conventions for platform authentication.
5.17	GET	/getMetadata	Get Batch Metadata	Provide station context as required.
5.18	GET	/getPPData	Get Full Programming Data (Compatibility API)	Station session; see common conventions for platform authentication.
5.19	POST	/uploadRequestData	Submit Intermediate Programming Request Data	Station session; see common cdeploymentns for platform authentication.
5.20	POST	/programmingResult	Report Programming Result	Station session; see common conventions for platform authentication.
5.21	POST	/executeApdu	Execute SE APDU Stepwise Programming	Station session; see common conventions for platform authentication.
5.22	POST	/getScriptFile	Get Programming Script URL	Provide station context as required.
5.23	GET	/download	Download File	Provide station context as required.
5.24	POST	/v1/eventLog/upload	Upload Device Event Logs	Station session; see common conventions for platform authentication.
5.25	POST	/generateVerifier	Generate Matter Verifier	Platform authentication depends on deployment configuration.
5.26	POST	/generateOnboardingData	Generate Matter Onboarding Payload	Platform authentication depends on deployment configuration.
5.27	POST	/generateUniqueId	Generate Matter Unique ID	Platform authentication depends on deployment configuration.
5.28	POST	/queryDeviceData	Query Device Data (Deprecated)	Provide station context as required.
5.29	POST	/updateDeviceInfo	Update Device Info (Deprecated)	Provide station context as required.

4.1 API Scope

This index covers the 29 external routes currently registered by `ProgrammingStationController` and `ExternalSystemController`. It excludes administration controllers, the debugging `TestController`, and the entirely commented-out `UsbKeyController`.

`/v1/testData/upload` remains in some filter or authentication exclusion configurations, but no current controller registers it. Upload device production data through `/v1/deviceData/upload`.

5. API List

5.1 Query Device Details

Query device profiles using locator and filter conditions combined with AND logic, and return the groups or fields specified by `queryKeys`. If no local record matches, query historical data and then cloud data. If multiple results refer to different products or `deviceId` values, return error code `8102`. If multiple records belong to the same product and device, return the most recent by production time.

5.1.1 Request Method

`POST`

5.1.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/v1/device/detail

5.1.3 Request Body

The request `body` contains the following parameters:

> **TIP**
> At least one non-empty value is required among `chipUid`, `bleMac`, `serialNumber`, `matterQrcode`, and `matterManualCode` in `deviceLocator`. Other parameters are auxiliary query conditions. All conditions are combined using AND logic. Provide sufficiently precise parameters to identify the target device. `queryKeys` only selects returned data groups or fields; it does not participate in device matching.

Name	Type	Mandatory	Description
deviceLocator	object	Yes	Used to retrieve basic device information. When `deviceLocator` is selected as a query condition, at least one of the following parameters must be provided: `chipUid`, `bleMac`, `serialNumber`, `matterQrcode`, or `matterManualCode`
├──chipUid	string	No	`mcuId` in the programmed device chip
├──bleMac	string	No	Bluetooth MAC address of the chip
├──serialNumber	string	No	Device serial number assigned during programming
├──matterQrcode	string	No	Matter commissioning QR code
├──matterManualCode	string	No	Matter manual commissioning code

Name	Type	Mandatory	Description
filters	object	No	Auxiliary filter conditions used to improve device identification accuracy
├──productionTimeStart	string	No	Device programming production start time. Date format: 2025-12-16T12:02:58+08:00
├──productionTimeEnd	string	No	Device programming production completion time. Date format: 2025-12-17T12:02:58+08:00
├──batchId	string	No	Production batch number
matter	object	No	Matter-related device parameters. Providing this parameter helps improve query accuracy
├──pid	string	No	Matter PID. Including this parameter helps improve device query accuracy
├──vid	string	No	Matter VID. Including this parameter helps improve device query accuracy
customParams	object	No	Custom filter parameters. For details, refer to the description of customParams parameter.
queryKeys	string[]	Recommended	Pass a string array naming groups (such as `matterData`) or fields (such as `chipData.chipUid`). If a local device profile matches but `queryKeys` is omitted or empty, the API returns an empty object `{}` rather than all fields by default. See the queryKeys V2 list for supported keys.

Description of customParams Parameter

This API accepts custom query conditions in customParams for more flexible filtering. To query fields uploaded for a business use case, add the field names and expected values to this object as key-value pairs.

Parameter structure

- Type: JSON object (key-value collection).
- Key: The extension field name to match (for example, `"customData"`).
- Value: The expected value for this field (for example, `"customDataValue"`).

Example

```
{"customData1": "customDataValue1", "customData2": "customDataValue2"}
```

This example indicates querying device records where the reported data customData1 equals "customDataValue1" and customData2 equals "customDataValue2".

Name	Type	Mandatory	Description
key	string	Yes	Used to query the basic key values in the device profile information. The available keys are either system-generated or a set of custom keys uploaded during the programming process
value	string	Yes	Each key corresponds to the value of the query parameter

5.1.4 Response Parameters

> **TIP**
> The following table shows the data returned when querying all `queryKeys`. In practice, querying only the required keys can help improve API performance.

Name	Type	Description
respCode	string	`8000` indicates success; any other value indicates an error. Detailed error information can be found in `respMsg`
respMsg	string	Response messages, such as `NoError` or `DataNotFound`
respData	object	Query result data
├──matterData	object	Matter data related
├──iterations	string	Iteration count
├──commissioningFlows	string	Commissioning flow
├──discoverableMode	string	Device discoverable mode
├──vid	string	Vendor ID
├──pid	string	Product ID
├──discriminator	string	Discriminator
├──enableRotatingDeviceId	string	Enable rotating device identifier
├──manualCode	string	Manual pairing code
├──passcode	string	Pairing code
├──qrcode	string	Matter commissioning QR code
├──salt	string	Salt
├──saltLength	string	Salt length
├──uniqueid	string	Unique device identifier
├──verifier	string	Password verifier
├──serialNumber	string	Matter device programming serial number
├──dacCertMd5	string	DAC certificate MD5
├──cdFileMd5	string	CD file MD5

Name	Type	Description
├──cdFileName	string	CD file name
├──paiCertMd5	string	PAI certificate MD5
├──factoryData	object	Factory data configuration related
├──vendorName	string	Vendor name
├──hardwareVersion	string	Hardware version
├──hardwareVersionString	string	Hardware version string
├──partNumber	string	Part number
├──productUrl	string	Product URL
├──productLabel	string	Product label
├──productName	string	Product name
├──startAddress	string	Start address
├──programmingData	object	Device programming results related
├──programmingStatus	string	Device programming status • Success: `success` • Failure: `failure` • Pending: `pending`
├──programmingLog	string	Programming log
├──programmingTime	string	Programming time
├──chipData	object	Chip data related
├──chipUid	string	mucId in the programmed device chip
├──bleMac	string	Bluetooth MAC address of the chip
├──firmwareData	object[]	Firmware related
├──fwName	string	Firmware name
├──fwMd5	string	Firmware MD5
├──customData	object	Custom data reported by the customer during production

5.1.5 Request Example

```
{
  "deviceLocator": {
    "chipUid": "f9e203b4cf3ce6323637364250 55f321",
    "bleMac": "f9:e2:03:b4:cf:3c:e6:32",
    "serialNumber": "8360224622727172372"
  },
  "filters": {
    "productionTimeStart": "2025-12-16T12:02:58+08:00",
    "batchId": "7603594830282752"
  },
  "matter": {
    "pid": "8001",
    "vid": "FFF2"
  },
  "customParams": {
    "customData1": "nPK3DqGQQwdE3fYxaofURw==",
    "customData2": "1"
  },
  "queryKeys": [
    "matterData.vid",
    "matterData.pid",
    "matterData.manualCode",
    "matterData.passcode",
    "matterData.qrcode",
    "matterData.salt",
    "matterData.uniqueid",
    "matterData.verifier",
    "matterData.serialNumber",
    "matterData.dacCertMd5",
    "matterData.paiCertMd5",
    "factoryData.vendorName",
    "factoryData.hardwareVersion",
    "factoryData.hardwareVersionString",
    "factoryData.partNumber",
    "factoryData.productUrl",
    "factoryData.productLabel",
    "factoryData.productName",
    "programmingData.programmingStatus",
    "chipData.chipUid",
    "chipData.bleMac",
    "firmwareData.fwName",
    "firmwareData.fwMd5"
  ]
}
```

5.1.6 Response Example

Note: This response example shows all result data. In practice, use `queryKeys` to retrieve only the required data.

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": {
    "matterData": {
      "vid": "FFF2",
      "pid": "8001",
```

5.1.7 Current Implementation Notes

- **Supported data groups**: `matterData`、`factoryData`、`programmingData`、`chipData`、`firmwareData`、`customData`、`seData`。
- **Field return rules**: `seData` can be returned only as a whole group; specifying `seData.xxx` does not filter its subfields. Unknown groups in `queryKeys` add no data. If a requested field is empty or has not been stored, its value may be empty.
- **Programming data**: `programmingData.programmingStatus` supports `success`, `failure`, and `pending`. The current local data construction logic does not populate `programmingLog` with actual log content; do not rely on it for programming logs.
- **Authentication**: `UkeyPermissionFilter` does not enforce station session validation for this API. See API Call Conventions and Authentication for platform authentication requirements.

5.2 Update Devices Profile

After a factory modifies device programming data, it can use this API to synchronize the updates to OBIS for centralized management.

5.2.1 Request Method

`POST`

5.2.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/v1/updateDevicesProfile

5.2.3 Request Body

The request `body` contains the following parameters:

Name	Type	Mandatory	Description
deviceLocator	object	Yes	Used to retrieve basic device information. When `deviceLocator` is selected as a query condition, at least one of the following parameters must be provided: `chipUid`, `bleMac`, `serialNumber`, `matterQrcode`, or `matterManualCode`
├——chipUid	string	No	`mcuId` in the programmed device chip
├——bleMac	string	No	Bluetooth MAC address of the chip
├——serialNumber	string	No	Device serial number assigned during programming
├——matterQrcode	string	No	Matter commissioning QR code
├——matterManualCode	string	No	Matter manual commissioning code
filters	object	No	Auxiliary filter conditions used to improve device identification accuracy
├——productionTimeStart	string	No	Device programming production start time. Date format: 2025-12-16T12:02:58+08:00
├——productionTimeEnd	string	No	Device programming production completion time. Date format: 2025-12-17T12:02:58+08:00
├——batchId	string	No	Production batch number

Name	Type	Mandatory	Description
matter	object	No	Matter-related device parameters. Providing this parameter helps improve query accuracy
├──pid	string	No	Matter PID. Including this parameter helps improve device query accuracy
├──vid	string	No	Matter VID. Including this parameter helps improve device query accuracy
customParams	object	No	Custom filter parameters. For details, refer to the description of customParams parameter.
updateParams	object[]	No	Pass updates as an array. Each item contains `type`, `key`, and `value`, all `strings`. An empty array does not update fields. See the updateParams description for supported keys and operations.

Description of updateParams Parameter

This API accepts system-supported keys in updateParams to modify specified fields in a device record. Provide the field name, target value, and operation type (type) together as an array.

Parameter structure

- Type: JSON array.
- When type is `add`, a new field record corresponding to the key will be created; if the key already exists, its field value will be updated.
- Some device data supports deletion. When type is `delete`, the field record corresponding to the key will be deleted.
- For details about the keys supported by this parameter, refer to the updatable keys list (updateDevicesProfile).

Example

```
[
    {
        "type": "add",
        "key": "qrcode",
        "value": "MT:XXX-S0XXXX"
    },
    {
        "type": "delete",
        "key": "factoryData.productURL"
    },
    {
        "type": "add",
        "key": "factoryData.qqbb",
        "value": "abcd"
    },
    {
        "type": "add",
        "key": "firmwareData.bootloader-xxx-1.1.s37",
        "value": "00000001"
    }
]
```

The above example indicates:

1. Add the `qrcode` field and set its value to MT:XXX-S0XXXX; if the field already exists in the device, the existing value will be

overwritten.

2. Delete the `productURL` field from `factoryData` of the device.
3. Add the `qqbb` field to `factoryData` of the corresponding device and set its value to `abcd`.
4. Add the firmware bootloader-xxx-1.1.s37 to the firmware list of the device and set its `startAddress` to `00000001`. If a firmware with the same name already exists in the device, update its start address to this value.

5.2.4 Response Parameters

Name	Type	Description
respCode	string	`8000` indicates success; any other value indicates an error. Detailed error information can be found in `respMsg`
success	bool	`True` indicates success
respMsg	string	Response messages, such as `NoError` or `DataNotFound`

5.2.5 Request Example

```
{
  "deviceLocator": {
    "chipUid": "f9e203b4cf3ce632363736425055f321"
  },
  "filters": {
    "productionTimeStart": "2025-12-16T12:02:58+08:00",
    "batchId": "7603594830282752"
  },
  "matter": {
    "pid": "8001",
    "vid": "FFF2"
  },
  "customParams": {
    "customData1": "nPK3DqGQQwdE3fYxaofURw=="
  },
  "updateParams": [
    {
      "type": "add",
      "key": "qrcode",
      "value": "MT:XXX-S0XXXX"
    },
    {
      "type": "delete",
      "key": "factoryData.productURL",
      "value": ""
    },
    {
      "type": "add",
      "key": "factoryData.qqbb",
      "value": "abcd"
    },
    {
      "type": "add",
      "key": "firmwareData.bootloader-xxx.s37",
      "value": "00000000"
    }
  ]
}
```

5.2.6 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respTime": 1765768529
}
```

5.2.7 Operation Rules and Limits

The following requirements apply to fields in `updateParams`:

Field	Type	Mandatory / condition	Description
updateParams[].type	string	By operation	`add` creates or overwrites; `delete` applies only to collection fields that support deletion.
updateParams[].key	string	Yes	A system key or collection prefix listed in Appendix 3.
updateParams[].value	string	For add	Only strings are accepted. Date and integer text must meet the target field format requirements.

Device Location and Update Scope

The device location rules for `deviceLocator`, `filters`, `matter`, and `customParams` are the same as for Query Device Details. Updates affect local device profiles only; they do not query historical or cloud data.

Field Value Rules

- The environment value supports only `test` and `prod`.
- `programmingStatus` supports `Complete`, `Failed`, and `Unreported`. Other values are currently treated as incomplete.

Field Update and Deletion Rules

- **Deletable fields**: `factoryData.*`, `firmwareData.*`, `otherFile.*`, `attachment.*`, and `programmingScript`.
- **Fields that cannot be deleted**: scalar keys and `extend.*` are currently assigned directly from `value`; `delete` is not supported.
- Unknown fields: unknown `keys` are currently skipped and do not necessarily produce a parameter error.

Other Processing Rules

- If `updateParams` is an empty array, no fields are updated, but the service still attempts to read the target device profile and metadata.
- After an update, the record is marked as awaiting synchronization. Receiving the HTTP response does not mean cloud synchronization is complete.

5.3 Upload Device Production Data

Factories can use this API to upload programming, test, and assembly data generated during device production and maintain the corresponding production record by `traceId`. At the end of programming, the API reuses the `programmingResult` processing logic. Test and assembly data are synchronized by a subsequent asynchronous flow.

5.3.1 Request Method

`POST`

5.3.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/v1/deviceData/upload

5.3.3 Request Body

The `Content-Type` for this API is `application/json`. In addition to the common request headers, provide the following header:

Field	Mandatory	Description
x-snbps-session	Yes	Station session `token` returned by `/login`.

5.3.4 Request Body

The request `body` contains the following parameters:

Name	Type	Mandatory / condition	Description
traceId	string	By scenario	Production record trace ID; `uuid` is an accepted alias. In the programming scenario, use the `uuid` returned by `/getProgrammingData`. If omitted, a new record may be created. A supplied value must refer to an existing production record or file task; otherwise the API returns `8006`.
scenarioType	string	Required for test or custom scenarios	Production scenario name, for example `RF-Test`. The `assembly` batch uses assembly, and an existing programming session uses `programming`. Other scenarios return `8069` if this field is absent.
deviceId	string	Required for test or assembly integration	Device MCU ID. If omitted, `deviceRecord.deviceId` is used.
batchId	string	Required for assembly integration	Production batch ID. Only a numeric string is treated as an ID. Assembly is identified by batch type, not by `scenarioType=assembly` alone.
deviceProfileId	string	No	Existing device profile ID; it must correspond to `deviceId`. If omitted, the service matches an existing profile by device ID.
stationId	string	No	Programming Station code. If omitted, the station in the current session is used. It need not be numeric.
factoryId	string	No	Factory ID. For new records, only a numeric string takes effect; otherwise the factory from the current session is used.
deviceSn	string	No	Device serial number.

Name	Type	Mandatory / condition	Description
result	boolean	Recommended for programming	Programming result: `true` for success, `false` for failure. If omitted at the end of programming, it is treated as failure.
failureReason	string	No	Failure reason, stored in the production record.
timestamp	string	No	This API does not use this field to set the production record time; the server generates the creation time.
status	string	No	Custom status. Programming and assembly may generate and overwrite it from the system result; other nonempty values are stored in `customStatus`.
deviceRecord	object	Per scenario integration contract	Device production data; `deviceProfile` is an accepted alias. See deviceRecord parameters for its fields.

5.3.5 deviceRecord Parameters

`deviceRecord` contains data produced by the device at the current production stage.

Name	Type	Required / Condition	Description
deviceId	string	Conditionally required	Device MCU ID; used when top-level `deviceId` in the request body is empty.
firmwareVersion	string	As needed for programming	Firmware version
seid	string	As needed for SE programming	SEID; use lowercase for the field name.
cplc	string	As needed for SE programming	CPLC data
matterData	map	As needed for Matter programming	Matter data actually programmed.
factoryData	map	No	Factory Data actually programmed. Field names must be nonempty and values must be strings.
customData	map	No	Custom data, up to 50 fields. Field names must be nonempty and no longer than 64 characters, and values must be strings; otherwise `8002` is returned. This upload replaces all existing `customData`.

Name	Type	Required / Condition	Description
testData	map	Required for test / assembly integration	Test data; also accepts the legacy field name `diagnosticData`. Values may be text or lists of file references. Fields with the same name are appended when both values are lists; otherwise the new value overwrites the old one.

5.3.6 Scenario Handling Rules

Parameter requirements and handling rules for different production scenarios are as follows:

Scenario	traceId and Batch	Data and Status
Programming	Use the existing programming session `traceId`, usually from `/getProgrammingData`.	Provide `deviceRecord` based on the actual programmed content and explicitly pass `result`. At completion, the API reuses `/programmingResult` handling. Merely setting `scenarioType=programming` does not create a complete programming flow.
Test / Custom	The first upload may omit `traceId`; later uploads use the `traceId` returned in the response. `scenarioType` is required.	Provide `deviceId` and `deviceRecord.testData` as specified by the integration contract. If a matching device profile exists, it is associated; otherwise only a production record may be created and `deviceProfileId` is `null`.
Assembly	`batchId` identifies the assembly batch. `traceId` may come from `/getProgrammingData` or a file task; a new record can also be created if the first upload omits it.	Provide `deviceId` and `deviceRecord.testData`. The scenario is fixed to `assembly`. `testData.result` affects result status and output counts. The current implementation deduplicates successful assembly records for the same device in the same batch.

Parameter Length and Format Limits

Database field length limits are as follows:

Field	Maximum Length
traceId	64 characters
deviceId	128 characters
deviceSn	64 characters
failureReason	1024 characters
status / scenarioType	64 characters

These length limits do not mean that the API automatically validates every request field. ID values must fall within the Java `Long` range. Overlong fields, invalid number formats, or violations of database constraints may cause data persistence to fail. Numeric strings must also satisfy the applicable length and value limits.

Other Handling Rules

- **Programming data**: The integration contract requires at least one of `deviceRecord.firmwareVersion`, `deviceRecord.seid`, `deviceRecord.cplc`, or `deviceRecord.matterData`, but the current API does not enforce this requirement.
- **Test and assembly data**: Upload test data in `deviceRecord.testData`. A top-level `customData` field in the request body is not recognized as `deviceRecord.customData`.

5.3.7 Request Example

Programming Scenario

Use the `uuid` returned by `/getProgrammingData` as `traceId`, and submit the programming result and device data.

```
{
  "traceId": "<uuid returned by getProgrammingData>",
  "deviceId": "DEVICE-001",
  "result": true,
  "deviceRecord": {
    "firmwareVersion": "1.2.3",
    "matterData": {
      "qrcode": "<actual commissioning payload>"
    },
    "customData": {
      "mac": "AA:BB:CC:DD:EE:FF"
    }
  }
}
```

Test Scenario

The first test report may omit `traceId`. Specify the test scenario through `scenarioType` and submit the test data in `testData`.

```
{
  "scenarioType": "RF-Test",
  "deviceId": "DEVICE-001",
  "status": "success",
  "deviceRecord": {
    "testData": {
      "voltage": "3.3",
      "result": "success"
    },
    "customData": {
      "fixture": "A01"
    }
  }
}
```

Assembly Scenario

Use `batchId` to specify the assembly batch. The first report may omit `traceId`.

```
{
  "batchId": "<actual assembly batch ID>",
  "deviceId": "DEVICE-001",
  "deviceRecord": {
    "testData": {
```

5.3.8 Associate Files with Production

To associate a test file, first upload it with `POST /v1/testFile/upload` and save the returned `traceId`. Then call this API with the same `traceId` and reference the file through `fileName` in `testData`.

If an existing production record has a `traceId`, you may pass it when uploading the file. `fileTraceId` is an internal association field and need not be added to the top level of this API request.

Request example:

```
{
  "traceId": "<traceId returned by testFile/upload>",
  "scenarioType": "RF-Test",
  "deviceId": "DEVICE-001",
  "deviceRecord": {
    "testData": {
      "report": [
        {
          "fileName": "report.txt"
        }
      ]
    }
  }
}
```

5.3.9 Response Parameters

This API uses the standard JSON response format. See "Standard Response and Error Handling" for common response fields. `respData` contains the following fields:

Field	Type	Description
traceId	string	Trace ID ultimately used by the server. Pass it unchanged on subsequent uploads.
deviceProfileId	integer (int64) / null	Associated device profile ID; `null` if none is associated.
programmingRecordId	integer (int64)	Device production record ID

5.3.10 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": {
    "traceId": "example-trace-id",
    "deviceProfileId": 200,
    "programmingRecordId": 11
  },
  "respTime": 1792108800
}
```

5.3.11 Main Error Codes

Error Code	Description
8002	Custom field does not meet requirements.
8006	`traceId` is not associated with an existing production record or file task.
8007	Data persistence failed.
8032	Batch does not exist.
8037	Device profile creation failed.
8050	Programming Station session expired.
8069	`scenarioType` is missing for a test or custom scenario.

5.3.12 HTTP Call Example and Subsequent Uploads

The following `curl` example uploads test data for the first time. `BASE_URL` contains only the scheme, host, and port, for example `http://edge-host:8005`. `SESSION` is the session token returned by `/login` for the Programming Station. If platform authentication is enabled in the target environment, also supply the `x-snbps-auth-token header`.

```
curl -X POST "$BASE_URL/api/iot-factory-edge-service/v1/deviceData/upload" \
  -H "Content-Type: application/json;charset=utf-8" \
  -H "x-snbps-session: $SESSION" \
  -d '{
    "scenarioType": "RF-Test",
    "deviceId": "DEVICE-001",
    "status": "success",
    "deviceRecord": {
      "testData": {"voltage": "3.3", "result": "success"},
      "customData": {"fixture": "A01"}
    }
  }'
```

Subsequent Upload Rules and Notes

- **Trace ID**: After the first successful upload, save `respData.traceId` from the response. Pass that value at the top level of subsequent uploads to the same production record, and continue to supply `scenarioType`. Independent tests must not reuse an existing `traceId`.
- **Identifiers**: `x-snbps-request-id` is only for request tracing and cannot replace the business `traceId`. Passing a `traceId` that does not exist and is not associated with a file task returns `8006`.
- **Test data**: Under the same `traceId`, list fields with the same name in `testData` are appended. Repeated requests may create duplicate attachment references.
- **Custom data**: Each upload replaces all existing `customData`. On a subsequent upload, include every custom field you need to retain.
- **Repeated requests**: After a network timeout, do not assume that the previous request did not run or that repeated uploads are fully idempotent.

5.4 Query Device Information

This API queries general information, Matter data, and specified custom data in the local device profile. Unlike `/v1/device/detail`, it cannot select returned fields through `queryKeys`.

5.4.1 Request Method

`POST`

5.4.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/v1/device/information/query

5.4.3 Request Headers

In addition to the common request headers, this API requires the following header:

Field	Required	Description
x-snbps-session	Yes	Programming Station session `token` returned by `/login`.

5.4.4 Request Body

The request `body` contains the following parameters. Supply at least one nonempty device identifier.

Field	Type	Required / Condition	Description
deviceId	string	At least one required	Exact match against `deviceId` in the device profile; maximum 128 characters.
chipId	string	At least one required	Exact match against `chipUid` in the device profile; maximum 128 characters.
uniqueId	string	At least one required	Match `uniqueId`, `uniqueid`, or the corresponding `snake_case` fields in Matter data; maximum 128 characters.

5.4.5 Notes

- **Query conditions**: Multiple nonempty device identifiers are combined with AND; all conditions must match.
- **Query scope**: Only local device profiles are queried, not the historical database or cloud. If no record matches, the API returns `8101`.
- **Query result**: If multiple records match, the first result is returned. This does not guarantee identifier uniqueness or that the returned device was programmed successfully.
- **Programming status**: `provisionStatus` currently always returns `Success` and cannot, by itself, establish the device's actual programming status.

5.4.6 Request Example

```
{
  "deviceId": "DEVICE-001"
}
```

5.4.7 Response Parameters

This API uses the standard JSON response format. `respData` contains the following data objects:

Field	Type	Description
generalData	object	Device identity, programming information, and product version information.
matterData	object / null	Matter certificate and commissioning data; may be `null` for non-Matter devices.
customData	object	Device serial number, business identifier, MAC address, and test and calibration data.

Fields without recorded values may be empty.

5.4.8 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": {
    "generalData": {
      "deviceId": "DEVICE-001",
      "provisionStatus": "Success",
      "provisionTime": "2026-10-16T02:00:00Z",
      "sourceProgramingBatchId": "10",
      "provisionFactoryId": "20",
      "provisionStationId": "30",
      "productId": "40",
      "productName": "sample product",
      "productVersionId": "50",
      "productVersionName": "1.0"
    },
    "matterData": null,
    "customData": {
      "serialNumber": "SN-001",
      "customId": "CUSTOM-001",
      "macAddress": "AA:BB:CC:DD:EE:FF",
      "factoryTestData": null,
      "calibrationParams": null
    }
  },
  "respTime": 1792108800
}
```

5.4.9 Response Field Descriptions

generalData: General information

> **TIP**
>
> `sourceProgramingBatchId` is the actual response field name; retain the original "`Programing`" spelling.

Field	Type	Description
deviceId	string	Device ID

Field	Type	Description
provisionStatus	string	Programming status; currently always returns `Success`.
provisionTime	string	Programming time in UTC ISO 8601 format.
sourceProgramingBatchId	string	Source programming batch ID
provisionFactoryId	string	Programming factory ID
provisionStationId	string	Programming Station ID; corresponds to the station primary key in the device profile.
productId	string	Product ID
productName	string	Product name
productVersionId	string	Product version ID
productVersionName	string	Product version name

matterData: Matter data

Field	Type	Description
dacCert	string	DAC certificate in its stored Base64 DER or PEM format.
paiCert	string	PAI certificate in its stored Base64 DER or PEM format.
certDeclaration	string	Certification Declaration (CD), Base64 encoded.
discriminator	integer	Matter Discriminator as a decimal number.
spake2plterationCount	integer	SPAKE2+ iteration count.
passcode	string	Matter commissioning passcode.
manualPairingCode	string	Matter manual pairing code.
setupPayload	string	Matter commissioning payload.
spake2pVerifier	string	SPAKE2+ verifier; retain the stored value.
spake2pSalt	string	SPAKE2+ salt; retain the stored value.

customData: Custom data

> **TIP**
> Before returning `factoryTestData` and `calibrationParams`, the API recursively removes defined credential fields and does not return the device private key.

Field	Type	Description
serialNumber	string	Device serial number
customId	string	Business identifier
macAddress	string	MAC address
factoryTestData	JSON	Factory test data
calibrationParams	JSON	Device calibration parameters

5.4.10 Main Error Codes

Error Code	Description
8002	All query parameters are empty.
8007	Stored data could not be parsed.
8050	Authentication failed.
8101	No matching device found.

See the appendix for other error codes.

5.5 Recover Matter QR Code

During assembly, this API uses the input programming batch associated with the assembly batch, Matter VID, PID, Discriminator, and the Rotating Device Identifier (RDI) in BLE advertising to find a successfully programmed local device profile and recover its Matter onboarding data.

5.5.1 Request Method

`POST`

5.5.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/v1/matter/onboarding-payloads/recover

5.5.3 Request Body

In addition to the common request headers, this API requires the following header:

Parameter	Required	Description
x-snbps-session	Yes	Programming Station session `token` returned by `/login`.

5.5.4 Request Body

The request `body` contains the following parameters:

Field	Type	Required	Description
batchId	string	Yes	Assembly batch ID. It must be associated with an input programming batch, have `RUNNING` status, and not be expired.
matterProductId	string	Yes	Matter PID as a hexadecimal string without a `0x` prefix. Do not use `productId` or `pid` as the field name.
matterVendorId	string	Yes	Matter VID as a hexadecimal string without a `0x` prefix. Do not use `vendorId` or `vid` as the field name.
discriminator	string	Yes	Hexadecimal string whose parsed value is in the range `0x000–0xFFF`, for example `0F00`. Do not pass a decimal string.
rotatingDeviceIdentifier	string	Yes	RDI from BLE advertising: 18 bytes (36 hexadecimal characters), without a `0x` prefix. Use the actual advertised value.

5.5.5 Notes

- **Query scope**: Only successfully programmed local device profiles in the input programming batch associated with the assembly batch are queried. The API does not look up RDI in the cloud.
- **Device matching**: The device is matched using VID, PID, Discriminator, and RDI. An RDI mismatch or no matching device returns `8101`.
- **Data format**: VID and PID are matched against their text values in the database. Use the same format as the batch data.

5.5.6 Request Example

```
{
  "batchId": "<assembly batch ID>",
  "matterProductId": "8001",
  "matterVendorId": "FFF2",
  "discriminator": "0F00",
  "rotatingDeviceIdentifier": "<actual 36-character hexadecimal BLE advertising RDI>"
}
```

5.5.7 Response Parameters

This API uses the standard JSON response format. The current implementation returns the following four fields in `respData`, rather than only `setupPayload`:

Field	Type	Description
setupPayload	string	Matter QR-code commissioning payload.
deviceId	string	Device ID
deviceProfileId	integer	Device profile ID
manualCode	string	Matter manual onboarding code. The field name is `manualCode`, not `manualPairingCode`.

5.5.8 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": {
    "setupPayload": "<actual QR code payload>",
    "deviceId": "DEVICE-001",
    "deviceProfileId": 200,
    "manualCode": "<actual manual pairing code>"
  },
  "respTime": 1792108800
}
```

5.5.9 Main Error Codes

Error Code	Description
8002	Invalid Discriminator parameter.
8009	Batch has expired.
8010	Batch is not in RUNNING status.
8032	Assembly batch or its association does not exist.
8050	Authentication failed.
8101	No matching device found.

See the appendix for other error codes.

5.6 Upload Device Test File

This API uploads device test files. After receiving a file, it saves the file locally and creates an asynchronous upload task. The HTTP response only confirms receipt and local processing; it does not mean the file has reached cloud storage.

5.6.1 Request Method

`POST`

5.6.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/v1/testFile/upload

5.6.3 Request Body

This API uses `multipart/form-data` as its Content-Type. In addition to the common request headers, supply the following header:

Parameter	Required	Description
x-snbps-session	Yes	Programming Station session `token` returned by `/login`.

5.6.4 Request Body

This API uploads files through multipart/form-data with the following parameters:

Field	Type	Required	Description
traceId	string (form field)	No	Existing script trace ID, if available; generated automatically if omitted. Pass the returned traceId unchanged to the production data upload API.
files	file[] (multipart parts)	Yes	List of files to upload; include at least one nonempty file. Repeat the files field to upload multiple files in one request. File names must not include a path, must not be `.` or `..`, and must be unique within the request.

5.6.5 Notes

- **File handling**: After receiving a file, the API saves it locally, then creates an asynchronous upload task.
- **Task tracking**: Pass an existing `traceId` if available; otherwise the API generates one. Pass the returned `traceId` unchanged to subsequent production data uploads.
- **Upload status**: A successful HTTP request does not mean the file has reached cloud storage. Errors during the background upload may not appear in the initial HTTP response.
- **File size limits**: File size is limited by the deployment's `multipart` configuration. `application.yml` defaults to 2 GB per file and 4 GB per request; the `dev` configuration sets 200 MB and 250 MB, respectively. Environment variables or external configuration may override the actual limits.

5.6.6 Request Example

```
curl -X POST "$BASE_URL/api/iot-factory-edge-service/v1/testFile/upload" \
  -H "x-snbps-session: $SESSION" \
  -F "files=@./report.txt" \
  -F "files=@./photo.png"
```

5.6.7 Response Parameters

This API uses the standard JSON response format. `respData` contains the following fields:

Field	Type	Description
traceId	string	File task trace ID.
fileCount	integer	Number of files received in this request.
status	integer	Asynchronous file upload task status: `1` pending sync; `2` uploading; `3` successful; `4` awaiting retry; `5` failed; `6` local file cleaned up.

5.6.8 Response Example

```
{
  "success": true,
  "respCode": "8000",
```

5.6.9 Main Error Codes

Error Code	Description
8057	Invalid upload parameters.
8058	File list or file name does not meet requirements.
8059	Local file storage failed.
8060	Failed to save the file upload task.
8061	Background cloud storage upload failed.
8051	Insufficient storage space or concurrent processing capacity.

Errors during asynchronous background upload may not appear in the initial HTTP response. See the appendix for other error codes.

5.7 Connection Heartbeat

The programming client uses this API to check connectivity to the Edge service.

5.7.1 Request Method

`GET`

5.7.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/heartBeat

5.7.3 Request Parameters

This is an initialization API. It does not require a Programming Station session token in advance and has no request parameters.

5.7.4 Notes

The client may call this API periodically as needed. A successful heartbeat only means the Edge HTTP service is reachable; it does not establish that other business services are healthy.

5.7.5 Request Example

```
GET {BASE_URL}/api/iot-factory-edge-service/heartBeat
```

5.7.6 Response Parameters

This API uses the standard JSON response format. On success, `respData` is `null`.

5.7.7 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": null,
```

5.8 Query Login Options

For compatibility with older clients, this API queries whether a U-Safe and a production password are required for the batch assigned to the Programming Station.

5.8.1 Request Method

`GET`

5.8.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/queryLoginOptions

5.8.3 Request Parameters

This is an initialization API and does not require a Programming Station session token in advance. Pass the following fields as query parameters:

Field	Type	Required / Condition	Description
usafeSN	string	Provide at least one of usafeSN or stationId.	Legacy U-Safe serial number.
stationId	string	Provide at least one of usafeSN or stationId.	Programming Station code; takes precedence when supplied.

5.8.4 Notes

- **Query conditions**: The Programming Station and its assigned batch must exist.
- **Compatibility**: This API supports older clients. In the newer login flow, call `/queryLoginType` first to determine the Programming Station login method.

5.8.5 Request Example

```
GET {BASE_URL}/api/iot-factory-edge-service/queryLoginOptions?stationId=STATION-001
```

5.8.6 Response Parameters

This API uses the standard JSON response format. `respData` contains the following fields:

Field	Type	Description
requireUsafe	boolean	Whether U-Safe is required; currently always `true`.
requirePassword	boolean	Whether a production password is required, as determined by the assigned batch setting.

5.8.7 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": {
    "requireUsafe": true,
```

5.8.8 Main Error Codes

Error Code	Description
8007	Programming Station or assigned batch not found.

See the appendix for other error codes.

5.9 Query Programming Station Login Method

This API queries the login method for a specified Programming Station and supports reporting client information for compatibility.

5.9.1 Request Method

`POST`

5.9.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/queryLoginType

5.9.3 Request Body

This is an initialization API and does not require a Programming Station session token in advance. The request `body` contains the following parameters:

Field	Type	Required / Condition	Description
factoryId	integer (int64)	Required for cloud factories	In cloud mode, verifies that the specified factory exists and is a cloud factory.
stationId	string	Yes	Programming Station code
clientPcMessage	string	No	JSON object string containing client information. Deprecated since 3.3.2; new clients should report it through `/login`.

5.9.4 Notes

Cloud factory login supports only password or Identity U-Safe authentication.

5.9.5 Request Example

```
{
  "factoryId": 20,
  "stationId": "STATION-001"
}
```

5.9.6 Response Parameters

This API uses the standard JSON response format. `respData` contains the following fields:

Field	Type	Description
loginType	integer	Login method: `1` passwordless; `2` password; `3` Identity U-Safe.

5.9.7 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": {
    "loginType": 2
  },
  "respTime": 1792108800
}
```

5.9.8 Main Error Codes

Error Code	Description
8016	Programming Station does not exist.
8036	Invalid cloud factory ID.

See the appendix for other error codes.

5.10 U-Safe Pre-login

This API performs the first step of U-Safe mutual authentication and returns the server certificate, challenge, and signature.

5.10.1 Request Method

`GET`

5.10.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/preLogin

5.10.3 Request Parameters

This is an initialization API and does not require a Programming Station session token in advance. Pass the following fields as query parameters:

Field	Type	Required / Condition	Description
usafeSN	string	Yes	Identity U-Safe serial number.
nonce	string	Yes	Client-generated random challenge.
type	integer	No	Compatibility parameter; the current method does not use its value.
factoryId	integer (int64)	Pass by factory in cloud mode	Used to match the factory scope in cloud mode.

5.10.4 Notes

- **Scope**: This API is for U-Safe mutual authentication, not direct password login.
- **Signature handling**: `sign` in the response is the server signature and cannot be used directly as the client login signature. The client should verify the server's identity, sign `challenge` according to the existing U-Safe authentication protocol, then call `/login`.

5.10.5 Request Example

```
GET {BASE_URL}/api/iot-factory-edge-service/
preLogin?usafeSN=USAFE-001&nonce=%3C%E5%AE%A2%E6%88%B7%E7%AB%AF%E9%9A%8F%E6%9C%BA%E6%95%B0%3E&factoryId=20
```

5.10.6 Response Parameters

This API uses the standard JSON response format. `respData` contains the following fields:

Field	Type	Description
cert	string	Server authentication certificate.
challenge	string	Server challenge data for the client to sign.
sign	string	Server signature for the client to verify server identity.

5.10.7 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": {
    "cert": "<authentication certificate>",
    "challenge": "<server challenge>",
    "sign": "<server signature>"
  },
  "respTime": 1792108800
}
```

5.10.8 Main Error Codes

Error Code	Description
8020	U-Safe does not exist.
8031	U-Safe is unavailable.
8024 / 8026	Authentication or signature verification failed.

See the appendix for other error codes.

5.11 Programming Station Login

This API authenticates the Programming Station and obtains a station session token. After a successful login, the programming client passes the returned `token` in the `x-snbps-session` header on subsequent requests requiring a station session.

5.11.1 Request Method

`POST`

5.11.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/login

5.11.3 Request Body

This is an initialization API and does not require a Programming Station session token in advance. The request `body` contains the following parameters:

Field	Type	Required / Condition	Description
stationId	string	Required for new login flow	When nonempty, authenticates the Programming Station; when omitted, uses the legacy U-Safe authentication flow.
factoryId	integer (int64)	Required for cloud factories	Factory ID
password	string	Required when loginType=2	Password encrypted with the existing RSA public key; do not pass plaintext.
usafeSN	string	Required for loginType=3 or legacy flow	Identity U-Safe serial number; must match the Programming Station.
sign	string	Required for U-Safe authentication	Client signature over the challenge returned by `/preLogin`.
usafeCert	string	No	Compatibility field; not directly used by the current controller branch.
clientPcMessage	string	No	JSON object string containing client information.

5.11.4 Notes

- **Login method**: Follow the authentication flow corresponding to the method returned by `/queryLoginType`.
- **Passwordless login**: `NO_PASSWORD` (`loginType=1`) does not require a password.
- **Password login**: `PASSWORD` (`loginType=2`) requires an RSA-encrypted password.
- **U-Safe login**: `IDENTITY_USAFE` (`loginType=3`) requires a prior `/preLogin` call and a client signature.
- **Session token**: The returned `token` authenticates the Programming Station session and must not be confused with a business operation's `uuid` or `traceId`.

5.11.5 Request Example

```
{
  "factoryId": 20,
  "stationId": "STATION-001",
  "password": "<RSA-encrypted password>",
  "clientPcMessage": "{\"hostname\":\"PS-01\"}"
}
```

5.11.6 Response Parameters

This API uses the standard JSON response format. `respData` contains the following fields:

Field	Type	Description
stationId	string	Programming Station code
token	string	Programming Station session token
factoryId	integer (int64)	Factory ID
partnerCode	string	Tenant code

5.11.7 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": {
    "stationId": "STATION-001",
    "token": "<station session token>",
    "factoryId": 20,
    "partnerCode": "<tenant code>"
  },
  "respTime": 1792108800
}
```

5.11.8 Main Error Codes

Error Code	Description
8012	Programming Station already logged in
8014	Incorrect password
8016	Programming Station does not exist
8017	Unsupported login method
8020 / 8021	U-Safe does not exist or does not match
8036	Invalid cloud factory ID

See the appendix for other error codes.

5.12 Programming Station Logout

This API removes the current Programming Station session from the cache to log the station out.

5.12.1 Request Method

`POST`

5.12.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/logout

5.12.3 Request Body

This API should include the following header to identify the current Programming Station and factory context:

Parameter	Required / Condition	Description
x-snbps-session	Required for an effective logout.	Programming Station session `token` returned by `/login`

5.12.4 Request Parameters

This API requires no request parameters.

5.12.5 Notes

- **Session identification**: A valid `x-snbps-session` is required to identify and clear the corresponding Programming Station session.
- **Validation**: This API is not on the mandatory `UkeyPermissionFilter` validation path, but a valid session token is still required to log out effectively.
- **Logout result**: If the Programming Station context cannot be identified, do not assume that the specified station session was cleared.

5.12.6 Request Example

```
curl -X POST "$BASE_URL/api/iot-factory-edge-service/logout" \
  -H "x-snbps-session: $SESSION"
```

5.12.7 Response Parameters

This API uses the standard JSON response format; `respData` is `null`.

5.12.8 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": null,
  "respTime": 1792108800
}
```

5.13 Query Batches by Order Number

This API queries running batches for a specified order number in the current factory.

5.13.1 Request Method

`GET`

5.13.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/v1/queryBatchByOrderNumber

5.13.3 Request Body

In addition to the common request headers, this API requires the following header:

Parameter	Mandatory	Description
x-snbps-session	Yes	Programming Station session `token` from `/login`

5.13.4 Request Parameters

This API accepts the following query parameter:

Field	Type	Mandatory	Description
orderNumber	string	Yes	Factory order number; it cannot be empty or whitespace only

5.13.5 Notes

- **Query scope**: Only batches in `RUNNING` status for the specified order number in the current factory are queried.
- **Batch assignment**: This API only queries batches; it does not automatically assign the returned batch to the current Programming Station.

5.13.6 Request Example

```
GET {BASE_URL}/api/iot-factory-edge-service/v1/queryBatchByOrderNumber?orderNumber=ORDER-001
```

5.13.7 Response Parameters

This API uses the standard JSON response format. `respData` is an array of batch information, or an empty array `[]` if no batch matches. Each object contains:

Field	Type	Description
batchId	integer (int64)	Batch ID
batchName	string	Batch name

5.13.8 Response Example


```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": [
    {
      "batchId": 10,
      "batchName": "BATCH-001"
    }
  ],
  "respTime": 1792108800
}
```


5.13.9 Main Error Codes

Error code	Description
8039	Order number is empty
8050	Authentication failed

See the appendix for other error codes.

5.14 Assign Programming Station Batch

This API assigns a specified batch to the currently logged-in Programming Station and synchronizes the station–batch association.

5.14.1 Request Method

`POST`

5.14.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/assignBatch

5.14.3 Request Headers

In addition to the common request headers, this API requires the following header:

Parameter	Mandatory	Description
x-snbps-session	Yes	Programming Station session `token` from `/login`

5.14.4 Request Body

The request `body` contains:

Field	Type	Mandatory	Description
batchId	integer (int64)	Yes	Batch ID to assign. It must belong to the current factory, be unexpired, and be running.

5.14.5 Notes

- **Batch requirements**: Only an unexpired, running batch in the current factory can be assigned.
- **Assignment restriction**: A Programming Station already assigned through Edge management cannot be reassigned through this API if overriding the association is prohibited.
- **Repeated assignment**: Assigning the same batch to a station already assigned to it returns `8011`.

5.14.6 Request Example

```
{
  "batchId": 10
}
```

5.14.7 Response Parameters

This API uses the standard JSON response format. `respData.batchInfo.batches` is an array of batch details with the same structure as the batch details returned by `/queryPPCBatchInfo`.

5.14.8 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": {
    "batchInfo": {
      "batches": [
        {
          "batchId": "10",
          "batchName": "BATCH-001",
          "programmingType": "SE",
          "bindType": 2
        }
      ]
    }
  },
  "respTime": 1792108800
}
```

5.14.9 Main Error Codes

Error code	Description
8009	Batch has expired
8010	Batch is closed
8011	Programming Station is already assigned to this batch
8013	Data parsing failed
8016	Programming Station does not exist
8032	Batch does not exist

Error code	Description
8035	Conflict with an assignment made in Edge management
8050	Authentication failed

See the appendix for other error codes.

5.15 Query Programming Station Batch Information

This API queries the batches associated with a Programming Station, programming scripts, and server load information.

5.15.1 Request Method

`GET`

5.15.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/queryPPCBatchInfo

5.15.3 Request Headers

In addition to the common request headers, this API requires the following header:

Parameter	Mandatory	Description
x-snbps-session	Yes	Programming Station session `token` from `/login`

5.15.4 Request Parameters

This API accepts the following query parameters:

Field	Type	Mandatory	Description
usafeSN	string	No	For legacy U-Safe Programming Station queries
batchId	integer (int64)	No	When positive, queries the specified running batch. If neither parameter is provided, queries by the station in the current session.

5.15.5 Notes

- **Query and assignment**: This API only queries batch information; it does not perform `/assignBatch`.
- **Programming type**: `programmingType` distinguishes programming types such as `MCU`, `SE`, and S`E_MCU`. Use the value in the actual batch.
- **Enum values**: Follow the `ProgrammingType` enum and the actual API response. `MATTER` and `SE_MATTER` in older comments must not be used directly as request enum values.

5.15.6 Request Example

```
GET {BASE_URL}/api/iot-factory-edge-service/queryPPCBatchInfo
```

5.15.7 Response Parameters

This API uses the standard JSON response format and mainly returns the station's associated batches and server load information.

Field	Type	Description
usafeSN	string	U-Safe serial number
batchInfo.batches	array	Array of batches associated with the Programming Station; may be empty if no batch is available
serverLoadInfo.pressureLevel	integer	Server load level
serverLoadInfo.pressurePercentage	number	Server load percentage

5.15.8 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": {
    "batchInfo": {
      "batches": [
        {
          "batchId": "10",
          "batchName": "BATCH-001",
          "total": 100,
          "produced": 10,
          "toBeProduced": 90,
          "programmingType": "SE",
          "scriptName": "burn-1.0",
          "scriptMd5": "<md5>",
          "scriptUrl": "<script download URL>"
        }
      ]
    },
    "serverLoadInfo": {
      "pressureLevel": 1,
      "pressurePercentage": 10.0
    }
  },
  "respTime": 1792108800
}
```

5.15.9 Batch Detail Fields

Each object in `batchInfo.batches` contains batch, chip, programming script, and production progress information, including:

Field	Type	Description
batchId	string	Batch ID
batchName	string	Batch name

Field	Type	Description
chips	object[]	Chip information array; `chipPart`, `chipVendor`, and `objectid` are strings
env	string	Environment information
extData	string	Extended data
mark	string	Batch marker
notes	string	Batch notes
orderNo	string	Order number
produced	integer	Quantity produced
toBeProduced	integer	Quantity remaining
total	integer	Total batch quantity
productName	string	Product name
version	string	Product version
region	string	Region
secureDebugReopen	integer	Secure debug switch
supportProtocols	string[]	Supported protocol list
scriptName	string	Programming script name
scriptMd5	string	Programming script MD5 digest
scriptUrl	string	Programming script download URL
vid	string	Matter VID
pid	string	Matter PID
bindType	integer	Assignment method: 1, Edge management; 2, manual client assignment
programmingType	string	Programming type, such as `MCU`, `SE`, or `SE_MCU`; choose the programming flow from the actual response value

5.15.10 Main Error Codes

Error code	Description
8013	Data parsing failed
8015	Programming Station has no assigned batch

Error code	Description
8016	Programming Station does not exist
8050	Authentication failed

See the appendix for other error codes.

5.16 Get Dynamic Programming Data

This API generates device programming data for the batch assigned to the current Programming Station and creates a production record for this programming operation.

5.16.1 Request Method

`GET`

5.16.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/getProgrammingData

5.16.3 Request Headers

In addition to the common request headers, this API requires the following header:

Parameter	Mandatory	Description
x-snbps-session	Yes	Programming Station session `token` from `/login`

5.16.4 Request Parameters

This API accepts the following query parameters:

Field	Type	Mandatory / condition	Description
uuid	string	No	Trace ID for a single programming operation. If omitted, `getProgrammingData` generates one; retain and reuse it in later calls.
ppcliVersion	string	No	Client script version
usafeSN	string	Legacy flow compatibility	Since 3.1.1, no longer a primary parameter in the new flow. Station information takes precedence when a session station exists.
chipIndex	integer	No	No longer used since 3.1.1.

5.16.5 Notes

- **Programming trace**: Omit `uuid` on the initial call. After obtaining the returned `uuid`, retain and reuse it in subsequent `/uploadRequestData`, `/executeApdu`, or programming result requests.
- **SE programming**: In an SE-only scenario, a programming record is created even if the returned `bundles` array is empty.
- **File download**: Use the `downloadUrl` in the actual response for large files.

- **Sensitive material**: Sensitive production material is not sent as plaintext in a `bundle`.

5.16.6 Request Example

```
GET {BASE_URL}/api/iot-factory-edge-service/getProgrammingData?ppcliVersion=1.0.0
```

5.16.7 Response Parameters

This API uses the standard JSON response format. `respData` is a batch Bundle object containing:

Field	Type	Description
batchId	string	Batch ID
batchName	string	Batch name
uuid	string	Trace ID for this programming operation
bundles	object[]	Data entries for this programming operation; may be an empty array in an SE-only scenario

5.16.8 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": {
    "batchId": "10",
    "batchName": "BATCH-001",
    "uuid": "example-programming-trace",
    "bundles": []
  },
  "respTime": 1792108800
}
```

5.16.9 Main Error Codes

Error code	Description
8013	Bundle data parsing failed
8030	No available Programming Station batch
8050	Authentication failed

See the appendix for other error codes.

5.17 Get Batch Metadata

This API returns complete batch metadata and download URLs for firmware and other files so the client can prepare device programming.

5.17.1 Request Method

`GET`

5.17.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/getMetadata

5.17.3 Request Headers

The following header is recommended to resolve the current Programming Station and factory context:

Parameter	Mandatory / condition	Description
x-snbps-session	Recommended	Programming Station session `token` from `/login`

This API is not on the mandatory `UkeyPermissionFilter` validation path.

5.17.4 Request Parameters

This API accepts the following query parameters:

Field	Type	Mandatory / condition	Description
uuid	string	No	Trace ID for a single programming operation. If omitted, a later `getProgrammingData` call may generate one; retain it for later calls.
ppcliVersion	string	No	Client script version
usafeSN	string	Legacy flow compatibility	Since 3.1.1, no longer a primary parameter in the new flow. Station information takes precedence when a session station exists.
chipIndex	integer	No	No longer used since 3.1.1.

5.17.5 Notes

- **Metadata retrieval**: Returns the complete batch Bundle, performs the necessary metadata processing, and removes specified key material.
- **Production record**: This API does not replace `/getProgrammingData` in creating a programming record. The `uuid` in its response does not prove that a new production session was created.
- **Programming Station context**: A valid `x-snbps-session` can be used to resolve the current Programming Station.

5.17.6 Request Example

```
GET {BASE_URL}/api/iot-factory-edge-service/getMetadata?usafeSN=USAFE-001
```

5.17.7 Response Parameters

This API uses the standard JSON response format. `respData` is a batch Bundle object containing:

Field	Type	Description
batchId	string	Batch ID
batchName	string	Batch name

Field	Type	Description
uuid	1string	Programming trace ID in the response; does not prove that a production record was created
bundles	object[]	Complete batch data entries

5.17.8 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": {
    "batchId": "10",
    "batchName": "BATCH-001",
    "uuid": "example-programming-trace",
    "bundles": []
  },
  "respTime": 1792108800
}
```

The empty `bundles` array in the example illustrates the response structure only; it does not mean every batch returns an empty array.

5.17.9 Main Error Codes

Error code	Description
8013	Data parsing failed
8030	No available Programming Station batch

See the appendix for other error codes.

5.18 Get Full Programming Data (Compatibility API)

This is the legacy API for retrieving a complete Bundle. Its service method is marked `@Deprecated`. New clients should use `/getMetadata` and `/getProgrammingData`.

5.18.1 Request Method

`GET`

5.18.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/getPPData

5.18.3 Request Headers

In addition to the common request headers, this API requires the following header:

Parameter	Mandatory	Description
x-snbps-session	Yes	Programming Station session `token` from `/login`

5.18.4 Request Parameters

This API accepts the following query parameters:

Field	Type	Mandatory / condition	Description
uuid	string	No	Trace ID for a single programming operation. In the new flow, `getProgrammingData` generates one if omitted; retain it for later calls.
ppcliVersion	string	No	Client script version
usafeSN	string	Legacy flow compatibility	Since 3.1.1, no longer a primary parameter in the new flow. Station information takes precedence when a session station exists.
chipIndex	integer	No	No longer used since 3.1.1.

5.18.5 Notes

- **Batch query**: This API identifies the batch by `usafeSN`, generates a new `uuid`, and returns the processed complete programming data.
- **No available batch**: The response may be `null` if no usable batch is found.
- **Compatibility**: This API supports legacy clients. New clients should use `/getMetadata` and `/getProgrammingData`.

5.18.6 Request Example

```
GET {BASE_URL}/api/iot-factory-edge-service/getPPData?usafeSN=USAFE-001
```

5.18.7 Response Parameters

This API uses the standard JSON response format. `respData` is a batch Bundle object containing:

Field	Type	Description
batchId	string	Batch ID
batchName	string	Batch name
uuid	string	Programming trace ID returned in this response
bundles	object[]	Processed complete programming data entries

5.18.8 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": {
    "batchId": "10",
    "batchName": "BATCH-001",
```

The empty `bundles` array in the example illustrates the response structure and is not the actual content for every batch. In an SE-only scenario, `/getProgrammingData` may return an empty `bundles` array.

5.18.9 Common Bundle Structure

Each Bundle object in the `bundles` array contains:

Field	Type	Description
type	string	Bundle type
section	string	Section to which the Bundle belongs
ownerId	string	Associated object ID
authToken	string	Authentication token
authCert	string	Authentication certificate
authSign	string	Authentication signature
items	object[]	List of Bundle data entries

Each entry in the `items` array contains:

Field	Type	Description
itemId	string	Entry ID
key	string	Entry key
objectId	string	Object ID
parentId	string	Parent object ID
fieldType	string	Field type
attribute	JSON object	Entry attributes; structure depends on field type

Bundle fields and content may differ by batch. Clients should follow the actual server response and download URLs; do not treat the empty `bundles` example as actual data for every batch.

5.18.10 Main Error Codes

Error code	Description
8050	Authentication failed

Other exceptional behavior of this legacy API depends on the old flow. See the appendix for specific error codes.

5.19 Submit Intermediate Programming Request Data

This API reports chip execution results for batch Bundle entries and lets the corresponding processor generate the next stage of programming data.

5.19.1 Request Method

`POST`

5.19.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/uploadRequestData

5.19.3 Request Headers

In addition to the common request headers, this API requires the following header:

Parameter	Mandatory	Description
x-snbps-session	Yes	Programming Station session `token` from `/login`

5.19.4 Request Body

The request `body` contains:

Field	Type	Mandatory / condition	Description
batchId	string	Yes	Batch ID in the current factory
uuid	string	Required by business rules	Programming trace ID returned by `/getProgrammingData`; IDRule number allocation requires a valid, stable `traceId`
bundles	object[]	As needed	Reported Bundle data; if empty, the current implementation returns `respData=null`
bundles[].type	string	Per template	Bundle type
bundles[].section	string	Per template	Section to which the Bundle belongs
bundles[].deviceId	string	Recommended for IDRule	Stable device identifier. If omitted, the service attempts to resolve it from the record for `uuid`. Do not change it after initial number allocation.
bundles[].items	object[]	Required for a nonempty Bundle	List of Bundle entries; empty `items` or `null` entries are not allowed
bundles[].items[].key	string	Yes	Nonempty entry key
bundles[].items[].response	object[]	Per item protocol	List of results from the previous step; fields are defined by `fieldType` and the corresponding processor protocol

5.19.5 Notes

- **Purpose**: This API submits intermediate programming data; it does not report the final programming result.
- **Data structure**: Populate `items[].key` and `response` according to the actual batch template and processor protocol. Do not use the placeholders in the example directly.
- **Device identifier**: In an IDRule scenario, reuse a stable device identifier for the same device. Do not change `deviceId` in later programming steps after the first number allocation.
- **Empty data**: If `bundles` is empty, the current implementation returns `respData=null`.

5.19.6 Request Example

```
{
  "batchId": "10",
  "uuid": "<uuid returned by getProgrammingData>",
  "bundles": [
    {
      "section": "<template section>",
      "deviceId": "DEVICE-001",
      "items": [
        {
          "key": "<template item key>",
          "response": [
            {
              "<protocol field>": "<actual execution result>"
            }
          ]
        }
      ]
    }
  ]
}
```

5.19.7 Response Parameters

This API uses the standard JSON response format. `respData` contains:

Field	Type	Description
batchId	string	Batch ID
batchName	string	Batch name
uuid	string	Programming trace ID for this operation
bundles	object[]	Next-stage programming data entries generated by the corresponding processor

5.19.8 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": {
    "batchId": "10",
```

5.19.9 Main Error Codes

Error code	Description
8002	Invalid request parameters
8003	Data processing failed
8032	Batch does not exist
8040–8046	Certificate or key agreement error
8065	Invalid IDRule rule
8066	Number range exhausted
8067	State conflict
8068	Stable device identifier is missing
8050	Authentication failed

See the appendix for other error codes.

5.20 Report Programming Result

This API closes an existing device programming record and creates or updates the device profile, while remaining compatible with legacy clients.

5.20.1 Request Method

`POST`

5.20.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/programmingResult

5.20.3 Request Headers

In addition to the common request headers, this API requires the following header:

Parameter	Mandatory	Description
x-snbps-session	Yes	Programming Station session `token` from `/login`

5.20.4 Request Body

The request `body` contains:

Field	Type	Mandatory / condition	Description
uuid	string	Yes (integration contract)	traceId of an existing programming record
batchId	integer (int64)	Recommended	Used as additional context when the programming record lacks batch information
result	boolean	Explicitly recommended	`true` means success; `false` or `null` is treated as failure
failureReason	string	No	Reason for programming failure
deviceProfile	object	No	Actual CPS programming data. Fields are described under `deviceRecord` in the Production Data Upload API, including `deviceId`, `firmwareVersion`, `seid`, `cplc`, `matterData`, `factoryData`, and `customData`.
changeKeys	object	No	Compatibility field. Accepts `null`, an empty string, or a JSON object; other types return `8002`. The current programming result flow does not use this field further.

5.20.5 Notes

- **Record handling**: If the programming record does not exist or the current factory batch cannot be identified, the current method skips device profile creation and returns normally.
- **Duplicate report**: A repeated report with the same status, or a report for a record already in success status, is not processed again.
- **Result confirmation**: `respCode="8000"` does not necessarily mean a new device profile was created. Confirm through a device query API.
- **Custom data**: `customData` values must be strings and may contain at most 50 entries.
- **Compatibility**: The newer unified production data upload flow can report programming results through `/v1/deviceData/upload`.

5.20.6 Request Example

```
{
  "uuid": "<current programming uuid>",
  "batchId": 10,
  "result": true,
  "deviceProfile": {
    "deviceId": "DEVICE-001",
    "firmwareVersion": "1.2.3",
    "customData": {
      "fixture": "A01"
    }
  }
}
```

5.20.7 Response Parameters

This API uses the standard JSON response format. On success, `respData` is `null`.

5.20.8 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": null,
  "respTime": 1792108800
}
```

5.20.9 Main Error Codes

Error code	Description
8002	Invalid request parameters
8007	Data processing or storage error
8037	Device profile creation failed
8050	Authentication failed

See the appendix for other error codes.

5.21 SE APDU Stepwise Programming

This API performs stepwise APDU programming of an SE device. The client executes the APDU commands returned by the server in order and submits their results to obtain the next commands. A `nextStep` value of `EOF` means the programming flow is complete.

5.21.1 Request Method

`POST`

5.21.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/executeApdu

5.21.3 Request Headers

In addition to the common request headers, this API requires the following header:

Parameter	Mandatory	Description
x-snbps-session	Yes	Programming Station session `token` from `/login`

5.21.4 Request Body

The request `body` contains:

Field	Type	Mandatory / condition	Description
batchId	string	Yes	SE configuration batch ID in the current factory. It cannot be omitted in reliance on the station assignment alone.
cplc	string	Provide for SE integration	CPLC protocol value of the SE device, usually 84 characters, used for the SE device record
seId	string	Provide for SE integration	SE identifier. The field name is case-sensitive: use `seId`.
nfcUid	string	For power keys	Protocol hex value of the device physical identifier; `nfcId` is accepted as an input alias. It must remain identical on initial and subsequent steps.
programmingTraceId	string	Optional for BOF; required for other steps	Programming trace ID; `uuid` is accepted as an input alias. In an `SE_MCU` flow, reuse the Matter session `traceId`. The response returns only `programmingTraceId`.
currentStep	string	Yes	Use BOF on the initial call. On subsequent calls, use `nextStep` from the previous response unchanged; do not construct a step cursor.
commandResults	object	Required for subsequent steps	Previous APDU execution result. Omit for BOF. For subsequent steps, `succeed` must be `true` and `results` must correspond to the previous commands.
commandResults.succeed	boolean	Required for subsequent steps	Whether the previous step succeeded. `false` stops the flow and returns `8053`.
commandResults.results	object[]	For subsequent steps, follow the previous response	List of APDU execution results. Each entry contains `index` (integer) and `command`, `checker`, and `result` (strings). `index` must numerically match the string sequence number of the previous command.

5.21.5 Notes

- **Initial call**: Start programming with `currentStep="BOF"` and save `programmingTraceId` from the response for later steps.
- **Command execution**: The client must execute APDU commands in the order of `commands` and submit the actual results of the previous batch in the next request.
- **Step control**: Use the previous response's `nextStep` unchanged in the next request. Do not parse or modify the server step cursor. Stop at `EOF`; do not submit `EOF` again.
- **Session recovery**: `8054` means the current session cannot continue. Restart from `BOF`; do not automatically replay earlier APDU commands.
- **Session lifetime**: The current SE session lifetime is defined by a server configuration constant.
- **Power key**: `nfcUid` (or its input alias `nfcId`) must stay the same on the initial and subsequent steps. Do not use `seId` as the NFC UID.

5.21.6 Request Example

Use `BOF` on the initial call:

```
{
  "batchId": "10",
  "cplc": "<device CPLC>",
  "seId": "<device SEID>",
  "currentStep": "BOF"
}
```

5.21.7 Response Parameters

This API uses the standard JSON response format. `respData` contains:

Field	Type	Description
nextStep	string	Next server step cursor; EOF means the flow is complete
programmingTraceId	string	Trace ID of this SE programming operation
commands	object[]	List of APDU commands to execute in this step
commands[].index	string	APDU command sequence number
commands[].checker	string	APDU response checker
commands[].command	string	APDU hex command to execute

5.21.8 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": {
    "nextStep": "<server step cursor>",
    "programmingTraceId": "example-programming-trace",
    "commands": [
      {
        "index": "1",
        "checker": "<server-provided checker>",
        "command": "<server-provided APDU hex>"
      }
    ]
  },
  "respTime": 1792108800
}
```

5.21.9 APDU Subsequent-Step Request Example

After executing the previous batch of APDU commands, the client submits the results using `programmingTraceId` and `nextStep` returned by the server.

```
{
  "batchId": "10",
  "programmingTraceId": "<programmingTraceId from previous response>",
  "currentStep": "<nextStep from previous response>",
  "cplc": "<same device CPLC>",
  "seId": "<same device SEID>",
  "commandResults": {
    "succeed": true,
    "results": [
      {
        "index": 1,
        "command": "<original command>",
        "checker": "<original checker>",
        "result": "<actual card response hex>"
      }
    ]
  }
}
```

For a power key, include the same `nfcUid` (or input alias `nfcId`) in the initial request and every subsequent request. Do not download the raw production key file separately to the client. Edge prepares the relevant production material and returns it through the APDU commands for the current step. After successful programming reaches `EOF`, use the existing production count and device profile handling flow. If command execution on the card clearly failed, do not report `commandResults.succeed` as `true`.

5.21.10 Power Key Device Binding Rules

- **Device identifiers**: The current power key flow requires both `nfcUid` (input alias `nfcId` is supported) and `seId`.
- **Data format**: Pass both identifiers as even-length hexadecimal strings as required by the device protocol. The current allocation endpoint requires non-empty values of at most 64 characters and converts them to uppercase.
- **Alias usage**: Do not submit `nfcUid` and `nfcId` with different values. Keep the same nfcUid and device identity in the initial call and every continuation request.
- **Material binding**: Production materials are stored against the device binding of factory, batch, NFC UID, and SEID. Even if a device changes `traceId` or resends `BOF`, previously bound materials are reused first.
- **Binding conflict**: If an NFC UID or SEID is already bound but the other device identifier differs, the API returns `8073`.
- **Material exhaustion**: If no production materials are available, the API returns `8072`.
- **Historical data**: If a batch still has an allocation state from legacy Redis or production records that has not been migrated, the API may also return `8073`. Resending BOF does not allocate new materials.

5.21.11 Main Error Codes

Error code	Description
8006	Batch or programming record is missing
8047	SE configuration is missing
8048	Data parsing failed
8049	CPLC-related error
8052	Step execution result does not match
8053	Previous step failed

Error code	Description
8054	Session cannot continue; restart required
8055	APDU processing failed
8056	Configuration decryption failed
8070–8074	Production data, material allocation, or configuration error
8050	Authentication failed

See the appendix for other error codes.

5.22 Get Programming Script URL

This API obtains the programming script download URL for a specified batch.

5.22.1 Request Method

`POST`

5.22.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/getScriptFile

5.22.3 Request Headers

Provide the following request header to resolve the current station and factory context:

Parameter	Mandatory	Description
x-snbps-session	Recommended	Station session `token` returned by `/login`

This API is not on the mandatory validation path list of `UkeyPermissionFilter`.

5.22.4 Request Body

The request `body` contains the following parameters:

Field	Type	Mandatory / condition	Description
batchId	integer (int64)	Required when querying by batch	Batch ID
productId	integer (int64)	No	Compatibility parameter for product-based queries
chipPart	string	As needed	Chip model
chipVendor	string	As needed	Chip vendor

5.22.5 Notes

- **Download URL**: The client should download the programming script from the `fileUrl` returned by the API.
- **Script verification**: An HTTP success response does not mean the client has completed local script download and verification.

5.22.6 Request Example

```
{
  "batchId": 10
}
```

5.22.7 Response Parameters

This API uses the standard JSON response format. `respData` contains the following field:

Field	Type	Description
fileUrl	string	Programming script download URL returned by the server

5.22.8 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": {
  "fileUrl": "<server-generated script download URL>"
  },
  "respTime": 1792108800
}
```

5.22.9 Main Error Codes

Error code	Description
8022	Failed to get the programming script

See the appendix for other error codes.

5.23 Download File

This API downloads a file by its local file record ID.

5.23.1 Request Method

`GET`

5.23.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/download

5.23.3 Request Headers

Calls requiring factory context should provide the following request header:

Parameter	Mandatory / condition	Description
x-snbps-session	When factory context is required	Valid station session `token` returned by `/login`

5.23.4 Request Parameters

Pass the following fields as query parameters:

Field	Type	Mandatory	Description
fileId	integer (int64)	Yes	Local file record ID
fileName	string	No	Download filename requested by the client

5.23.5 Notes

- **Response format**: On success, the API returns a binary file stream with download headers such as `Content-Disposition`. It does not use the standard `ClientR` JSON response structure.
- **Download URL**: The client can use the full download URL returned by `/getScriptFile` or `/getMetadata` rather than building it manually.
- **Session requirement**: Provide a valid `x-snbps-session` when factory context is required.

5.23.6 Request Example

```
curl -L "$BASE_URL/api/iot-factory-edge-service/download?fileId=100&fileName=burn.zip" \
  -H "x-snbps-session: $SESSION" \
  -o burn.zip
```

5.23.7 Response Parameters

On success, the API returns a binary file stream, not JSON `respData`.

5.23.8 Main Error Codes

Error code	Description
8022	File record does not exist
8023	File download failed

See the appendix for other error codes.

5.24 Upload Device Event Logs

This API receives event logs produced during device programming, filters them by configured priority, and reports them through MQTT. If sending fails, it saves a compensation record.

5.24.1 Request Method

`POST`

5.24.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/v1/eventLog/upload

5.24.3 Request Headers

In addition to the common request headers, provide the following header:

Parameter	Mandatory	Description
x-snbps-session	Yes	Station session `token` returned by `/login`

5.24.4 Request Body

The request `body` contains the following parameters:

Field	Type	Mandatory / condition	Description
programmingTraceId	string	Yes (non-null)	Programming trace ID; `uuid` is also accepted as an input alias
events	object[]	Yes (non-null)	Event log list; an empty array may be ignored
events[].eventName	string	Recommended	Event name
events[].timestamp	string	Recommended	Client event time in the format defined by the event protocol
events[].source	string	Recommended	Event source
events[].priority	string	Recommended	Event priority as a numeric string from 1 to 5; larger values mean higher priority. Empty or invalid values are filtered out.
events[].status	string	Recommended	Event status: `success` or `failure`
events[].content	string	No	Log content; the current implementation truncates it after 256 characters.

5.24.5 Notes

- **Event filtering**: The server filters events by the configured priority threshold. Events below the threshold or with invalid priority are not reported further.
- **Log synchronization**: A successful HTTP request means only that the API received and processed the request; it does not mean every event has been synchronized to the cloud.
- **Failure compensation**: If MQTT sending fails, the server saves a compensation record.

5.24.6 Request Example

```
{
  "programmingTraceId": "<current programming traceId>",
  "events": [
    {
      "eventName": "ProgrammingFinished",
      "timestamp": "2026-10-16T10:00:00+08:00",
      "source": "PS",
      "priority": "5",
      "status": "success",
      "content": "Burn completed"
```

5.24.7 Response Parameters

This API uses the standard JSON response format. On success, `respData` is `null`.

5.24.8 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": null,
  "respTime": 1792108800
}
```

5.24.9 Main Error Codes

Error code	Description
8050	Authentication failed

Request parameter binding or validation may also produce errors. See the appendix for other error codes.

5.25 Generate Matter Verifier

This API calls the Matter SDK to generate a Matter passcode, Salt, and Verifier.

5.25.1 Request Method

`POST`

5.25.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/generateVerifier

5.25.3 Request Headers

The current `application.yml` does not exclude this API from platform authentication. If platform authentication is enabled in the deployment, provide the following header:

Parameter	Mandatory / condition	Description
x-snbps-auth-token	When platform authentication is enabled	Platform authentication token, distinct from the station session token `x-snbps-session`

5.25.4 Request Parameters

This API requires no request parameters.

5.25.5 Notes

- **Generation parameters**: The current implementation fixes `saltLen` at 32 bytes and `iterations` at 1000. They cannot be overridden in the request body.
- **Runtime dependency**: The corresponding native Matter SDK must be deployed on the server.
- **Status interpretation**: The outer `respCode` and `respData.status` have different meanings. The former indicates the API business

response status; the latter is the status returned by the Matter SDK.

- **Example data**: Numeric values in the response example illustrate field types only.

5.25.6 Request Example

```
POST {BASE_URL}/api/iot-factory-edge-service/generateVerifier
```

5.25.7 Response Parameters

This API uses the standard JSON response format. `respData` contains the following fields:

Field	Type	Description
status	integer	Status code returned by the Matter SDK
passcode	integer	Matter passcode
salt	string	Base64-encoded Salt
verifier	string	Base64-encoded Verifier

5.25.8 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": {
    "status": 0,
    "passcode": 20202021,
    "salt": "<Base64 salt>",
    "verifier": "<Base64 verifier>"
  },
  "respTime": 1792108800
}
```

5.26 Generate Matter Onboarding Payload

This API calls the Matter SDK with specified Matter parameters to generate a QR code onboarding payload and manual pairing code.

5.26.1 Request Method

`POST`

5.26.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/generateOnboardingData

5.26.3 Request Headers

The current `application.yml` does not exclude this API from platform authentication. If platform authentication is enabled in the deployment, provide the following header:

Parameter	Mandatory / condition	Description
x-snbps-auth-token	When platform authentication is enabled	Platform authentication token, distinct from the station session token `x-snbps-session`

5.26.4 Request Parameters

The API accepts the following query or form parameters, not a JSON request body.

Field	Type	Mandatory	Description
vid	integer	Yes	Matter Vendor ID (VID) as a decimal integer, unlike the hexadecimal text used by the Matter payload recovery API
pid	integer	Yes	Matter Product ID (PID) as a decimal integer
passcode	integer	Yes	Matter passcode
commissioningFlow	integer	Yes	Matter SDK commissioning flow parameter
discoveryMode	integer	Yes	Matter SDK device discovery mode parameter

5.26.5 Notes

- **Parameter transport**: Although this API uses `POST`, all five parameters are bound with `@RequestParam`. Pass them through URL query or form parameters, not a JSON request body.
- **Parameter validation**: The underlying Matter SDK determines the valid range for each parameter.
- **VID/PID format**: This API uses decimal integers, unlike the hexadecimal strings used by `/v1/matter/onboarding-payloads/recover`.

5.26.6 Request Example

```
POST {BASE_URL}/api/iot-factory-edge-service/
generateOnboardingData?vid=65522&pid=32769&passcode=20202021&commissioningFlow=0&discoveryMode=2
```

5.26.7 Response Parameters

This API uses the standard JSON response format. `respData` contains the following fields:

Field	Type	Description
status	integer	Status code returned by the Matter SDK
discriminator	integer	Matter onboarding `Discriminator`
qrcode	string	Matter QR code onboarding payload
manualcode	string	Matter manual pairing code; note that the field name `manualcode` is all lowercase

5.26.8 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": {
    "status": 0,
    "discriminator": 3840,
    "qrcode": "<SDK-generated QR code payload>",
    "manualcode": "<SDK-generated manual pairing code>"
  },
  "respTime": 1792108800
}
```

5.27 Generate Matter Unique ID

This API calls the Matter SDK to generate a Matter Unique ID.

5.27.1 Request Method

`POST`

5.27.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/generateUniqueId

5.27.3 Request Headers

The current `application.yml` does not exclude this API from platform authentication. If platform authentication is enabled in the deployment, provide the following header:

Parameter	Mandatory / condition	Description
x-snbps-auth-token	When platform authentication is enabled	Platform authentication token, distinct from the station session token `x-snbps-session`

5.27.4 Request Parameters

This API requires no request parameters or request body.

5.27.5 Notes

- **Generation**: The Matter SDK generates the Unique ID; the client supplies no generation parameters.
- **Status interpretation**: A successful Matter SDK status is usually `0`, while the outer API business success code is `8000`. They have different meanings.

5.27.6 Request Example

```
POST {BASE_URL}/api/iot-factory-edge-service/generateUniqueId
```

5.27.7 Response Parameters

This API uses the standard JSON response format. `respData` contains the following fields:

Field	Type	Description
status	integer	Status code returned by the Matter SDK
result	string	Base64-encoded Unique ID

5.27.8 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": {
    "status": 0,
    "result": "<Base64 uniqueId>"
  },
  "respTime": 1792108800
}
```

5.28 Query Device Data (Legacy API, Deprecated)

This API supports device data queries by legacy clients. New clients should use `/v1/device/detail`, or `/v1/device/information/query` for station query scenarios.

5.28.1 Request Method

`POST`

5.28.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/queryDeviceData

5.28.3 Request Body

Provide the following request header to resolve the current station and factory context:

Parameter	Mandatory / condition	Description
x-snbps-session	Recommended	Station session `token` returned by `/login`

This API is not on the mandatory validation path list of `UkeyPermissionFilter`.

5.28.4 Request Body

The request `body` contains the following parameters:

Field	Type	Mandatory / condition	Description
batchId	string	Yes	Batch ID
paramKey	string	Yes	Key identifying the parameter group
paramField	string	Yes	Field to locate

Field	Type	Mandatory / condition	Description
paramValue	string	Yes	Field value identifying the device
queryKeys	string[]	Yes	See the queryable keys list (queryKeys V1) for supported keys.

5.28.5 Notes

- **Compatibility**: This API was declared deprecated in the v1.3 document and remains only for legacy clients.
- **Query source**: If the batch was deleted, a file cannot be read, or no local device profile matches, the API may query historical records or cloud data.
- **Query parameters**: `queryKeys` must follow the legacy Bundle query protocol. Do not use field definitions from the newer QueryKeys V2 directly.

5.28.6 Request Example

```
{
  "batchId": "10",
  "paramKey": "<legacy protocol key>",
  "paramField": "<legacy protocol field>",
  "paramValue": "<actual value>",
  "queryKeys": [
    "<legacy query key>"
  ]
}
```

5.28.7 Response Parameters

This API uses the standard JSON response format. `respData` contains the following fields:

Field	Type	Description
batchId	string	Batch ID
paramKey	string	Parameter Key used by the query
paramField	string	Field used by the query
paramValue	string	Field value used by the query
data	map<string,map<string,object>>	Device information in the legacy data structure

5.28.8 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": {
    "batchId": "10",
    "paramKey": "<legacy protocol key>",
    "paramField": "<legacy protocol field>",
    "paramValue": "<actual value>",
```

5.28.9 Main Error Codes

Error code	Description
8006	Batch does not exist
8003	Legacy device information query failed

See the appendix for other error codes.

5.29 Update Device Information (Legacy API, Deprecated)

This API supports updates using the legacy device data model. New clients should use `/v1/updateDevicesProfile`.

5.29.1 Request Method

`POST`

5.29.2 Request URL

{BASE_URL}/api/iot-factory-edge-service/updateDeviceInfo

5.29.3 Request Headers

Provide the following request header to resolve the current station and factory context:

Parameter	Mandatory / condition	Description
x-snbps-session	Recommended	Station session `token` returned by `/login`

This API is not on the mandatory validation path list of `UkeyPermissionFilter`.

5.29.4 Request Body

The request `body` contains the following parameters:

Field	Type	Mandatory / condition	Description
batchId	integer (int64)	Yes	Batch ID; note that this differs from the string type used by the legacy query API
paramKey	string	Yes	Key identifying the parameter group
paramField	string	Yes	Field to locate
paramValue	string	Yes	Field value identifying the device
data	map<string,map<string,string>>	Yes	Device data to update

5.29.5 Notes

- **Compatibility**: This API uses the legacy device data model. New clients should use `/v1/updateDevicesProfile`.

- **Batch handling**: If the specified batch does not exist, the current service returns directly, while the controller may still return the `data` from the request.
- **Update confirmation**: `respCode="8000"` and an echoed `data` value do not prove that data was persisted. Do not infer a successful device information update solely from the API response.

5.29.6 Request Example

```
{
  "batchId": 10,
  "paramKey": "<legacy protocol key>",
  "paramField": "<legacy protocol field>",
  "paramValue": "<actual value>",
  "data": {}
}
```

5.29.7 Response Parameters

This API uses the standard JSON response format. `respData` is the `data` object from the request, of type `map<string, map<string, string>>`. It only echoes request data; it does not represent a fresh read of persisted state.

5.29.8 Response Example

```
{
  "success": true,
  "respCode": "8000",
  "respMsg": "NoError",
  "respData": {},
  "respTime": 1792108800
}
```

6. Appendix

6.1 Appendix One: Queryable Keys (queryKeys V2)

Data item	Key
Matter commissioning: PBKDF iterations (1000)	matterData.iterations
Matter commissioning: salt length (32 bytes)	matterData.saltLength
Matter commissioning flow type (0 = standard flow)	matterData.commissioningFlows
Device discovery mode	matterData.discoverableMode
Matter Vendor ID	matterData.vid
Matter Product ID	matterData.pid
Discriminator	matterData.discriminator
Matter commissioning: rotating device ID enabled (true = enabled)	matterData.enableRotatingDeviceId

Data item	Key
Matter manual pairing code	matterData.manualCode
Matter passcode	matterData.passcode
Matter onboarding QR code	matterData.qrcode
Matter salt used by the password derivation algorithm	matterData.salt
Matter device Unique ID (Base64 encoded)	matterData.uniqueid
Matter Verifier	matterData.verifier
Matter programming device serial number	matterData.serialNumber
Matter DAC certificate MD5	matterData.dacCertMd5
Matter PAI certificate MD5	matterData.paiCertMd5
Matter CD filename	matterData.cdFileName
Matter CD file MD5	matterData.cdFileMd5
Factory data; see the OBIS product configuration UI	factoryData.vendorName
Factory data; see the OBIS product configuration UI	factoryData.hardwareVersion
Factory data; see the OBIS product configuration UI	factoryData.hardwareVersionString
Factory data; see the OBIS product configuration UI	factoryData.partNumber
Factory data; see the OBIS product configuration UI	factoryData.productUrl
Factory data; see the OBIS product configuration UI	factoryData.productLabel
Factory data; see the OBIS product configuration UI	factoryData.productName
Factory data; see the OBIS product configuration UI	factoryData.startAddress
Programming status	programmingData.programmingStatus
Programming log	programmingData.programmingLog
Programming time	programmingData.programmingTime
MCU ID in the programmed chip	chipData.chipUid
Chip Bluetooth MAC address	chipData.bleMac

Data item	Key
Firmware name (returns firmware list)	firmwareData.fwName
Firmware MD5 (returns firmware list)	firmwareData.fwMd5
Custom field	customData.<custom field>

6.1.1 Additional Query Keys

> **TIP**
> This API follows `queryKeys V2` naming. Matter data uses `matterData.manualCode`, `matterData.qrcode`, and `matterData.uniqueid`, which differ from `manualPairingCode` and `setupPayload` in the "Query Device Information" API; do not mix them.

Data item	Key	Description
SE data	seData	Returns the complete SE data object in the device profile
Query entire group	matterData / factoryData / programmingData / chipData / firmwareData / customData	A group name without a dot queries its supported fields; the exact response depends on the current implementation.

6.2 Appendix Two: Error Code List

The following table lists client business error codes used by the APIs and their descriptions.

- Each API uses only a subset of these codes. See the corresponding API description for causes and handling.
- Parameter validation, platform authentication, and framework processing may produce other error codes not listed here.
- An SDK status of 0 has a different meaning from `respCode="8000"` in the standard API response; do not confuse them.

respCode	Enum name	Description
8000	NO_ERROR	Operation succeeded (NoError)
8001	REQUEST_FAILED	Request failed (RequestFailed)
8002	PARAM_ERROR	Invalid parameter (ParamError)
8003	METHOD_ERROR	Invalid method (MethodError)
8004	AUTH_ERROR	Authentication failed (AuthError)
8005	USAFE_TYPE_ERROR	Invalid U-Safe type (USafeTypeError)
8006	DATA_NOT_FOUND	Data not found (DataNotFound)

respCode	Enum name	Description
8007	DATA_ERROR	Invalid data (DataError)
8008	NOT_ALLOWED	Operation not allowed (NotAllowed)
8009	BATCH_EXPIRED_ERROR	Batch expired
8010	BATCH_STATUS_CLOSED	Batch closed
8011	STATION_BATCH_HAS_BIND	Batch already assigned to this station
8012	STATION_HAS_LOGIN	Station already logged in (StationHasLogin)
8013	BATCH_BUNDLE_PARSE_ERROR	Failed to parse the Bundle file generated for the batch
8014	STATION_PASSWORD_ERROR	Incorrect station password (StationPasswordError)
8015	STATION_NOT_BIND_BATCH	Station has no assigned batch
8016	STATION_NOT_EXIST	Station does not exist
8017	UNKNOWN_LOGIN_TYPE	Unknown login type
8018	FACTORY_NOT_EXIST	Factory does not exist
8019	FACTORY_STATUS_LOCKED	Factory is locked
8020	USAFE_NOT_FOUNT	U-Safe not found
8021	STATION_USAFE_NOT_MATCH	U-Safe does not match station ID
8022	FETCH_BURN_SCRIPT_ERROR	Failed to get programming script
8023	DOWNLOAD_BURN_SCRIPT_ERROR	Failed to download programming script
8024	PRELOGIN_AUTH_ERROR	Pre-login authentication failed
8025	HSM_CERT_SYNC_REQUIRED	HSM authentication certificate not found. Edit factory HSM information in the cloud to trigger certificate synchronization.
8026	SIGN_VERIFY_ERROR	Signature verification failed
8027	BATCH_HAS_DONE	Batch completed
8028	BATCH_FILE_NOT_EXIST	Operation failed: product batch file does not exist
8029	BATCH_IMPORT_FAILED	Failed to import batch
8030	PS_BURN_STATION_RECORDING_FINISHED	U-Safe SN has completed programming or is not assigned to a batch
8031	USAFE_UNAVAILABLE	U-Safe unavailable

respCode	Enum name	Description
8032	BATCH_NOT_FOUND	Batch not found
8033	BATCH_DOWNLOAD_STATUS_NOT_WAIT	Batch download status is not pending
8034	DEVICES_PROGRAMMER_DATA_UPLOAD_ERROR	Failed to upload device adapter data
8035	STATION_HAS_EDGE_BIND	Station already assigned a batch from the Edge Service
8036	CLOUD_FACTORY_ID_ERROR	Invalid cloud factory ID
8037	DEVICE_PROFILE_GENERATE_ERROR	Failed to generate device profile
8038	UNSUPPORTED_CERTIFICATE_FORMAT	Unsupported certificate format
8039	ORDER_NUMBER_CANNOT_BE_EMPTY	Order number must not be empty
8040	CERT_CHAIN_VERIFY_FAILED	Silicon Labs certificate chain verification failed
8041	PUBKEY_SIGNATURE_VERIFY_FAILED	Chip temporary public key signature verification failed
8042	ECDH_KEY_AGREEMENT_FAILED	ECDH key agreement failed
8043	SESSION_KEY_NOT_FOUND	Session key does not exist or has expired
8044	KEY_WRAP_FAILED	Key encapsulation failed
8045	DAC_PUBKEY_SIGNATURE_VERIFY_FAILED	DAC public key signature verification failed
8046	BUNDLE_PUBLIC_KEY_NOT_FOUND	Bundle public key material for the specified keyCode not found
8047	SE_PROFILE_NOT_FOUND	SE Profile for the batch not found
8048	SE_PROFILE_PARSE_ERROR	Failed to parse SE Profile
8049	CPLC_INVALID	Invalid CPLC; length must be 84
8050	PERMISSION_CHECK_DENY	Authorization verification failed; obtain verification permission again
8051	SERVER_PERFORMANCE_LIMIT	Concurrency limit reached; ask the factory administrator to check the server
8052	APDU_SESSION_INVALID	Invalid executeApdu session: command result for the current step does not match the expected result
8053	APDU_STEP_FAILED	Previous APDU execution failed
8054	APDU_RESTART_REQUIRED	APDU session invalid or expired; restart programming for this device
8055	APDU_PROCESS_FAILED	Failed to process APDU Profile step

respCode	Enum name	Description
8056	SE_PROFILE_DECRYPT_ERROR	Failed to decrypt SE Profile
8057	TEST_FILE_PARAM_ERROR	Invalid test file upload parameters
8058	TEST_FILE_MANIFEST_ERROR	Invalid test file manifest
8059	TEST_FILE_STORAGE_ERROR	Failed to save test file locally
8060	TEST_FILE_TASK_ERROR	Failed to save test file upload task
8061	TEST_FILE_UPLOAD_ERROR	Failed to upload test file to cloud storage
8064	TEST_DATA_TASK_ERROR	Failed to save test data report task
8065	ID_RULE_INVALID	Invalid ID Rule description
8066	ID_RULE_RANGE_EXHAUSTED	ID Rule allocation range exhausted
8067	ID_RULE_CONFLICT	ID Rule status conflicts with batch description
8068	ID_RULE_DEVICE_KEY_REQUIRED	A stable device identifier is required to parse the ID Rule
8069	SCENARIO_TYPE_NOT_BE_NULL	scenarioType must not be empty
8070	DATA_INVALID	Invalid production data
8071	KEY_DECRYPT_FAILED	Failed to decrypt batch protection key
8072	DATA_EXHAUSTED	Production data exhausted
8073	ASSIGNMENT_CONFLICT	Production data allocation conflicts with batch or device
8074	PROFILE_INVALID	Programming requires a PERSONALIZE operation and corresponding Applet selection information
8101	DEVICE_NOT_FOUND	Device not found; check query parameters
8102	MULTI_DEVICES_FOUND	Multiple devices matched; add filters
8103	QUERY_CONDITION_REQUIRED	Specify at least one valid query condition in deviceLocator

6.3 Appendix Three: Updatable Keys (updateDevicesProfile)

Keys supported by the Update Devices Profile API:

Data item	Key	Value description
Device extension field	extend.<extension field key>	Supported extension fields: vid, pid, chipId, qrcode, pairingCode
env	environment	test / prod

Data item	Key	Value description
Production time	manufactureTime	• string • Time format: 2020-11-11T01:11:11+08:00
Version name	productVersion	string
Batch name	productBatch	string
Factory name	factory	string
Programming station name	programmingStation	string
Programming status	programmingStatus	Complete / Failed / Unreported
DAC certificate serial number	dacSerialNo	string
DAC certificate commonName	dacCommonName	string
DAC certificate VID	dacVid	string
DAC certificate PID	dacPid	string
Subject key identifier	subjectKeyIdentifier	string
Authority key identifier	authorityKeyIdentifier	string
DAC effective date	notBefore	• string • Time format: 2020-11-11T01:11:11+08:00
DAC expiration date	notAfter	• string • Time format: 2020-11-11T01:11:11+08:00
Pairing code	pairingCode	string
QR code	qrcode	string
DAC expiration time (date, time, and time zone)	expiryDate	• string • Time format: 2020-11-11T01:11:11+08:00
Remaining days of DAC validity	leftDays	string
Issued DAC certificate	certificate	string
Field in factoryData	factoryData.<key name>	• string • Supports deletion of the corresponding key in factory data.

Data item	Key	Value description
Script filename	programmingScript	• string • Supports deletion of the script file with this name. Only one script file may exist; uploading a new script filename overwrites the prior file record.
Firmware	firmwareData.<firmware name>	• The startAddress of the named firmware may be updated. • Deletion of the named firmware is supported.
otherFiles	otherFile.<filename>	• The key of the named file in otherFile may be updated. • Deletion of the named file is supported.
attachmentFiles	attachment.<filename>	• The key of the named attachment may be updated. • Deletion of the named file is supported.

6.4 Appendix Four: Queryable Keys (queryKeys V1)

> **TIP**
> - If a data item is not configured for the product, the API returns an empty value for it.
> - The digit `1` in the key `fw1.fileName` denotes the first firmware file.

Data item	Key
CD file MD5	cdFile.md5
CD filename	cdFile.fileName
Commissioning flow	matterCommissioningParameter.commissioningFlows
DAC MD5	dacTemplate.md5
Discovery mode	matterCommissioningParameter.discoverableMode
Discriminator	matterCommissioningParameter.discriminator
EnableRotatingDeviceId	matterCommissioningParameter.enableRotatingDeviceId
hardwareVersion	hardwareVersion
hardwareVersionString	hardwareVersionString
mcuId	mcuId
PAI MD5	paiCert.md5
partNumber	partNumber
PID	pid
productLabel	productLabel

Data item	Key
productUrl	productUrl
salt	matterCommissioningParameter.salt
softwareVersion	softwareVersion
softwareVersionString	softwareVersionString
SPAKE2+Iterations	matterCommissioningParameter.iterations
UniqueID	matterCommissioningParameter.uniqueId
vendorName	vendorName
verifier	matterCommissioningParameter.verifier
VID	vid
QR code	matterCommissioningParameter.qrcode
Firmware filename	fw1.fileName
Device programming result	programmingStatus
Manual pairing code	matterCommissioningParameter.manualCode
Original passcode	matterCommissioningParameter.passcode

6.5 Typical Business Call Flows

This section gives typical API call sequences for station programming, testing, and assembly. See each API description for its request parameters, response structure, and handling rules.

6.5.1 Station Programming

After login, batch query, and assignment, the station obtains programming data, programs the device, and reports the result. A typical call flow is:

```
queryLoginType
    ↓
 preLogin (as needed)
    ↓
 login
    ↓
 queryPPCBatchInfo / queryBatchByOrderNumber
    ↓
 assignBatch (as needed)
    ↓
 getMetadata
    ↓
 getProgrammingData
    ↓
 uploadRequestData (as required by programming items)
```

For SE programming, also observe the following rules:

- SE-only scenario: A production record may be created at the BOF stage.
- Mixed SE_MCU programming: Reuse the same `traceId` from the MCU or Matter programming stage, then execute SE programming operations through `executeApdu` until EOF.

6.5.2 Testing and Assembly

Testing and assembly data is reported through `/v1/deviceData/upload`. To associate a test file, upload the file first, then submit the related production data. A typical call flow is:

```
login
   ↓
 testFile/upload (as needed)
   ↓
 deviceData/upload
```

When calling these APIs, note:

- **Testing**: For the first upload without attachments, `traceId` may be omitted, but `scenarioType` is required. Use the `traceId` returned by the API for subsequent uploads.
- **File association**: To attach a test file, first upload it through `/v1/testFile/upload`, then use the returned `traceId` in `/v1/deviceData/upload`.
- **Assembly**: Use the `batchId` of the actual assembly batch and report the execution result through `deviceRecord.testData`.result.

Depending on the business flow, a scan may first call `/v1/matter/onboarding-payloads/recover` to recover Matter onboarding data. Use `/v1/device/information/query` when detailed device information is needed.

Version: v3.2.1

OnBoard™ IoT Security FW Signing and Pushing CLI - v1.0.0

1. Revision Record

Version	Modified Date	Notes
v1.0.0	2025/02/17	Initial release

2. Overview

OnBoard™ IoT Security (OBIS) FW Signing and Pushing CLI is a command-line tool built specifically for OBIS, dedicated to firmware signing and pushing. It integrates several core functions, including key configuration based on the PKI (Public Key Infrastructure) certificate signing, KMS (Key Management System) firmware signing, public key export, and firmware pushing to OBIS.

This article introduces the common commands of the OBIS FW Signing and Pushing CLI and provides parameter descriptions and usage examples to help you efficiently complete firmware signing and pushing operations.

3. Syntax

`snb [command] [subcommand] [parameters]`

4. Command List

4.1 Key Configuration Command

Used to set the Access Key and Secret Key.

4.1.1 Command

`config`

4.1.2 Parameters

Parameters	Type	Mandatory	Description
`--accessKey`	string	Yes	Set the Access Key, which can be viewed in the user page in OBIS.
`--secretKey`	string	Yes	Set the Secret Key, which can be viewed in the user page in OBIS.

4.1.3 Example

```
snb config --accessKey <AK> --secretKey <SK>
```

4.2 Firmware Signing Command

Used to perform a signing operation on input data with the specified key pair.

4.2.1 Command

`kms`

4.2.2 Subcommand

`sign`

4.2.3 Parameters

Parameters	Type	Mandatory	Description
`--algorithm`	string	Yes	Configure the signing algorithm, currently supporting `ecdsa_sha_256`.
`--keyId`	string	Yes	Specify the key ID to use for the signing operation.
`--inputFile`	string	No	Specify the target file to be signed.
`--inputData`	string	No	Input the data to be signed, in hexadecimal string format.

Parameters	Type	Mandatory	Description
`--inputHash`	string	No	Input the pre-hashed data to be signed, in hexadecimal string format.
`--outputFile`	string	No	Specify the output file for the signature result, in the **.der** format. Defaults to standard output (stdout).
`--resourcePackageId`	string	No	Specify the resource package ID for the product. This parameter is required when accessing the product keys.

TIP

- At least one of the parameters (`inputFile`, `inputData`, or `inputHash`) must be provided.
- When outputting to standard output (stdout), the signature result in the **.der** format is represented as a hexadecimal string.

4.2.4 Example

```
snb kms sign --algorithm ecdsa_sha_256 --keyId <keyID> --
inputFile ./firmware.bin --outputFile ./signedResult.der
```

4.3 Public Key Export Command

Used to export the public key of a specified key pair.

4.3.1 Command

`kms`

4.3.2 Subcommand

`exportPubKey`

4.3.3 Parameters

Parameters	Type	Mandatory	Description
`--keyId`	string	Yes	Specify the key ID to export.
`--outputFormat`	string	Yes	Specify the output format for the exported key, options are `PEM` or `DER`. Default is `DER`.
`--outputFile`	string	Yes	Specify the output file for the exported key. Defaults to standard output (stdout).
`--resourcePackageId`	string	No	Specify the resource package ID for the product. This parameter is required when the Access Key used belongs to a personal account.

> **TIP**
>
> When outputting to standard output (stdout), the signature result in the **.der** format is represented as a hexadecimal string.

4.3.4 Example

```
snb kms exportPubKey --keyId <keyID> --outputFormat PEM --
outputFile pubKey.pem
```

4.4 Certificate Signing Command (such as Matter DAC)

Used to sign a certificate signing request (CSR) file using the specified certificate template.

4.4.1 Command

`pki`

4.4.2 Sub-command

`sign`

4.4.3 Parameters

Parameters	Type	Mandatory	Description
`--inputFormat`	string	Yes	Specify the input format (`PEM` or `DER`). Default is `PEM`.

Parameters	Type	Mandatory	Description
`--certTemplateId`	string	Yes	Specify the certificate (such as Matter DAC) template ID.
`--outputFormat`	string	Yes	Specify the output file format(`PEM` or `DER`). Default is `DER`.
`--csrFile`	string	No	Specify the path of the CSR file.
`--csrContent`	string	No	Provide the content of the CSR file.
`--outputFile`	string	No	Specify the output filename. Defaults to standard output (stdout).

TIP

- At least one of the parameters (`csrFile` or `csrContent`) must be provided.
- When outputting to standard output (stdout), the signature result in the **.der** format is represented as a hexadecimal string.

4.4.4 Example

```
snb pki sign --csrFile ./csr.der --inputFormat DER --
certTemplateId 1F118749-EEB1-11EF-A430-B8CEF6C25058
```

4.5 Firmware Push Command

Used to push the signed firmware file to OBIS.

4.5.1 Command

`obis`

4.5.2 Subcommand

`push`

4.5.3 Parameters

Parameters	Type	Mandatory	Description
`--file`	string	Yes	Specify the file to be pushed to OBIS.
`--fileType`	string	Yes	Specify the type of the file being pushed: `Application` (default), `Bootloader`, or `Others`.
`--resourcePackageId`	string	Yes	Specify the resource package ID under the target product in OBIS.
`--env`	string	Yes	Specify the target environment for pushing: `TEST` or `PROD`.
`--startAddress`	string	No	Specify the firmware programming start address, in hexadecimal format starting with `0x`.

Parameters	Type	Mandatory	Description
`--version`	string	No	Set the firmware file version, with a maximum length of 16 characters. Allowed characters include text, numbers (0-9), letters (a-z, A-Z), dot (.), underscore (_), and hyphen (-).

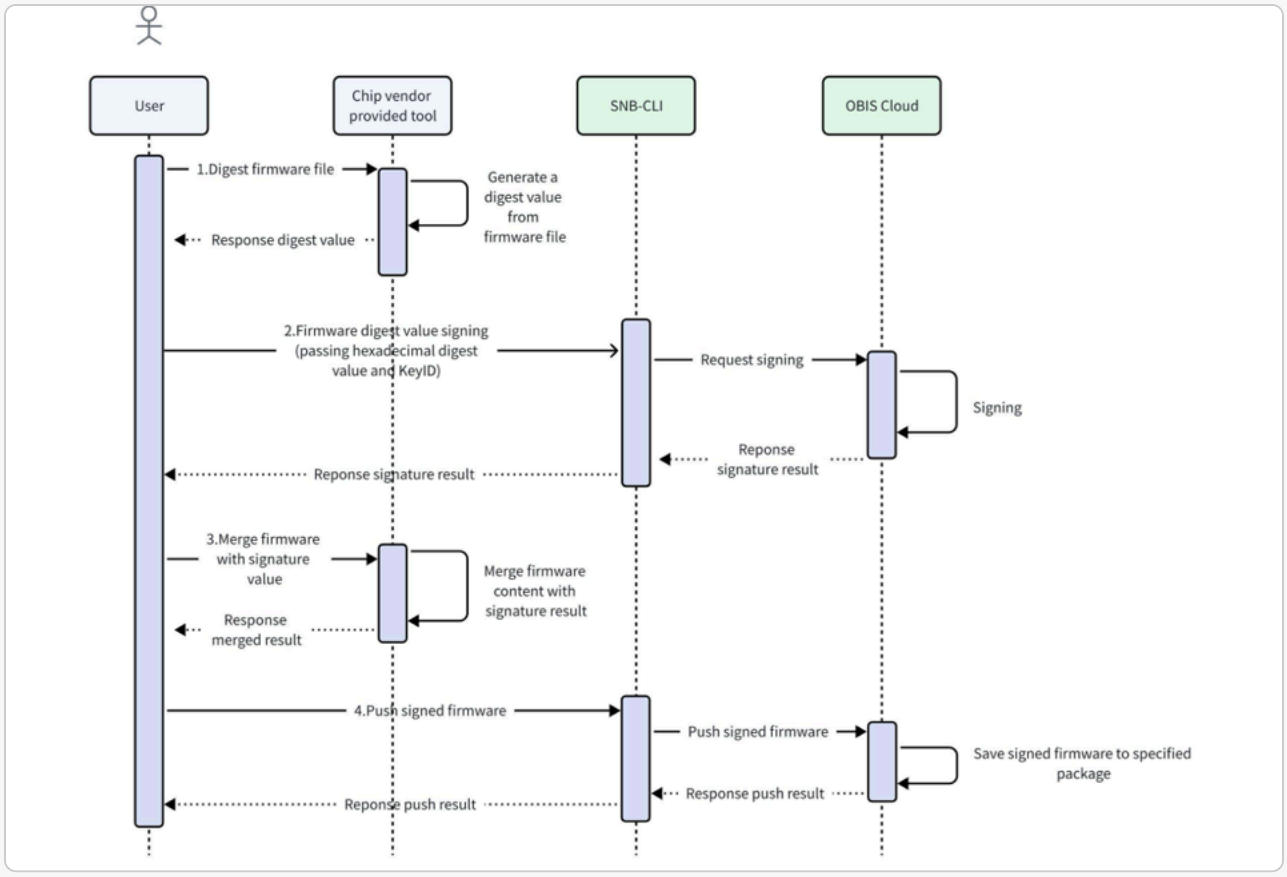
TIP

- When the file being pushed is in the **.bin** format, the `startAddress` parameter is required.
- The combination of `version`, `env`, and `file` must be unique and can't be duplicated.

4.5.4 Example

```
snb obis push --file ./lighting.s37 --fileType Application --
resourcePackageId <id> --env TEST --version 1.0.0
```

5. Execution Flow Example



Version: v3.2.1

OnBoard™ IoT Security Programming CLI - v3.0.5

1. Revision Record

Version	Modified Date	Notes
v1.0.0	2025/05/12	Initial release
v3.0.0	2026/01/22	Added the following nine commands: • Log out of Edge Service • Query Login Type • Query Batch List • Assign Batch • Push Programming Events • Extract Programming Script • View Log Configuration • Start Log Collection • Stop Log Collection Deleted the following three commands: • Query Identity U-Safe Information • Query Tool Version • Download Tool Update Package Updated the following two commands: • Added the `--stationId` parameter for the "Log in to Edge Service" command • Added the `scriptUrl` field to the output of the "Query Batch Information" command
v3.0.1	2026/03/13	The `--stationId` parameter in the "Log in to Edge Service" command is now optional instead of mandatory
v3.0.2	2026/03/17	In the Query Programming Script command section, add instructions for downloading the programming script, and update the corresponding command example
v3.0.3	2026/06/02	• Added the Download Large Firmware command • Added the `--stationVersion` parameter for the "Log in to Edge Service" command
v3.0.4	2026/09/19	• Added PPCLI (Programming Process Control CLI) and the dedicated Telink programming interface, with UI (Programming Station client) integration guidance • Added the complete call flow from login, batch and programming script preparation through device programming • Added process startup, output reading, result determination, and error handling guidance for programming • Updated related command parameters and programming flow descriptions

Version	Modified Date	Notes
v3.0.5	2026/09/22	Added the Execute Batch APDU Operation command, including parameters, call examples, HTTP mapping, response, and exit code descriptions

2. Overview

OnBoard™ IoT Security (OBIS) Programming CLI is a command-line tool built specifically for OBIS, dedicated to supporting the device programming process. It offers a comprehensive set of features: you can use it to log in to the Edge Service, query Identity U-Safe list, retrieve and upload batch data, upload programming results, and query or download programming scripts.

This article introduces common OBIS Programming CLI commands, with parameter descriptions and usage examples to help you perform device programming and related operations. It also explains how the UI (Programming Station client) starts device programming through PPCLI (Programming Process Control CLI) or a dedicated programming interface, and covers the full call flow from login, batch and programming script preparation through device programming and result handling.

3. OBIS Programming CLI Syntax

`obiscli [command] [subcommand] [parameters]`

> **INFO**
>
> Strings containing spaces in command-line arguments must be enclosed in double quotes `"`; otherwise, the command may fail to parse.
>
> - Correct Example: `--chipVendor="Test Labs"`
> - Error Example: `--chipVendor=Test Labs`

4. OBIS Programming CLI Commands

4.1 Query Tool Help

Used to query the usage instructions of the command-line tool and the list of available parameters.

4.1.1 Parameter

`--help`

4.1.2 Example

```
obiscli.exe --help
positional arguments:
                OBIS CLI Tool Command
    login       Login to OBIS Local Service.
    logout      logout to OBIS Local Service.
    usafe       OBIS usafe commands.
    batch       OBIS batch commands.
    script      OBIS batch commands.
    obisexe     OBIS commands.
    logCollect  OBIS batch commands.
    addPath     add sys Path.

options:
```

4.2 Log in to Edge Service

Used to log in to the Edge Service.

4.2.1 Command

`login`

4.2.2 Parameters

Parameters	Type	Mandatory	Description
`--ip`	string	Yes	Programming Station client IP, default value: 127.0.0.1
`--port`	string	Yes	Programming Station client port, default value: 32333
`--stationId`	string	No	Programming Station ID
`--stationVersion`	string	No	Version of the Programming Station client
`--factorytype`	string	No	Factory type, default value: local
`--password`	string	No	Login password, required only when password-based login is enabled
`--usafesn`	string	No	U-Safe serial number

4.2.3 Example

```
# no-password login:
obiscli login --ip=192.168.40.100 --port=32333 --stationId 81045379
# u-safe login:
obiscli login --ip=192.168.40.100 --port=32333 --usafesn=00003F00AB013668 --stationId 81045379
# password login:
obiscli login --ip=192.168.40.100 --port=32333 --password=12345678 --stationId 81045379
or
obiscli login --ip=192.168.40.100 --port=32333 --password=12345678
output:
{
    "stationid": "94221070",
    "token": "b69a9fe7-9df4-4bdc-819e-9b84a7bc0b23",
    "tokenexpire": "unlimited"
}
```

4.3 Log out of Edge Service

Used to log out of the Edge Service.

4.3.1 Command

`logout`

4.3.2 Example

```
# Logout
obiscli logout
```

4.4 Query Login Type

Used to query the login type for the Edge Service.

4.4.1 Command

`login`

4.4.2 Subcommand

`queryType`

4.4.3 Parameters

Parameters	Type	Mandatory	Description
`--ip`	string	Yes	Programming Station client IP, default value: 127.0.0.1
`--port`	string	Yes	Programming Station client port, default value: 32333
`--stationId`	string	Yes	Programming Station ID

4.4.4 Example

```
obiscli login queryType --ip=192.168.40.205 --port=32333 --stationId=97123586

output◈◈1 No password, 2 Password, 3 U-safe◈
{
    "loginType": 3
}
```

4.5 Query Login Options

Used to query the specific login options for the Edge Service.

4.5.1 Command

`login`

4.5.2 Subcommand

`queryoptions`

4.5.3 Parameters

Parameters	Type	Mandatory	Description
`--ip`	string	Yes	Programming Station client IP, default value: 127.0.0.1
`--port`	string	Yes	Programming Station client port, default value: 32333
`--usafesn`	string	Yes	U-Safe serial number
`--factoryType`	string	No	Factory type, default value: local
`--stationId`	string	No	Programming Station ID

4.5.4 Example

```
obiscli login queryoptions --factoryType=local --ip=192.168.40.100 --port=32333 --usafesn=00003F00AB013668
```

4.6 Query Identity U-Safe List

Used to list the Identity U-Safes inserted in the local device.

4.6.1 Command

`usafe`

4.6.2 Subcommand

`list`

4.6.3 Example

```
obiscli usafe list
output⏎
{
    "usafelist": [
        {
            "usafesn": "0001BC00B86DA2BB"
        }
    ]
}
```

4.7 Query Batch Information

Used to query information about the batch corresponding to the devices to be programmed.

4.7.1 Command

`batch`

4.7.2 Subcommand

`query`

4.7.3 Example

```
obiscli batch query

output⏎
{
    "batches": [
        {
            "batchId": "7346273795440640",
            "batchName": "UI-jason-0925",
            "chips": [
                {
                    "chipPart": "QPG6200",
                    "chipVendor": "Qorvo",
                    "objectid": "4bb81efb504248fe813c83a4ed4602ac"
                }
            ],
            "env": "test",
            "extData": "VID : 0xFFF2  PID : 0x8001",
```

4.8 Query Batch List

Used to query the batch list associated with a factory order number.

4.8.1 Command

`batch`

4.8.2 Subcommand

`queryList`

4.8.3 Parameter

Parameter	Type	Mandatory	Description
`--orderNumber`	string	Yes	Order number

4.8.4 Example

```
obiscli batch queryList --orderNumber "LDS-19990101-1030BTC"

output⏀
[
    {
        "batchId": "8725582734950400",
        "batchName": "Test-Order-5"
    }
]
```

4.9 Assign Batch

Used to bind the batch to the programming station.

4.9.1 Command

`batch`

4.9.2 Subcommand

`assign`

4.9.3 Parameter

Parameter	Type	Mandatory	Description
`--batchId`	string	Yes	Batch ID

4.9.4 Example

```
obiscli batch assign --batchId 7176076344164352

output⏀
{
    "batches": [
        {
```

4.10 Query Batch Full Data

Used to query the complete data to be programmed into a device. This command is typically called by PPCLI or a vendor programming script; the UI does not need to call it directly.

4.10.1 Command

`batch`

4.10.2 Subcommand

`getfulldata`

4.10.3 Example

```
obiscli batch getfulldata
```

4.11 Query Batch Fixed Data

Used to query fixed data to be programmed into a device. This command is typically called by PPCLI or a vendor programming script; the UI does not need to call it directly.

4.11.1 Command

`batch`

4.11.2 Subcommand

`getFixData`

4.11.3 Example

```
obiscli batch getFixData
```

4.12 Query Batch Changed Data

Used to query changing data to be programmed into a device. This command is typically called by PPCLI or a vendor programming script; the UI does not need to call it directly.

4.12.1 Command

`batch`

4.12.2 Subcommand

`getChangingData`

4.12.3 Example

```
obiscli batch getChangingData

output
{
    "batchId": "7115488079380480",
    "batchName": "V1.8.5 Dual Button 0906 MP",
    "bundles": [
```

4.13 Push Batch Data

Used to push data to the Edge Service. This command is typically called by PPCLI or a vendor programming script; the UI does not need to call it directly.

4.13.1 Command

`batch`

4.13.2 Subcommand

`putdata`

4.13.3 Parameters

Parameters	Type	Mandatory	Description
`--batchId`	string	Yes	Batch ID
`--chipVendor`	string	Yes	Chip vendor name, for example: Silicon Labs
`--chipPart`	string	Yes	Chip part
`--data`	json	Yes	Push data
`--uuid`	string	Yes	Universally unique identifier

4.13.4 Example

```
obiscli batch putdata --uuid=24a1c3da-459a-429b-a717-29989f926d43 --batchId=4808741360762880 --chipVendor="Silicon Labs" --chipPart=EFR
[{\"items\": [{\"key\": \"dacTemplate\", \"response\": [{\"csr\":
\"MIHmMIGNAgEAMCsxKTAnBgNVBAMMIERBQyA1SFNmblpxVSBNdmlkOjE0NDEgTXBpZDowMTIzMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEXeJ5k1UtgF7YJDRm37XvHu47m
FNF1JjcucbEgaIR+KAAMAoGCCqGSM49BAMCA0gAMEUCIQDeR2rM9FOdCg1GTJyAOpQR2FtARVwIw3nEjXv9t7mg6wIgBcvUbH1G8NAlB/I7rOY48b/Fgnf5UkEOMx/7eekcU0k=
```

4.14 Execute Batch APDU Operation

Used to execute an APDU (Application Protocol Data Unit) operation through the platform service in programming flows that require APDU interaction. This command is typically called by PPCLI or a vendor programming script; the UI does not need to call it directly.

4.14.1 Command

`batch`

4.14.2 Subcommand

`executeApdu`

4.14.3 Parameters

Parameter	Type	Mandatory	Description
`--batchId`	string	Yes	Current programming batch ID. It must match the batch to which this device data belongs.

Parameter	Type	Mandatory	Description
`--uuid`	string	Yes	`uuid` of this device data or programming record. Use the identifier obtained in the current programming flow; do not substitute the adapter `serialno`.
`--data`	string	No	Additional request fields as a JSON object. After parsing, they are merged into the top level of the request body. Examples include `programmingTraceId` and command response data required by the server protocol. Follow the actual server protocol for field names, types, and which stages require them.

TIP

- `--batchId` and `--uuid` are not marked required by the current argument parser, but the command does not fill them in from the assigned batch or data cache. Supply valid `batchId` and `uuid` values. If omitted, their request values are `null`; final business validation is performed by the server.
- Log in first and obtain identifiers for the current batch and device data. OBIS Programming CLI reads the service address and session token from local configuration; do not pass login parameters again.
- `--data` must be a valid JSON object, not a file path, plain string, or array. The current implementation merges its fields directly into the top level of the request body, without adding a `data` layer. Duplicate fields overwrite existing values, so do not repeat `batchId` or `uuid` in `--data`.
- This command does not support `--chipVendor`, `--chipPart`, or `--serialno`.
- Whether `executeApdu` must be called more than once, how to submit the previous command response, and how to determine when the APDU flow is complete depend on the server protocol.

4.14.4 Examples

Without additional request fields:

```
obiscli batch executeApdu --batchId="<BATCH_ID>" --uuid="<UUID>"
```

When programmingTraceId is required, add other business fields according to the server protocol:

```
obiscli batch executeApdu --batchId="<BATCH_ID>" --uuid="<UUID>" --
data="{\"programmingTraceId\":\"<PROGRAMMING_TRACE_ID>\"}"
```

The two examples show alternative argument forms; they do not mean the command must be run twice in succession. Follow the actual server protocol for the use of `--data` and the business fields it contains.

The `\"` escaping in `--data` above applies to the Windows `cmd.exe` example. When an application invokes the process API, pass the JSON string directly in the argument array, without additional shell quotes or escaping.

4.14.5 HTTP Mapping

OBIS Programming CLI sends `executeApdu` requests to the following endpoint:

Item	Description
Method	`POST`
Path	`/api/iot-factory-edge-service/executeApdu`
Content-Type	`application/json`
Session	`x-snbps-session` uses the current login token

Item	Description
Request ID	`x-snbps-request-id` is generated automatically by OBIS Programming CLI
Timestamp	`x-snbps-timestamp` is generated automatically by OBIS Programming CLI

Local mode uses the saved IP and port. Cloud mode uses the saved host and automatically includes `factoryId` when configured.

For example, a command containing `programmingTraceId` produces this request body:

```
{
  "uuid": "<UUID>",
  "batchId": "<BATCH_ID>",
  "programmingTraceId": "<PROGRAMMING_TRACE_ID>"
}
```

4.14.6 Response and Exit Codes

After parsing the HTTP response, OBIS Programming CLI formats `respData` itself as JSON and exits with code `0` when `respCode` is the string `"8000"` and `respData` is not `null`.

The CLI does not also output the outer `respCode` and `respMsg`. The server defines the business structure of respData; the current CLI does not prescribe fixed fields.

Exit code	Description
`0`	This interface call succeeded. The calling script must still process the response according to the business protocol; this does not establish that the full device programming flow is complete.
`1`	`--data` is not valid JSON text.
`45`	The HTTP client call failed. The CLI outputs `executeApdu failed: Network error`
`46`	`respData` is missing or `null` in a successful response. The current implementation does not return this error solely for an empty object `{}` or empty array `[]`.
Integer converted from the server `respCode`	When `respCode` is not `"8000"`, the CLI outputs `respMsg` and exits with the corresponding business error code.
`200`	An uncaught exception occurs, for example when the response cannot be parsed or a business error code cannot be converted to an integer.

After the APDU step, PPCLI or the vendor programming script continues the original programming flow and reports the final programming result with batch `putresult`. `executeApdu` does not replace batch `putresult`.

4.15 Push Programming Results

Used to push programming results to the Edge Service. This command is typically called by PPCLI or a vendor programming script; the UI does not need to call it directly.

4.15.1 Command

`batch`

4.15.2 Subcommand

`putresult`

4.15.3 Parameters

Parameters	Type	Mandatory	Description
`--batchId`	string	Yes	Batch ID
`--changeKeys`	string	Yes	If service data is not used, enter the key. Supported keys are defined in the list of queryKeys V2
`--result`	string	Yes	Programming results
`--uuid`	string	Yes	Universally unique identifier

4.15.4 Example

```
obiscli batch putresult --batchId=4808741360762880 --uuid=24a1c3da-459a-429b-a717-29989f926d43 --result=success --
changeKeys=''
```

4.16 Push Programming Events

Used to push programming events to the Edge Service.

4.16.1 Command

`batch`

4.16.2 Subcommand

`uploadEvent`

4.16.3 Parameters

Parameters	Type	Mandatory	Description
`--batchId`	string	Yes	Batch ID
`--uuid`	string	Yes	Unique device identifier
`--event`	string	No	Programming events

4.16.4 Example

```
obiscli batch uploadEvent --batchId=4808741360762880 --uuid=24a1c3da-459a-429b-a717-29989f926d43 --event
"{\"uuid\":\"24a1c3da-459a-429b-a717-29989f926d43\", \"events\": [{\"eventName\": \"Generate Trace ID\",
\"timestamp\": \"2026-01-09T09:46:02.672Z\", \"source\": \"script\", \"priority\": \"1\", \"status\": \"success\",
\"content\": \"\"}, {\"eventName\": \"Fetch Batch Dynamic Data\", \"timestamp\": \"2026-01-09T09:46:04.538Z\",
\"source\": \"script\", \"priority\": \"1\", \"status\": \"success\", \"content\": \"\"}, {\"eventName\": \"Fetch
Device ID\", \"timestamp\": \"2026-01-09T09:46:07.340Z\", \"source\": \"script\", \"priority\": \"1\", \"status\":
\"success\", \"content\": \"\"}, {\"eventName\": \"Fetch CSR\", \"timestamp\": \"2026-01-09T09:46:27.433Z\",
\"source\": \"script\", \"priority\": \"1\", \"status\": \"success\", \"content\": \"\"}, {\"eventName\": \"Program
Factory Data\", \"timestamp\": \"2026-01-09T09:46:33.631Z\", \"source\": \"script\", \"priority\": \"1\", \"status\":
\"success\", \"content\": \"\"}, {\"eventName\": \"Report Programming Result\", \"timestamp\":
```

4.17 Query Programming Script

> **TIP**
>
> It is recommended to directly use the Query Batch Information command (`batch query`) to obtain the batch information for the devices to be programmed, retrieve the script download link, and then execute the Download Programming Script command (`script download`) to perform the download.

Used to check whether a programming script exists for the batch to be programmed.

4.17.1 Command

`script`

4.17.2 Subcommand

`query`

4.17.3 Parameters

Parameters	Type	Mandatory	Description
`--batchId`	string	Yes	Batch ID
`--chipPart`	string	No	Chip part
`--chipVendor`	string	No	Chip vendor name, for example: Silicon Labs
`--productId`	string	No	Product ID

4.17.4 Example

> **TIP**
>
> When `scriptExist` is `false`, the script should be downloaded by executing the Download Programming Script command (`script download`); if `true`, downloading is not required.

```
obiscli script query --batchId=4808741360762880 --chipVendor="Silicon Labs" --chipPart=EFR32MG24B210F1536IM48
output:
{
    "scriptExist": bool,
    "path": ...
}
```

4.18 Download Programming Script

Used to download the programming script.

4.18.1 Command

`script`

4.18.2 Subcommand

`download`

4.18.3 Parameters

Parameters	Type	Mandatory	Description
`--url`	string	Yes	Download link for the programming script
`--extractFolder`	string	No	Extraction path after downloading the programming script
`--savePath`	string	No	Save path after downloading the programming script

4.18.4 Example

```
obiscli script download --url="http://192.168.40.171:8005/api/iot-factory-edge-service/
download?fileId=1866310684527935298&fileName=0d232bf966e516f4bf2a75e1880e59ee.zip" --
savePath="C:\\Users\\Administrator\\AppData\\Roaming\\SNB\\chipVendorCmd" --
extractFolder="C:\\Users\\Administrator\\AppData\\Roaming\\SNB\\chipVendorCmd"
```

4.19 Extract Programming Script

Used to extract the programming script.

4.19.1 Command

`script`

4.19.2 Subcommand

`extract`

4.19.3 Parameters

Parameters	Type	Mandatory	Description
`--extractFolder`	string	No	Extraction path of the file
`--zipFile`	string	No	Path of the compressed file, must be in the **.zip** format

4.19.4 Example

```
obiscli script extract --zipFile="C:\\Users\\Administrator\\AppData\\Roaming\\SNB\\chipVendorCmd\\Silicon Labs.zip"
--extractFolder="C:\\Users\\Administrator\\AppData\\Roaming\\SNB\\chipVendorCmd"
```

4.20 Log Configuration

Used to configure parameters related to log collection.

4.20.1 Command

`logCollect`

4.20.2 Subcommand

`config`

4.20.3 Parameters

Parameters	Type	Mandatory	Description
`--logLevel`	string	No	Log level control
`--collectLogPath`	string	No	Directory for storing log files, default: **C:\Users\<User Name>\AppData\SNB**
`--collectLogSuffix`	string	No	Suffix for log files, default: `.log`
`--mqttIp`	string	No	IP of the MQTT Broker (server)
`--mqttPassword`	string	No	Password for connecting to the MQTT Broker user
`--mqttPort`	string	No	Port number of the MQTT Broker
`--mqttUser`	string	No	User name for connecting to the MQTT Broker

4.20.4 Example

```
obiscli logCollect config --logLevel=INFO
```

4.21 View Log Configuration

Used to view parameters related to log collection.

4.21.1 Command

`logCollect`

4.21.2 Subcommand

`showConfig`

4.21.3 Example

```
obiscli logCollect showConfig
output
{
  "DEFAULT": {
    "log_level": "INFO",
    "log_file_path": "C:\\Users\\Administrator\\AppData\\Roaming\\iot-log-collecter\\collecter.log",
    "log_rotation": "2 hour",
    "log_retention": "1 days",
    "send_times_save_status": "1",
    "collect_log_path": "C:\\Users\\Administrator\\AppData\\Roaming\\SNB",
    "collect_log_suffix": "[\".log\"]",
    "send_log_mode": "1",
    "source_type": "\"PS\" # ps:programming station  es:edge service"
  },
  "MQTT": {
    "broker": "192.168.40.144",
    "port": "32222",
    "user": "snowball",
    "password": "vX4hR9tV7iE8pO9d",
    "connect_crt": "",
    "connect_key": "",
```

4.22 Start Log Collection

Used to start log collection.

4.22.1 Command

`logCollect`

4.22.2 Subcommand

`start`

4.22.3 Example

```
obiscli logCollect start
output
iot-log-collect.exe run successfully.
```

4.23 Stop Log Collection

Used to stop log collection.

4.23.1 Command

`logCollect`

4.23.2 Subcommand

`stop`

4.23.3 Example

```
obiscli logCollect stop
output
iot-log-collecter.exe stop successfully.
```

4.24 Download Large Firmware

Used to download large firmware files.

4.24.1 Command

`downloadFw`

4.24.2 Parameters

Parameters	Type	Mandatory	Description
`--url`	string	Yes	Download link for the large firmware file
`--savePath`	string	No	Save path after downloading the large firmware file

4.24.3 Example

```
obiscli downloadFw --url <URL> --savePath <Path>
```

5. PPCLI Programming Interface

5.1 Overview

PPCLI (Programming Process Control CLI) controls the device programming flow. To integrate device programming, the UI (Programming Station client) first uses OBIS Programming CLI to log in, assign a batch, and prepare the programming script. It then uses the PPCLI entry point supplied with the programming script package to list adapters and start device programming.

PPCLI or the vendor programming script uses OBIS Programming CLI to retrieve batch data, request certificates, perform APDU interactions when needed, and report programming results. The UI generally does not need to call those interfaces directly.

5.2 Python PPCLI

This applies when the programming script package includes `ppcli.py` and its Python runtime. `ppcli.py` is the PPCLI entry point; `process.py` contains the vendor programming implementation that it calls. The UI should start programming through `ppcli.py` rather than executing `process.py` directly. Follow the package documentation for the actual entry point.

5.2.1 Prerequisites

Before calling Python PPCLI, ensure that:

- You have logged in to the Edge Service and assigned the current batch;
- The programming script package matches the current batch and chip model, has been downloaded and prepared, and contains all required dependencies;
- The child process working directory (`cwd`) is the actual entry directory containing `ppcli.py`, rather than the ZIP download directory or UI installation directory. Relative paths are resolved from this directory;
- The current process and its child processes can find the correct `obiscli.exe` version through `PATH` and use the same Windows user and configuration environment as the earlier OBIS Programming CLI calls, so they share the login session and current batch assignment. Using an absolute OBIS CLI path only in the UI does not ensure that PPCLI can find `obiscli.exe` internally;
- The adapter driver is installed and connected, and the target device is powered on;
- Parameters such as `--type`, `--speed`, chip vendor, and chip model match the actual programming script package.

5.2.2 List Adapters

Used to list currently available programmers.

5.2.2.1 Command

`adapter`

5.2.2.2 Subcommand

`list`

5.2.2.3 Parameters

Parameter	Type	Mandatory	Description
`--chipVendor`	string	Yes	Chip vendor name recognized by the delivered package, usually from batch `chips[].chipVendor`. Map it according to the package contract if needed; do not assume that case and spaces are interchangeable.
`--type`	string	Yes	Adapter type. Use the value specified by the programming script package.

5.2.2.4 Example

> **TIP**
> `--type=1` is the value used in this example. Use the value specified by the actual programming script package.

```
".\python-3.12.9-embed-amd64\python.exe" ".\ppcli.py" adapter list --chipVendor="<CHIP_VENDOR>" --type=1
Output
{
  "adapterList": [
    {
      "serialno": "<SERIALNO>"
    }
  ]
}
```

After a successful query, select the target adapter from `adapterList` and pass its `serialno` unchanged to the `program` command. Treat `serialno` as a string. Preserve leading zeros, for example `0008`. Do not start device programming if `adapterList` is empty. Treat missing required fields or an unparseable response as an adapter query failure. An exit code of `0` alone does not establish that an adapter is available.

5.2.3 Start Programming

Used to start device programming with the selected adapter.

5.2.3.1 Command

`program`

5.2.3.2 Parameters

Parameter	Mandatory	Type	Description
`--serialno`	string	Yes	Adapter or channel selector returned by `adapter list` in this package. Preserve the original string; do not substitute `batchId`, `stationId`, chip UID, or `uuid`.
`--type`	string	Yes	Use the value required by the package, consistent with `adapter list`. The package defines the meaning of `1` in this example.
`--chipVendor`	string	No	Vendor of the chip selected for the current batch. The UI should pass it; map it from configuration if the package uses a vendor alias.
`--chipPart`	string	No	Chip model selected for the current batch, usually from `chips[].chipPart`. The UI should pass it.
`--speed`	string	No	Programming communication speed; `115200` is an example. The package defines its units, valid range, and default. Do not interpret it universally as a UART baud rate or debug clock.
`--extInfo`	string	No	Vendor extension parameters. Pass them only when the package explicitly defines them; otherwise omit them. Do not pass arbitrary batch JSON.

5.2.3.3 Example

In the Python PPCLI version reviewed, `--serialno` and `--type` are required; the other parameters above are syntactically optional. Because the vendor implementation may depend on chip vendor and model, the UI should also pass `--chipVendor` and `--chipPart`. The `program` command does not need `batchId`; the preceding `obiscli batch assign` command binds the current batch.

```
".\python-3.12.9-embed-amd64\python.exe" ".\ppcli.py" program --chipVendor="<CHIP_VENDOR>" --chipPart="<CHIP_PART>"
--serialno="<SERIALNO>" --speed=115200 --type=1
```

> **TIP**
>
> Use `--chipVendor`, `--chipPart`, and `--type` values defined by the actual Python PPCLI package. Do not copy `TELINK` or `telinkevk` from the dedicated Telink interface example.

If the programming script package explicitly requires vendor extension parameters, add `--extInfo="<EXT_INFO>"`.

5.2.4 Check Parameters

Use these commands to check the parameters supported by the actual Python PPCLI package:

```
".\python-3.12.9-embed-amd64\python.exe" ".\ppcli.py" adapter list --help
".\python-3.12.9-embed-amd64\python.exe" ".\ppcli.py" program --help
```

> **TIP**
>
> `--help` only displays command and parameter definitions; it does not program a device.

5.2.5 Internal PPCLI Calls

This section describes the calls between PPCLI and OBIS Programming CLI after device programming starts. The UI prepares the runtime, selects an adapter, starts `program`, and handles its output. PPCLI then follows the current batch and chip configuration to perform the programming flow, including:

1. Calling `obiscli batch getChangingData` to retrieve changing data and the programming identifier for the current device;
2. When fixed data is not cached, calling `obiscli batch getFixData` to retrieve and cache it, then combining it with the changing data;
3. Calling the vendor programming logic in `process.py` to access the device and perform certificate and Factory Data processing, firmware writing, and verification according to the chip and batch configuration;
4. Using `obiscli batch putdata` to exchange data with the Edge Service when required during programming, for example to submit a CSR and obtain a DAC; using `obiscli batch executeApdu` for APDU programming steps when APDU interaction is required;
5. After programming, reporting `success` or `failure` through `obiscli batch putresult`. Reporting may be impossible if preliminary data retrieval fails, a valid `uuid` or `batchId` is unavailable, or the flow stops unexpectedly.

The UI does not need to retrieve changing device data, request a DAC, or report the same programming result again. PPCLI or the programming script performs these operations.

5.3 Dedicated Telink Programming Interface

Some Telink programming script packages use a dedicated executable to list adapters and start device programming directly. The UI should select the programming entry point supplied by the actual package. Python PPCLI and the dedicated Telink executable are alternative integration methods; use one of them. If Python PPCLI already calls the Telink programming tool internally, the UI should call only the outer PPCLI to avoid starting programming twice for the same device.

5.3.1 List Adapters

Used to list available adapters through the dedicated Telink programming executable.

5.3.1.1 Command

`adapter`

5.3.1.2 Subcommand

`list`

5.3.1.3 Parameters

Parameter	Type	Mandatory	Description
`--chipVendor`	string	Yes	Chip vendor identifier. This example uses `TELINK`; do not substitute the vendor name used by Python PPCLI without checking the package.
`--type`	string	Yes	Adapter type. This example uses `telinkevk`; do not mix it with `--type=1` from the Python PPCLI example.

5.3.1.4 Example

```
".\Cmd\_download\_tool.exe" adapter list --chipVendor="TELINK" --type=telinkevk
```

The path above refers to `_tool.exe` in the `_download` subdirectory of `Cmd`; it does not refer to a single file named `Cmd_download_tool.exe`.

Use the executable path in the delivered programming script package. For example, if the executable is in the package root, use:

```
".\Cmd_download_tool.exe" adapter list --chipVendor="TELINK" --type=telinkevk
```

Packages may have different directory structures. The UI should determine the executable path and working directory from package configuration, without hard-coding a fixed path or guessing a substitute when the target file is absent.

5.3.2 Start Programming

Used to start device programming with the selected Telink adapter.

5.3.2.1 Command

program

5.3.2.2 Parameters

Parameter	Type	Mandatory	Description
`--serialno`	string	Yes	Target adapter device ID returned by `adapter list` of the same dedicated Telink executable. Preserve the original string.

5.3.2.3 Example

```
".\Cmd\_download\_tool.exe" program --serialno="<SERIALNO>"
```

If the delivered executable is in the package root, use:

```
".\Cmd_download_tool.exe" program --serialno="<SERIALNO>"
```

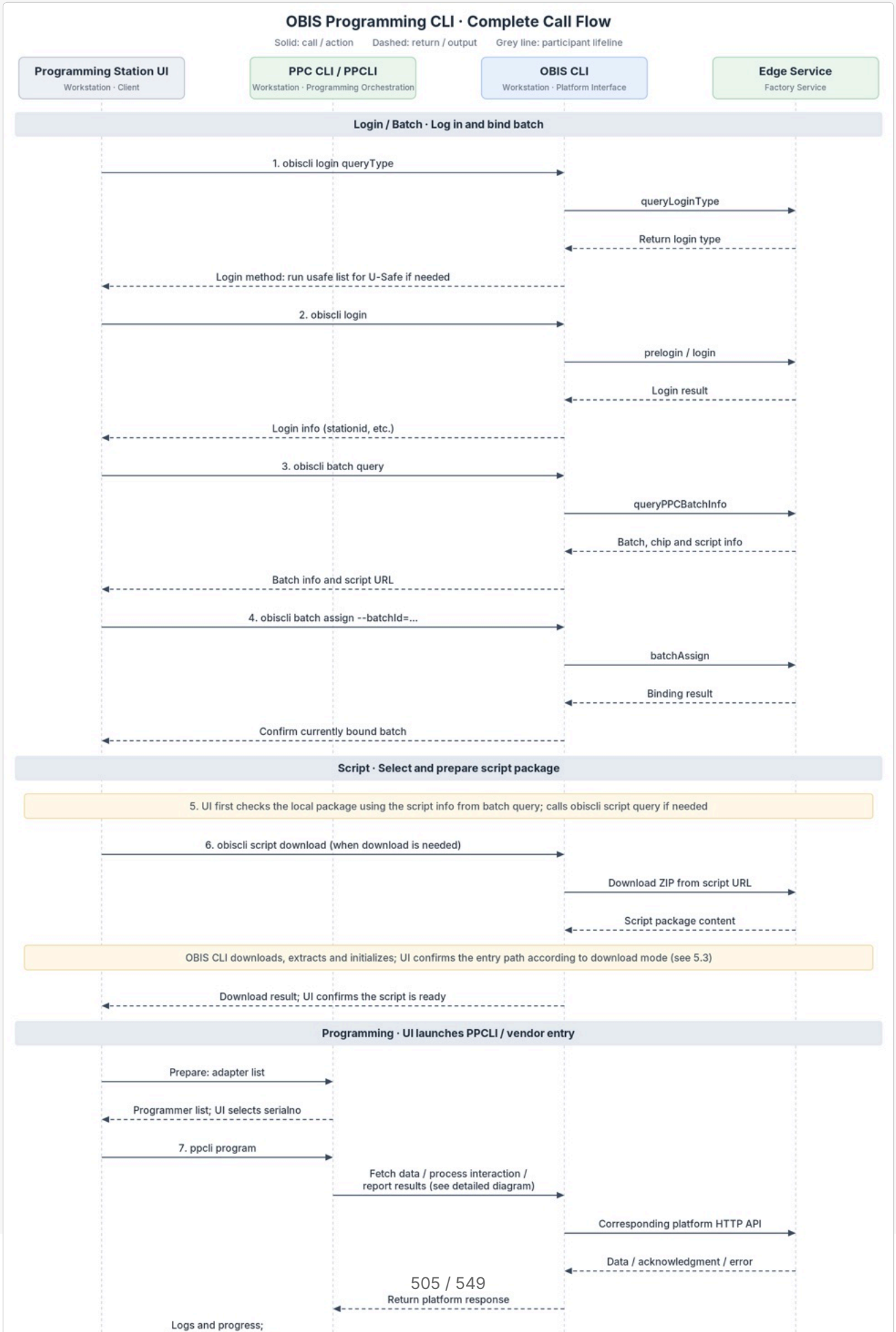
The `program` command of the dedicated Telink interface does not use the `--chipVendor`, `--chipPart`, `--speed`, or `--type` parameters shown for Python PPCLI. Do not append them to the Telink command. Follow the actual Telink package documentation for other programming settings.

The response, progress format, timeout, and exit codes of the dedicated Telink interface depend on the delivered package version.

6. Device Programming Execution Flow

6.1 Overall Call Flow

This section describes the overall UI flow from Edge Service login and batch and programming script preparation through device programming with PPCLI.


OBIS Programming CLI · Complete Call Flow
Solid: call / action
Dashed: return / output
Grey line: participant lifeline
Programming Station UI
Workstation · Client
PPC CLI / PPCLI
Workstation · Programming Orchestration
OBIS CLI
Workstation · Platform Interface
Edge Service
Factory Service
Login / Batch · Log in and bind batch
1. obiscli login queryType
queryLoginType
Return login type
Login method: run usafe list for U-Safe if needed
2. obiscli login
prelogin / login
Login result
Login info (stationid, etc.)
3. obiscli batch query
queryPPCBatchInfo
Batch, chip and script info
Batch info and script URL
4. obiscli batch assign --batchId=...
batchAssign
Binding result
Confirm currently bound batch
Script · Select and prepare script package
5. UI first checks the local package using the script info from batch query; calls obiscli script query if needed
6. obiscli script download (when download is needed)
Download ZIP from script URL
Script package content
OBIS CLI downloads, extracts and initializes; UI confirms the entry path according to download mode (see 5.3)
Download result; UI confirms the script is ready
Programming · UI launches PPCLI / vendor entry
Prepare: adapter list
Programmer list; UI selects serialno
7. ppcli program
Fetch data / process interaction /
report results (see detailed diagram)
Corresponding platform HTTP API
Data / acknowledgment / error
Return platform response
Logs and progress;

Stage	Caller → Receiver	Command / action	Prerequisite for next action
1	UI → OBIS CLI	`login queryType`; run `usafe list` if needed for U-Safe login	Obtain the login type and required identity information.
2	UI → OBIS CLI	`login`	Login succeeds; save the `stationid` required by the client.
3	UI → OBIS CLI	`batch query`; or `batch queryList` for an order	Obtain full batch information and identify `batchId`, chip, and programming script. A list containing only IDs cannot replace full batch information.
4	UI → OBIS CLI	`batch assign`	Batch assignment succeeds; subsequent programming uses the current assigned batch.
5	UI	Prefer script URL, name, and version or digest from `batch query`; run `script query` if needed.	Decide whether to reuse a matching local package or download it again.
6	UI → OBIS CLI	`script download`	Download, extract, and initialize the package; locate its actual entry directory. Skip download if a prepared package is already cached.
7 (prepare)	UI → PPCLI / package entry	`adapter list`	List available adapters successfully and select the target `serialno`.
7	UI → PPCLI / package entry	`program`	Read programming output in real time, wait for process completion, and handle success, failure, or a result reporting error.
Finish	UI → OBIS CLI	Run `batch query` to refresh batch statistics if needed; run `logout` at session end.	End the login session after programming and result reporting have finished on all channels.

Step 5 should use the programming script information returned by `batch query` first. The UI need not always call `script query`; call it only if the flow requires more information. Batch responses may have an outer wrapper that varies by version. Parse the actual OBIS Programming CLI output rather than assuming its JSON structure from a raw HTTP response.

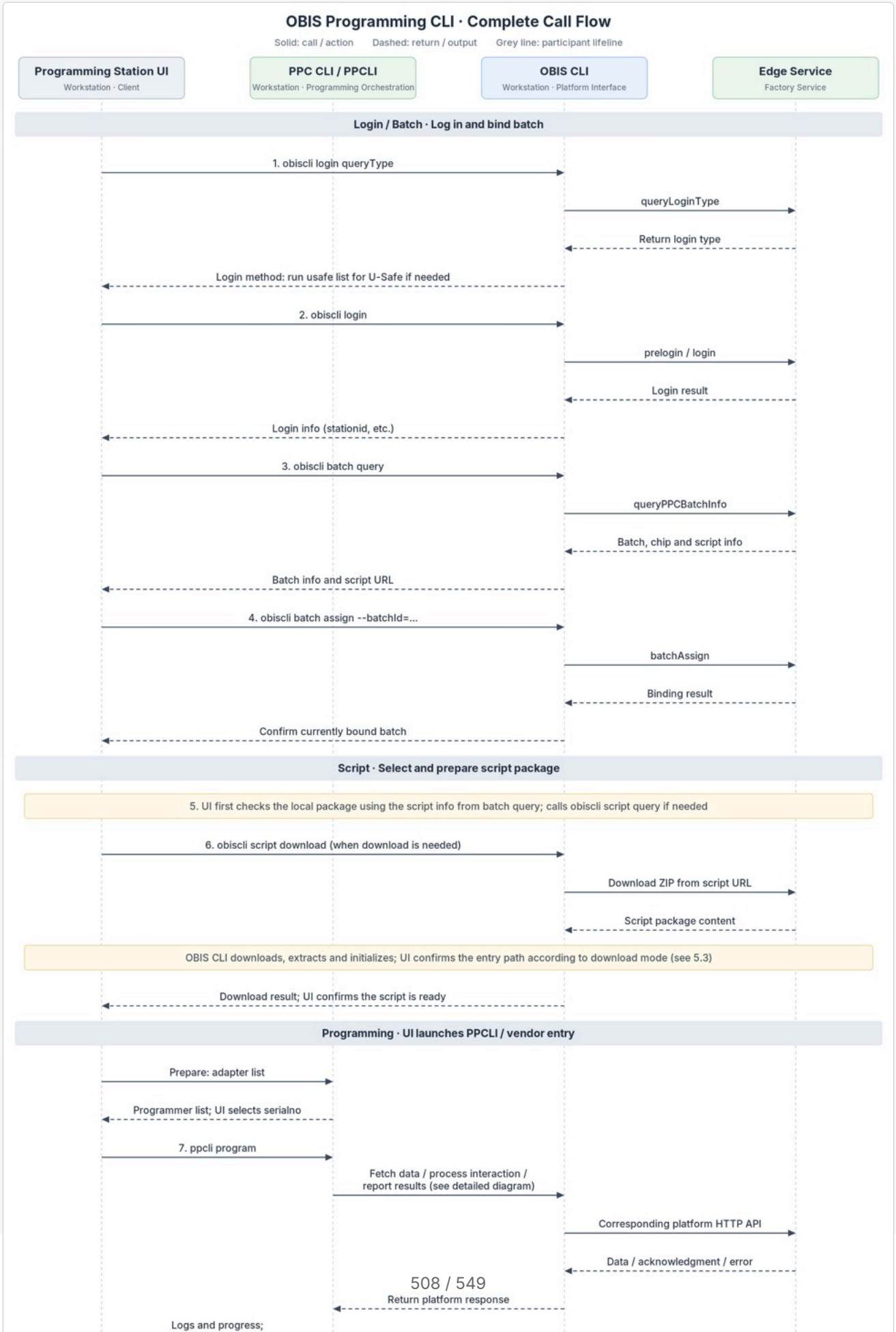
6.2 Programming Stage Call Flow

This section describes stage 7 of the overall flow, including how the UI starts programming, retrieves programming data, programs the device, and reports the result.

TIP

- getChangingData and getFixData are OBIS Programming CLI commands for changing and fixed device data. Their HTTP endpoints are getProgrammingData and getMetadata, respectively; do not confuse CLI command names with HTTP endpoint names.
- If fixed data is cached, getFixData may be skipped.
- The chip and batch configuration determine device erase, Device ID retrieval, CSR/DAC processing, Factory Data writing, firmware programming, reset, and other steps.
- Non-Matter devices or flows without device-generated CSRs may not include CSR/DAC processing.
- putdata exchanges data and requests/responses during programming, for example CSR/DAC requests; its HTTP endpoint is uploadRequestData.
- putresult reports the final programming result; its HTTP endpoint is programmingResult. putdata and putresult have different purposes and are not interchangeable.

The diagram shows a typical Python PPCLI flow. The internal flow of the dedicated Telink interface depends on the delivered version.


OBIS Programming CLI · Complete Call Flow
Solid: call / action
Dashed: return / output
Grey line: participant lifeline
Programming Station UI
Workstation · Client
PPC CLI / PPCLI
Workstation · Programming Orchestration
OBIS CLI
Workstation · Platform Interface
Edge Service
Factory Service
Login / Batch · Log in and bind batch
1. obiscli login queryType
queryLoginType
Return login type
Login method: run usafe list for U-Safe if needed
2. obiscli login
prelogin / login
Login result
Login info (stationid, etc.)
3. obiscli batch query
queryPPCBatchInfo
Batch, chip and script info
Batch info and script URL
4. obiscli batch assign --batchId=...
batchAssign
Binding result
Confirm currently bound batch
Script · Select and prepare script package
5. UI first checks the local package using the script info from batch query; calls obiscli script query if needed
6. obiscli script download (when download is needed)
Download ZIP from script URL
Script package content
OBIS CLI downloads, extracts and initializes; UI confirms the entry path according to download mode (see 5.3)
Download result; UI confirms the script is ready
Programming · UI launches PPCLI / vendor entry
Prepare: adapter list
Programmer list; UI selects serialno
7. ppcli program
Fetch data / process interaction /
report results (see detailed diagram)
Corresponding platform HTTP API
Data / acknowledgment / error
Return platform response
Logs and progress;

6.3 Complete Command Example

This example shows the basic command sequence from Edge Service login through device programming. It uses Python PPCLI and Windows `cmd.exe` syntax. Set actual parameters according to the site, batch, and programming script package. Confirm that each step succeeds before proceeding. Python PPCLI and the dedicated Telink interface are alternative entry points; select the one provided by the package.

```
REM 1. Query login type; run obiscli usafe list if needed for U-Safe login
obiscli login queryType --ip="<EDGE_IP>" --port=32333
f
REM 2. Log in using the type returned by queryType; this example uses passwordless login
REM For password login add --password="<PASSWORD>"; for U-Safe login add --usafesn="<USAFE_SN>"
obiscli login --ip="<EDGE_IP>" --port=32333 --stationVersion="<UI_VERSION>"

REM 3. Query batch information including batchId, chips, and scriptUrl
obiscli batch query

REM 4. Assign the current batch
obiscli batch assign --batchId="<BATCH_ID>"

REM 5. Get scriptUrl from the batch response; check for a matching prepared local package

REM 6. Download when no usable local package exists
obiscli script download --url="<SCRIPT_URL>"

REM Locate the actual entry directory from the download result and change directory
cd /d "<SCRIPT_ENTRY_DIRECTORY>"

REM 7 (prepare). List adapters and select SERIALNO
".\python-3.12.9-embed-amd64\python.exe" ".\ppcli.py" adapter list --chipVendor="<CHIP_VENDOR>" --type=1

REM 7. Start device programming
".\python-3.12.9-embed-amd64\python.exe" ".\ppcli.py" program --chipVendor="<CHIP_VENDOR>" --chipPart="<CHIP_PART>"
--serialno="<SERIALNO>" --speed=115200 --type=1

REM Check the exit code after program completes; the UI should read the child exit code directly rather than rely on
this echo
echo %ERRORLEVEL%

REM Refresh batch statistics if needed
obiscli batch query

REM Log out after programming and result reporting finish on all channels
obiscli logout
```

When using the dedicated Telink interface, replace only the stage 7 adapter query and programming commands:

```
REM 7 (prepare). List Telink adapters and select SERIALNO
".\Cmd\_download\_tool.exe" adapter list --chipVendor="TELINK" --type=telinkevk

REM 7. Start device programming
".\Cmd\_download\_tool.exe" program --serialno="<SERIALNO>"
```

The login, batch assignment, and programming script preparation flow remains the same. Use the executable path and working directory from the delivered Telink package.

6.4 Programming Script Download and Entry Directory

This section explains how the UI locates the actual programming entry point after downloading the script.

When only `--url` is passed, `script download` uses the default cache directory. The `scriptExist` and path values in its response indicate whether the script already exists and where it is located.

If both `--savePath` and `--extractFolder` are specified, the UI should not assume the command output contains the actual entry path. Locate the entry point from the chosen extraction directory and package structure, and verify that the entry executable and dependencies exist.

`--savePath` specifies the downloaded file directory. A programming script ZIP can contain one or more nested directories, so the download directory may differ from the actual programming entry directory.

Before starting, perform the initialization and dependency checks required by the package.

6.5 UI Process and Result Handling

This section describes process startup, output reading, and result handling when the UI calls PPCLI or a dedicated Telink programming executable.

Item	Integration requirement
Process startup	Set `executable` (path), `args[]` (arguments), and `cwd` (working directory) separately. Prefer an absolute executable path and `shell=false` to start the target process directly. Argument values in `args[]` need no extra shell quoting.
Parameter sources	Chip vendor and model come from the selected batch; `serialno` comes from `adapter list` of the selected programming entry point. Determine type, speed, and entry path from package configuration.
Output reading	Read `stdout` and `stderr` continuously and asynchronously until the process exits and output reading completes, so an unconsumed stream cannot block the process. Python PPCLI usually uses UTF-8; check the actual output encoding of older EXE versions.
Adapter output	Parse the complete `adapter list` response as defined by the package. Handle an empty list, missing fields, unexpected output, and nonzero exit codes. Parse multiline JSON as a complete response, not as separate JSON on each line.
Programming output	Programming output may contain logs and events; do not parse all `stdout` as a single JSON object. The Python PPCLI version reviewed may emit `[Progress]:<value>%` or `[Finish].` for progress or completion and `{"errMsg":"..."}` on failure. Other packages may use different formats.
Success criteria	Wait for normal process termination, then evaluate the exit code and final result defined by the package. Reaching 100% progress or emitting `[Finish].` indicates progress or flow completion only; do not release the channel early or treat either alone as proof that the platform received the result.
Result reporting	Check whether a `putresult` reporting failure affects the script's final status. If not, verify the platform record separately. A change in batch statistics alone does not prove that a particular `uuid` was reported successfully.
Failure, timeout, or cancellation	Retain serialno, package version, exit code, `stdout`, `stderr`, and any `uuid` obtained. Never mark a timed-out or terminated process successful. Before retrying, check device state and the previous result to avoid obtaining duplicate programming data or certificates.
Concurrency	One active programming task corresponds to one `serialno`. Actual concurrency depends on the package, driver, and service. While tasks share an OBIS login session, do not switch the current batch, log out, or replace the package in use.

6.5.1 UI Process Invocation Example

This example shows how the UI organizes process arguments when starting Python PPCLI. It does not define new CLI parameters or a fixed interface format:

- `executable`: path to the Python executable;
- `args`: commands and arguments passed to Python PPCLI;
- `cwd`: actual entry directory of the programming script;
- `shell`: `false` in this example, so the target process starts directly without running the full command through a shell.

```
{
  "executable": "<ENTRY_DIRECTORY>\\python-3.12.9-embed-amd64\\python.exe",
  "args": [
    "<ENTRY_DIRECTORY>\\ppcli.py",
    "program",
    "--chipVendor=<CHIP_VENDOR>",
    "--chipPart=<CHIP_PART>",
    "--serialno=<SERIALNO>",
    "--speed=115200",
    "--type=1"
  ],
  "cwd": "<ENTRY_DIRECTORY>",
  "shell": false
}
```

6.6 Integration Verification

This section lists key scenarios to verify after integrating the programming interface into the UI.

Verification item	Verification scope
Normal flow	Verify login, batch query and assignment, script preparation, adapter query, single-device programming, process exit status, platform record confirmation, and logout.
Parameters and paths	Verify directories and chip vendor names containing spaces, preservation of leading zeros in `serialno`, and correct executable, working directory, and `type` mapping.
Device errors	Verify that the UI ends tasks correctly and retains errors when no device is present, a device is unplugged, `serialno` is wrong, or a device is busy.
Platform errors	Verify that device data retrieval, certificate request, and result reporting failures are distinguished and that reaching 100% progress alone is not treated as success.
Multiple devices	Verify that cached and freshly downloaded packages both start; `serialno` and `uuid` stay separate across channels; and the shared batch and login session remain unchanged during programming.

Version: v3.2.1

Glossary in OnBoard™ IoT Security

B

BYOK (Bring Your Own Key)

A key management model in which the customer retains ownership and control over cryptographic key generation, while importing or delegating keys to an external platform for managed use.

C

CLI (Command-Line Interface)

A text-based interface for interacting with software systems through typed commands, commonly used for automation, scripting, and integration with development and production workflows.

CSR (Certificate Signing Request)

A cryptographic request generated by a device or secure environment containing a public key and identity information, used to request a digital certificate from a Certificate Authority (CA).

D

Device Identity

The cryptographic identity assigned to a device during provisioning, typically represented by a unique key pair and corresponding certificate.

Digital Assets

Product-related artifacts used during device production and provisioning, including firmware, cryptographic keys, certificates, and configurations.

E

EdgeHSM

A hardware-protected cryptographic execution device deployed at factory sites, designed to perform security-sensitive operations locally under externally issued authorization.

Edge Service

The Edge Service of OnBoard™ IoT Security provides on-site production management capabilities for factories. It is typically deployed on on-premises servers along the production line, where it caches, processes, and stores batch data and device data. It also provides cryptographic services through the Edge HSM.

H

HSM (Hardware Security Module)

A tamper-resistant hardware device designed to securely generate, store, and use cryptographic keys.

HSM U-Safe

A hardware security token used for authentication when logging in to OnBoard Edge Service.

I

Identity U-Safe

A hardware security token used for authentication when logging in to the Programming Station client.

K

KMS (Key Management System)

A system for managing the full lifecycle of cryptographic keys, including generation, storage, distribution, rotation, revocation, and destruction.

M

MES (Manufacturing Execution System)

A factory-side system used to manage, monitor, and control manufacturing operations, including production scheduling, process tracking, and quality control.

O

OBIS

OnBoard™ IoT Security, a security service platform built by Snowball for IoT scenarios. It is a security control plane that governs how product digital assets may be used across the device manufacturing lifecycle.

OBIS CLI

OnBoard™ IoT Security CLI establishes a secure channel with the Local Service using authentication provided by U-Safe for PPC, and retrieves the corresponding device data from the Local Service for use by upper-layer applications.

OBIS Edge Web Portal

The portal website of OBIS Edge, through which factory managers can manage on-site operations for factories visually.

OBIS Programming Station software

The controlled, last-mile software environment installed on a factory Programming Station where devices are physically connected, programmed, and initialized.

OBIS Web Portal

The portal website of OnBoard™ IoT Security. Users can perform product-related operations and management through this website, such as product management, factory management, user management, key management, and certificate management.

ODM (Original Design Manufacturer)

A manufacturer that provides end-to-end product design and manufacturing services on behalf of an OEM. ODMs typically develop products based on their own reference designs or platforms, customized to OEM specifications, with the final product shipped under the OEM's brand.

OEM (Original Equipment Manufacturer)

The entity that designs, owns, and markets a product under its own brand. The OEM bears ultimate responsibility for product specifications, quality, and regulatory compliance, even when some or all manufacturing activities are delegated to external partners.

OIDC (OpenID Connect)

An identity layer built on top of the OAuth 2.0 protocol, used for authentication and single sign-on (SSO). OBIS supports OIDC-based integration with enterprise identity and access management systems.

P

PKI (Public Key Infrastructure)

A framework of technologies, policies, and procedures for managing digital certificates and public-private key pairs, enabling secure authentication, encryption, and digital signing.

PPC (Programming Process Control) CLI

Programming Process Control Command Line Interface, a command-line tool used for executing programming process control tasks.

Production Order

A production order (also referred to as a Work Order or Manufacturing Order depending on the system) is a directive created in the OEM's SAP, ERP, or MES platform that authorizes and tracks the manufacturing of a specific quantity of products.

Programming CLI

Programming Command Line Interface, a command-line tool used for executing device programming.

Programming Station client

The factory programming client used on production lines to perform chip programming operations.

S

Secure Provisioning

The end-to-end process of securely initializing a device with its identity, cryptographic materials, firmware, configurations, and security settings required for authenticated operation and lifecycle management.

SnowBall

Shanghai Snowball Digital Intelligence Technology Co., Ltd., referred to as Snowball. A technology company with deep expertise in secure device provisioning and cryptographic infrastructure.

SSO (Single Sign-On)

An authentication mechanism that lets users access multiple systems using a single set of credentials.

Version: v3.2.1

FAQs in OnBoard™ IoT Security

This article lists the issues that you may encounter when using OnBoard™ IoT Security (OBIS) and provides corresponding solutions.

Account & Sign-in

How to self-register for an OBIS account?

When your organization has enabled the self-registration feature, click **Sign Up** on the OBIS sign-in page. After completing the registration, you can sign in to OBIS by using the verification code method.

Why can't I self-register for an OBIS account?

If you see the message "Your organization has not enabled self-registration. Please contact your admin", please contact the System Manager to confirm whether the self-registration feature has been enabled for your organization.

Why can't I sign in to OBIS through a verification code?

- If you see the message "The email address does not exist. Please check and try again", please contact the System Manager to confirm whether an OBIS account has been created for your email address.
- If you see the message "This account has been disabled. If you have any questions, please contact the administrator", please contact the System Manager to check your account status. Only accounts with the **Active** status can sign in to OBIS. For more information, refer to Manage accounts.

Why can't I sign in to OBIS through SSO?

- If you see the message "Your organization has not enabled SSO", please contact the System Manager to check if SSO has been enabled for your organization.
- If you see the message "SSO service is currently unavailable. Please try again later", please contact the System Manager to check if there are any issues with the SSO service of your organization.
- If you see the message "Your organization does not allow automatic account creation via SSO. Please contact your admin", please contact the System Manager to check if the automatic account creation via SSO feature has been enabled for your organization.

User & Role

Can a user be assigned multiple roles, and can these assignments be updated?

- The System Manager can assign multiple OBIS roles to a user and modify these assignments afterward. For more information, refer to Manage accounts.
- The Product Owner and Product Manager can assign multiple product roles to a user and modify these assignments afterward. For more information, refer to Manage members.

Can users and user groups from other tenants be added to a product team?

Users from other tenants can be added, but user groups can't be added across tenants. Only users with the **Active** status can be added.

For example, a Product Manager from Tenant A can add a user from Tenant B to their product team. For more information, refer to Manage members.

Why do I see the message "Cannot be invited. Please check the address and try again" when adding a user to a product team?

Please check whether the user already has an OBIS account. Only OBIS users can be invited to join a product team. Contact the System Manager to create an account for the user before attempting to add them.

Product

Why are some products in the product list not clickable?

The product list includes both the products you are involved in and all products created within the tenant. Please check whether you are a member of the product team for the specific product. If not, contact the Product Owner or Product Manager of the product to invite you to the team and assign a role. Once added, you will receive an email containing a link, which you can use to access the product details.

Why is the product status shown as Draft?

After a product is created, its initial status is **Draft**. Once a version is created, the product status will change to **Released**. For more information, refer to the description of product status.

Which chips are currently supported by OBIS?

For the detailed list, refer to the currently supported chip settings. Additional chip vendors, series, and parts will be added in the future based on demand.

What does "Mandatory" mean when configuring Factory Data for a product?

If the checkbox in the **Mandatory** column for a Factory Data item is selected, members are required to enter a value for that Factory Data when creating a version.

How can I check the QR code after a device is programmed?

After the device has been programmed, you can check its QR code on the Device tab of the product.

Version

Why is the Version tab grayed out?

If the Version tab appears grayed out, it is currently unavailable. Please check whether all required resources and configuration items for your product have been configured. Once everything is properly configured, the **Version** tab will be enabled automatically.

Why am I unable to edit or delete a version?

Before editing or deleting a version, please ensure that:

- The version isn't in the **Pending for approval** status (versions in **Pending for approval** can't be edited or deleted).
- This version isn't associated with any batches (once a version has been used to create batches, it can't be edited or deleted).

Batch

Why is the Batch tab grayed out?

If the **Batch** tab appears grayed out, it is currently unavailable. Please check whether a version has been created. After a version is created, the **Batch** tab will be enabled automatically.

Why are some versions grayed out and can't be selected when creating a batch?

When creating a batch, only versions with the **Approved** status can be selected. If version approval is enabled for the product, versions created in the PROD environment will require approval (approve or reject) by the Product Owner or Product Manager.

- Once the version is approved, the version status changes to **Approved** and it can be used to create a batch.
- Once the version is rejected, the version status changes to **Rejected**, and the version needs to be updated and resubmitted for approval.

Factory

Why is the factory status shown as Pending and can't be selected when creating a batch?

When creating a batch, only factories with the **Normal** status can be selected. After a local factory is created, its initial status is **Pending**. The status will change from **Pending** to **Normal** after adding HSM U-Safe to the local factory. For more information, refer to the description of factory status.

Why am I unable to unbind the U-Safe?

A U-Safe can't be unbound if it is locked. For more information, refer to unbinding, locking, or unlocking HSM U-Safe and Identity U-Safe.

Certificate

Why am I unable to delete a certificate?

- If you see the message "Certificate cannot be deleted. It has the following sub certificates" along with a list of the sub-certificates, you must first delete all sub-certificates under the certificate.
- If you see the message "Certificate cannot be deleted, the certificate is currently used by the following list," along with a list of the referencing resources, you must first remove all references to the certificate.

OnBoard Edge Service & Programming Station

How can I check the service address used by the local factory for logging in to OnBoard Edge Service and the Programming Station client?

After contacting Snowball for assistance in deploying OnBoard Edge Service, you can find the **Service Address** in the **Service Info** area on the **Batch** tab of the **Factory Details** page. For more information, refer to Check the batch.

Why am I unable to sign in to OnBoard Edge Service?

Please check whether the prerequisites for logging in to OnBoard Edge Service have been met.

How can I check the password for logging in to the Programming Station client?

The initial sign-in password is randomly generated by the system after a programming station is added to the factory. You can check the sign-in password in OBIS (local factory & cloud factory) or OnBoard Edge Service (local factory). Currently, the randomly generated password can't be changed.

Version: v3.2.1

Release Notes

This article provides an overview of the release dates, key feature updates, and related documentation for OnBoard™ IoT Security (OBIS), covering versions up to **v3.2.1**. For CLI and API version updates and changes, refer to the revision records in the corresponding documentation.

2026 Release Versions

v3.2.1 (January 15, 2026)

Version **v3.2.1** introduces updates to the Registration and Sign-in, User Group, Product, Version, Batch, Factory, KMS, OnBoard Edge Service, and Programming Station features.

Feature Categories	Key Updates	Description	Related Documents
Registration and Sign-in	Added **Sign in with SSO** and **Self-Registration** features	• Sign in with SSO: Sign in to OBIS directly using your company's IdP account, without needing to remember a separate OBIS account • Self-Registration: Register with your company email address and automatically receive a default role, enabling quick access to your organization's tenant	Registration and Sign-in

Feature Categories	Key Updates	Description	Related Documents

User Group	Added the **User Group** menu	• User group management: Enables unified user management by organizing users into groups
		• Product permission assignment: Allows user groups to be added directly to a product, granting product access to all group members at once
			• Product - Manage members
			• User:
			  ◦ Group:
			  ▪ Create a group
			  ▪ Check groups
			  ▪ Manage groups

Feature Categories	Key Updates	Description	Related Documents

Product	• Splits the key and certificate resource package
	• Supports downloading product resource files
	• Added product archiving feature
	• Removed the Mark field when creating a product
		Split the key and certificate resource package:
		• You need to add all certificates and keys used by the product to the corresponding resource packages in advance
		• The product's Resource Manager allows you to select the required certificates or keys from the resource packages within configuration items (such as Matter Config and Chip Config)
		
		Download product resource files:
		• Product members can download uploaded files
		• Downloads are supported on the resource
			• Quick Start
			• Product:
			  ◦ Create a product
			  ◦ Configure a product
			  ◦ Manage products

Feature Categories	Key Updates	Description	Related Documents
		configuration page, including Firmware, CD files in Matter Config, files in Factory Data, Programming Scripts, Attachment Files, and Other Files Product archiving feature: • Inactive products can be archived or unarchived, with their status clearly displayed in the product list. Archived or unarchived products can be quickly filtered • Archived products can't be edited (including information modification, resource configuration, version or batch creation, etc.), but programming of already created	

Feature Categories	Key Updates	Description	Related Documents
		batches remains unaffected Remove the Mark field from the product: • The Mark field is no longer required when creating a product and will no longer appear on the product page • For existing products, the Mark field will automatically be appended to the product name	

Feature Categories	Key Updates	Description	Related Documents
Version	Supports tab switching, long names, mass production marking, and multi-select firmware addition	• Tab switching: When creating a version, you can switch tabs to check other product information • Version name: Supports a length of 128 characters • Mass production mark: You can mark a version as **Mass Production**. This mark will then appear in the batch list • Firmware addition optimization: ◦ Clicking **Add** displays a firmware list, supporting searching by filename or version ◦ Supports multi-selection for adding firmware, improving operational efficiency	Product - Create a version

Feature Categories	Key Updates	Description	Related Documents
Batch	• Supports tab switching, long names, and display of version mass-production information; simplifies factory selection and enables automatic script matching • Enhances batch and data synchronization status management, with a resynchronization feature	Batch optimization: • Tab switching: When creating a batch, you can switch tabs to check other product information • Batch name: Supports a length of 128 characters • Version information display: Displays whether a selected version is in mass production and its configuration region • Simplified factory selection: Only the factory needs to be selected; selecting a Factory Manager is no longer required • Automatic script matching: Automatically matches the programming script based on	Product: • Create a batch • Check batches

Feature Categories	Key Updates	Description	Related Documents
		the chip configured for the selected version Batch status optimization: • Batch status: Indicates the complete lifecycle of a batch, including **Generating**, **Generated**, **Generate Failed**, **Running**, **Completed**, **Stopped**, and **Expired** • Batch synchronization status: Indicates the status of batch data synchronization to the factory, including **Waiting**, **Syncing**, **Sync Failed**, **Synced**, and **Deleted** • Programming data synchronization status: Indicates the data synchronization	

Feature Categories	Key Updates	Description	Related Documents
		status of batch programming, including **No Records**, **Uploading**, and **Uploaded** • Resynchronization feature: For batches that fail to synchronize (batch status is **Generate Failed** or batch synchronization status is **Sync Failed**), click **Resync** in the batch list to resend the batch to the factory	

Feature Categories	Key Updates	Description	Related Documents
Factory	Supports creating cloud factories	• Cloud factory creation: Enables creating cloud factories in OBIS, including programming stations that support password or Identity U-Safe login • Cloud factory management: Allows management of cloud factory batches and programming station assignment in OBIS, including starting and stopping batches	Factory: • Create a factory • Manage factories and programming stations: Cloud factory: ▪ Configure and manage factories ▪ Configure and manage programming stations ▪ Manage batches and assign programming stations • Check factories

Feature Categories	Key Updates	Description	Related Documents
KMS	Added **Complete**, **Download**, and **Archive** options	• Key creation: Select **Complete** to proceed with key creation • Key download: Supports downloading the public key for asymmetric keys • Key archiving: Allows archiving of inactive keys. Active keys are displayed first, while archived keys are hidden by default for easier daily management	KMS - Manage keys

Feature Categories	Key Updates	Description	Related Documents
OnBoard Edge Service	• Added **Factory**, **Batch**, and **Station** menus • Supports multiple HSM U-Safes programming	• Factory management: Supports factory management within OnBoard Edge Service, including batch management, programming station assignment, and programming station configuration • Multiple HSM U-Safes programming: The Edge-Service server can accommodate multiple HSM U-Safes, enhancing programming performance and concurrency	OnBoard Edge Service: • Sign-in • Manage factories • Manage batches and assign programming stations • Configure and manage programming stations

Feature Categories	Key Updates	Description	Related Documents
Programming Station	• Added new login methods for cloud factories • Supports switching programming scripts and custom download paths • Added display and alerts for programming concurrent load	• Cloud factory sign-in: Allows sign-in using password or Identity U-Safe • Local programming script management: Supports switching programming scripts and customizing the script download path • Programming concurrent load alerts: ◦ The client displays real-time concurrent load in the bottom-left corner ◦ When the load ≥ 90%, a warning is displayed; at 100% load, a high-load alert appears, facilitating operation and	Programming Station: • Sign-in • Execute device programming

Feature Categories	Key Updates	Description	Related Documents
		monitoring	

Version: v3.2.1

How to configure SSO for Microsoft work accounts in OnBoard™ IoT Security

> **TIP**
>
> The Single Sign-On (SSO) feature in OnBoard™ IoT Security (OBIS) allows users to sign in with their company email through the Enterprise Identity Provider (IdP). OBIS will automatically create an account for them and assign default role permissions.

This article will guide you on how to configure and enable the Sign in with SSO feature in OBIS for Microsoft work accounts. Once enabled, users can sign in to OBIS using their work accounts and passwords.

Prerequisites

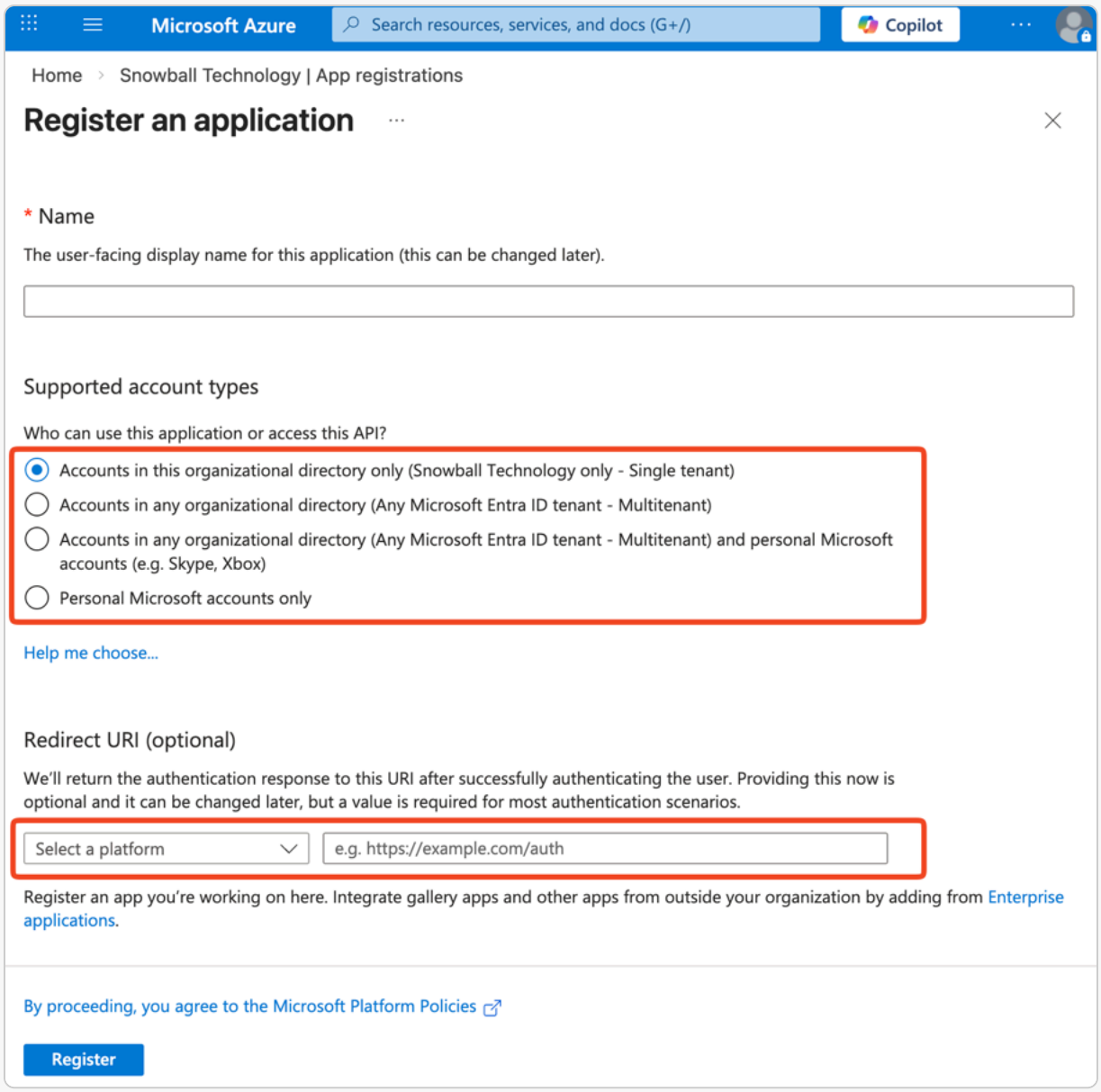
Before getting started, please ensure that you have obtained the Redirect URL required to configure SSO from Snowball.

Instructions

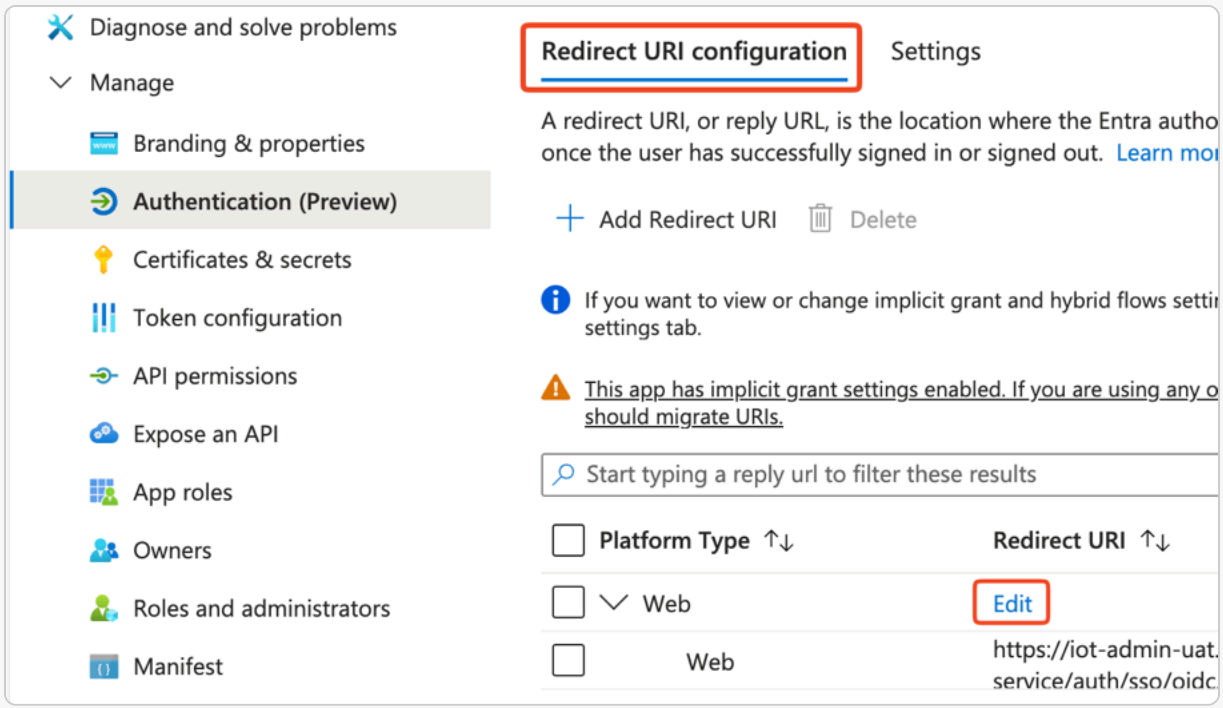
1. Sign in to the Microsoft Azure portal.
2. In the search box at the top, search for and select **Microsoft Entra ID**.

3. In the left menu, click **Manage** > **App registrations** > **New registration**. On the **Register an application** page, enter the application name and then select the **Supported account types**.

4. In the **Redirect URI** area, select **Web** as the platform, enter the Redirect URL obtained from Snowball, and then click **Register**.

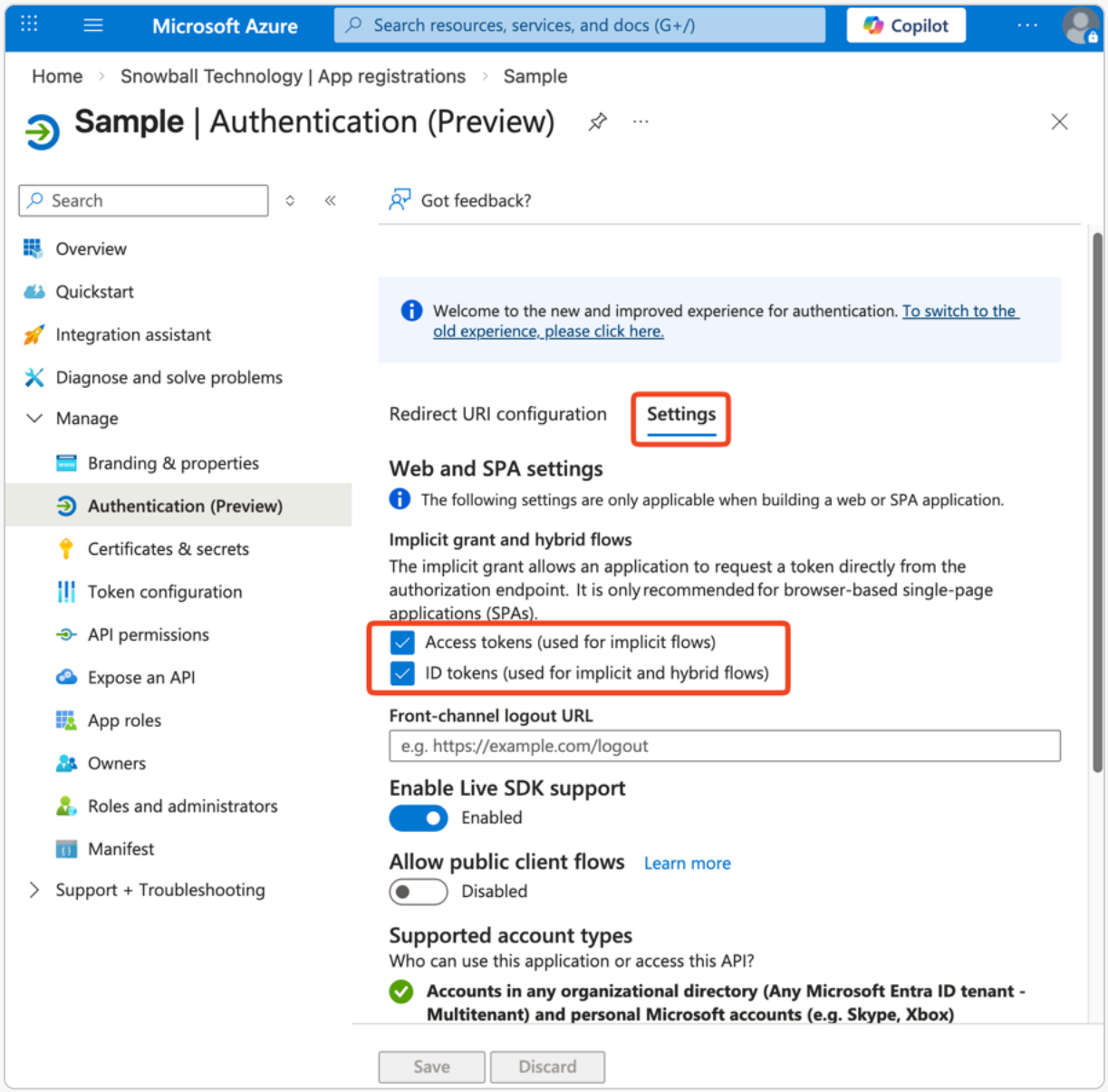


If you need to change the Redirect URI later, click **Manage** > **Authentication (Preview)** in the left menu of the application, and then make the edit on the **Redirect URI configuration** tab.



5. In the left menu of the application, click **Manage** > **Authentication (Preview)** > **Settings**, and then select the **Access tokens (used for implicit flows)** and **ID tokens (used for implicit and hybrid flows)** checkboxes.

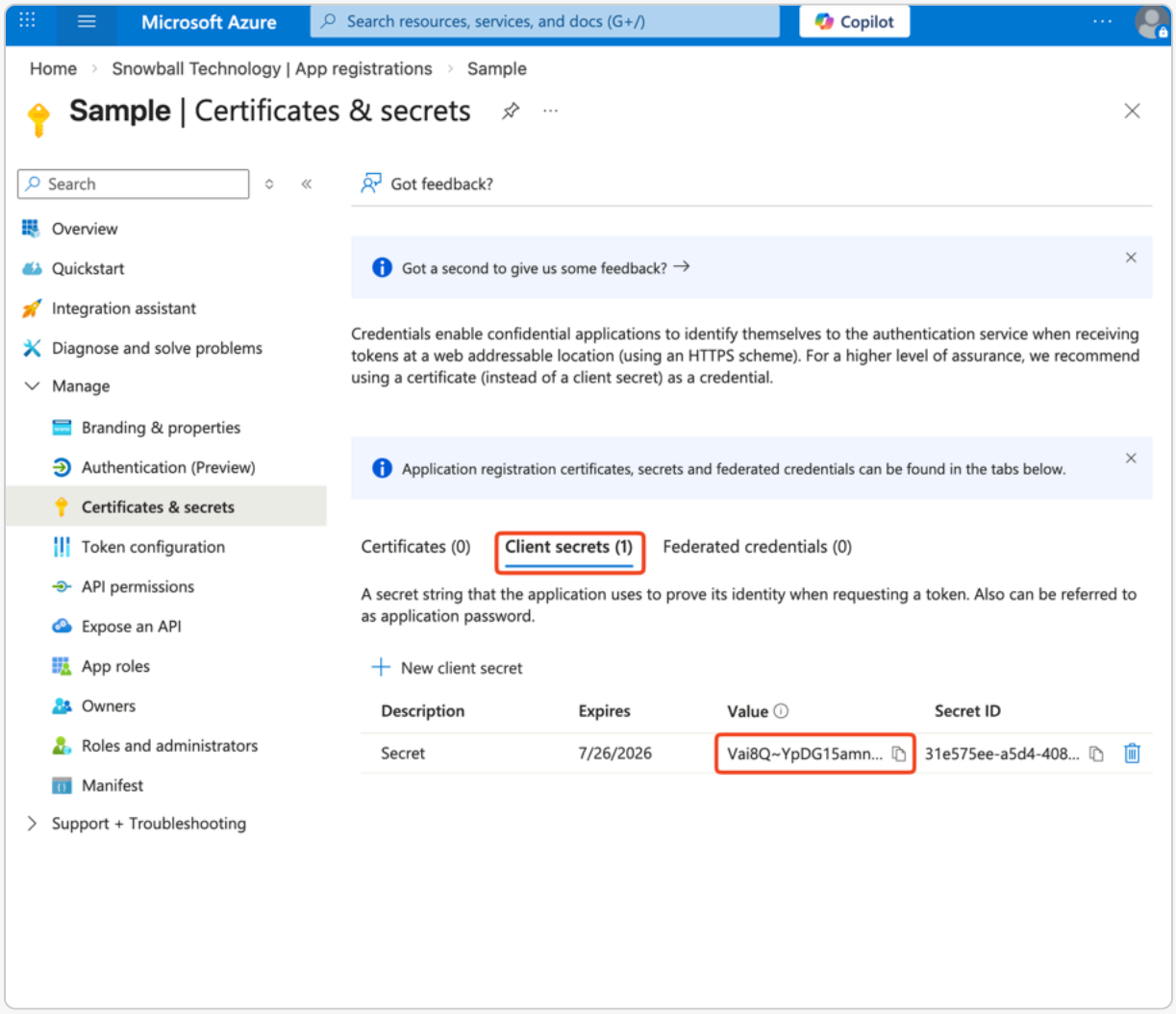
> **TIP**
>
> The Access token is used to call the API for retrieving user information, while the ID token is used for parameter validation during the SSO sign-in process.



6. In the left menu of the application, click **Manage** > **Certificates & secrets** > **Client secrets** > **New client secret**. On the **Add a client secret** panel, enter the client secret description, select the expiration date, and then click **Add**.

7. Record the client secret value. This secret value is never displayed again after you leave this page.

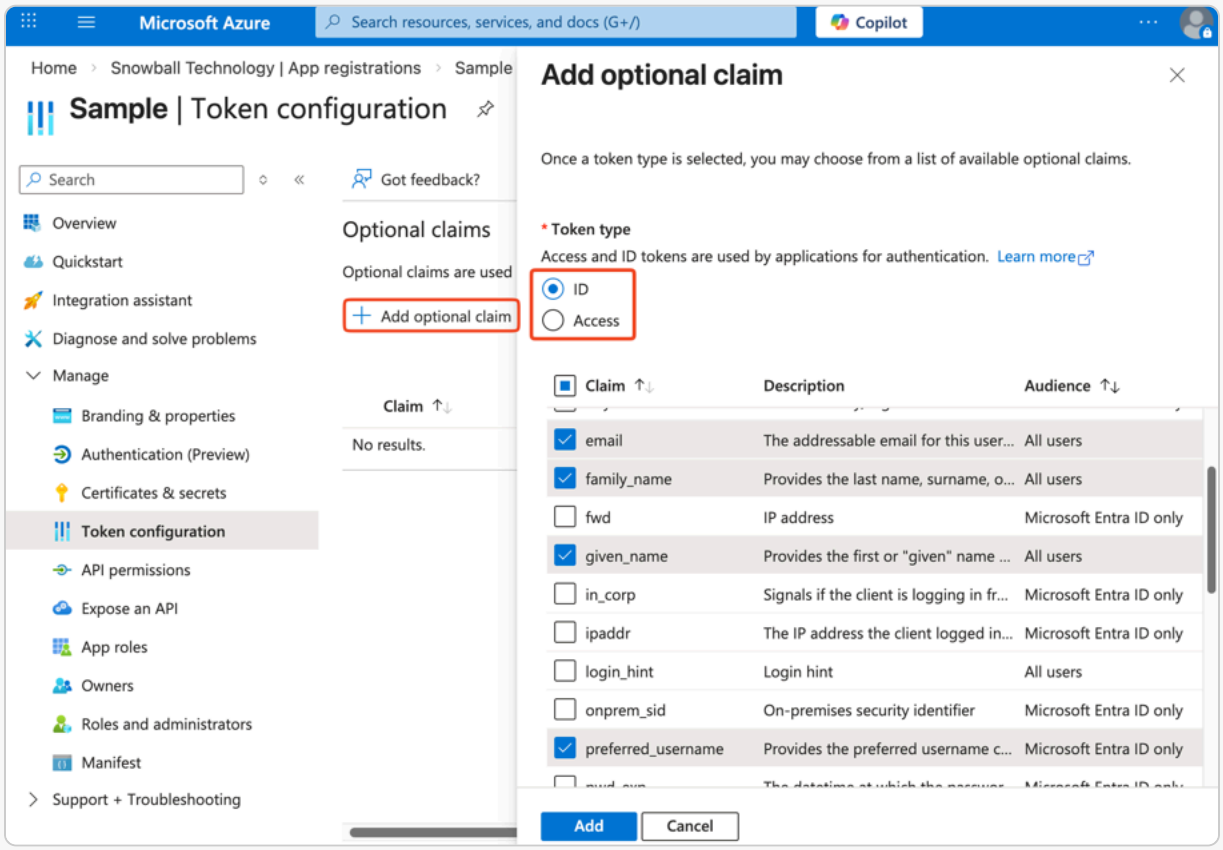
> **TIP**
>
> The client secret of the currently authorized application is used to verify application information through callbacks, ensuring that the source of the authorization request is trusted.



8. In the left menu of the application, click **Manage** > **Token configuration** > **Add optional claim**, then configure and add the ID token and access token separately.

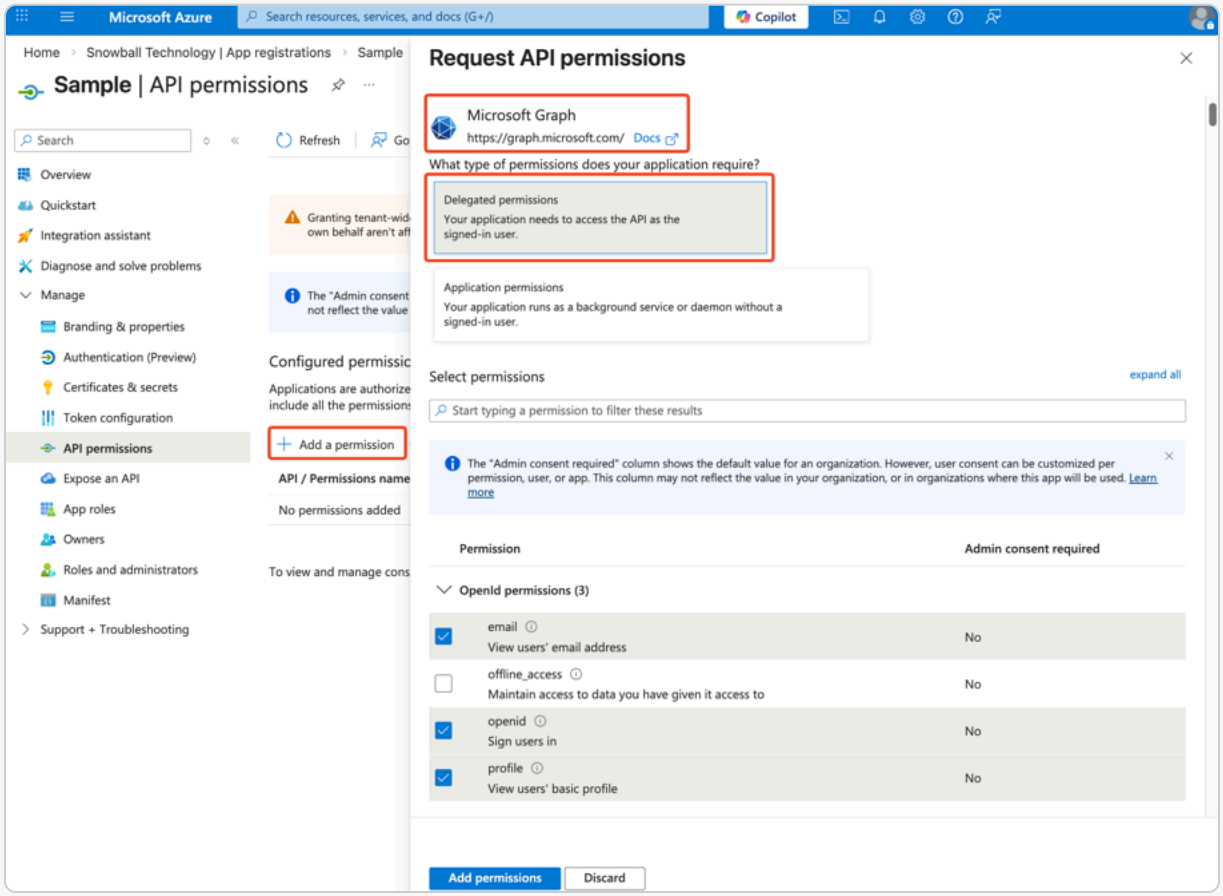
> **TIP**
>
> The optional claims correspond to the information returned by Microsoft to the platform during the authentication callback.

- Select **ID** as the **Token type**, and then select **email**, **family_name**, **given_name**, and **preferred_username**
- Select **Access** as the **Token type**, and then select **email**, **family_name**, **given_name**, and **preferred_username**



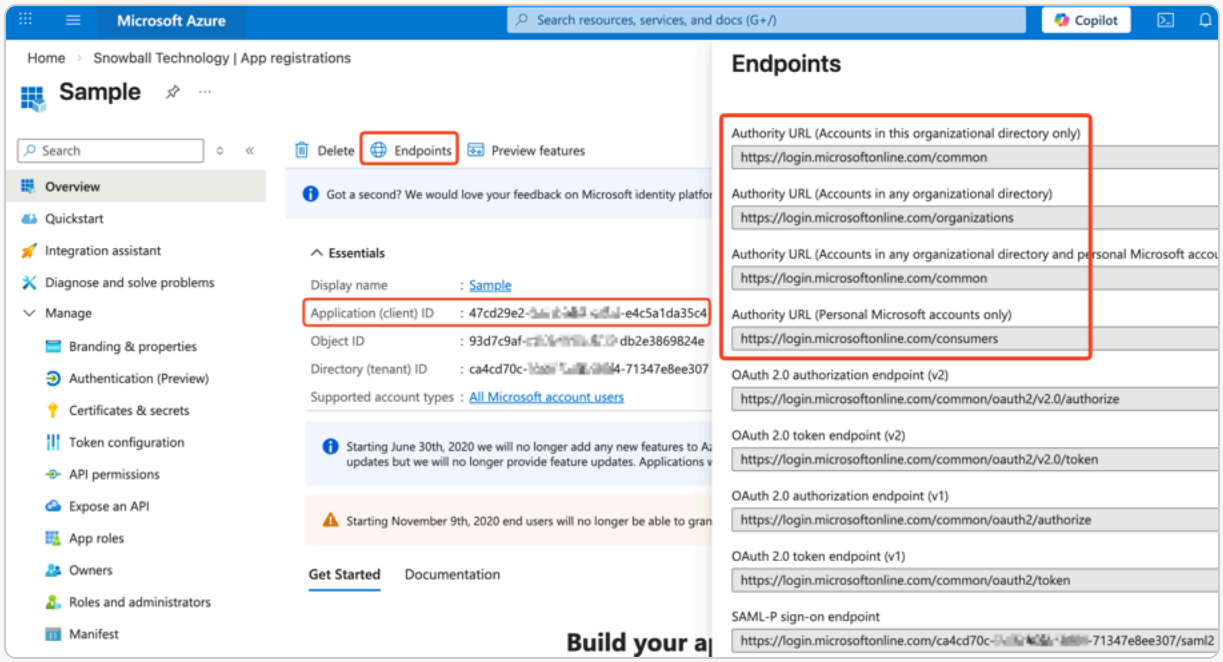
9. In the left menu of the application, click **Manage** > **API permissions** > **Add a permission**. On the **Request API permissions** panel, select **Microsoft Graph** > **Delegated permissions**. In the **OpenId permissions** area, select the corresponding permissions (it is recommended to select **email**, **openid**, and **profile**), and then click **Add permissions**.

> **TIP**
>
> The API permissions configured here correspond to the scope settings in OBIS for SSO.



10. Once the application configuration is complete, please provide the following information to Snowball, so that we can enable the **Sign in with SSO** feature for your tenant in OBIS.

    - **Email Domain**: The email domain of your Microsoft work account
    - **Issuer**: On the **Overview** page of the application, click **Endpoints**, and then copy the relevant **Authority URL**
    - **Client ID**: On the **Overview** page of the application, copy the **Application (client) ID**
    - **Client Key**: The client secret recorded in step 7



Make Security Simple

上海雪球数智科技有限公司

Shanghai Snowball Digital Intelligence Technology Co., Ltd

www.snowballtech.com

info@snowballtech.com

Floor 19, Building T2, Yuqiao Smart Bay, 968 Yuqiao Road, Pudong New Area, Shanghai

Visit Website

www.snowballtech.com

© 2026 Shanghai Snowball Digital Intelligence Technology Co., Ltd. All rights reserved.

This document is for reference only and does not constitute any warranty, express or implied.

To cite or use this document, please contact Shanghai Snowball Digital Intelligence Technology Co., Ltd. for authorization.