PKI

Production Public Key Infrastructure Establishment

Application Note

Abstract

This document describes the process to establish a production Public Key Infrastructure PKI. Starting from the theory of operation and the initial requirements and going through the steps needed to create the certificate authorities CA using the *PKI4mfg periMICA* container. The security chain of trust makes possible the usage of SSL certificates to be used in production of periCORE-based products.

Doc-No.: PRN.100.811 rev: 2 2025-02-03

Document Information

Title	PKI
Subtitle	Production Public Key Infrastructure Establishment
Type	Application Note
Status	Release
Version	2
Date	2025-02-03
Disclosure Restriction	

Contents

1 Overview

Cybersecurity, especially in the realm of IoT within automation industries, is a critical area that needs to receive adequate attention. Recognizing the complexity of security, *Perinet GmbH* is committed to simplifying this process to assist integrators in implementing state-of-the-art security measures and and ensuring compliance with the newly introduced European regulations, such as the Cyber Resilience Act (CRA).

To comply with the CRA hardware and software products need to integrate comprehensive security measures throughout their lifecycle, from design to disposal. The goal is to enhance the overall cybersecurity posture of digital products and protect users from cyber threats. For more information about CRA please read the consolidate online version [16].

Secure networks employ cryptographic operations based on digital signatures to provide secure communication between the participants. Trusted Certificate Authority (CA) distributes signed certificates to trusted parts in the network and the communication can be done based on a chain of trust. The secure certificates are fundamental to employing cryptography effectively across networks communication. The creation of a CA itself must adhere to stringent security measures to meet the necessary security requirements. It is crucial that the CA supports recovery functions and is distributed to avoid single points of failure as well.

The CA, while critical, is not solitary but a part of a broader Public Key Infrastructure (PKI). This infrastructure establishes a chain of trust among multiple CAs, ensuring that the authenticity and validity of certificates are maintained across the network. The establishment of a production PKI is the main subject of this document.

YubiKeys play a pivotal role in enhancing the security infrastructure within the PKI. As Hardware Security Modules (HSMs), YubiKeys provide a secure environment for generating and managing cryptographic keys. Their robust design prevents external tampering and unauthorized access, which is crucial for maintaining the integrity of the keys throughout their lifecycle. The utilization of YubiKeys ensures that cryptographic operations are performed in a secure hardware boundary, significantly reducing the risk of key exposure.

Figure 1: YubiKey Devices

This document is designed to facilitate the creation of a complete production PKI that intend to manage the production security certificates. These certificates are vital for authenticating *periCORE*-based products and for ensuring the integrity of code firmware signatures. The goal is having security measures from design to end-user operations.

The document follows with fundamental chapters like *The Theory of Operations* [2] where is explained the usage of the certificates in production and the *PKI* structure overall and the chapter *Initial Setup Requirements* [3] where the instructions to start are given. Finally, the chapter *Creating the Certificate Authorities* [4] brings the practical usage of *PKI4mfg* container and Yubikeys to create the complete production PKI.

2 Theory of Operations

This section presents the theory of operation. It starts with a brief clarify for each component and their interactions, followed by the security measures during the production phase, up until the device is integrated with the customer OEM. Ensuring security from the beginning of the product lifecycle aligns with the security by design principles advocated by the Cyber Resilience Act (CRA).

2.1 Understand the components and their roles

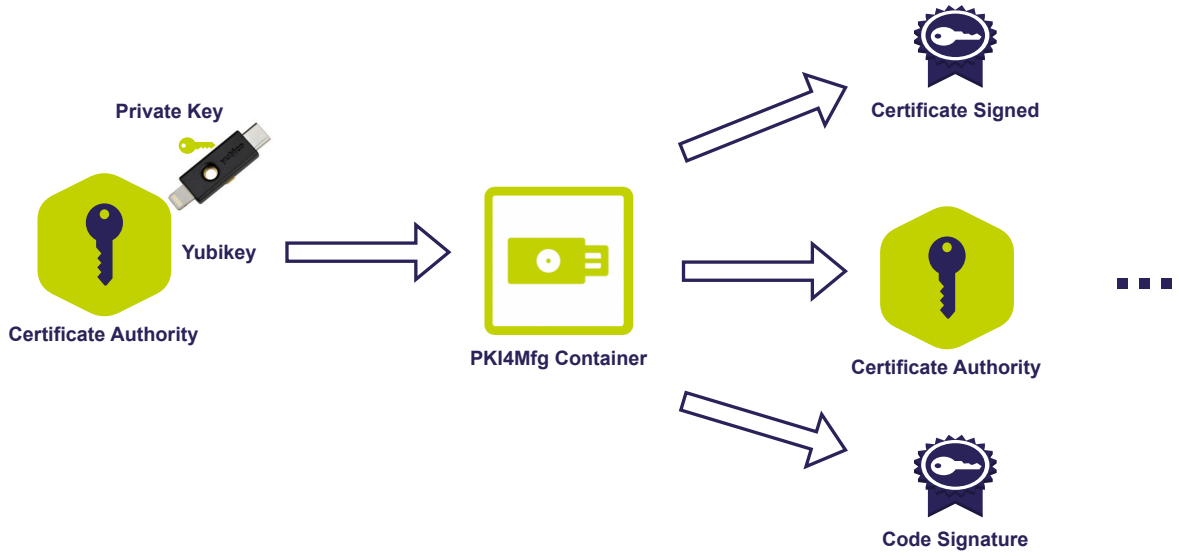

Figure 2: Components and their Roles

- *PKI4mfg Container* is the tool provided by Perinet, which operates Yubikeys to establish a production PKI. It deploys product certificates during production and generates code signatures to ensure secure updates.
- *Yubikey* is responsible for internally generating private keys and securely storing them. The usage of the private key to generate signatures or certificate sign request CSRs is done internally on the device, protecting the data.
- *Certificate Authorities* (CA) are entities that have their own private keys and identity. They are able to sign subsequent certificates, which can be other CA certificates or end certificates, such as product certificates or code signatures.
- *CA Certificate* is a certificate signed by an authority, or self signed ones in case of Root CAs, containing the subject name and the public key of the entity.
- *Product Certificates* are endpoint certificates that include the signature of the authority and the identification subject name and a determined validity time.
- *Code Signature* is a signature that proves the authenticity of the code (firmware, container, etc.). When a user installs an update, only code that can prove its authenticity

through the signature can be installed.

- *PKI* (Public Key Infrastructure) is the complete structure of CAs that comprise the secure infrastructure.

2.2 periCORE Attestation Certificate

During the periCORE production process, the firmware is flashed, and the OTP data, along with the periCORE attestation certificate for the OEM, is stored in the periCORE storage area (see Figure 3).

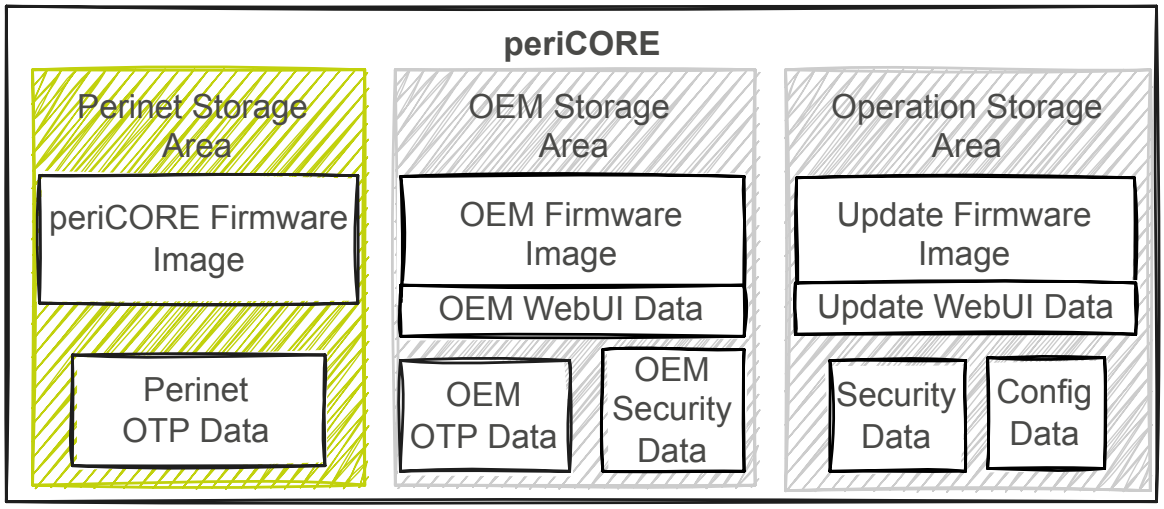

Figure 3: Structure of the persistent memory of a periCORE based product.

Each periCORE attestation certificate is signed by a *Perinet Batch CA YYYY-WW*, which is designated for that production batch. The certificate carries the *periCORE-sermn* hostname and also concatenate the chain of trust. For this reason, it can authenticate itself in the subsequent steps of its lifecycle. periCORE is now in the production state (see Figure 4).

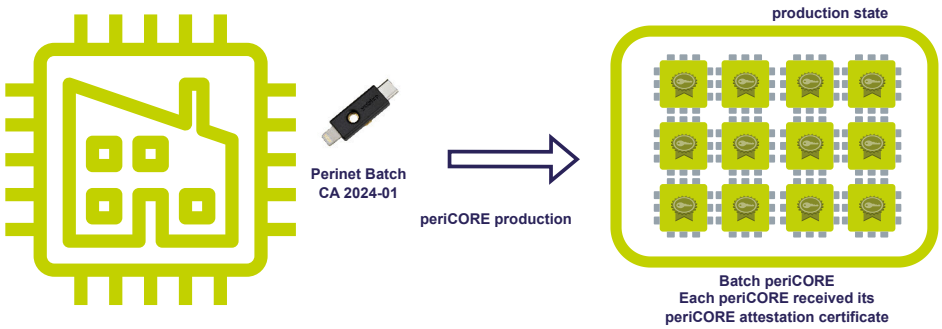

Figure 4: Each periCORE in production is patched with an attestation certificate

2.3 OEM Certificate

The OEM manufacturer intending to integrate periCORE should verify the periCORE attestation certificate against *Perinet Ecc Root CA* ensuring the originality of the periCORE, see Figure 5. At the end of production each produced device should receive a new certificate, which will be stored in the OEM Security Data Area (see Figure 3) trusted by the *Manufacturer Root CA*.

This process is similar with the periCORE production, where each device receives a certificate signed by a dedicated *Perinet Batch CA YYYY-WW* and patched with the chain of trust.

During the production, the device receives the production data according to the manufacturer, its name is changed, and from this point forward, its authenticity should be validated according to the manufacturer PKI and its *Manufacturer Root CA* (see Figure 5) and not any more by *Perinet Root CA*.

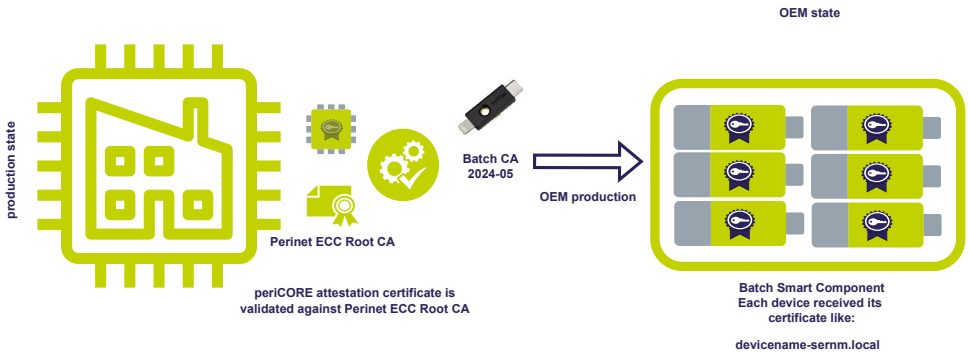

Figure 5: OEM Production

With the certificate installed the device is in the OEM production state, customers can ensure the authenticity of devices by validating against the *Manufacturer Root CA* certificate. The fact of the certificate stays in the *OEM Security Data* storage, enables customers to recover the device to the OEM state when necessary.

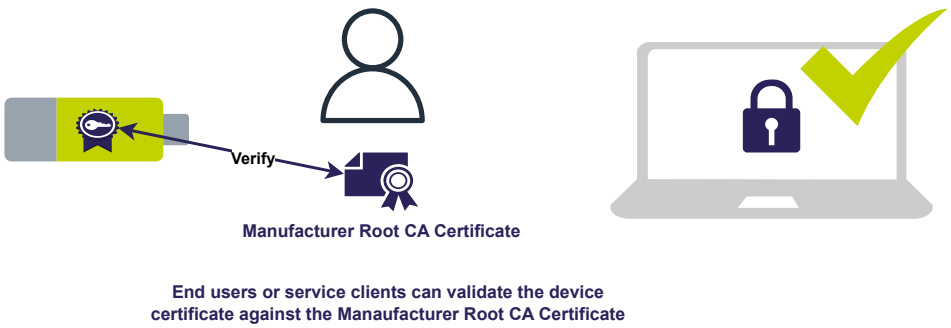

Figure 6: End clients can verify authenticity of device against Manufacturer Root CA Certificate

The *periCORE Attestation and OEM Certificates* are deployed in the production state and are first used by the periCORE customers (called here manufacturer) to authenticate the devices in production time.

During the normal lifecycle of the devices, security continues to play a role in the operation, and the users can even upload their own defined certificate as they need, or take advantage of the container PKI2go [12] restricting the device access through the Role Based Access Control RBAC and the Security API, but the OEM Certificate is never overwritten and it will take place again once the device is *reset*.

The next section provides the overview of the PKI designed by Perinet Products and used in this Application Note.

2.4 Public Key Infrastructure

The figure 7 illustrates the PKI Certificate Authority CA hierarchy utilized in this application note. This hierarchy can be adapted to meet specific organizational needs. However, the recommended hierarchy ensures distributed responsibilities throughout the company, enhancing preparedness for recovery when necessary.

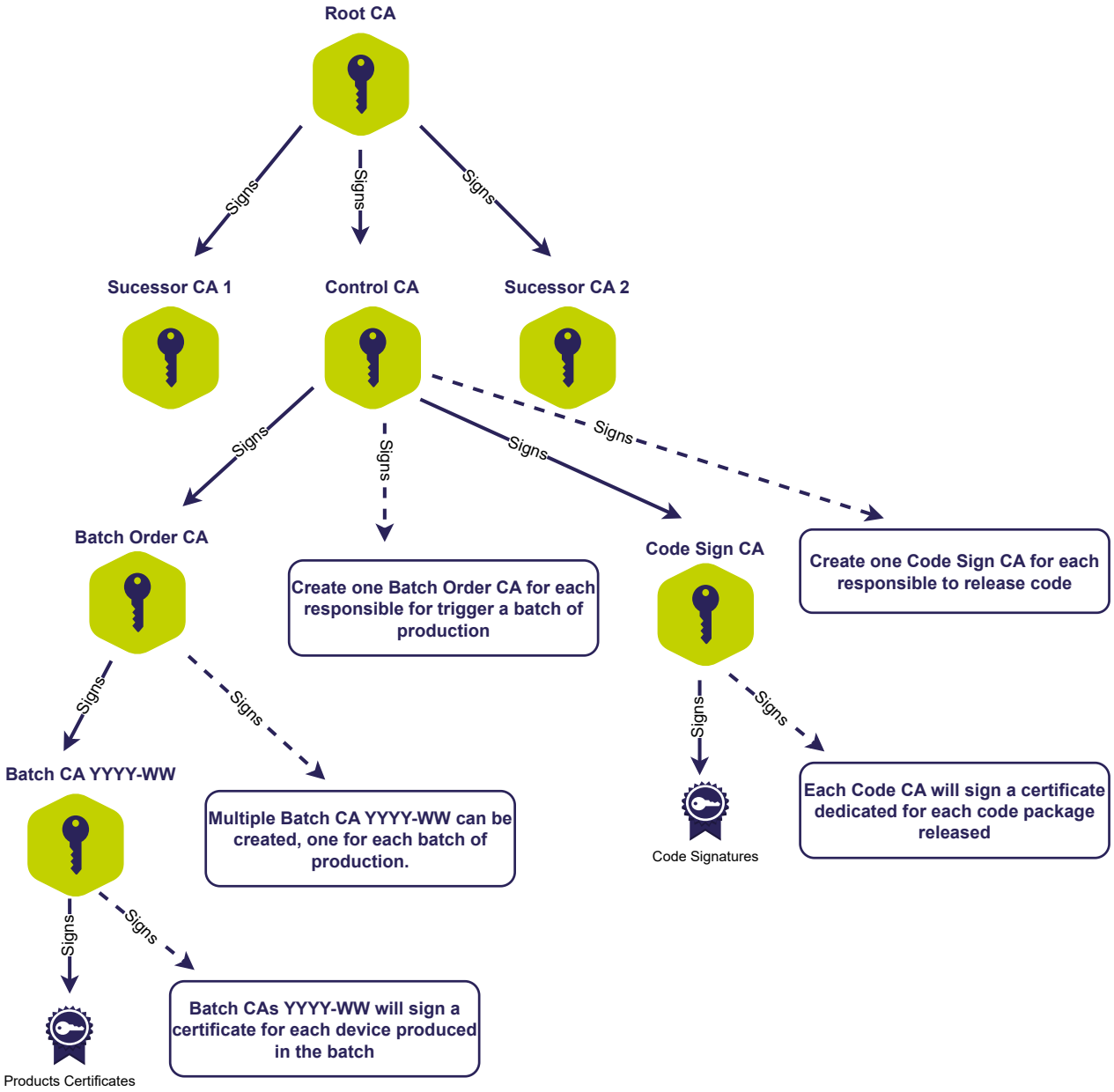

Figure 7: PKI Hierarchy Overview

A brief understanding of the structure is beneficial to follow the deployment. To clarify the structure used we present, the function of each CA is described below:

- *Root CA* is the top authority, its certificate is self-signed. Its public key should be published, and it will be used to recognize the authenticity of the device, service or firmware published in the future. The private key if this authority has a huge importance and should be kept in the most safe locate. Once created it is crucial that the minimum people have access to it.
- *Control CA* dedicated as well for small number of people and should be used to create new Batch Orders CA. The responsible to trigger a batch of production hold this key and will be able to create/sign the batch CA key.

- *Successor CA* is the successor CA that will be used in case of necessity of replacement of the Control CA.
- *Batch Order CA* the production manager of the company keeps this key and will be able to sign the batch CA certificate.
- *Code Sign CA* also signed by the Control CA, the responsible to sign the code and proof the authenticity the update data.
- *Batch CA* used in production time, each device created with this batch will have the certificate signed by this entity.

The chain of trust is established through hierarchical levels of authority. This means that if an entity is trusted, such as the Root CA, and an endpoint presents a certificate signed by an intermediary authority, which in turn is endorsed by the Root CA, the chain of trust is confirmed. Consequently, with this trust validated, operations involving the endpoint can proceed securely.

The application note continues with the initial requirements and follow with an example of creating a CA, providing specific commands applicable to each CA.

For further information, please refer to where additional documents are referenced. These resources can provide a deeper understanding of the security measures implemented in *Perinet* products.

Perinet ECC Root CA is available in https://perinet.io and it includes the public key to be used to verify the certificates signed direct or indirect by this authority.

3 Initial Setup Requirements

In order to be prepared to start the *Production PKI* establishment make sure to have the following items:

- Computer with a *browser* and *ssh client* and connection to *periMICA*
- periMICA with USB module (see section3.1) with basesystem version 24-01 or superior.
- PKI4mfg container (see section 3.2)
- YubiKey Devices (one) for each entity that should be created
- USB adapter (when necessary)

Note: Due to the sensitive nature of handling cryptographic keys and certificates, it is strongly recommended to perform the PKI establishment process in an offline environment. Conducting this process offline significantly reduces the risk of exposure to cyber threats and attacks, thereby fortifying the security of the entire system.

3.1 periMICA

Ensure that the datetime on *periMICA* is accurately set. This is crucial to guarantee that the certificates generated will have the correct datetime associated with them. On the *periMICA* homepage click on *Settings* → *Time & Date*, if necessary adjust the correct data (see figure 8).

Note: *periMICA* basesystem version should be at least *24-01*. Make sure to update the periMICA basesystem in order to proceed.

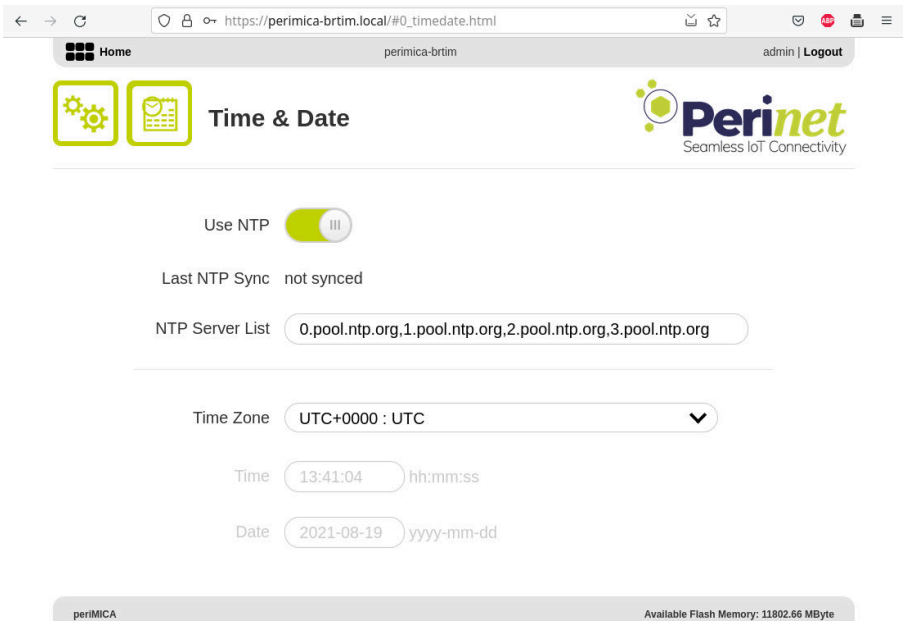

Figure 8: periMICA Time & Date

3.2 PKI4mfg Container

Download the PKI4fmg container from:

https://downloads.perinet.io/periMICA-containers/periMICA-container-pki4mfg-24-01.tar.

On *periMICA* Homepage click on *Install*:

Figure 9: periMICA Install Icon

Start the container.

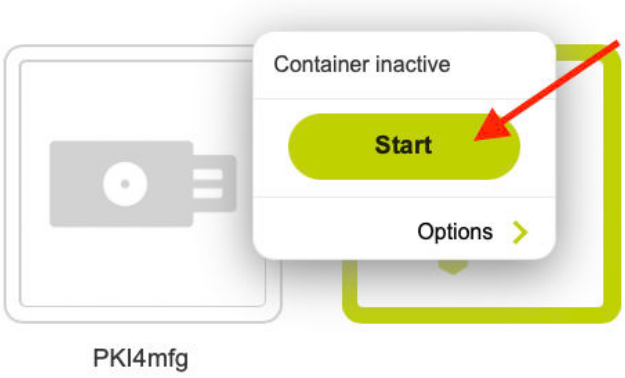

Figure 10: PKI4fmg Start the container

Open the container clicking on the icon:

Figure 11: PKI4fmg Icon

The container is already prepared with the needed tools.

Note: Remember, after downloading the container you can turn off the internet to proceed with the *PKI* establishment.

3.3 Access the PKI4mfg container

In the *periMICA* homepage click on *PKI4mfg*, scroll until the *SSH access* and enable the development mode in order to have access through port 22 and to be able to generate the One Time Password OTP (see figure 12).

Figure 12: SSH enable development mode

Generate the SSH one time password clicking on *Generate*:

Open a terminal within the operating system, for instance, use *Powershell* on Windows. To access the system as *root*, run the `ssh` command. Substitute the installed container name and the `<sernm>` of *periMICA*, then execute the following command:

```
ssh root@<container-name>-perimica-<sermn>.local
```

Listing 1: Access SSH periMICA PKI4mfg

To access the container through SSH using the One Time Password OTP provided in the webUI: Copy and paste the password generated (6 digits number), and press ENTER

When the connection is opened, the prompt will be ready, as an example:

```
Linux pki4mfg-perimica-sermn 5.4.266 #1 SMP Tue Apr 30 09:52:25 UTC 2024 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Apr 26 09:49:48 2024 from fe80::85:cd41:7cda:a32a%eth0
root@pki4mfg-perimica-sermn:~#
```

Listing 2: Prompt SSH periMICA PKI4mfg

The procedure for creating the PKI relies on bash scripts located within a specific directory. Therefore, it is essential to navigate to this directory before initiating the process. To go into the directory use the following command:

```
cd /var/pki
```

Listing 3: Change Directory PKI4mfg

3.4 Initialize the YubiKeys

The YubiKey devices come from manufacture with default PIN, PUK and management key.

- default PIN code is 123456
- default PUK code is 12345678
- default mgm key is 010203040506070801020304050607080102030405060708

Before start, please set new passwords for each device. The management of the password and keys and the identification of the devices is up to the user and it is not in the scope of this document. Just remember to keep it safe and how to fetch the information when necessary.

To change the passwords, connect the YubiKey to *periMICA* and run the following commands and follow the instructions:

```
$ yubico-piv-tool -a change-pin
$ yubico-piv-tool -a change-puk
$ yubico-piv-tool -a set-mgm-key
```

Listing 4: Initialize YubiKey

At any time you can check the status of the Yubikey and check which slots are in use and which certificate are store, for this use the command:

```
$ yubico-piv-tool -a status
```

Listing 5: Check YubiKey status

4 Creating the Certificate Authorities

Basically the process to create any certificate authority (CA), explained here, will involve a single script stored in the container. The process can differ a bit to the *Root CA* because it is the self signed entity and the *Batch CA* which add the creation of the production certificate.

In this section will be presented the usage of the script `ca-create.bash` in order to create the entity. Since the procedure is similar, it will be presented the example with the outputs for the Root CA and for the others CA only examples on how to call the script will be given.

Remember, to create a CA, except to the Root CA, it is required a signature from the entity above it, confirming its validity and authority. Because of it, it is crucial to respect the hierarchical order of entities in the certificate authority structure.

The procedure involves the following steps:

- Generate the private key, internally in YubiKey
- Generate the Sign Request Certificate CSR, based on the private key
- Sign the CSR from parent CA in the chain
- Import certificates

4.1 Usage of *ca-create.bash* script to create CAs

The script `ca-create.bash` follow this steps in order to create each entity and it provides some options described in table 1. Basically it expects to receive the ca certificate name to be created in its call, the tags available are listed in table 2. It means for each CA that needs to be created a call of the script is necessary.

ca-create.bash options	
-e	override the validity time for the certificate
-d	enable debug mode
-h|–help	show help
-n	subject common name
-s	subject string

Table 1: ca-create.bash option

ca-create.bash CA certificate name	
ecc-root-ca	Root CA
ctrl-ca	Control CA
batch-order-ca	Batch Order CA
batch-ca	Batch CA
codesign-ca	Code Sign CA

Table 2: CA certificate name to be created

The CA identifier is called Subject distinguished name and should be defined using the option -s, otherwise the *Perinet* names will be used as default values. The minimum mandatory fields *Country, Organization and Common Name* should be given. Besides it can also be extended according to the *X.509* specifications [1].

- *C* is the country acronym, e.g. DE
- *O* is the Organization Name e.g. Perinet GmbH
- *CN* is the common Name e.g. Perinet ECC Root CA

It is given below a list of the example commands for each entity. Remember to replace at least subject according to the company information:

Root CA:

```
$./scripts/ca-create.bash -s ”/C=DE/O=Perinet GmbH/CN=Perinet ECC Root CA” ecc-root-ca
```

Listing 6: Create ca script

Control CA:

```
$./scripts/ca-create.bash -s ”/C=DE/O=Perinet GmbH/CN=Perinet Ctrl CA” ctrl-ca
```

Listing 7: Create ctrl CA

Code Sign:

```
$./scripts/ca-create.bash -s ”/C=DE/O=Perinet GmbH/CN=Perinet Code Sign CA” codesign-cert
```

Listing 8: Create Code Sign CA

Batch Order CA:

```
$./scripts/ca-create.bash -s ”/C=DE/O=Perinet GmbH/CN=Perinet Batch Order CA 1” batch-order-ca
```

Listing 9: Create ctrl CA

Batch CA:

```
$./scripts/ca-create.bash -s ”/C=DE/O=Perinet GmbH/CN=Perinet Batch CA 2024-10” batch-ca
```

Listing 10: Create Production Batch CA

In the subsequent section, a detailed example is provided for executing the script to create the *Root CA*, followed by an example for creating the *Batch CA*. As previously mentioned, the process to establish any CA is similar. The script facilitates this process by prompting the user to input the necessary data.

Note: When copy and paste the commands don't forget to change the CA names, e.g. Company Name Batch CA 2024-15

4.2 Example to Create Perinet ECC Root CA

In this section the creation of the *Perinet ECC Root CA* is described with all the steps including the output messages. The *Root CA* entity is the top of the chain and will be created with a *self-sign certificate*. Define the *Subject* and use the parameter to define the *distinguished Name* of the entity.

Note: Root CA is self-signed, no other entity signed it since it is the beginning of the chain.

Use the script *create-ca.bash*, to create the *ecc-root-ca*, below is an example to run the script:

```
$./scripts/ca-create.bash -s ”/C=DE/O=Perinet GmbH/CN=Perinet ECC Root CA” ecc-root-ca
```

Listing 11: Create ca script

The script will guide the user with messages when the credential of the YubiKey need to be given and guide the user overall through the steps. Below is shown as an example of the execution of script:

```
    root@pki4mfg-perimica-sernm:/var/opt/pki# ./scripts/ca-create.bash -s "/C=DE/O=
Perinet GmbH/CN=Perinet ECC Root CA"  ecc-root-ca
    Cmdline override, Subject: /C=DE/O=Perinet GmbH/CN=Perinet ECC Root CA
    1. Generate private key on YubiKey for \"ecc-root-ca\". (writes YubiKey)
    2. Generate certificate sign request \"ecc-root-ca\".
    3. Sign CSR from parent CA in the chain \"ecc-root-ca\".
    4. Import certificate into YubiKey \"ecc-root-ca\". (writes YubiKey)
    5. Import certificate chain into YubiKey "ecc-root-ca". (writes YubiKey)

    "NOTE": when prompted for the 'management key'
    the script is going to write to the YubiKey.
    Asure you have connected the correct YubiKey to prevent
    overwriting the wrong key.


    Step 1: Generate private key on YubiKey. (writes YubiKey)
    Step 1: Insert YubiKey dedicated to hold "ecc-root-ca" and press <ENTER>...
```

Listing 12: Output Example creating Root CA

Connect the YubiKey dedicated to store the *ecc-root-ca* in the *periMICA* USB and press <ENTER>. In this step the private key will be written and for that the *management key* is needed:

```
    $ Enter management key:
```

Listing 13: Ask for management key

Input the management key and press <ENTER>. When first step is done the second step take place:

```
    Successfully generated a new private key.
    Step 1: done

    Step 2: Generate certificate sign request
    Step 2: Insert YubiKey dedicated to hold "ecc-root-ca" and press <ENTER>...
```

Listing 14: Step 1 done, start Step 2

Since the Root CA is a self signed the same YubiKey will be used, and because of it, it is not necessary to change the device. Just Press <ENTER> to proceed.

```
Enter PKCS#11 token PIN for YubiKey PIV #11183984:
```

Listing 15: Ask PIN to create the CSR

```
Enter PKCS#11 key PIN for Private key for Digital Signature:
```

Listing 16: Ask PIN to sign the certificate

Note: The terms ”token PIN” and ”key PIN for Private key” refer to the same PIN used to access the private key stored in the smart card with Personal Identity Verification (PIV) functionality.

Step 3 will start and the CSR will be signed:

```
Step 2: done

Step 3: Sign CSR from parent CA in the chain.
Step 3: wait for signed CA before continue "ecc-root-ca" ( "ecc-root-ca" ) and
press <ENTER>...
```

Listing 17: Step 2 Done, follow with Step 3

Enter the PIN as requested:

```
Engine "pkcs11" set.
Using configuration from conf/ca.conf
Enter PKCS#11 token PIN for YubiKey PIV #11183984:
```

Listing 18: Step 3 - Sign the Certificate

To create the signature, the PIN is requested again:

```
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number:
            c4:5e:f7:9a:ca:4f:43:72:11:2d:d7:b7:fe:06:b2:25
        Validity
            Not Before: May  3 11:22:29 2024 GMT
            Not After : Apr 26 11:22:29 2054 GMT
        Subject:
            countryName               = DE
            organizationName          = Perinet GmbH
            commonName                = Perinet ECC Root CA
        X509v3 extensions:
            X509v3 Key Usage: critical
                Certificate Sign, CRL Sign
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Subject Key Identifier:
                FF:D2:29:BD:16:A2:C6:66:DC:94:79:91:B6:FC:05:9D:92:FB:B1:3E
            X509v3 Authority Key Identifier:
                FF:D2:29:BD:16:A2:C6:66:DC:94:79:91:B6:FC:05:9D:92:FB:B1:3E
Certificate is to be certified until Apr 26 11:22:29 2054 GMT (10950 days)
Enter PKCS#11 key PIN for Private key for Digital Signature:
```

Listing 19: The Certificate was signed correct and present to the user

Since the signature is done, the signed certificate will be imported in YubiKey (step 4).

```
Write out database with 1 new entries
Database updated
Step 3: done

Step 4: Import certificate into YubiKey "ecc-root-ca". (writes YubiKey)
Step 4: Insert YubiKey dedicated to hold "ecc-root-ca" and press <ENTER>...
```

Listing 20: Step 4: import certificate to the YubiKey

To write the certificate in the YubiKey, again the *Management Key* will be requested:

```
subject=C = DE, O = Perinet GmbH, CN = Perinet ECC Root CA
Warning, unable to reset locale
Enter management key:
```

Listing 21: Request Management Key to store certificate in YubiKey

Step 5 is importing the chain certificate in the YubiKey, for *ecc-root-ca* this can not be so important since it is the root of the chain.

```
Successfully imported a new certificate.
Step 4: done

Step 5: Import certificate chain into YubiKey "ecc-root-ca". (writes YubiKey)
Step 5: Insert YubiKey dedicated to hold "ecc-root-ca" and press <ENTER>...
```

Listing 22: Step 5: importing certificate

When all steps are done, the script finish the execution.

```
Step 5: done
Process finished press <ENTER>...
```

Listing 23: Step 5 Done!

And the metadata of the certificate is presented.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            c4:5e:f7:9a:ca:4f:43:72:11:2d:d7:b7:fe:06:b2:25
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C = DE, O = Perinet GmbH, CN = Perinet ECC Root CA
        Validity
            Not Before: May  3 11:22:29 2024 GMT
            Not After : Apr 26 11:22:29 2054 GMT
        Subject: C = DE, O = Perinet GmbH, CN = Perinet ECC Root CA
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:3f:8b:d4:74:47:bb:5d:84:9c:91:e6:17:b0:00:
                    d1:01:2a:31:61:74:71:90:f9:a8:f1:af:da:58:58:
                    f7:4f:dc:25:05:64:c8:9c:c9:ca:eb:fe:c4:66:ee:
                    ba:c7:93:46:9d:11:79:bf:c8:bf:76:b5:95:57:b4:
                    8f:27:f6:e5:06
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Key Usage: critical
                Certificate Sign, CRL Sign
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Subject Key Identifier:
                FF:D2:29:BD:16:A2:C6:66:DC:94:79:91:B6:FC:05:9D:92:FB:B1:3E
            X509v3 Authority Key Identifier:
                FF:D2:29:BD:16:A2:C6:66:DC:94:79:91:B6:FC:05:9D:92:FB:B1:3E
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:45:02:21:00:87:63:1f:3e:fc:a0:a3:43:54:0f:91:3d:c2:
        5b:b2:fb:07:e7:64:97:e3:15:d6:61:c0:1d:7d:b7:3d:ad:2d:
        50:02:20:7c:b6:36:7e:d8:c1:3f:73:61:bf:61:62:77:aa:c1:
        f8:24:8b:1a:93:bc:4c:a8:55:6b:99:99:1b:b5:83:af:0a
```

Listing 24: Output Root CA Certificate

4.3 Create Batch CA

The creation of a *batch-ca* requires the production manufacturing data and it is described in this section.

The owner/responsible to manage the Batch Order CA Yubikey can create a Batch CA, which will be used in production to create the endpoints certificates in production.

Every batch of production requires a dedicated Yubikey and the production certificate stored on it.

To create a Batch CA select the Batch Number, which consists of the year followed by the week, like *2024-10*. Embedded it into the *Common Name (CN)* of the batch-ca, as presented in the example:

```
    ./scripts/ca-create.bash -s "/C=DE/O=Perinet GmbH/CN=Perinet Batch CA 2024-10"
batch-ca
```

Listing 25: Create Production Batch CA

When creating a Yubikey dedicated to the production (batch-ca) the production manufacturing data will be requested like in the example:

```
Step 2.1: Generate certificate sign request for production certificate. For that
input the production manufacturing data as requested.
Type PRODUCT NAME and press <ENTER>:
periSNOOP
Type PRODUCT PART NUMBER and press <ENTER>:
PRN.000.069
Type PRODUCT VERSION and press <ENTER>:
2
Step 2: done
```

Listing 26: Type the manufacturing data due to production certificate

In the *batch-ca*, the PIN and the management key will be requested more times since it is signing the certificate and the production certificate as well.

```
Step 3: Sign CSR from parent CA in the chain.
Step 3: Insert the ( "perinet-batch-order-ca" ) and press <ENTER>...
Warning, unable to reset locale
Engine "pkcs11" set.
Using configuration from conf/ca.conf
Enter PKCS#11 token PIN for Perinet Batch Order CA 1:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number:
            07:88:70:ce:c3:30:97:57:6b:e3:a3:ab:88:6c:a8:b9
        Validity
            Not Before: May  8 08:16:23 2024 GMT
            Not After : May  6 08:16:23 2034 GMT
        Subject:
            countryName               = DE
            organizationName          = Perinet GmbH
            commonName                = Perinet Batch CA 2024-10
        X509v3 extensions:
            X509v3 Key Usage: critical
                Certificate Sign, CRL Sign
            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:0
            X509v3 Subject Key Identifier:
                48:60:70:E6:DE:0D:F8:E7:74:CE:F3:83:1D:D2:A8:BD:C3:8E:43:1D
            X509v3 Authority Key Identifier:
                C3:7B:FF:AA:C2:7D:1E:5C:77:34:EC:12:96:C4:69:E9:18:B3:21:BA
Certificate is to be certified until May  6 08:16:23 2034 GMT (3650 days)
Enter PKCS#11 key PIN for SIGN key:
```

Listing 27: Step 3 Create Batch CA

4.3.1 Production Certificate

An example of production certificate looks like the presented into listing 28.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            f4:ee:ec:f0:99:80:90:6f:52:d6:1d:be:7d:cf:87:11
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C = DE, O = Perinet GmbH, CN = Perinet Batch Order CA
        Validity
            Not Before: May  8 08:16:31 2024 GMT
            Not After : Jun  7 08:16:31 2024 GMT
        Subject: C = DE, O = Perinet GmbH, CN = production-2024-10, description = {"
product_name": "periSNOOP","product_part_number": "PRN.000.069", "product_version":
2, "product_charge": "2024-10"}
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:76:42:df:33:9c:b7:c2:f6:60:41:93:19:5b:2e:
                    bd:7b:77:df:db:a8:83:45:be:23:97:e7:ce:93:0d:
                    33:f2:ed:81:61:88:52:70:20:39:3d:49:1c:57:d1:
                    68:c6:53:f7:d9:d0:a7:83:b9:ed:c8:23:b0:d5:02:
                    e3:73:26:77:dc
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Key Usage: critical
                Data Encipherment
            X509v3 Subject Key Identifier:
                30:B8:10:21:DA:C5:92:A7:E5:DC:E8:90:27:FE:C3:87:0F:80:58:31
            X509v3 Authority Key Identifier:
                C3:7B:FF:AA:C2:7D:1E:5C:77:34:EC:12:96:C4:69:E9:18:B3:21:BA
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:45:02:20:3e:c0:76:fd:ad:87:06:96:d5:eb:da:3f:43:d5:
        78:e9:d8:14:7b:23:d1:ac:de:bd:ca:81:35:4e:ee:82:56:4d:
        02:21:00:90:3b:d1:38:43:13:57:48:e8:d7:5f:50:e9:6c:22:
        37:45:d5:10:69:a6:09:09:49:ca:e7:c3:e5:a4:ed:b7:9f
```

Listing 28: Production Certificate Example

The production data will be stored in the OEM according to the Perinet API, the variable data like serial number is defined in the test place, but the data of the batch will be fetched from the production certificate stored in the Yubikey device.

The production certificate is set by default to have the expiry days for 30 days. It can be changed it using the `-e` option.

5 Further Documentation

5.1 periCORE

Document Name	Description
periCORE Product Summary [6]	A product features summary documentation for the product *periCORE*.
periCORE Datasheet [2]	A detailed reference documentation of the product *periCORE*.
periCORE Development Kit Product Summary [3]	A product features summary documentation for the product *periCORE Development Kit*.
periCORE Development Kit Setup Application Note [4]	A setup guide for the product *periCORE Development Kit*. The starting point when you are new to the product which describes how to quickly set up a development environment for firmware development for a periCORE based product.
periCORE Development Kit User Guide [5]	A guide and reference documentation for the product *periCORE Development Kit*.

5.2 periMICA

Document Name	Description
periMICA Product Summary [9]	A product features summary documentation for the *periMICA*.
periMICA Quick Start Guide [10]	A setup guide for the product *periMICA*. The starting point when you are new to the product.
periMICA User Guide [11]	A detailed description and reference documentation of the product *periMICA*.
PKI2go Container User Guide [12]	A detailed description of the public key infrastructure (PKI) periMICA container 'PKI2go'.
Security Certificates Installation Guide [15]	A guide on how to install prerequisites certificates on various platforms (MAC, Windows and Linux).
periMICA Debian Container User Guide [7]	A detailed description of the base container of a periMICA lxc-based container. It's the starting point for container development.
periMICA Product ReBranding Application Note [8]	A detailed description for development to rebrand the product periMICA to a dedicated customer corporate design.

5.3 Perinet Security

Document Name	Description
PKI2go Container User Guide [12]	A detailed description of the public key infrastructure (PKI) periMICA container 'PKI2go'.
PKI4mfg Container User Guide [13]	The (PKI) periMICA container 'PKI4mfg' User Guide.
Production Public Key Infrastructure Establishment [14]	Production Public Key Infrastructure Establishment.
Security Certificates Installation Guide [15]	A guide on how to install prerequisites certificates on various platforms (MAC, Windows and Linux).

6 Contact & Support

For customer support, please call us at **+49 30 863 206 701** or send an e-mail to *support@perinet.io*.

For complete contact information visit us at www.perinet.io

A List of Figures

B List of Listings

C List of Tables

D Glossary

API Application Programming Interface. 27

CA Certification Authority, a trusted entity which is represented by a certificate that is used to verify the signature on a certificate issued by that authority (trust anchor). 1, 4, 6, 7, 9–11, 17–19, 24

CRA Cyber Resilience Act is a proposal for an EU regulation on cybersecurity requirements for products with digital elements. 4, 6

CSR Certificate Sign Request. 6, 17, 21

HSM Hardware Security Module. 4

OEM Original Equipment Manufacturer is generally perceived as a company that produces parts and equipment that may be marketed by another manufacturer. 6, 7

OTP One-Time Programmable memory. 7, 14

PKI Public Key Infrastructure. 1, 4, 5, 8, 9, 28, 29

PKI2go PKI2go is a patent-pending security service provided by Perinet. 9

RBAC Role Based Access Control. 9

SSL Secure Sockets Layer. 1

E References

[1] et al. Cooper. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC. https://www.rfc-editor.org/rfc/rfc5280.txt. RFC Editor, May 2008. URL: https://www.rfc-editor.org/rfc/rfc5280.txt.

[2] Perinet GmbH. periCORE Datasheet. PRN.100.375. https://docs.perinet.io/PRN100375-periCOREDatasheet.pdf.

[3] Perinet GmbH. periCORE Development Kit Product Summary. PRN.100.546. https://docs.perinet.io/PRN100546-periCOREDevelopmentKitProductSummary.pdf.

[4] Perinet GmbH. periCORE Development Kit Setup Application Note. PRN.100.376. https://docs.perinet.io/PRN100376-periCOREDevelopmentKitSetupApplicationNote.pdf.

[5] Perinet GmbH. periCORE Development Kit User Guide. PRN.100.378. https://docs.perinet.io/PRN100378-periCOREDevelopmentKitUserGuide.pdf.

[6] Perinet GmbH. periCORE Product Summary. PRN.100.301. https://docs.perinet.io/PRN100301-periCOREProductSummary.pdf.

[7] Perinet GmbH. periMICA Debian Container User Guide. PRN.100.462. https://docs.perinet.io/PRN100462-DebianContainerUserGuide.pdf.

[8] Perinet GmbH. *periMICA Product ReBranding Application Note*. PRN.100.583. https://docs.perinet.io/.

[9] Perinet GmbH. periMICA Product Summary. PRN.100.389. https://docs.perinet.io/PRN100389-periMICAProductSummary.pdf.

[10] Perinet GmbH. periMICA Quick Start Guide. PRN.100.391. https://docs.perinet.io/PRN100391-periMICAQuickStartGuide.pdf.

[11] Perinet GmbH. periMICA User Guide. PRN.100.392. https://docs.perinet.io/PRN100392-periMICAUserGuide.pdf.

[12] Perinet GmbH. PKI2go Container User Guide. PRN.100.465. https://docs.perinet.io/PRN100465-PKI2goContainerUserGuide.pdf.

[13] Perinet GmbH. PKI4mfg Container User Guide. PRN.100.809. https://docs.perinet.io/PRN100809-PKI4mfgContainerUserGuide.pdf.

[14] Perinet GmbH. Production Public Key Infrastructure Establishment. PRN.100.811. https://docs.perinet.io/PRN100811-ProductionPublicKeyInfrastructureEstablishment.pdf.

[15] Perinet GmbH. Security Certificates Installation Guide. PRN.100.447. https://docs.perinet.io/PRN100447-SecurityCertificatesInstallationGuide.pdf.

[16] European Union. *Regulation - 2024/2847*. URL: https://eur-lex.europa.eu/eli/reg/2024/2847/oj.

F Revision History

Revision	Date	Author(s)	Description
1	2025-01-28	dshe	Initial Release
2	2025-02-03	dshe	Enhance language, add note about PIN, avoid break lines in commands