nShield Post-Quantum Cryptography Option Pack

nShield® HSM Integration Guide

2026-03-16

© Entrust Corporation. All rights reserved.

Table of Contents

1. Introduction . . . 1
    1.1. Product configurations . . . 1
    1.2. Requirements . . . 1
    1.3. More information . . . 2
2. Procedures . . . 3
    2.1. Install the HSM . . . 3
    2.2. Install the Security World Software and create the Security World . . . 3
    2.3. Set the PATH environment variable . . . 3
    2.4. Install the nShield PQC Option Pack . . . 4
    2.5. CodeSafe Setup . . . 4
    2.6. Test the PQCOP . . . 16
    2.7. Create LMS Keys . . . 17
    2.8. Uninstall the PQCOP CodeSafe machines . . . 26
    2.9. Manage LMS Keys in a Mixed State Environment . . . 29
3. Additional resources and related products . . . 40
    3.1. nShield HSMs . . . 40
    3.2. nShield as a Service . . . 40
    3.3. Entrust products . . . 40
    3.4. nShield product documentation . . . 40

Chapter 1. Introduction

This guide describes how to integrate the nShield Post-Quantum Cryptography (PQC) Option Pack library, also known as PQCOP, with an Entrust nShield® Hardware Security Module (HSM) to provide secure solutions.

The nShield PQC Option Pack enables Security World Software users to generate and use keys with public-key cryptographic algorithms selected by NIST as part of the Post-Quantum Cryptography standardization process.

The nShield PQC Option Pack is installed on top of your Security World Software, allowing you to use your existing keys and algorithms alongside post-quantum algorithms.

1.1. Product configurations

We have successfully tested nShield HSM integration with the nShield PQC Option Pack on a Linux server running Ubuntu 22.04 and a Windows server running Windows Server 2022 in the following configurations:

PQCOP	CodeSafe	Security World	HSM	Firmware	Netimage	FIPS Level 3
1.4.1	13.9.3	13.9.3	nShield Connect XC	13.8.3	13.9.3	No
1.4.1	13.9.3	13.9.3	nShield 5c	13.8.4	13.9.3	No

1.2. Requirements

Ensure that you have supported versions of all products as described in Product configurations.

To perform the integration tasks, you must have:

- `root` access on the operating system. This is required to install Security World. This can be done by having `sudo` access.
- Access to `nfast` accounts.

`sudo` is used in some commands; however, it is not necessary if the user has `nfast` group access.

Before starting the integration process, familiarize yourself with the documentation and setup processes for:

- The HSM (Documentation only. This guide assumes your HSM is already set up.)
- CodeSafe running on the nShield Connect XC
- CodeSafe 5 running on the nShield 5c
- The nShield PQC Option Pack

Before using the nShield software, you need to know:

- The number and quorum of Administrator Cards in the Administrator Card Set (ACS), and the policy for managing these cards.
- The number and quorum of Operator Cards in the OCS, and the policy for managing these cards.

Entrust recommends that you allow only unprivileged connections unless you are performing administrative tasks.

For more information, see the documentation for the HSM.

ML-DSA is now supported in firmware and is better used in the main product line than through the nShield PQC Option Pack. LMS will be the default type in nShield PQC Option Pack for releases *after* 1.4.1.

1.3. More information

For more information about OS support, contact Entrust nShield Support at https://nshieldsupport.entrust.com.

Access to the Entrust nShield Support Portal is available to customers under maintenance. To request an account, contact nshield.support@entrust.com.

Chapter 2. Procedures

This guide describes some use case scenarios demonstrating the usage of the PQCOP with an nShield HSM. When running the commands in this guide, use a `terminal shell` (**Linux**) or a `PowerShell` (**Windows**).

2.1. Install the HSM

Install the HSM by following the instructions in the *Installation Guide* for the HSM.

We recommend that you install the HSM before configuring the Security World Software on the PQCOP server.

2.2. Install the Security World Software and create the Security World

Install the Security World Software and create the security world:

1. On the computer that you want to use PQCOP, install the latest version of the Security World Software as described in the *Installation Guide* for the HSM.

   Entrust recommends that you uninstall all existing nShield software before you install new nShield software.

2. Create the Security World and required ACS and OCS as described in the *User Guide*.

2.3. Set the PATH environment variable

After Security World is installed, add the `bin` directory containing the Security World binaries to your PATH. The commands in this guide do not include the path, so they will only work if your `PATH` variable has been updated.

Linux

```
export PATH=/opt/nfast/bin:$PATH
```

Windows

```
$path = [Environment]::GetEnvironmentVariable('Path','Machine')
$newpath = $path + ';C:\Program Files\nCipher\nfast\bin'
[Environment]::SetEnvironmentVariable("Path", $newpath, 'Machine')
```

2.4. Install the nShield PQC Option Pack

1. Obtain the PQCOP release file from Entrust Professional Services.

   In this guide we use `pqsdk-1.4.1-win64.zip` as the release file.

2. Transfer the file to the PQCOP server.
3. Unzip the file.

```
mkdir ~/pqsdk
cd ~/pqsdk
mv ~/Downloads/pqsdk-1.4.1-win64.zip .
unzip pqsdk-1.4.1-win64.zip
```

2.5. CodeSafe Setup

The PQCOP functions as a CodeSafe machine running on the HSM. Because CodeSafe setup differs between an nShield Connect XC and an nShield 5c, this section is divided into two sections, one for each HSM type. Before proceeding with the setup, you must install the CodeSafe licenses in the HSM, if not present yet:

- For the nShield Connect XC: **SEE Activation (EU+10)**
- For the nShield 5c: **SEE Activation (CodeSafe 5)**
  1. Check whether the licenses are installed in the HSM:

```
/opt/nfast/bin/fet -m 1

                        Feature Enable Tool
                        ===================

                      ISO Smart Card Support
                      |  Remote Operator
                      |  |  Korean Algorithms
                      |  |  |  SEE Activation (EU+10)
                      |  |  |  |  SEE Activation (Restricted)
                      |  |  |  |  |  SEE Activation, CodeSafe 5
                      |  |  |  |  |  |  Elliptic Curve algorithms
                      |  |  |  |  |  |  |  Elliptic Curve MQV
                      |  |  |  |  |  |  |  |  Fast RNG for ECDSA
                      |  |  |  |  |  |  |  |  |  HSM Speed Rating
Mod   Electronic      |  |  |  |  |  |  |  |  |  |
No.  Serial Number
1   7852-268D-3BF9 -- Y  Y  Y  N  N  Y  Y  Y  N  High Speed
```

  2. If the required license is not installed, request a license file from Entrust Support.
  3. Transfer the license file to the following folder on the RFS: `/opt/nfast/kmdata/hsm-<ESN>/features`, replacing `<ESN>` with the ESN of your HSM.

4. In the RFS, run the following commands to install the license (taking the module number into account):

```
cd /opt/nfast/kmdata/hsm-<ESN>/features
/opt/nfast/bin/nethsmadmin -m 1 -s RFS_IP_ADDRESS -a LICENSE_FILE.txt
/opt/nfast/bin/nopclearfail -c -m 1
/opt/nfast/bin/fet -m 1
```

2.5.1. nShield XC Setup

This section describes the process of getting the PQCOP CodeSafe machine running in a nShield XC HSM.

1. Change directory to where the PQCOP release file was unzipped.

```
cd pqsdk-1.4.1-win64
```

2. Make sure the ACS card is presented on the front of the HSM.
3. Use the `generatekey` utility to create the signing key.

   The signing key must have the `seeinteg` appname and must be protected by an operator card set.

If you need to share LMS keys between nShield Connect XC and nShield 5c devices, the key must be an ECDSA key using the NIST 521 bit prime curve.

```
generatekey seeinteg type=ecdsa curve=nistp521 plainname=pqsdk-signer nvram=no recovery=yes protect=token
cardset=testOCS

slot: Slot to read cards from? (0-5) [0] > 3
key generation parameters:
 operation    Operation to perform       generate
 application  Application                seeinteg
 protect      Protected by               token
 slot         Slot to read cards from    3
 recovery     Key recovery               yes
 verify       Verify security of key     yes
 type         Key type                   ecdsa
 plainname    Key name                   pqsdk-signer
 nvram        Blob in NVRAM (needs ACS)  no
 curve        Elliptic curve             nistp521

Loading `testOCS':
 Module 1: 0 cards of 1 read
 Module 1 slot 2: `testOCS' #5
 Module 1 slot 0: empty
 Module 1 slot 4: empty
 Module 1 slot 5: empty
 Module 1 slot 3:- passphrase supplied - reading card
Card reading complete.
```

```
Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_seeinteg_pqsdk-signer
```

4. Use the same key name to create an NVRAM delegation certificate:

   You must have the ACS card presented to do this step.

   - **Linux**

     ```
     /opt/nfast/python3/bin/python3 userdatatool.py -A seeinteg -I pqsdk-signer -f userdata.raw
     ```

   - **Windows**

     ```
     c:\> & 'C:\Program Files\nCipher\nfast\python3\python3.exe' C:\Users\Administrator\pqsdk\pqsdk-1.4.1-win64\userdatatool.py -A seeinteg -I pqsdk-signer -f userdata.raw
     ```

   Example output:

   ```
   Loading ACS KNV for NVMem delegation:
    Module 1 slot 0: Admin Card #1
    Module 1 slot 2: `testOCS' #5
    Module 1 slot 3: `codesign' #1
    Module 1 slot 4: empty
    Module 1 slot 5: empty
    Module 1 slot 0:- passphrase supplied - reading card
   Card reading complete.
   ```

5. Sign the SEE machine.

   This example uses a single key to sign both the SEE machine and the user data; however, you can use different keys.

   The `pqsdk.elf` file in the distribution has a version in it. In our case `1.4.1`. Notice the module being used is module #1.

   ```
   tct2 --sign-and-pack -m 1 -k pqsdk-signer --is-machine --machine-type PowerPCELF -o pqsdk-1.4.1.sar pqsdk-1.4.1.elf

   Signing machine as `PowerPCELF'.

   Loading `testOCS':
    Module 1: 0 cards of 1 read
    Module 1 slot 0: Admin Card #1
    Module 1 slot 2: `testOCS' #5
    Module 1 slot 4: empty
    Module 1 slot 5: empty
    Module 1 slot 3:- passphrase supplied - reading card
   Card reading complete.
   ```

6. Sign the user data file:

```
tct2 --sign-and-pack -m 1 -k pqsdk-signer --machine-key-ident pqsdk-signer -o userdata.sar userdata.raw

Loading `codesign':
 Module 1: 0 cards of 1 read
 Module 1 slot 3: `codesign' #1
 Module 1 slot 0: Admin Card #1
 Module 1 slot 2: `testOCS' #5
 Module 1 slot 4: empty
 Module 1 slot 5: empty
 Module 1 slot 3:- passphrase supplied - reading card
Card reading complete.
```

7. Create the `/opt/nfast/kmdata/custom-seemachines` directory and copy the `pqsdk-1.4.1.sar` and the `userdata.sar` file to it:

   - **Linux**

     ```
     sudo mkdir -p /opt/nfast/kmdata/custom-seemachines
     sudo cp pqsdk-1.4.1.sar /opt/nfast/kmdata/custom-seemachines/.
     sudo cp userdata.sar /opt/nfast/kmdata/custom-seemachines/.
     ```

   - **Windows**

     ```
     mkdir 'C:\ProgramData\nCipher\Key Managemeent Data\custom-seemachines'
     cp pqsdk-1.4.1.sar 'C:\ProgramData\nCipher\Key Managemeent Data\custom-seemachines\.'
     cp userdata.sar 'C:\ProgramData\nCipher\Key Managemeent Data\custom-seemachines\.'
     ```

8. Configure one or more nShield XC devices to run the SEE machine.

   This will prompt you to reset the nShield HSM. Doing so should start the SEE machine and restart it whenever the HSM is restarted.

   - **Linux**

     ```
     loadsee-setup --setup \
          -m 1 -M /opt/nfast/kmdata/custom-seemachines/pqsdk-1.4.1.sar \
          -U /opt/nfast/kmdata/custom-seemachines/userdata.sar -p pqsdk
     ```

   - **Windows**

     ```
     loadsee-setup --setup -m 1
       -M 'C:\ProgramData\nCipher\Key Managemeent Data\custom-seemachines\pqsdk-1.4.1.sar'
       -U 'C:\ProgramData\nCipher\Key Managemeent Data\custom-seemachines\userdata.sar'
       -p pqsdk
     ```

   Example output:

   ```
   Module #1 new SEE configuration saved, new configuration follows:

   Module #1:
     Machine file:             /opt/nfast/kmdata/custom-seemachines/pqsdk-1.4.1.sar
   ```

```
  Encryption key:
  Signing key hash:
  Userdata file:            /opt/nfast/kmdata/custom-seemachines/userdata.sar
  WorldID published object: pqsdk
  Postload helper:
  Postload args:

Reload new configuration now? (yes/no): yes
Clear modules before reloading new configuration? (yes/no): yes
Modules cleared for new configuration.
Check the hardserver log to ensure the SEE machine was loaded correctly after clearing.
```

You can also use the front panel for SEE configuration. To do this step using the configuration file:

1. In the RFS, create a copy of the `/opt/nfast/kmdata/hsm-<ESN>/config/config` file, where `<ESN>` is the ESN of the nShield Connect.

   In this example, the new copy of the file is named `config_new` and is in the current directory. The file can have any name, as long as it is used consistently.

2. Edit the `[load_seemachine]` section in the new copy to resemble the following example:

   ```
   module=1
   machine_file=pqsdk-1.4.1.sar
   encryption_key=
   signing_hash=
   userdata=userdata.sar
   worldid_pubname=pqsdk
   postload_prog=
   postload_args=
   pull_rfs=yes
   ```

3. Use `cfg-pushnethsm` to push the new configuration, where `<HSM_IP_ADDRESS>` is the IP address of the HSM being used.

   ```
   cfg-pushnethsm -a <HSM_IP_ADDRESS> config_new
   ```

9. Check that the SEE machine has been loaded:

```
hsc_loadseemachine -m1
```

2.5.2. nShield 5c Setup

This section describes the process of getting the PQCOP CodeSafe machine running in a

nShield 5c HSM. You must install CodeSafe 5 software before proceeding.

2.5.2.1. Install CodeSafe 5 (Linux)

For more information on installing CodeSafe 5, refer to the https://nshielddocs.entrust.com/security-world-docs/codesafe5/.

1. Download the `Codesafe_Lin64-13.9.3.iso`.
2. Transfer the `Codesafe_Lin64-13.9.3.iso` to the PQCOP server.
3. Make a mountpoint directory:

```
mkdir -p ~/codesafe_iso_mountpoint
```

4. Mount the ISO image:

```
sudo mount ~/Downloads/Codesafe_Lin64-13.9.3.iso ~/codesafe_iso_mountpoint/
```

5. Untar the tarballs into the root directory:

```
sudo tar -zxvf ~/codesafe_iso_mountpoint/linux/amd64/csd.tar.gz -C /
sudo tar -zxvf ~/codesafe_iso_mountpoint/linux/amd64/csdref.tar.gz -C /
```

6. Umount the mount directory:

```
sudo umount ~/codesafe_iso_mountpoint
```

2.5.2.2. Install CodeSafe 5 (Windows)

For more information on installing CodeSafe 5, refer to the CodeSafe 5 Installation Guide.

1. Download the `Codesafe_Windows-13.9.3.iso`.
2. Transfer the `Codesafe_Windows-13.9.3.iso` to the PQCOP server.
3. Mount the ISO image and navigate into the mounted directory.
4. Launch `setup.msi`.
5. Follow the on-screen instructions.
6. Accept the license terms and select **Next** to continue.
7. Specify the installation directory and select **Next** to continue.
8. Select **Install**.
9. Select **Finish** to complete the installation.

This installs the nShield CodeSafe 5 SDK under `C:\Program Files\nCipher\nfast\c\csd5` and the nShield CodeSafe 5 SDK Python files under `C:\Program Files\nCipher\nfast\python3\csd5`.

2.5.2.3. Set up the PQCOP CodeSafe machine

1. Change directory to where the PQCOP release file was unzipped:

```
cd pqsdk-1.4.1-win64
```

2. Create a Developer Key and certificate signing request (CSR).

   Running a CodeSafe 5 container requires a developer key and certificate in addition to an application signing key (ASK). You must generate the key pair first, and then send a signing request to Entrust Support. When it has been accepted, Entrust Support will send you a certificate.

   When you create the CSR, you must have an OCS card to protect the key presented.

```
csadmin ids create --keyname devkeyident -m 1 --x509cname COMMON_NAME --x509org ORGANIZATION_NAME

Generate key 'devkeyident' ...

Loading `testOCS':
 Module 1: 0 cards of 1 read
 Module 1 slot 2: `testOCS' #4
 Module 1 slot 0: empty
 Module 1 slot 3: empty
 Module 1 slot 4: empty
 Module 1 slot 5: empty
 Module 1 slot 2:- passphrase supplied - reading card
Card reading complete.

OK
Generate a CSR in 'devkeyident.csr' ...

Loading `testOCS':
 Module 1 slot 2: `testOCS' #4
 Module 1 slot 0: empty
 Module 1 slot 3: empty
 Module 1 slot 4: empty
 Module 1 slot 5: empty
 Module 1 slot 2:- passphrase supplied - reading card
Card reading complete.

OK
Created CSR file 'devkeyident.csr'. Please send it to Entrust Support
```

   This creates a CSR, `devkeyident.csr`.

3. Send the CSR file to Entrust Support.

4. When Entrust Support sends you the X.509 developer certificate, save it as `devkeyident.pem`.
5. Get the CodeSafe 5 client info:

```
hsmadmin cs5 clientinfo

ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBpxuj1sCIjvq5dCoprxVCUFTfTVImwO/7WW95njfZOmnw8/aAuW6nQ
uq4IiBiXeqkAkGbmY2SB5NU3QeUxeyxM=
```

6. Set up the CodeSafe 5 client with the details from the previous command:

```
appliance-cli -m <module> cs5 setclientinfo <keytype> <publickey>
```

   In the previous step, the `keytype` is `ecdsa-sha2-nistp256` and the `publickey` is the string commencing with `AAAAE2VjZH`, for example:

```
appliance-cli -m 1 cs5 setclientinfo \
  ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBpxuj1sCIjvq5dCoprxVCUFTfTVImwO/7WW95njfZOmnw8/aAuW6nQ
uq4IiBiXeqkAkGbmY2SB5NU3QeUxeyxM=

Client info set
```

7. Enable the services on the 5c:

```
appliance-cli -m <module> cs5 ports enable <IPv4>
```

   The IPv4 address is the IPv4 address of the HSM.

```
appliance-cli -m 1 cs5 ports enable XXX.XXX.XXX.XXX

Enabled ports for CodeSafe 5

Current ports settings:
enable_ports: yes
address: XXX.XXX.XXX.XXX
```

8. Get the connection information for the launcher service from the nShield 5c:

```
appliance-cli -m {MODULE} cs5 getserverinfo
```

   For example:

```
appliance-cli -m 1 cs5 getserverinfo

CodeSafe 5 server info in format <ESN> <IP Address> <Port> <Key Type> <Public Key>
7852-268D-3BF9 XXX.XXX.XXX.XXX 2205 ecdsa-sha2-nistp256
```

```
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBCwlIbn33m9WceI2uuQif5ZU/LvLZ1NqZa8W0JtOlNPZOWx5q3vdUxo
U9T+l4Ny74qDoB/9eqIYa5h6JU5vA2JU=
```

9. Enroll as a client of CS5 launcher services on this HSM:

   ```
   hsmadmin cs5 enroll <esn> <IPv4> <port> <keytype> <keybody>
   ```

   The output from the previous step contains all the details you need to pass as arguments to this command. In the previous step:

   - ESN: `7852-268D-3BF9`
   - IPv4: `XXX.XXX.XXX.XXX`
   - Port: `2205`
   - Keytype: `ecdsa-sha2-nistp256`
   - Keybody: The string commencing with `AAAAE2VjZH`

   For example:

   ```
   hsmadmin cs5 enroll 7852-268D-3BF9 XXX.XXX.XXX.XXX 2205 \
          ecdsa-sha2-nistp256
   AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBCwlIbn33m9WceI2uuQif5ZU/LvLZ1NqZa8W0JtOlNPZOWx5q3vdUxo
   U9T+l4Ny74qDoB/9eqIYa5h6JU5vA2JU=
   ```

10. Run `csadmin list` to confirm the RFS connection to this nShield 5c.

    The output displays the ESN and informs you that there are no SEE machines:

    ```
    csadmin list

    7852-268D-3BF9        No SEE machines currently installed
    ```

11. Transfer the `devkeyident.pem` developer certificate to your PQCOP server.
12. Register this developer certificate as the first trusted code-signing certificate for the SEE machine on this HSM:

    ```
    csadmin ids --esn {esn} add {CERTFILE}
    ```

    For example:

    ```
    csadmin ids --esn 7852-268D-3BF9 add devkeyident.pem

    7852-268D-3BF9        SUCCESS
    ```

13. Check that the new certificate is present:

```
csadmin ids list --json

{
    "7852-268D-3BF9": {
        "succeeded": true,
        "data": {
            "Certificates": [
                {
                    "subject": "Common Name: CodeSafe Production, Organizational Unit: nCipher,
Organization: Entrust, Country: GB",
                    "keyid": "c9c01efbe14ff33cb9afec71d66639c6e038a378",
                    "serialNumber": "143541911711567699079218604000066011212901051119",
                    "authKeyid": "e193ba2ea9acf291ac3c072261db107d2ef08706",
                    "notBefore": "2023-03-14 09:36:09+00:00",
                    "notAfter": "2033-03-24 09:36:09+00:00"
                },
                {
                    "subject": "Common Name: entrust.com, Organization: ENTRUST, Organizational Unit: PQ
SDK Test",
                    "keyid": "400d62ecb83cfd1b6a1a1b844a653e8614c3c737",
                    "serialNumber": "226364443189258673605810532466534336860",
                    "authKeyid": "c9c01efbe14ff33cb9afec71d66639c6e038a378",
                    "notBefore": "2026-02-13 17:18:39+00:00",
                    "notAfter": "2029-02-13 17:18:38+00:00"
                },
                {
                    "subject": "Common Name: interops.local, Organization: Hurricanes, Organizational Unit:
PQ SDK Test",
                    "keyid": "a1d2a0415f313f50a1485bec75c45f20619812e4",
                    "serialNumber": "238627680622732813176583323266486551324",
                    "authKeyid": "c9c01efbe14ff33cb9afec71d66639c6e038a378",
                    "notBefore": "2026-02-19 16:04:34+00:00",
                    "notAfter": "2029-02-19 16:04:33+00:00"
                }
            ]
        }
    }
}
```

14. Use the `generatekey` utility to create the application signing key (ASK).

    The ASK must have the `simple` appname and must be an ECDSA key using the NIST 521 bit prime curve.

    Remember the slot number because this is the slot where you need to present the OCS card that will be used to protect the key.

```
generatekey simple type=ecdsa curve=nistp521 ident=pqsdk-signer plainname=pqsdk-signer
    nvram=no recovery=yes protect=token cardset=testOCS slot=2
    trusted-certifier= seeintegname=
```

15. Use the same key name to create an NVRAM delegation certificate.

    Make sure the ACS card is presented on the front slot of the HSM.

    - **Linux**

```
/opt/nfast/python3/bin/python3 userdatatool.py -A simple -I pqsdk-signer -f userdata.raw
```

- **Windows**

```
c:\> & 'C:\Program Files\nCipher\nfast\python3\python3.exe' userdatatool.py -A simple -I pqsdk-signer
-f userdata.raw
```

Example output:

```
Loading ACS KNV for NVMem delegation:
 Module 1 slot 0: Admin Card #1
 Module 1 slot 2: `testOCS' #6
 Module 1 slot 3: empty
 Module 1 slot 4: empty
 Module 1 slot 5: empty
 Module 1 slot 0:- passphrase supplied - reading card
Card reading complete.
```

16. Create the `container` CodeSafe 5 machine directory structure.

    Create a directory with files laid out the way they are intended to be in the container. Include the `pqsdk.bin` file as the entry point.

    Remember the release number on the `pqsdk.bin` file.

```
mkdir -p container/usr/bin
mkdir -p container/etc
cp pqsdk-1.4.1.bin container/usr/bin/entrypoint
cp userdata.raw container/etc/userdata.codesafe
```

17. Create an unsigned container image using the `csadmin` tool:

```
csadmin image generate
  --package-name "{pqcop_short}-1.4.1"
  --version-str "1.4.1"
  --entry-point /usr/bin/entrypoint
  --network-conf network.json
  --packages-conf packages.json
  --rootdir container pqsdk.cs5
```

Example output:

```
Copying  /opt/nfast/python3/csd5/ppc64/usr/bin to /home/user/pqsdk/pqsdk-1.4.1-
win64/rfs_build_YlydVwJozb/usr/bin
Copying  /opt/nfast/python3/csd5/ppc64/usr/lib/python3.11 to /home/user/pqsdk/pqsdk-1.4.1-
win64/rfs_build_YlydVwJozb/usr/lib/python3.11
Copying  /opt/nfast/c/csd5/rootfs to /home/user/pqsdk/pqsdk-1.4.1-win64/rfs_build_YlydVwJozb/
```

The `network.json` and `package.json` files provided with PQCOP contain example

configurations that can be used to create a container with reasonable settings.

18. Sign the container image.

    We are going to use the `devkeyident.pem` used earlier to sign our container image. Ensure that this file is inside the `pqsdk-1.4.1-win64` folder (PQCOP unzip folder).

    ```
    csadmin image sign
      --devkeyname devkeyident --devcert devkeyident.pem
      --askeyname pqsdk-signer --out pqsdk-signed.cs5 pqsdk.cs5
    ```

    Example output:

    ```
    Loading `testOCS':
     Module 1: 0 cards of 1 read
     Module 1 slot 2: `testOCS' #6
     Module 1 slot 0: Admin Card #1
     Module 1 slot 3: empty
     Module 1 slot 4: empty
     Module 1 slot 5: empty
     Module 1 slot 2:- passphrase supplied - reading card
    Card reading complete.

    Loading `testOCS':
     Module 1: 0 cards of 1 read
     Module 1 slot 2: `testOCS' #6
     Module 1 slot 0: Admin Card #1
     Module 1 slot 3: empty
     Module 1 slot 4: empty
     Module 1 slot 5: empty
     Module 1 slot 2:- passphrase supplied - reading card
    Card reading complete.
    ```

19. Load the container image into each nShield 5c where it will be used:

    ```
    csadmin load --esn 7852-268D-3BF9 pqsdk-signed.cs5

    7852-268D-3BF9        UUID: 3d362d35-fc17-448a-b501-914fe0a680b0
    ```

    This outputs a UUID which should be recorded for subsequent use.

20. Start the container to make the code run inside the nShield HSM:

    ```
    csadmin start --esn 7852-268D-3BF9 --uuid 3d362d35-fc17-448a-b501-914fe0a680b0

    7852-268D-3BF9        SUCCESS; IP ADDRESS: XXX.XXX.XXX.XXX
    ```

21. Publish it so that it is easy for services and command line tools to find it.

    - **Linux**

      ```
      % /opt/nfast/python3/bin/python3 ./bootstrap.py -m 1 --uuid 3d362d35-fc17-448a-b501-914fe0a680b0
      ```

```
--cs5file pqsdk-signed.cs5
```

- **Windows**

```
> c:\> & 'C:\Program Files\nCipher\nfast\python3\python3.exe' ./bootstrap.py -m 1
      --uuid dd8756e4-0bdf-4b25-a640-9e6f6f700fa4
      --cs5file pqsdk-signed.cs5
```

Example output:

```
Creating SEE Connection succeeded
Published worldid object status: OK
```

Containers can be stopped with the `stop` subcommand and erased when no longer needed using the `destroy` subcommand.

Published objects become invalid when the container is stopped. This does not prevent them being published again later and they should not show up in pipsqueak, but they may create an error message.

2.6. Test the PQCOP

The primary interface to PQCOP is a shared library that exposes all features. However, a test program named `pipsqueak` is provided both as example code and a testing tool. During the execution of the examples in this guide, we will be generating keys, so the user needs to have permissions to write to `kmdata/local`.

1. (**Linux**) Add the user to the nfast group:

```
% sudo usermod -aG nfast user
```

2. Exit the shell and launch a new shell.
3. (**Linux**) Set `LD_LIBRARY_PATH`.

   `pipsqueak` depends on the shared library, so it is usually best to add the directory to the path. Assuming you are currently in the folder where PQCOP was unzipped, use:

```
% export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
```

This only applies to linux environments.

4. Test `pipsqueak`.

You should see the list of HSMS connected.

```
./pipsqueak

PQSDK: 1.4.1 (5F08-02E0-D947: 1.4.1)
```

2.7. Create LMS Keys

When you create LMS Keys, you can do them in two formats:

- Singular
- Sectorized

2.7.1. Singular LMS Keys

`pipsqueak` can generate LMS keys using the `generate` command. This command defaults to ML-DSA and so must be provided to create an LMS key. When doing so, six additional parameters are important:

- `lmscode` and `lmotscode`: These determine the specific kind of LMS key that will be generated and correspond to the values defined in RFC 8554.
- `expchunk`: This determines how the LMS key Merkle tree will be partitioned. This is especially useful for large height trees which can take a long time to generate.
- `expcache`: This determines how much of the Merkle tree to store for future use when loading the key. This avoids expensive recalculation of parts of the Merkle tree that might not be in use.
- `expcards` and `explease`: These determine whether LMS keys will be sectorized across smart cards, see Sectorized LMS Keys. When both are zero, which is the default, all signatures associated with the LMS key will be assigned to the nShield HSM where the key was generated. This is more secure but means that if the nShield HSM fails then all remaining signatures will be lost.

LMS keys currently are module protected only. Softcard and OCS cards are not currently supported.

1. Generate a singular LMS key:

```
./pipsqueak generate alias=test-lms algorithm=lms lmscode=sha256_m32_h5 lmotscode=sha256_n32_w8

PQSDK: 1.4.1 (5F08-02E0-D947: 1.4.1)
Generate LMS key started: test-lms (0.002 seconds)
  chunk: test-lms (0.316 seconds) 4 remaining
  chunk: test-lms (0.245 seconds) 3 remaining
```

```
  chunk: test-lms (0.250 seconds) 2 remaining
  chunk: test-lms (0.250 seconds) 1 remaining
  chunk: test-lms (0.010 seconds) 0 remaining
Public key (alias: test-lms):
  00000005 00000004 32345290 ee7b23cd 95f535ac 59cc8f34 6b4004d1 94236988
  b3439180 b273eeb9 5813bc40 25d21588 0c6a09b3 cdbd1fbd
```

2. List available keys:

```
./pipsqueak list

```
PQSDK: 1.4.1 (5F08-02E0-D947: 1.4.1)
Available Keys:
  test-lms LMS
```

3. Create an LMS signature:

```
./pipsqueak alias=test-lms message=test

PQSDK: 1.4.1 (5F08-02E0-D947: 1.4.1)
Public key (alias: test-lms):
  00000005 00000004 32345290 ee7b23cd 95f535ac 59cc8f34 6b4004d1 94236988
  b3439180 b273eeb9 5813bc40 25d21588 0c6a09b3 cdbd1fbd
Message (test):
  74657374
Signature (alias: test-lms, size: 1292, time: 0.018):
  00000003 00000004 cde06ca3 fe784329 22f6d3ef 3d34a20a b4fbe886 5b5e8413
  ba1851b1 54c09d11 b41ddeef c3a30966 350a23c1 be609294 5baa5489 55364b17
  7a29b75c e61175e6 ebb195db 9cab2d3e 0ac11f78 a9ef6d65 5364071c 37879bcd
  ca5c50c7 8061b93a 4d7869c2 f3946553 8d2e8330 8a35e778 016bf187 cc2b3faa
  a3c4d178 4e1e08df 3e784958 d613571a 5604bdc6 cd5dd11a d3767c62 9f675abf
  135b37fe 56bc351a 598e5b72 fc3e0b88 2ad78af5 b1e4bf6f 0f274aa5 31d4ff54
  9eaee715 28a7f59b b437f42e 0fb88b11 e414fb84 5c5b4fe4 f4553499 ffcf80bd
  a2f640bc 01b3e9a9 3e46a0c4 392b3b7c 23a676c5 3b63014e 2377e8e1 49d0dd10
  1a689f4f ffc44389 da34469f ab9ded46 09a2a92b eb0fb093 6a552d74 9ab91780
  ff69fce4 757bc52a 4a0a84ba 8abc4704 c75cd4b6 54cfaea6 9175230a 46dcfcd2
  28cc99a8 55c7b837 7580f435 4e6b448b 8bcbee94 5f50fd0a d5fdd0bb 7d813bc0
  0d4f702f dd0dda22 17ee771a 5de0e19d 73a037fb 8f2e6ac1 cb334f3c 7af39aa6
  eb62c6ad dd3df983 68f47d1d f3fac585 9198b508 1f8267a9 a87f8282 1c27b800
  2e8134ff e87c72c6 2633e135 d09013fd 81c777fa ff65953b 30346cbb 6f1e09d0
  d460308d aa835c07 39268e67 d169bcfc 6f83ba52 05c78aa3 26d54f94 53b8681e
  ba144ce0 db191db1 1c88f77d 4899ae13 3dd2b2c7 d18dd5d4 35fc9f54 d817b4d4
  b68613ea c7e8eb40 4f2930ed f2f6fea9 669c4e91 b05c3758 33f242c7 e4cd11b7
  df0e1cda 3e600e93 c5c08e11 96cee473 648cbb31 975c6b23 a4d6a51d aec1341d
  4f7dda8e 1814e7cb 71f0f82d d64c1e86 2686d8e1 4f2e9c04 f6c259c6 d9454b34
  04e87b11 383b5aa0 8d8ec976 06e5993c 41afa0ef 60d51e5c 2e4c65a3 a56b7118
  93e0a657 91c2602f bd231e6c e4c3ec3a 3af16461 4af9afc0 c19bcea9 4a5d9ee7
  8b57433c 724b6eb5 c4996f3c cf975407 38ea4c65 76570e12 aca86caa f2766fdd
  7bf164f9 618afa62 78e798d7 0a72d8b3 84657fdf 5925c9d4 128e6a89 957202fa
  0bf32def 4439b704 a8d9f004 fcf5f628 be4e833f d3d7f646 e773278f c99d9d14
  4e711a8f c967bb94 fe435691 45537c80 68fb5b1c abd24177 d57dc473 c8fdf8c1
  f67b9394 6d57905f f619608f ac7f1ec5 70b3d400 3fde29a5 498bf007 cbf6bcc2
  93aee25f 7338b4aa 52922c47 fc68b5d9 fff1c165 a90bb320 7e4d0523 6ed7ba94
  d03e5627 fa1fdf09 75d93724 c5f5ee3d a112456e cd41968b da0eb254 e5103787
  3e918f8b 64feeda7 9fc80fb7 923347bd 3e742578 cf5039e9 ffd8d596 c16a0ead
  dd0888fd e1adeb31 b744200a 3fb368b0 9021cc7a 44b3de85 d1395ace 261b845a
  a900756b a129f3fa a885f40a 453c7783 482ef060 c743fa94 0104175a ccbc4849
  cc75d233 9305b241 04b30f3b be7ae48a b12ea5b4 ed466f67 b96af4d0 c754f020
  d2ea73cf ec5fe52a ae2ca1fb eba6d9aa 2db47944 458fcafd 4bc5fd94 5e1ad494
  a919e4ec 6f723793 ac0dd16d 93c07fe7 b3a34bb3 9a5dc336 885bfd33 ef72fea5
  6e1399af bf9091c5 3d72bf03 458f9716 d61300b4 9ac98c7a 9f174c01 2b24ae68
```
```

```
  f56786a9 e27898f3 00000005 0d83249a 5f25e276 cc03e074 314f5194 8b4e6c47
  a75cbdeb 1fc87c9f 8c6776b9 69299a8a 47ecadd9 a6617052 edf4722d f0faef11
  ad8ddd84 1afa5ca9 c0d8d20a fa020143 9b6140cb 84a83989 32c8ac71 15aef0df
  8a9a73a6 a6b420ae a7894abf 50f686f0 0e3f7b3d de77b80c 59e7b5a1 fc90b250
  f2b96fde 637f5095 03889250 74f40760 2cb68947 f2f7f618 6093e36c ccd0092d
  0aa2a005 9a60eea8 6300875b
Verify (alias: test-lms, time: 0.013): true
```

4. Create an LMS signature where the message is in a file.

   In this example, the message to be signed is stored in a file. The signature is written to a file.

   - **Linux**

     ```
     echo "hello world" > /tmp/message.txt
     ./pipsqueak alias=test-lms msgfile=/tmp/message.txt sigfile=/tmp/message.sig
     ```

   - **Windows**

     ```
     mkdir c:\tmp
     echo "hello world" > c:\tmp\message.txt
     ./pipsqueak alias=test-lms msgfile=c:\tmp\message.txt sigfile=c:\tmp\message.sig
     ```

   Example output:

   ```
   PQSDK: 1.4.1 (5F08-02E0-D947: 1.4.1)
   Public key (alias: test-lms):
     00000005 00000004 32345290 ee7b23cd 95f535ac 59cc8f34 6b4004d1 94236988
     b3439180 b273eeb9 5813bc40 25d21588 0c6a09b3 cdbd1fbd
   Message (/tmp/message.txt):
     68656c6c 6f20776f 726c640a
   Signature (alias test-lms, size: 1292, time: 0.019)
     written to /tmp/message.sig
   Verify (alias: test-lms, time: 0.012): true
   ```

   Verification is free and it does not decrement the signature count.

5. Change the contents of the `message.txt` file to demonstrate that verification will fail if you try to verify the file:

   - **Linux**

     ```
     echo "Fail Signature" >> /tmp/message.txt
     ```

   - **Windows**

     ```
     echo "Fail Signature" >> c:\tmp\message.txt
     ```

6. Attempt to verify the file.

   It should fail.

```
./pipsqueak alias=test-lms msgfile=/tmp/message.txt verify=/tmp/message.sig

PQSDK: 1.4.1 (5F08-02E0-D947: 1.4.1)
Public key (alias: test-lms):
  00000005 00000004 32345290 ee7b23cd 95f535ac 59cc8f34 6b4004d1 94236988
  b3439180 b273eeb9 5813bc40 25d21588 0c6a09b3 cdbd1fbd
2026-02-11T15:43:43 lms_public_verify[hsslms.c:2463] VerifyFailed(11): expected
6b4004d1:94236988:b3439180:b273eeb9:5813bc40:25d21588:0c6a09b3:cdbd1fbd but computed
6995fff9:f101e5d0:1cadd5dd:8b67be4c:e5a65fb3:2e8d1a0e:63814c29:039b1ccc
ERROR: verify failed
```

7. Find out how many signatures remain:

```
./pipsqueak showinfo alias=test-lms

PQSDK: 1.4.1 (5F08-02E0-D947: 1.4.1)
  5F08-02E0-D947
Alias: test-lms
Algorithm: LMS
 sha256_m32_h5 sha256_n32_w8
  Remaining: 28 signatures
Public key (alias: test-lms):
  00000005 00000004 32345290 ee7b23cd 95f535ac 59cc8f34 6b4004d1 94236988
  b3439180 b273eeb9 5813bc40 25d21588 0c6a09b3 cdbd1fbd
```

2.7.2. Sectorized LMS Keys

Setting either or both of `expcards` and `explease` to a non-zero value will result in a sectorized LMS key. The sum of these values must be less than or equal to the height (h value) of the key.

The sum of `explease` and `expcards` can not be greater than `h`.

A user will be prompted to insert a blank card "two to the power of `expcards` times", for example:

- If `expcards` is one the user must supply two blank cards.
- If `expcards` is two the user must supply four blank cards.
- If `expcards` is three the user must supply eight blank cards.

Each of these cards will be usable for provisioning two to the power of `explease` times in total. As before, if `explease` is one, each card can be used twice. If `explease` is two, each card can be used four times. If `explease` is three, each card can be used eight times, and so on.

`explease` can be zero if `expcards` is greater than zero, in which case each card will be exhausted after the first use.

The remaining height (`h` minus `expcards` minus `explease`) determines how many signatures are provided with each lease provision operation. For example, an h10 key with `expcards` one and `explease` three will have two smart cards, each of which can be used eight times and each provision operation will provide 64 signatures (which is 2 to the power of (ten minus one minus three)).

Immediately after a sectorized key generation there are no nShield HSMs capable of creating signatures with the new key.

To create a sectorized LMS Key, use `pipsqueak generate`, for example, assuming h = 5:

- 2 smart cards: (expcards=1 means 2^1)
  - Each card can be used 8 times: (explease=3 means 2^3)
  - Each provision will provide 4 signatures: 2 ^ (5 - 1 - 3) = 2^2 = 4 signatures

```
./pipsqueak generate alias=test-lms algorithm=lms lmscode=sha256_m32_h5 lmotscode=sha256_n32_w8
expcards=1 explease=3
```

- 4 smart cards: (expcards=2 means 2^2)
  - Each card can be used 2 times: (explease=1 means 2^1)
  - Each provision will provide 4 signatures: 2 ^ (5 - 2 - 1) = 4 signatures

```
./pipsqueak generate alias=test-lms algorithm=lms lmscode=sha256_m32_h5 lmotscode=sha256_n32_w8
expcards=2 explease=1
```

- 1 smart card: (expcards=0)
  - The card can be used 4 times: (explease=2 means 2^2)
  - Each provision will provide 30 signatures: 2 ^ (5 - 0 - 2) = 8 signatures

```
./pipsqueak generate alias=test-lms algorithm=lms lmscode=sha256_m32_h5 lmotscode=sha256_n32_w8
expcards=0 explease=2
```

See the following reference table for more information:

h	sigs	expcard	explease	expsig	cards	lease / card	sig / lease	leases
5	32	0	3	2	1	8	4	8
10	1024	0	3	7	1	8	1284	8

h	sigs	expcard	explease	expsig	cards	lease / card	sig / lease	leases
15	32768	0	3	12	1	8	4096	8
20	1048576	0	3	17	1	8	131072	8
5	32	1	2	2	2	4	4	8
10	1024	1	3	6	2	8	64	16
15	32768	1	3	11	2	8	2048	16
20	1048576	1	3	16	2	8	65536	16
10	1024	3	3	4	8	8	16	64
15	32768	3	3	9	8	8	512	64
20	1048576	3	3	14	8	8	16384	64

If the combined `explease` and `expcard` values are greater than the height, the `pipsqueak generate` command will fail, as illustrated in the following example where the height is 5, `expcard` is 4 and `explease` is 3:

```
./pipsqueak generate alias=test-lms algorithm=lms lmscode=sha256_m32_h5 lmotscode=sha256_n32_w8 expcards=4
explease=3

PQSDK: 1.4.1 (5F08-02E0-D947: 1.4.1)
Generate LMS key started: test-slms (0.002 seconds)
2026-02-11T20:58:01 continueLMS_cache_next[see-commands.c:532] InvalidParameter(24): private key tree height is
less than the sum of card and lease exponents (5 < 4 + 3) in continueLMS_cache_next
ERROR: continueLMS failed on chunk
LMS key requires 16 cards
```

2.7.2.1. Sectorized Key LifeCycle.

This section explains the lifecycle of a sectorized key:

1. Key creation
2. Generate a key with some number of cards and leases.
3. Load a lease into the HSM with provesn.
4. A lease is loaded into the HSM with provesn. After that, signature can be used by calling the HSM or using pipsqueak to sign.
5. Signing with the HSM will stop working when it runs out of signatures.
6. Key can not be used anymore, requiring another provision.

7. The HSM cannot sign anymore so use card to provision another lease into the HSM; i.e. load some more signatures for the LMS key into the HSM.
8. Card runs out of leases.
9. The card runs out leases. If there are 8 lease per card, then after 8 uses, this card will be blank.

To create and use a sectorized key:

1. Format a blank card so that it can be used:

```
slotinfo --format --ignoreauth -m1 -s2
```

2. Ensure the card is inserted.
3. Create a sectorized LMS key.

   For this example, we will configure the LMS key as one card, two uses, and 16 signatures. The key name is `sec-lms` to indicated is sectorized.

```
./pipsqueak generate alias=sec-lms algorithm=lms lmscode=sha256_m32_h5 lmotscode=sha256_n32_w8 expcards=0
explease=1

PQSDK: 1.4.1 (5F08-02E0-D947: 1.4.1)
Generate LMS key started: sec-lms (0.004 seconds)
  chunk: sec-lms (0.247 seconds) 4 remaining
  chunk: sec-lms (0.245 seconds) 3 remaining
  chunk: sec-lms (0.247 seconds) 2 remaining
  chunk: sec-lms (0.273 seconds) 1 remaining
  chunk: sec-lms (0.002 seconds) 0 remaining
LMS key requires 1 card
Attempting to write to card in slot 2...
Card processed (0 remaining)
Public key (alias: sec-lms):
  00000005 00000004 81908b48 36be539b fc553a0e 720c695c 7d62e3b1 957663b4
  aa46dfd3 de008a62 ddf4a720 f8eb89cd fbadc6ca db93e2e2
```

4. Attempt to create a signature. This step should fail because you need to provision the key from the card to the HSM first.

```
./pipsqueak alias=sec-lms message=test

PQSDK: 1.4.1 (5F08-02E0-D947: 1.4.1)
Public key (alias: sec-lms):
  00000005 00000004 81908b48 36be539b fc553a0e 720c695c 7d62e3b1 957663b4
  aa46dfd3 de008a62 ddf4a720 f8eb89cd fbadc6ca db93e2e2
2026-02-11T16:35:18 dispatch_see_nvmemfile[dispatch.c:837] NotAvailable(73): no HSMs with NVRAM
6c6d737d:62e3b195:7663b4 file
ERROR: sign failed
```

5. Provision the key from the card to the HSM by inserting one of the associated smartcards and using `pipsqueak` with the `provesn` and `slot` options:

```
./pipsqueak provesn=5F08-02E0-D947 slot=2

PQSDK: 1.4.1 (5F08-02E0-D947: 1.4.1)
Attempting to provision 5F08-02E0-D947 from slot 2...
Provision successful.
```

6. Find out how many signatures remain:

```
./pipsqueak showinfo alias=sec-lms

PQSDK: 1.4.1 (5F08-02E0-D947: 1.4.1)
  5F08-02E0-D947
  - Slot 2: ic=3 lms
Alias: sec-lms
Algorithm: LMS
 sha256_m32_h5 sha256_n32_w8
  Remaining: 16 signatures
Public key (alias: sec-lms):
  00000005 00000004 81908b48 36be539b fc553a0e 720c695c 7d62e3b1 957663b4
  aa46dfd3 de008a62 ddf4a720 f8eb89cd fbadc6ca db93e2e2
```

7. Create an LMS signature:

   You should be able to do this 16 times before needing to provision again.

```
./pipsqueak alias=sec-lms message=test

PQSDK: 1.4.1 (5F08-02E0-D947: 1.4.1)
Public key (alias: sec-lms):
  00000005 00000004 81908b48 36be539b fc553a0e 720c695c 7d62e3b1 957663b4
  aa46dfd3 de008a62 ddf4a720 f8eb89cd fbadc6ca db93e2e2
Message (test):
  74657374
Signature (alias: sec-lms, size: 1292, time: 0.021):
  00000000 00000004 33e4b7e7 d7682df7 a7b9a31a 7bc774c0 15f80cec 9c21af35
  4c33d720 7e6d9140 8148c6bd bb74b160 6d696187 0e6d82ff 0e48143a c5ddde5e
  32c34d08 98e011d9 fe2fa2b1 68d0fe27 801dd029 76958b0d 97af07fd fcce6500
  e2376760 00509a62 71f0251a ff0f31b0 e5051e66 ff4d0079 138ca99c 34d3f7d2
  087d1881 47bfa5e2 e8f73c02 18e01e75 f63f5c73 852db8e6 9153f976 084a9718
  4b6ff4c0 745395ca 893b4b58 d1a74c7c 155688bc c11dda1a 232e0a86 5965bae6
  e5cd7d5d e716a446 6eb1c9a1 d5e1472b 0eb59941 ec466e12 3864fd25 5c520537
  61379195 afb67717 313803f0 eaa976d5 10caef40 07b1b013 5f182cbc d9916589
  1cfd60c6 6a2a3a5e a9c6c2ca 5d4c3a25 d8660354 cf8c7775 1e4a5880 7951967e
  02fd55e7 b04f7a2d 573464de c4a7a958 3eb55d03 d887c1b3 94761145 5f1bb42c
  04b4e582 4590ddd8 57fbcf77 91ce5e1b 2bdc979b 83568b2d 1fd3daae c3b95891
  90f37e90 c2c5cc60 3b924cb0 befc7abd ea08ac5b e1235559 4af27a4d 3fb30c43
  865481f0 6cbd72f1 b3ebf0d6 d879ce81 a0add27a 82fdf392 1f15d58f d05bf6ef
  2a162271 d18c7825 f79ec3c3 9ef124a3 2d929206 1df35759 c5de4432 a03c1974
  a46e0bba b63e91a1 9ca9ff21 9b7d07c6 dc394fdb 1aa66cb6 37db4ede d1f0fe61
  e67d2026 ef2adbbb 754e24f4 1a175ecf 7a1225f4 6424d21b 8084dd63 60d9b184
  22d7d3bd ae255cb5 be68df28 bc63475b a3cb2dac 1e353e46 d5814d0a e45fb197
  82f68e34 082663a2 40f1a2a5 1f875047 3c7d0b26 b7875765 7f8eabe0 0ef21d12
  0b4efae2 7fc0441c d3ac09c8 ad3a85ff eef81f30 da1d193f e7b50e52 00de854f
  3b30468a 85f459f4 338cc1a1 094fc765 121e8b92 48e10ab4 c366dd69 771dccd6
  22317de7 c0b4abba 3f9e2435 9fadaa4d 95d8be8f d12d2087 d146255c 26d87839
  6cb44665 29413820 bfc76423 0a6b2634 591d719e 50354c8c c46cf667 3dd7990e
  f76dd8ef f37bcc96 72d374e8 01c3cbca c3c903d0 a1afe2f0 b0b8ece6 3a22f905
  42749315 4110892f 510146eb 284e38d0 87d22838 66334ac5 e02db1ed 9be976ee
  0b31abf6 1a3ff113 5ddc733c 2e103edb 285da6ce 043bb4e5 adc664ae a4e3d40d
  f00891b8 6253d95a 6e855ec7 aa0657fe 733959f5 700cda16 76eb2fe7 e4dd51f0
  a5e21d3d fb5624f6 b1d816ba 48db98b1 f0dc0566 be88a357 107613ba 34de598f
```

```
  3f72fdee fe8484cb d9096361 1caede2a c6a67838 c999ae8c 3841a2c3 6f96cb99
  0125af73 2aaf26b5 6a44d1b6 aaac48cf 5b6cdf1f 33353076 a137044e 78087658
  da3c4230 83850c1a 6a1934bc d8658bc1 7ab7597f 5e3bde9c 677101cb 064e5e6b
  42aec66c 6871d26e 149bb1b6 b38f1e8f 7d3e6cde b3bc2c4c 4d8a193e 01fdbe9f
  3cc8985f 139546d4 edc376aa 28184f43 03e0ad86 edb41d85 76b86768 c7f7ed3f
  4874be58 45d50bb1 03868e8f bf8d39e0 41db4124 9a15030b 1d11c39d 8b430152
  9b0da619 531cd368 96065b3a 83120bbc bead562b cc8cec61 99c895c2 568843ac
  5c8e4e92 b85aaed8 b108c844 2961574f 247c50b8 bafc25e4 22352cc1 02788b95
  c8c699b7 4598c017 00000005 3bdf8a8c 949c3493 0079f1f4 2b5dd0e2 af9cfd48
  b10b8c80 79c888a4 750d9e6f 5a6fea41 10ee8371 1e589752 c59d657c 9bc1284a
  b99a1f6d d30a0cae 5f7ed61f db075210 acf8f858 69121eab 04787132 c9ef1dfa
  b515d302 19bb1c78 2edad85e 3e4665f5 a2391be5 11fd5d5c c9ea6dd9 7a1d21e8
  8a46f4a7 77ddaa84 0abfc339 dd9e1495 a7264087 9558d4ba adbda82d 5c68d697
  9ee47fd0 ef256ec3 f89a1c7c
Verify (alias: sec-lms, time: 0.012): true
```

On the last one available, you will see a message indicating it was the last one:

```
2026-02-11T16:47:20 dispatch_see_nvmemfile[dispatch.c:837] NotAvailable(73): no HSMs with NVRAM
6c6d737d:62e3b195:7663b4 file
Verify (alias: sec-lms, time: 0.001): false
```

If you run `showinfo` again, it will show that there are no remaining signatures:

```
./pipsqueak showinfo alias=sec-lms

PQSDK: 1.4.1 (5F08-02E0-D947: 1.4.1)
  5F08-02E0-D947
  - Slot 2: ic=3 lms
Alias: sec-lms
Algorithm: LMS
 sha256_m32_h5 sha256_n32_w8
  Remaining: 0 signatures
Public key (alias: sec-lms):
  00000005 00000004 81908b48 36be539b fc553a0e 720c695c 7d62e3b1 957663b4
  aa46dfd3 de008a62 ddf4a720 f8eb89cd fbadc6ca db93e2e2
```

If you try to sign, you will get an error:

```
./pipsqueak alias=sec-lms message=test

PQSDK: 1.4.1 (5F08-02E0-D947: 1.4.1)
Public key (alias: sec-lms):
  00000005 00000004 81908b48 36be539b fc553a0e 720c695c 7d62e3b1 957663b4
  aa46dfd3 de008a62 ddf4a720 f8eb89cd fbadc6ca db93e2e2
2026-02-11T16:50:18 dispatch_see_nvmemfile[dispatch.c:837] NotAvailable(73): no HSMs with NVRAM
6c6d737d:62e3b195:7663b4 file
ERROR: sign failed
```

8. Provision an nShield HSM again by inserting one of the associated smart cards and using `pipsqueak` with the `provesn` and `slot` options:

```
./pipsqueak provesn=5F08-02E0-D947 slot=2

PQSDK: 1.4.1 (5F08-02E0-D947: 1.4.1)
Attempting to provision 5F08-02E0-D947 from slot 2...
```

```
Provision successful.
```

If you run `showinfo` again, it indicates that there are signatures available now:

```
./pipsqueak showinfo alias=sec-lms

PQSDK: 1.4.1 (5F08-02E0-D947: 1.4.1)
Attempting to provision 5F08-02E0-D947 from slot 2...
Provision successful.user@dev-ubuntu:~/pqsdk/pqsdk-1.4.1-win64$ ./pipsqueak showinfo alias=sec-lms
PQSDK: 1.4.1 (5F08-02E0-D947: 1.4.1)
  5F08-02E0-D947
  - Slot 2: ic=3 lms
Alias: sec-lms
Algorithm: LMS
 sha256_m32_h5 sha256_n32_w8
  Remaining: 16 signatures
Public key (alias: sec-lms):
  00000005 00000004 81908b48 36be539b fc553a0e 720c695c 7d62e3b1 957663b4
  aa46dfd3 de008a62 ddf4a720 f8eb89cd fbadc6ca db93e2e2
```

9. Sign again.

   Do this 16 times until all signatures are used.

   ```
   ./pipsqueak alias=sec-lms message=test
   ```

   After using all of the signatures, if you try to provision again it will give you an error because you already used all of the signatures in the card.

10. Attempt to provision an nShield HSM again.

    This time you should get an error because the card run out of leases. You cannot use the key anymore.

    ```
    ./pipsqueak provesn=5F08-02E0-D947 slot=2

    PQSDK: 1.4.1 (5F08-02E0-D947: 1.4.1)
    Attempting to provision 5F08-02E0-D947 from slot 2...
    2026-02-11T21:48:37 command_see_provisionLMS[see-commands.c:785] UseLimitExceeded(7): no leases remaining
    on card in command_see_provisionLMS
    ERROR: provisionLMS command failed
    ```

This concludes the lifecycle of the sectorized key: one card, two uses, 16 signatures. You were able to see the usage of the card. It could only be used twice for provisioning the key signatures. Each time you got 16 signatures to be used.

2.8. Uninstall the PQCOP CodeSafe machines

In this section we describe how to uninstall the PQCOP CodeSafe machines from the HSM.

We are doing this here so our environment can be cleared before we create a mixed state environment were we will demonstrate the use of the LMS key using two HSMs of different types, the nShield Connect XC and the nShield 5c.

2.8.1. Uninstall the PQCOP CodeSafe machine from the nShield 5c

To uninstall the PQCOP CodeSafe machine from the nShield 5c, use the following commands. You will need the UUID of the CodeSafe machine.

1. Run `pipsqueak`:

```
./pipsqueak
PQSDK: 1.4.1 (7852-268D-3BF9: 1.4.1)
```

2. Stop the PQCOP CodeSafe machine:

```
csadmin stop --esn 7852-268D-3BF9 --uuid 607fc44d-953b-4b83-8c2b-945a22cef78e
7852-268D-3BF9        SUCCESS
```

3. Destroy the CodeSafe machine:

```
csadmin destroy --esn 7852-268D-3BF9 --uuid 607fc44d-953b-4b83-8c2b-945a22cef78e
7852-268D-3BF9        SUCCESS
```

4. Make sure it is not listed:

```
csadmin list
7852-268D-3BF9        No SEE machines currently installed
```

5. Run pipsqueak.
   You should get a connection error.

```
./pipsqueak
2026-02-18T11:19:56 dispatch__socket_connect[dispatch.c:172] OSErrorErrno(57): connecting to
XXX.XXX.XXX.XXX on port 8888: Connection refused
PQSDK: 1.4.1 ()
```

6. Clear the module:

```
nopclearfail -c -m 1
Module 1, command ClearUnit: OK
```

7. Remove published objects:

- **Linux**

```
/opt/nfast/python3/bin/python3 ./removepubobj.py -m 1 -p pqsdk
```

- **Windows**

```
c:\> & 'C:\Program Files\nCipher\nfast\python3\python3.exe' ./removepubobj.py -m 1 -p pqsdk
```

The script was provided by Entrust Support. It is not part of the release.

8. Run `pipsqueak` again.
   No HSMs should appear in the output.

```
./pipsqueak
PQSDK: 1.4.1 ()
```

2.8.2. Uninstall the PQCOP CodeSafe machine from the nShield XC

1. Run `pipsqueak`:

```
./pipsqueak
PQSDK: 1.4.1 (5F08-02E0-D947: 1.4.1)
```

2. Remove the CodeSafe machine:

```
loadsee-setup -r -m 1

Module #1:
  Machine file:             /opt/nfast/kmdata/custom-seemachines/pqsdk-1.4.1.sar
  Encryption key:
  Signing key hash:
  Userdata file:            /opt/nfast/kmdata/custom-seemachines/userdata.sar
  WorldID published object: pqsdk
  Postload helper:
  Postload args:

Erase this configuration? (yes/no): yes
Module #1 SEE auto-loading configuration removed.
Reload new configuration now? (yes/no): yes
Clear modules before reloading new configuration? (yes/no): yes
Modules cleared for new configuration.
Check the hardserver log to ensure the SEE machine was loaded correctly after clearing.
```

3. Clear the module:

```
nopclearfail -c -m 1
Module 1, command ClearUnit: OK
```

4. Remove published objects:

   - **Linux**

     ```
     /opt/nfast/python3/bin/python3 ./removepubobj.py -m 1 -p pqsdk
     ```

   - **Windows**

     ```
     c:\> & 'C:\Program Files\nCipher\nfast\python3\python3.exe' ./removepubobj.py -m 1 -p pqsdk
     ```

     The script was provided by Entrust Support. It is not part of the release.

5. Run `pipsqueak` again.
   No HSMs should appear in the output.

   ```
   ./pipsqueak
   PQSDK: 1.4.1 ()
   ```

2.9. Manage LMS Keys in a Mixed State Environment

This section demonstrates how to use a mixed state environment, where we have multiple HSMs used to manage the LMS Key. In our scenario, we will have 2 HSMs. The first HSM is an nShield 5c and the second HSM is an nShield Connect XC.

2.9.1. Enroll both HSMs.

We are going to enroll the nShield 5c as the first HSM and the Connect XC as the second HSM. After enrolling, make sure world is usable for both HSMs.

2.9.2. Setup CodeSafe again.

This time we are going to setup the CodeSafe environment first on the Connect XC and then set the `seeinteg` key used on the Connect XC as the `simple` key created during the nShield 5c CodeSafe setup.

1. Set up the Connect XC:

   Follow the CodeSafe setup as usual for the Connect XC.

   The Connect XC is module #2 in our setup.

2. Set up the nShield 5c:

   Setup the nShield 5c as usual. The only difference is that when you create the simple key for the ASK, you will retarget the same key from the Connect XC.

```
generatekey -r simple

module: Module to use? (1, 2) [1] >
slot: Slot to read cards from? (0-5) [0] > 2
from-application: Source application? (seeinteg, simple) [seeinteg] >
from-ident: Source key identifier? (pqsdk-signer) [pqsdk-signer] >
ident: Key identifier? [] > pqsdk-signer
plainname: Key name? [] > simple
key generation parameters:
 operation         Operation to perform     retarget
 application       Application              simple
 module            Module to use            1
 slot              Slot to read cards from  2
 verify            Verify security of key   yes
 from-application  Source application       seeinteg
 from-ident        Source key identifier    pqsdk-signer
 ident             Key identifier           pqsdk-signer
 plainname         Key name                 simple

Loading `testOCS':
 Module 1: 0 cards of 1 read
 Module 1 slot 0: Admin Card #1
 Module 1 slot 2: empty
 Module 1 slot 3: empty
 Module 1 slot 4: empty
 Module 1 slot 5: empty
 Module 1 slot 2: `testOCS' #6
 Module 1 slot 2:- passphrase supplied - reading card
Card reading complete.

Key successfully retargetted.
Path to key: /opt/nfast/kmdata/local/key_simple_pqsdk-signer
```

3. Run `pipsqueak` to check that both HSMs are visible:

```
./pipsqueak
PQSDK: 1.4.1 (7852-268D-3BF9: 1.4.1, 5F08-02E0-D947: 1.4.1)
```

2.9.3. Test the Mixed State Setup

On a mixed state setup, we have a nShield 5c setup as the first HSM and an nShield XC as the second HSM with the following settings:

- Module 1: 7852-268D-3BF9 - Slot: 2
- Module 2: 5F08-02E0-D947 - Slot: 2

To test the setup:

1. Format the card where the LMS key will be provisioned:

```
slotinfo --format --ignoreauth -m1 -s2
Formatting token in module 1 slot 2:
Formatted token OK
```

2. Check that `pipsqueak` works:

```
./pipsqueak

PQSDK: 1.4.1 (7852-268D-3BF9: 1.4.1, 5F08-02E0-D947: 1.4.1)
```

2.9.4. Singular LMS Keys

1. Generate a singular LMS Key:

```
./pipsqueak generate alias=test-lms-78b2f8 algorithm=lms lmscode=sha256_m32_h5 lmotscode=sha256_n32_w8

PQSDK: 1.4.1 (7852-268D-3BF9: 1.4.1, 5F08-02E0-D947: 1.4.1)
Generate LMS key started: test-lms-78b2f8 (0.002 seconds)
  chunk: test-lms-78b2f8 (0.199 seconds) 4 remaining
  chunk: test-lms-78b2f8 (0.200 seconds) 3 remaining
  chunk: test-lms-78b2f8 (0.203 seconds) 2 remaining
  chunk: test-lms-78b2f8 (0.197 seconds) 1 remaining
  chunk: test-lms-78b2f8 (0.011 seconds) 0 remaining
Public key (alias: test-lms-78b2f8):
  00000005 00000004 ffc4d719 ba63ca01 2e759d65 8237177a 4f079fe8 192235ae
  c3b148c1 25073113 dfa85315 aab5079c 960b5ba2 85d65a05
```

2. List available keys:

```
./pipsqueak list

PQSDK: 1.4.1 (7852-268D-3BF9: 1.4.1, 5F08-02E0-D947: 1.4.1)
Available Keys:
  sec-lms-3c6e64 LMS
  test-lms-78b2f8 LMS
  test-lms-28624b LMS
  test-lms-e77f9b LMS
  sec-lms-c98a6a LMS
  test-lms-dcc777 LMS
  sec-lms-8d134d LMS
```

3. Create an LMS signature:

```
./pipsqueak alias=test-lms-78b2f8 message=test

PQSDK: 1.4.1 (7852-268D-3BF9: 1.4.1, 5F08-02E0-D947: 1.4.1)
Public key (alias: test-lms-78b2f8):
  00000005 00000004 ffc4d719 ba63ca01 2e759d65 8237177a 4f079fe8 192235ae
  c3b148c1 25073113 dfa85315 aab5079c 960b5ba2 85d65a05
Message (test):
  74657374
Signature (alias: test-lms-78b2f8, size: 1292, time: 0.020):
  00000000 00000004 ad57253c 2eb71ee7 1d070b65 c7c48a07 ed88b378 3d6d1aef
  1f4046dc daba8ff3 0aaadf7f e726ccc3 2f757007 35b7e289 a904c1a7 b7b695da
```

```
  70101c74 b346d63d 6c6c16bf 626f6f89 4b163dc7 27e7f252 fc0b243b 515cc1b4
  0e5cd8be bcc5ed52 27e56816 88ed4770 cdb71737 3341a04c 5471b29d 29ec3d27
  4dd2ca04 91dbefdb 2200d156 d6136536 e7c4b96d 55cdbbe6 b2ea301c 236021ef
  831b1f2d 2efd7901 5f8f97c4 732306ef 0412cd65 b8cafb37 247ab7f7 6f83a416
  9c979373 bf5b5d95 7845a483 20be4426 e92c6cdb 9013c14b a0af42d7 b019ff55
  9b0d0a34 a9e4f8b9 519b24e2 8ad80fa0 ed25e283 9251a180 36e82bf5 84d3d65b
  b38986e9 50bac8f7 0f4c2b8c 253e5fd1 0384bbde 8e751aa9 02364e58 aebbea4e
  4749b8ee 5b4f2259 41404d1c 7d806867 83fe4864 405ab0f0 2086e5b0 9f4c86ca
  ffb60486 50b42dee 1dca2506 83c831dc c3a988fc 14d533f9 b2bda8c5 03ecce04
  aa4a91a1 481749d6 d4f9b9f3 ad353d31 72360d1b b12208a5 b8ae75cd 390c83ca
  5fdc899e af2d98d5 ea8632bf 654ea12b a8c6596d 17fa5216 8e50dfc9 af2230b7
  bba5beb3 ccb7fd35 688b3213 8606520f 7b490834 29096355 4fa22a46 5fd90c64
  a23d4380 f2493111 d75e0165 57b1d695 2fe3622e 9eae3314 98d8ca80 dc641752
  bca49978 65f40c88 7bfd6592 7f327c97 990f0570 91214da6 3b20a6bc dfc7092a
  a622f991 a4f906a5 8b3f3909 68aa8e22 7d9ce7a1 8a4fc8e7 ee15d79b 77b51536
  88436aef aa8235c1 b9ada358 0e01e1a5 1d3e0371 a6416096 15bf0a7d 3706fc3b
  4f2f750d 3c8b3d6c 1a308c5e 7686a50e 35eb70b1 ec920b25 346f2814 cb6fc7ca
  08884559 0bfb8853 3d08aa4b 5c980a0e bf2820e3 65661987 39638f66 3dee1f1e
  c3c4d81d aadb6fc4 77dc6cdd ad2618a0 e58e5769 11d22c78 b6730a1b 1c702bd8
  8e9eb983 36120b48 ec84bc59 1c7f5c70 22f65766 2e432634 867b0ab9 cb78267d
  d4c9990e c54cde98 706f62c1 8b795b5d 240e8ffc 9f3f813c 3864e39b 89ab56f6
  6e1e5e03 cd1d5c0b 9aec052a d5e55905 dfad70d9 aecfdcb9 1038e43b 544f3f6b
  a6a45726 c985dc29 46d83fbd 4aa370d1 ee79a2a0 063b5169 3b98298c 8d36363b
  832b73d3 f9c50646 1cc2c715 8fd258e1 a1fa1183 41a87488 e8d34976 71c0ee09
  b8ab6fc8 8437e67d 44202255 b1641eee 8d1cdff7 d230492c 86912f47 b7045b73
  8c3b09f9 2c7d1b87 9f091045 7304f18a 812e7c62 8af66ff7 fb0b4ba3 38a40e2a
  88c6afee 1ab5e19f c3afb32d c12851ed bdc395a4 9d8daa17 af0a3076 152b4334
  9652e4fe 3b0a845a be0aacdb 666cceae f98b90e2 4bb48bf8 37d301ab 75ac8af5
  cdd39a5d b803b99c 0e13b2af 46946eb6 a17abd1a 4a443a4c 0745b16b fc5da670
  7e38f81a 25abd8ec 3a665144 120ea192 284b9c22 91308953 07846cf5 29f4d1f9
  e36321a0 c065e7d0 fc14a502 987c2da2 c23dc605 af35120d 3ef84377 a3c5c071
  f34d9969 031cf75b af9db47b c7b5629c 09c79cf6 6c179854 cbc0baef 07235a58
  0982d343 e9680ca6 ec811f35 7c718783 07189a7d dcb1e3af dd60ac5a c03ca42a
  9af3a8ba 13a44351 00000005 08f7b4e8 3d91331d 7c745417 550137b9 84cde78b
  9d8f9db6 673baf8d 23c33865 b0a588ce 1a98d80d ed5223a9 2ad40b0b 082233c3
  b6d0275b aec38f4c 41c67b60 c0ae5641 1e5a6d76 0d376eb4 18530254 a305ea5f
  b7e52479 1c2697c8 f4e7b742 84f9adce b71bb5c9 fbd770ab 766f1b56 649640a1
  e3f5d7a8 ea590443 0d4fdc16 4bf153a3 8b333780 6feb2b70 444f0ada 5b38a420
  b81fa95a 2256dbef d508170c
Verify (alias: test-lms-78b2f8, time: 0.014): true
```

4. Create an LMS signature where the message is in a file:

   - **Linux**

     ```
     echo "hello world" > /tmp/message.txt
     ./pipsqueak alias=test-lms-78b2f8 msgfile=/tmp/message.txt sigfile=/tmp/message.sig
     ```

   - **Windows**

     ```
     mkdir c:\tmp
     echo "hello world" > c:\tmp\message.txt
     ./pipsqueak alias=test-lms-78b2f8 msgfile=c:\tmp\message.txt sigfile=c:\tmp\message.sig
     ```

   Example output:

   ```
   PQSDK: 1.4.1 (7852-268D-3BF9: 1.4.1, 5F08-02E0-D947: 1.4.1)
   Public key (alias: test-lms-78b2f8):
     00000005 00000004 ffc4d719 ba63ca01 2e759d65 8237177a 4f079fe8 192235ae
   ```

```
  c3b148c1 25073113 dfa85315 aab5079c 960b5ba2 85d65a05
Message (/tmp/message.txt):
  68656c6c 6f20776f 726c640a
Signature (alias test-lms-78b2f8, size: 1292, time: 0.021)
  written to /tmp/message.sig
Verify (alias: test-lms-78b2f8, time: 0.012): true
```

5. Verify the signature:

   - **Linux**

     ```
     ./pipsqueak alias=test-lms-78b2f8 msgfile=/tmp/message.txt verify=/tmp/message.sig
     ```

   - **Windows**

     ```
     ./pipsqueak alias=test-lms-78b2f8 msgfile=/tmp/message.txt verify=/tmp/message.sig
     ```

   Example output:

   ```
   PQSDK: 1.4.1 (7852-268D-3BF9: 1.4.1, 5F08-02E0-D947: 1.4.1)
   Public key (alias: test-lms-78b2f8):
     00000005 00000004 ffc4d719 ba63ca01 2e759d65 8237177a 4f079fe8 192235ae
     c3b148c1 25073113 dfa85315 aab5079c 960b5ba2 85d65a05
   Message (/tmp/message.txt):
     68656c6c 6f20776f 726c640a
   Verify (alias: test-lms-78b2f8): true
   ```

6. Change the contents of the `message.txt` file and attempt to verify it again to demonstrate that verification will fail if the file changes.

   - **Linux**

     ```
     % echo "Fail Signature" >> /tmp/message.txt
     % ./pipsqueak alias=test-lms-78b2f8 msgfile=/tmp/message.txt verify=/tmp/message.sig
     ```

   - **Windows**

     ```
     echo "Fail Signature" >> c:\tmp\message.txt
     ./pipsqueak alias=test-lms msgfile=c:\tmp\message.txt verify=c:\tmp\message.sig
     ```

   Example output:

   ```
   PQSDK: 1.4.1 (7852-268D-3BF9: 1.4.1, 5F08-02E0-D947: 1.4.1)
   Public key (alias: test-lms-78b2f8):
     00000005 00000004 ffc4d719 ba63ca01 2e759d65 8237177a 4f079fe8 192235ae
     c3b148c1 25073113 dfa85315 aab5079c 960b5ba2 85d65a05
   2026-02-24T16:35:48 lms_public_verify[hsslms.c:2463] VerifyFailed(11): expected
   4f079fe8:192235ae:c3b148c1:25073113:dfa85315:aab5079c:960b5ba2:85d65a05 but computed
   80be279a:69c5c69e:bcb908ba:fe177958:08cc234d:2af11f34:d5b521b7:0dd3e48b
   ERROR: verify failed
   ```

7. Check how many signatures remain:

```
./pipsqueak showinfo alias=test-lms-78b2f8

PQSDK: 1.4.1 (7852-268D-3BF9: 1.4.1, 5F08-02E0-D947: 1.4.1)
  7852-268D-3BF9
  - Slot 2: ic=13 blank
  5F08-02E0-D947
Alias: test-lms-78b2f8
Algorithm: LMS
 sha256_m32_h5 sha256_n32_w8
  Remaining: 30 signatures
Public key (alias: test-lms-78b2f8):
  00000005 00000004 ffc4d719 ba63ca01 2e759d65 8237177a 4f079fe8 192235ae
  c3b148c1 25073113 dfa85315 aab5079c 960b5ba2 85d65a05
```

2.9.5. Sectorized LMS Keys

1. Check that the combined values of `explease` and `expcard` is not greater than the `height`.

   The following example should fail because the sum of `explease` and `expcard` is greater than the `height`:

```
./pipsqueak generate alias=sec-lms-def0cc algorithm=lms lmscode=sha256_m32_h5 lmotscode=sha256_n32_w8
expcards=4 explease=3

PQSDK: 1.4.1 (7852-268D-3BF9: 1.4.1, 5F08-02E0-D947: 1.4.1)
Generate LMS key started: sec-lms-def0cc (0.002 seconds)
2026-02-24T16:35:48 continueLMS_cache_next[see-commands.c:532] InvalidParameter(24): private key tree
height is less than the sum of card and lease exponents (5 < 4 + 3) in continueLMS_cache_next
ERROR: continueLMS failed on chunk
LMS key requires 16 cards
```

2. Create a sectorized LMS key.

   For this example, we will configure the LMS key as one card, two uses, and 16 signatures. You must have a blank card presented for sectorized LMS keys to work.

```
./pipsqueak generate alias=sec-lms-def0cc algorithm=lms lmscode=sha256_m32_h5 lmotscode=sha256_n32_w8
expcards=0 explease=1

PQSDK: 1.4.1 (7852-268D-3BF9: 1.4.1, 5F08-02E0-D947: 1.4.1)
Generate LMS key started: sec-lms-def0cc (0.002 seconds)
  chunk: sec-lms-def0cc (0.200 seconds) 4 remaining
  chunk: sec-lms-def0cc (0.199 seconds) 3 remaining
  chunk: sec-lms-def0cc (0.197 seconds) 2 remaining
  chunk: sec-lms-def0cc (0.207 seconds) 1 remaining
  chunk: sec-lms-def0cc (0.002 seconds) 0 remaining
LMS key requires 1 card
Attempting to write to card in slot 2...
Card processed (0 remaining)
Public key (alias: sec-lms-def0cc):
  00000005 00000004 63b057b9 575d3dc3 639b85b9 d2533fad 1ef37030 8b8688f7
  02f44fc2 9c697464 9d28ea08 6f172909 04c0aa8a 9ca20d35
```

3. Attempt to create a signature.

   It should fail because it has not been provisioned.

```
./pipsqueak alias=sec-lms-def0cc message=test

PQSDK: 1.4.1 (7852-268D-3BF9: 1.4.1, 5F08-02E0-D947: 1.4.1)
Public key (alias: sec-lms-def0cc):
  00000005 00000004 63b057b9 575d3dc3 639b85b9 d2533fad 1ef37030 8b8688f7
  02f44fc2 9c697464 9d28ea08 6f172909 04c0aa8a 9ca20d35
2026-02-24T11:26:41 dispatch_see_nvmemfile[dispatch.c:837] NotAvailable(73): no HSMs with NVRAM
6c6d731e:f370308b:8688f7 file
ERROR: sign failed
```

4. Provision from the card to the HSM:

```
./pipsqueak provesn=7852-268D-3BF9 slot=2

PQSDK: 1.4.1 (7852-268D-3BF9: 1.4.1, 5F08-02E0-D947: 1.4.1)
Attempting to provision 7852-268D-3BF9 from slot 2...
Provision successful.
```

5. Check how many signatures remain:

```
./pipsqueak showinfo alias=sec-lms-def0cc

PQSDK: 1.4.1 (7852-268D-3BF9: 1.4.1, 5F08-02E0-D947: 1.4.1)
  7852-268D-3BF9
  - Slot 2: ic=13 lms
  5F08-02E0-D947
Alias: sec-lms-def0cc
Algorithm: LMS
 sha256_m32_h5 sha256_n32_w8
  Remaining: 16 signatures
Public key (alias: sec-lms-def0cc):
  00000005 00000004 63b057b9 575d3dc3 639b85b9 d2533fad 1ef37030 8b8688f7
  02f44fc2 9c697464 9d28ea08 6f172909 04c0aa8a 9ca20d35
```

6. Create an LMS signature.

   We can do this 16 times before needing to provisioning again.

```
./pipsqueak alias=sec-lms-def0cc message=test

PQSDK: 1.4.1 (7852-268D-3BF9: 1.4.1, 5F08-02E0-D947: 1.4.1)
Public key (alias: sec-lms-def0cc):
  00000005 00000004 63b057b9 575d3dc3 639b85b9 d2533fad 1ef37030 8b8688f7
  02f44fc2 9c697464 9d28ea08 6f172909 04c0aa8a 9ca20d35
Message (test):
  74657374
Signature (alias: sec-lms-def0cc, size: 1292, time: 0.020):
  00000000 00000004 948ddb4a 8381238e 6a579c13 e5c78a26 42fb4953 867aec18
  3969c380 0fc1f97f 2cde9765 b63e1ee1 ad9d2008 487239b2 f16dc501 227b74db
  f115341c a1f6add1 6c961cad 5dd8d294 7fd08115 0b04d8b4 e1587f91 ba2c59fd
  6a2db924 87d6f5df 1d7e1dff cfcf599c bc64f138 9808fd36 f21de8c4 b1513da9
  711f2e4e 40ba9828 337e2ef7 09c3cc40 699190fd eec808d1 35f5bc60 9ec4dedc
  62945804 7e529ccc a98ecae0 c10b4791 1fe3b23f 009fe02a f14f486f 267f7c23
```

```
  9fcce888 a1dca9c1 9c77891a aa7b163c edc0f48a 2ff90407 35264425 bf6559a8
  f2793316 99a8d2d7 9db383be 9571b8ee 23626967 ef3d21b9 0108eb63 f8058b61
  ed50755b 69b6cdeb b56d2863 ab2c968b d9241d13 c2d8123f 16c723e1 c866401f
  f436ae04 1572188a 05be22ab 80ed4983 8692cee4 3fcd8b68 91ba8d85 64f4b8e2
  3d3db884 4038695c fedfeeb9 60b6976f 5ecb89f4 85217db1 a6a7e4b9 cdd29d10
  4aeaf91e 77df74a7 654f01f2 5123b1fc b17687a9 deca941f 6b785541 23ab7539
  a3cbde83 bf911077 e960bc6f 25a6e9e1 d5e7402c bcd58cd6 89149061 d47b598d
  99cbcb24 b8bed1ed 69896fcf 901dc451 d5d090e2 7d9b12cb 2a41f42c fc538034
  d0c77eb6 0c6bc8e1 fafb2276 5f167785 ba9d1a73 c626eaae c3a8dd06 924f2d18
  0b9ffaf9 af356ee1 b61a2ef6 a4bdeea2 f57aa60a 05281ded 93d6e381 8a1950f3
  0dd8aa39 4d19b6f8 a53bd4ab 3f06524c 8170bce6 c4e4aeb6 6710373d 7a27c77b
  1c599d9a 368faa81 7c715d57 4254529d 5624d471 d2914767 475a02d4 94e34a8f
  6d2caa74 fa3c04e8 7e5d85fd 6350ae8c 6c370703 8651c751 2e03bbb5 86475a4f
  9b0f1052 a7e3b4da 424034a2 8eefafa9 695a2e5d 556087c1 79100bf8 c3be152b
  b7bf1c40 075bb1b3 44ad9146 7d7bc042 a5fca7b6 54a591b6 6cda67bc cacc6e38
  c49056d5 4f30ca5c 82b1255e 6d8cbc72 7f150178 61c182b8 11300f69 bb1a638f
  89938dd0 c2aaa78a 15617f0a 8fd15f78 78c51440 feda9e7f dcde2942 d1b6f59b
  e9de1960 3100115e 9027eb7a 7d99c874 4950ad86 29acfce9 c70fa7e4 73918010
  8ca77bea 4d9c9b8f 1d549885 fc676750 9b32cc20 1519d7e6 73b7c9ab 7bab1fc3
  8e32f22c da15a9c2 54dbc82f 08e8eed3 1538847d 7bb0dd0d 867176be f53eb25a
  ba8d5edd b3a83589 836dc0ae 51c9fe8b 6dcdeb67 44afd958 75655355 64fc5b7e
  27a7c726 56050ff9 68b2a24a 0028de0c 37b3f42c 8184a493 96439a54 bf8b96e5
  777c0b92 f636df8f 3241b3e7 5d824671 7fd4fdcf 6da3f535 45cd502c 9f362cce
  8ed0d379 a2d11a98 13371b39 ae6fa58b 12e6b058 5c3864c1 037db52d c876f60a
  054d0134 4d1658e7 43209a64 f7cddb93 7f8e7e1a 949542c5 38503815 9237090f
  eab5063e ba5ca970 32c6901a 3d583105 5a960cdb 1b721cf7 7030c3ee 94064e4e
  ae73189a 2972cf82 e8fe485e b6ac21f2 1e63939b b53bc723 c5272d8b f927e659
  3fa97648 f276fbad 22d3990e 7ece8fc4 5b8d76df 6a4715a6 0e28476f 7192aecc
  f726a121 d8ca7778 11c29e75 d095c274 86210a80 c6ab0051 35fc6cb0 04c981e0
  01940181 717107f1 00000005 9e671d09 fe3ba52f f9fc57bd 31270480 3242375e
  56e0ef3e ad17bb40 3f24437d 863cc431 4eb2cab6 022cac64 6701fbe9 8f85d928
  3897d3bd aa17efa7 8cec8438 1510974c ad689517 2333ab90 4bcd2772 2782ea40
  05b9cf44 a079101f 8ee08ba9 d605830f 1b1c89ef 2957bc36 4b54cde4 3ca8ca7d
  cd39268b e1a722c1 583a7be3 77342141 77f591ed 44040420 fcaae51f 60378c93
  920b7d36 99448634 56a3f517
Verify (alias: sec-lms-def0cc, time: 0.013): true
```

7. Check how many signatures remain.

   After signing 16 times, there should be no signatures left.

```
./pipsqueak showinfo alias=sec-lms-def0cc

PQSDK: 1.4.1 (7852-268D-3BF9: 1.4.1, 5F08-02E0-D947: 1.4.1)
  7852-268D-3BF9
  - Slot 2: ic=13 lms
  5F08-02E0-D947
Alias: sec-lms-def0cc
Algorithm: LMS
 sha256_m32_h5 sha256_n32_w8
  Remaining: 0 signatures
Public key (alias: sec-lms-def0cc):
  00000005 00000004 63b057b9 575d3dc3 639b85b9 d2533fad 1ef37030 8b8688f7
  02f44fc2 9c697464 9d28ea08 6f172909 04c0aa8a 9ca20d35
```

   If you try to sign now, you will get an error.

```
./pipsqueak alias=sec-lms-def0cc message=test

PQSDK: 1.4.1 (7852-268D-3BF9: 1.4.1, 5F08-02E0-D947: 1.4.1)
Public key (alias: sec-lms-def0cc):
```

```
  00000005 00000004 63b057b9 575d3dc3 639b85b9 d2533fad 1ef37030 8b8688f7
  02f44fc2 9c697464 9d28ea08 6f172909 04c0aa8a 9ca20d35
2026-02-24T11:27:06 dispatch_see_nvmemfile[dispatch.c:837] NotAvailable(73): no HSMs with NVRAM
6c6d731e:f370308b:8688f7 file
ERROR: sign failed
```

8. Present the card to the second HSM to provision on it:

```
./pipsqueak provesn=5F08-02E0-D947 slot=2

PQSDK: 1.4.1 (7852-268D-3BF9: 1.4.1, 5F08-02E0-D947: 1.4.1)
Attempting to provision 5F08-02E0-D947 from slot 2...
Provision successful.#
```

9. Run `showinfo` again, to show that we now have 16 more signatures.

```
./pipsqueak showinfo alias=sec-lms-def0cc

PQSDK: 1.4.1 (7852-268D-3BF9: 1.4.1, 5F08-02E0-D947: 1.4.1)
  7852-268D-3BF9
  5F08-02E0-D947
  - Slot 2: ic=13 lms
Alias: sec-lms-def0cc
Algorithm: LMS
 sha256_m32_h5 sha256_n32_w8
  Remaining: 16 signatures
Public key (alias: sec-lms-def0cc):
  00000005 00000004 63b057b9 575d3dc3 639b85b9 d2533fad 1ef37030 8b8688f7
  02f44fc2 9c697464 9d28ea08 6f172909 04c0aa8a 9ca20d35
```

10. Sign again.

    We can do this 16 times before we need to provision again.

```
./pipsqueak alias=sec-lms-def0cc message=test

PQSDK: 1.4.1 (7852-268D-3BF9: 1.4.1, 5F08-02E0-D947: 1.4.1)
Public key (alias: sec-lms-def0cc):
  00000005 00000004 63b057b9 575d3dc3 639b85b9 d2533fad 1ef37030 8b8688f7
  02f44fc2 9c697464 9d28ea08 6f172909 04c0aa8a 9ca20d35
Message (test):
  74657374
Signature (alias: sec-lms-def0cc, size: 1292, time: 0.025):
  00000010 00000004 116798bf 26fdd928 0d6983b6 085e4787 39367eef 3b276c63
  8ba0ac83 a82b857e 9647cd3b 83451492 191f0c5d 1a7ccee0 ac0aa00f 51e8e2f4
  bb3c7476 21559b13 8681f55f 707baa33 57b4e6a9 16876ea5 00997f03 7f92a387
  2ae77660 410e9102 f306e44c 71b43f15 c10d8ca3 ef2bc15e 99006a24 4f70800d
  9f08b92b 43214120 7240d518 ff38a9d4 2631c51d fa063d71 b67ccd92 9faa257e
  c95343bc adec959e c42641be 7801280b e621a22f afb480cb 0046f733 0a609caf
  621e22de 446269f7 59238cd6 6c230a40 2f21bd69 0cb61e92 9f91c7fe e174faa7
  42fe74b1 d8b04695 94b0e006 fc7780fc 8b13d073 5f82b01a 3ce70852 49299d3c
  de87dbfd ab24a5d7 08b23d6c 87299298 94698dcc f1d5d413 4cfaed83 5e582b9c
  f3324388 4080c5ef e4e50d9c b5f340e9 afca45de a09945c9 2becfcb9 16aca544
  cc8a0e25 6ee9ca86 a1ffad5e c5e76d5f 657f71b1 84c1f0f3 b74ed796 9df17463
  19b8729d 7a583c68 c6f6d111 a963ba90 534ae294 f61cc4a8 156e3dab 2be05be5
  5fdb7dce 77c2ca10 e14b29f5 bde90f6b a4772a73 5eb69f5f 803d6d70 cf147494
  7b4ebdf3 00630b72 cde20cf3 3322f475 765e659e f791f102 d0d16bd0 05e2df67
  e06e9cbb b0d20d36 6d3f2c67 97430ef6 347623ca 1e4ea554 5b5e31a5 86eefca6
  9fce4011 7816d617 bd090935 4ba14651 1688af72 ea4eb515 f68d96a1 fb56c183
```

```
  e770e634 6489015e 7990a4d0 cc0c5369 f6e82810 82fd25b1 c117c9b4 efea6492
  58f7118a 1af50d54 b623ff6a c6be9745 bda0c0ef c2b825bb 672e9115 50c4b657
  62ad50d0 ee79b79d ee93c78c 43a02a23 d7b008ee 93dfb87a e8776e15 534f5168
  20649bbf a4a1ca03 d74038ad 710d2d95 57ea03a1 b3e70bbc b0a373f0 2747fcc3
  f9657d9f 5e2d73fa c88cd5f7 e0e1187f 966fc4df 1042d61a 3f2615b2 117556ca
  24169a3f 125e38d8 2a9907a8 8fc843db a88452bb 8042a34a cffda58d b9a7d352
  8e1c62c6 9b6616e1 171e1592 7cb5c237 0409fccd f4cb7842 d41f19e3 4ff4a3cf
  45dec48b 99d0799b 4f3226a1 3706197c 3c069b4f 492d31b2 7f852081 7efc84ca
  56cda44a c9f5560f 5677a38e 40ec1749 4e10588d 387cf3bf ad4a30b8 0c8c70a7
  27710184 073715e6 900873c4 3bdbe68d 6060c032 a1da8446 e8705b44 c8509c23
  a42e0ba2 758e273a 242675c2 c868ed41 f4780582 c7dae642 c42cfc68 ee9003c2
  3739cadc 0a226f63 27c95803 0f9d710b 846a144a 65c96588 d9d92f32 bc980c76
  3b81e6bc 61945a51 1c790678 61a4e56d a036b940 9fb1ec03 ae0c5c9f 5bd59c99
  9f312dbf 1ec00568 de8f9f74 1bfe9585 4572cffb 5d77edd7 75701328 113c69ef
  64292073 cc1bf195 27581487 0195329f 1f5e3b67 419c0108 a4eadc5f 9564219c
  e8d6db35 b47ccfd2 e4a7e748 49286841 98e6971e 09d5ced3 b58314c9 a5660717
  c746af8b 7f42259d c68791c2 1091fed6 a48dbfa1 f36e7b7f e15f9e68 c9e8a5ef
  3251402b f916ffdc 875dc79f e579e1a1 d5bdfa73 59c2ca4c 07ed6052 95c7bb44
  e3eb2568 335bf1be eac51dac ff621b27 ed17fb4a 4ba81fda 7afe862c 240708ad
  6b408fa5 c5dc8dc3 00000005 e0293d06 e1ad683a d757fe8c 906c46e5 57d4c196
  28364cc0 9899c976 97f451fa b218dea7 8346e0ba ac731e33 f22fe31c 84a8e585
  f7f8dbe6 1ea8ca4e 6696c73f 7b8f1969 a093b36c 7722d26f b83bae1b 5336a7ff
  b7549767 41e26651 a12c5cde 34154a68 88663b34 d6c35bb9 c089b847 584f3401
  0765fabf de306f6b 4588d78e 4ba0bf8e 31b04d42 3fbbccda 8eda7df7 121223e0
  bb13c50f 8da186ff b2db269e
Verify (alias: sec-lms-def0cc, time: 0.013): true
```

11. Check the number of signatures left.

    After signing 16 times, there should be no signatures left.

```
./pipsqueak showinfo alias=sec-lms-def0cc

PQSDK: 1.4.1 (7852-268D-3BF9: 1.4.1, 5F08-02E0-D947: 1.4.1)
  7852-268D-3BF9
  5F08-02E0-D947
  - Slot 2: ic=13 lms
Alias: sec-lms-def0cc
Algorithm: LMS
 sha256_m32_h5 sha256_n32_w8
  Remaining: 0 signatures
Public key (alias: sec-lms-def0cc):
  00000005 00000004 63b057b9 575d3dc3 639b85b9 d2533fad 1ef37030 8b8688f7
  02f44fc2 9c697464 9d28ea08 6f172909 04c0aa8a 9ca20d35
```

    If you try to sign again, you will get an error.

```
./pipsqueak alias=sec-lms-def0cc message=test

PQSDK: 1.4.1 (7852-268D-3BF9: 1.4.1, 5F08-02E0-D947: 1.4.1)
Public key (alias: sec-lms-def0cc):
  00000005 00000004 63b057b9 575d3dc3 639b85b9 d2533fad 1ef37030 8b8688f7
  02f44fc2 9c697464 9d28ea08 6f172909 04c0aa8a 9ca20d35
2026-02-24T11:28:02 dispatch_see_nvmemfile[dispatch.c:837] NotAvailable(73): no HSMs with NVRAM
6c6d731e:f370308b:8688f7 file
ERROR: sign failed
```

12. Attempt to provision again.

It will fail as you only can lease two times for this key.

```
./pipsqueak provesn=5F08-02E0-D947 slot=2

PQSDK: 1.4.1 (7852-268D-3BF9: 1.4.1, 5F08-02E0-D947: 1.4.1)
Attempting to provision 5F08-02E0-D947 from slot 2...
2026-02-24T16:18:03 command_see_provisionLMS[see-commands.c:785] UseLimitExceeded(7): no leases remaining
on card in command_see_provisionLMS
ERROR: provisionLMS command failed
```

2.9.6. Conclusion

On a mixed state environment, the operations are essentially the same. The main difference is that the `ASK` key used by the nShield 5c during the CodeSafe setup had to be regenerated from the `seeinteg` key used in the Connect XC CodeSafe setup.

Chapter 3. Additional resources and related products

3.1. nShield HSMs

3.2. nShield as a Service

3.3. Entrust products

3.4. nShield product documentation