nShield Security World

CodeSafe 5 v13.9.3 Developer Guide

03 March 2026

Table of Contents

1. Introduction

CodeSafe is a runtime on the Entrust nShield HSM that allows third-party developers to run their own code within the secure boundary of the module. Using the CodeSafe Developer Kit, developers write their own CodeSafe Apps, cross-compile them and package them to run on the HSM. While on the HSM, the CodeSafe App is segregated from the actual keys loaded onto the module, including the keys the App uses. This means that CodeSafe can be used without affecting the FIPS 140 validation of the module it runs on.

Where the HSMs provide security controls on key usage, CodeSafe provides control over application code. Depending on the runtime used, you are either sending nCore commands to the HSM, or designing your own protocol to send data and commands back and forth.

The CodeSafe Developer Kit includes the Secure Execution Engine (SEE) technology. The CodeSafe product comprises a suite of cross-compilers and support tools that allow you to develop SEE machines.

With CodeSafe, you can build and deploy Trusted Agents to perform application-specific security functions on your behalf on unattended servers, or in unprotected environments where the operation of the system is outside of your direct control. Examples of Trusted Agents include digital meters, authentication agents, timestamp servers, audit loggers, digital signature agents and custom encryption processes.

Traditionally, HSMs have protected cryptographic keys within a defined security boundary; SEE allows you to extend that security boundary to include code that utilizes those protected keys. The code itself is signed to provide additional protection.

2. Overview of CodeSafe 5

2.1. Applications as container images

In CodeSafe 5, the application is a container image, meaning a complete filesystem image that can contain multiple executables, libraries, scripts, and data files.

This has the following benefits:

- Data files can be written to the local filesystem and persisted over container shutdown and restart.
- The application can comprise multiple co-operating processes. This can enhance security by separating memory spaces and reliability by allowing individual processes to be restarted if they crash or leak memory.
- Third-party or pre-existing source code that works on Linux can be built and run without modification.
- Standalone tools can be executed as subprocesses.
- Dynamically-loaded libraries work in a regular way. Code architectures that make use of plug-in modules make code development easier and reduce the attack surface by excluding unwanted code.

2.2. Easy and fast network connectivity

nShield 5 HSMs and CodeSafe 5 containers are logically connected via TCP and UDP networking. The container running the SEE Machine can receive incoming connections from the host side app, establishing two-way communication between host side app and SEE machine. Existing software that makes use of incoming or outgoing network connections can run with little or no modifications.

Kernel-implemented networking provides good performance both for throughput and for latency compared to previous HSM models.

2.3. 'Secure by default' client communication

The CodeSafe 5 execution environment includes both a configurable firewall and an SSH server. The firewall is set according to configuration in the signed CodeSafe 5 application package so that only the network ports required by the application are allowed. The CodeSafe SSH server also allows a secure tunnel to be configured to the CodeSafe 5 application to encapsulate a plaintext TCP protocol within a secure layer. The client credentials required

to access this tunnel can be configured using the support tools, or (when using CSEE) can be configured automatically by the nShield software.

This means that applications, including applications ported from older CodeSafe SEE machines, can benefit from strong authentication of their clients and protection from unauthorized network traffic without additional code.

2.4. Better language support

The CodeSafe 5 SDK supports:

- C and C++
- Python

  The container environment has a regular Linux filesystem and supports system calls for network and file I/O, so a wide range of standard and third-party Python modules can be used without modification.

  Refer to the nShield Security World Release Notes for information about the supported version for this release.

CodeSafe applications can be written using mixed languages with the usual range of IPC and calling mechanisms available to the developer.

2.5. Developer authentication

CodeSafe 5 uses Entrust X.509 certificates to link the CodeSafe application to a real-world developer identity through code signing.

This allows the administrator of an HSM to, for example, restrict the HSM to authorized in-house applications or to those provided by trusted development partners.

3. CodeSafe 5 Application Authentication

3.1. Introduction

CodeSafe on Solo XC, Connect XC, and earlier HSMs allowed any operator with access to the nCore API to load executable code onto the HSM. The CodeSafe functionality uses the signature on this code to control access to other Security World keys, but there is no other significance attached to these signatures. So, simply loading code onto these HSMs does not require access to any particular signing key.

CodeSafe 5 aims to improve deployment security by ensuring that every CodeSafe applica tion is cryptographically linked to a developer with a real-world identity, and by allowing the administrator of each HSM to choose which developer identities are permitted to run Code Safe applications on that HSM. nShield 5s and 5c HSMs will not run applications which have not been signed by an authenticated developer.

3.1.1. CodeSafe 5 signing keys

The developer of a CodeSafe 5 application (the "Developer") uses the following keys:

- A *Developer ID* key
- One or more *Application Signing Keys* ("ASK"s)

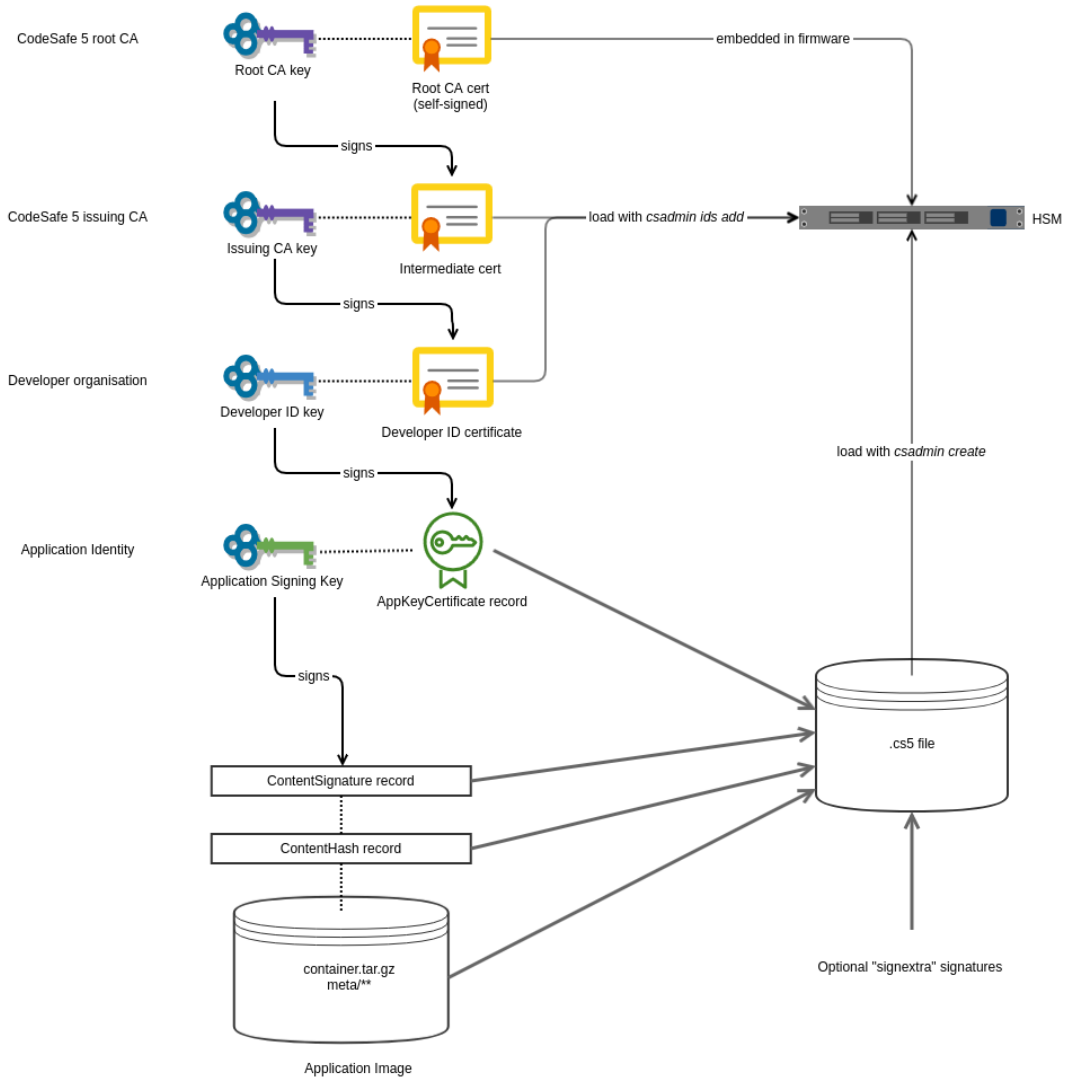
The Developer ID key belongs to a company or other organization, and that organization has an active nShield support contract or similar relationship with Entrust. Entrust issues a *Developer ID Certificate*, which is an X.509 certificate binding the Developer ID key to a organization name, location, and DNS domain name ("Common Name").

Application Signing Keys are owned by the Developer's organization, and should be distinct for each CodeSafe 5 application they produce.

An ASK is associated with an 'Package Name', which is a short name for the application, assigned by the Developer. Taken together, the Package Name and Common Name form a unique identifier for that application, which is cryptographically assured through the authen tication mechanism.

3.2. Application Authentication chain of trust

This is shown in the diagram below:



The **CodeSafe 5 root CA** is owned by Entrust; it has a corresponding X.509 root certificate which is embedded in the HSM firmware. This certificate is the root of trust for the application authentication mechanism.

The **CodeSafe 5 issuing CA** is also owned by Entrust; it is an online CA operated by nShield Technical Support. This creates a Developer ID Certificate in response to a CSR generated by the `csadmin ids create` command.

The **Developer ID Key** here is responsible for authenticating Application Signing Keys as belonging to the Developer's organization. It does this by creating a certificate (an `AppKeyCertificate` record) binding that ASK to a Package Name.

The **Application Signing key** is used to generate the signature on the CodeSafe 5 application itself. This signature, and the `AppKeyCertificate` record which describes the ASK, are embedded in the *.cs5* file for delivery to the HSM. This is done with the `csadmin image sign`

command.

3.2.1. Application signatures and ACLs

A central feature of CodeSafe is the ability to restrict use of Security World keys to specific CodeSafe images. For Solo XC and earlier HSMs, this was done by signing the machine image (actually, signing the `userdata` associated with the machine) using a `seeinteg` key, and then referring to the key-hash of that `seeinteg` key in the Access Control List (ACL) of another Security World key. For instance, this can be done by using `generatekey` with the `seeintegname` or `--trusted-certifier` options (see Working with keys).

When the CodeSafe machine wishes to use a key which is restricted in this way, it adds a `CertType_SEECert` certificate to the nCore API commandas described in the Command Certificates documentation.

In CodeSafe 5, the Application Signing Key's identity is available to authorize actions. When creating keys with `generatekey`, the ASK's key name can be used for the `seeintegname` parameter, or its hash passed as the `--trusted-certifier` value. The ASK key hash can then be used in `CertType_SEECert` command certificates.

However, in some situations it is desirable to use existing Security World keys created for Codesafe XC applications, i.e. to use an existing `seeinteg` key's identity for command authorization rather than a new ASK. To do this, the existing `seeinteg` key can be used to add an additional signature on the CodeSafe 5 machine image. For details see the csadmin image signextra command.

3.2.2. Handling of Developer ID certificates

As the diagram above shows, X.509 certificates (the Developer ID and Issuing CA certificates) are not placed in the *.cs5* file itself. Instead, the HSM maintains a certificate database, separate from the CodeSafe 5 machine image itself. The following commands access that database:

- `csadmin ids add` is used to upload a Developer ID or Issuing CA certificate to the HSM.
- `csadmin ids list` lists the current contents of the certificate database.
- `csadmin ids remove` deletes a given certificate by the database.

These are described in detail at Developer Identity management.

It is possible to have more than one certificate for the same Developer ID key (e.g. with different validity dates) loaded. The validation procedure (described below) will accept any such certificate that is currently valid.

The `csadmin ids add` command will therefore set which Developer(s) are authorized to run applications on that HSM. An organization deploying CodeSafe 5 will normally use this to lock down each HSM to a single identified Developer.

3.2.3. Offline signing using the Developer ID key

The diagram above shows that the Developer ID key signs an `AppKeyCertificate` record, but it is not involved in creating a signature on the application itself. This allows the Developer ID key to be kept in a separate environment to the ASK; it is only necessary to use the Developer ID key when an ASK is first created. After that, the `AppKeyCertificate` can be stored and reused repeatedly. Only the ASK needs to be available when creating the signed *.cs5* file.

The workflow for using the Developer ID key in a separate (e.g. offline) environment is as follows:

- Use `generatekey` to create a new ASK
- In the same Security World, run `csadmin image presignask`. This will generate a small file containing data to be signed.
- Transfer this file to the environment hosting the Developer ID key (this can be a different Security World).
- In this environment, run `csadmin image signask`. This accepts the file above and signs it. The output is another file containing an `AppKeyCertificate` record.
- Transfer this output file back to the ASK's environment.
- When the ASK is signing an application, use `csadmin image sign` with the `--signedask` option to pass in the `AppKeyCertificate` file from the previous step.

These operations need a Developer ID certificate file to be available in both environments, but note this is just used to identify the Developer ID key; it does not form part of the signed *.cs5* file.

3.2.4. Validation of CodeSafe 5 images

The contents of a *.cs5* file are validated when it first loaded onto the HSM (using `csadmin create`), and when the machine is started (using `csadmin start`, or when the HSM boots if auto-start is enabled).

Validation of a *.cs5* file consists of the following steps:

- Check that the file has at least one signature (a `ContentSignature` record, and a `ContentHash` record of the appropriate type).
- Check that the verification (public) key for each signature can be located.
  - If the key is an ASK, it needs to have a matching `AppKeyCertificate` record.
  - If the key is a "signextra" key, it needs a matching `KeyData` record.
- Check that the signature can be verified with the public key data, and that the `ContentHash` record matches the hash of the file's payload.
- For each `AppKeyCertificate` record:
  - Locate a matching Developer ID certificate in the Certificate Store. Certificates are matched by the X.509 Subject Key ID field, which is set from the key hash of the Developer ID key.
  - Check that the Developer ID certificate is valid (including a check for expired certificates, and a check that a valid Issuing CA certificate has been loaded). If not, the database is searched for another Developer ID certificate with a matching Subject Key ID. If any valid certificate is found, the key is accepted.
  - Check that the public key in the Developer ID certificate can be used to verify the signature in the `AppKeyCertificate` record.
  - If the `AppKeyCertificate` signed message includes a start date or expiry date, check that the current date is within the allowed range.
- Check that there is at least one complete chain from a `ContentSignature` through an ASK back to the CodeSafe 5 root CA.

You can use the `npkgtool inspect` command to view the records in a *.cs5* file.

4. Install the CodeSafe 5 SDK on Linux

1. Make sure that the following nShield ISO images are available locally:
   - `SecWorld_Lin64-13.x.y.iso`
   - `Codesafe_Lin64-13.x.y.iso`

   Where `<x.y>` are the same versions for Security World and CodeSafe.

2. Create a mount directory for each ISO:

```
mkdir ~/secworld_iso_mountpoint
mkdir ~/codesafe_iso_mountpoint
```

3. Mount the ISO images to their respective directories:

```
sudo mount <PATH_TO>/SecWorld_Lin64-13.x.y.iso ~/secworld_iso_mountpoint/
sudo mount <PATH_TO>/Codesafe_Lin64-13.x.y.iso ~/codesafe_iso_mountpoint/
```

   The nShield CodeSafe 5 hostside is located in tarballs under:

```
ls ~/codesafe_iso_mountpoint/linux/amd64/
csdref.tar.gz  csd.tar.gz
```

   The nShield Security World hostside is located in tarballs under:

```
ls ~/secworld_iso_mountpoint/linux/amd64/
ctd.tar.gz   devref.tar.gz  javasp.tar.gz  ncsnmp.tar.gz
ctls.tar.gz  hwsp.tar.gz    jd.tar.gz      raserv.tar.gz
```

4. Untar the tarballs into the root directory:

```
tar -zxvf ~/codesafe_iso_mountpoint/linux/amd64/csd.tar.gz -C /
tar -zxvf ~/codesafe_iso_mountpoint/linux/amd64/csdref.tar.gz -C /
tar -zxvf ~/secworld_iso_mountpoint/linux/amd64/ctd.tar.gz -C /
tar -zxvf ~/secworld_iso_mountpoint/linux/amd64/devref.tar.gz -C /
tar -zxvf ~/secworld_iso_mountpoint/linux/amd64/javasp.tar.gz -C /
tar -zxvf ~/secworld_iso_mountpoint/linux/amd64/ncsnmp.tar.gz -C /
tar -zxvf ~/secworld_iso_mountpoint/linux/amd64/ctls.tar.gz -C /
tar -zxvf ~/secworld_iso_mountpoint/linux/amd64/hwsp.tar.gz -C /
tar -zxvf ~/secworld_iso_mountpoint/linux/amd64/jd.tar.gz -C /
tar -zxvf ~/secworld_iso_mountpoint/linux/amd64/raserv.tar.gz -C /
```

   This installs the nShield CodeSafe 5 SDK to `/opt/nfast/c/csd5` and the nShield CodeSafe 5 SDK Python files to `/opt/nfast/python3/csd5`.

5. Install the CodeSafe 5 SDK on Windows

5.1. Prerequisites

Make sure that the following nShield ISO images are available locally:

- `SecWorld_Windows-13.x.y.iso`
- `Codesafe_Windows-13.x.y.iso`

Where `<x.y>` are the same versions for Security World and CodeSafe.

5.2. Install the Security World Software

1. Log in as Administrator or as a user with local administrator rights.
2. Mount the Security World Software ISO image and navigate into the mounted directory.
3. Launch `setup.msi`.
4. Follow the on-screen instructions.
5. Accept the license terms and select **Next** to continue.
6. Specify the installation directory and select **Next** to continue.
7. Select **Install**.
8. Select **Finish** to complete the installation.

5.3. Install CodeSafe 5

1. Mount the CodeSafe 5 SDK ISO image and navigate into the mounted directory.
2. Launch `setup.msi`.
3. Follow the on-screen instructions.
4. Accept the license terms and select **Next** to continue.
5. Specify the installation directory and select **Next** to continue.
6. Select **Install**.
7. Select **Finish** to complete the installation.

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

6. Create and sign CodeSafe 5 applications with csadmin

The `csadmin` tool is new to CodeSafe 5. It performs a variety of operations for creating and managing CodeSafe 5 machines. It replaces a number of other utilities such as `tct2`, `elftool` and `loadmache`; these are not needed for CodeSafe 5.

This page describes functionality available in version 13.9.3 or later of the CodeSafe 5 SDK, and version 13.8.4 and later of the nShield 5s firmware. Earlier version of the SDK and firmware do not support all the features described here.

6.1. Running csadmin

`csadmin` itself is included in the main Security World software distribution. Ensure you have completed the steps described at Install the Security World software, and that `/opt/nfast/bin` (Linux) or `C:\Program Files\nCipher\nfast\bin` (Windows) is included in your `PATH`.

A number of `csadmin` subcommands create or use Security World keys. Your HSM must be enrolled in a Security World (see nShield Security World v13.9.3 Management Guide and sub-pages) to perform these operations.

Subcommands for management of CodeSafe 5 machines on an HSM require access to the *launcher* service's SSH client key. You should run these commands as `root` or `Administrator`, or as a user in the appropriate group as described in Permissions on SSH keys.

All `csadmin` subcommands support the `-h` or `--help` option for displaying the help text associated with that command.

6.2. Developer Identity management

When your company or organization starts to develop CodeSafe 5 applications, you must create a Developer ID key and obtain a certificate provided by Entrust nShield Technical Support, which links that key to the organization's real-world identity. For a description of Developer ID keys and related topics, see CodeSafe 5 Application Authentication.

6.2.1. Create a Developer ID key and CSR

Developer ID certificates are created with `csadmin ids create`:

```
$ csadmin ids create --help
usage: csadmin ids create [-h] --keyname KEYNAME [--appname APP] [-m MODULE] --x509cname COMMON_NAME [--
x509country COUNTRY]
                          [--x509province STATE_OR_PROVINCE] [--x509locality LOCALITY] --x509org ORGANIZATION
                          [--x509orgunit ORGANIZATIONAL_UNIT] [--verbose]

options:
  -h, --help            show this help message and exit
  --keyname KEYNAME     Name for the certificate's key.
  --appname APP         appname (e.g. simple, seeinteg) of the certificate's key
  -m MODULE, --module MODULE
                        Module to generate the key with.
  --x509cname COMMON_NAME
                        The CN part of the key's DN.
  --x509country COUNTRY
                        The C part of the key's DN.
  --x509province STATE_OR_PROVINCE
                        The ST part of the key's DN.
  --x509locality LOCALITY
                        The L part of the key's DN.
  --x509org ORGANIZATION
                        The O part of the key's DN.
  --x509orgunit ORGANIZATIONAL_UNIT
                        The OU part of the key's DN.
  --verbose             Print verbose logs
```

The **KEYNAME** parameter is the Security World name (ident) for the key. If a key with this name doesn't exist it is created using the appropriate parameters; if it already exists a new CSR is generated for it.

The X.509 **ORGANIZATION** and **COMMON_NAME** fields should match the name and DNS name associated with your organization, as understood by Entrust Technical Support.

6.2.1.1. Example usage

```
$ csadmin ids create --keyname developerid --x509cname yourcompany.com --x509org "yourorganization" --x509orgunit
"CodeSafe App Development"

Generate key 'developerid' ...

Loading `TestOCS':
 Module 1: 0 cards of 1 read
 Module 1 slot 0: empty
Card reading complete.

OK
Generate a CSR in 'developerid.csr' ...
OK
Created CSR file 'developerid.csr'. Please send it to Entrust Support
```

The output file `developerid.csr` is a short ASCII text file which can be send via email, or uploaded to the support portal.

The certificate returned will be another short text file; in these examples we use `developerid.pem` to refer to this file. This contains no secret information so can be made available

in source repositories or from file servers.

6.2.2. Upload and view Developer ID certificates

If you use hsc_codesafe automatic configuration utility, you do not need to upload Developer ID certificates manually. You should ensure the certificate file is available at `/opt/nfast/kmdata/cscerts` for Linux, or `C:\ProgramData\nCipher\Key Management Data\cscerts` for Windows. The certificate will be uploaded when required by the `hsc_codesafe` utility.

Developer ID certificates can be uploaded to an HSM using the `csadmin ids add` subcommand:

```
$ csadmin ids add developerid.pem
FEDC-BA09-8765        SUCCESS
```

The list of currently loaded certificates is retrieved using `csadmin ids list`:

```
$ csadmin ids list
FEDC-BA09-8765        SUCCESS
Certificates:
{'serialNumber': '705640309799691626348045512084088225642474535983', 'subject': 'Common Name: CodeSafe
Development, Organizational Unit: nCipher, Organization: Entrust, Country: GB', 'keyid':
'406386a0f3de84618ce0ee4022f67aa86768e6a1', 'authKeyid': 'd81e9f34b89c76b5c20cddd489b6f2d8d66cafd5', 'notBefore':
'2023-03-03 09:25:25+00:00', 'notAfter': '2033-03-13 09:25:25+00:00'}
{'serialNumber': '116622002464887670342028053697306624797', 'subject': 'Common Name: yourcompany.com,
Organization: yourorganization, Organizational Unit: CodeSafe App Development', 'keyid':
'b6687020bb527f56ef426d174eee61ac16baa16d', 'authKeyid': '406386a0f3de84618ce0ee4022f67aa86768e6a1', 'notBefore':
'2025-10-30 16:28:00+00:00', 'notAfter': '2028-10-30 16:27:59+00:00'}
```

The certificate file you receive back from Entrust Support will contain a chain of certificates, including the Entrust issuing CA's certificate. Both certificates will be loaded by `csadmin ids add`, so it is normal to see two or more entries in the above list.

6.2.3. Check a Developer ID certificate

The `csadmin ids validate` command checks whether a certificate in the HSM's database can currently be validated. The certificate is identified by its serial number.

```
$ csadmin ids validate -s 705640309799691626348045512084088225642474535983
FEDC-BA09-8765        SUCCESS
```

The certificate will be validated when it is first uploaded. If this operation fails the likely reasons are (a) the certificate has expired, or (b) the Entrust Issuing CA certificate has been

removed from the database.

6.2.4. Retrieve a Developer ID certificate

A copy of a certificate in the HSM's database can be retrieved with `csadmin ids get`. Again, the certificate is identified by its serial number.

```
$ csadmin ids get -s 705640309799691626348045512084088225642474535983
DE5C-23C7-E473        Certificate: -----BEGIN CERTIFICATE-----
MIIDKDCCAoqg...
-----END CERTIFICATE-----
```

6.2.5. Remove a Developer ID certificate

This is done with the `csadmin ids remove` command, specifying the certificate's serial number:

```
$ csadmin ids remove -s 116622002464887670342028053697306624797
FEDC-BA09-8765        SUCCESS
```

6.3. Image creation and signing

6.3.1. Create an image file

The `csadmin image generate` command takes a directory containing a CodeSafe 5 machine image, and packs it into an CodeSafe 5 image file. By convention these have the extension **.cs5**. The file generated by this command does not contain any signatures; these are added later using `csadmin image sign`.

```
$ csadmin image generate --help
usage: csadmin image generate [-h] --package-name PACKAGE_NAME --version-str VERSION_STR --entry-point
ENTRY_POINT --network-conf NETWORK_CONF
--packages-conf PACKAGES_CONF --rootdir ROOTDIR [--verbose] CS5FILE

positional arguments:
  CS5FILE               The cs5 file to be handled

optional arguments:
  -h, --help            show this help message and exit
  --package-name PACKAGE_NAME
                        Short name describing the product contents
  --version-str VERSION_STR
                        Version number of this package contents
  --entry-point ENTRY_POINT
                        Full path, within the container, to the entry point application to be executed upon start
  --network-conf NETWORK_CONF
                        Full path, outside the container, to the network config file to be copied into the
```

```
container meta data
  --packages-conf PACKAGES_CONF
                        Full path, outside the container, to the extra packages config file used to copy
additional packages into container rootfs
  --rootdir ROOTDIR     Directory where the contents of the new container are located
  --verbose             Print verbose logs
```

PACKAGE_NAME is the package name associated with the ASK; it does not need to match either the Security World name (ident) of the ASK, or of the output file, but it is recommended that a consistent naming convention is adopted by the developer.

VERSION_STR can be chosen freely. It can be retrieved using `csadmin image inspect` (below) but is not otherwise interpreted by the HSM.

ENTRY_POINT, **NETWORK_CONF**, **PACKAGES_CONF** and **ROOTDIR** specify the image contents and control information. They are described in detail at Image creation options.

6.3.1.1. Example usage

```
$ csadmin image generate --package-name "MyCodeSafeApp" --entry-point /usr/bin/entrypoint --network-conf network-
conf.json --packages-conf extra-packages-conf.json --version-str 1.0  --rootdir container/ myapp.cs5
INFO: creating content package
INFO: Creating content tar ball
INFO: Creating copy of source file: network-conf.json into dest: cs5_build/meta/network-conf.json
INFO: Creating copy of source file: extra-packages-conf.json into dest: cs5_build/meta/extra-packages-conf.json
INFO: Creating compressed tar ball cs5_build/extra-packages.tar.gz out of cs5_build/extra-packages
INFO: Creating compressed tar ball cs5_build/container.tar.gz out of container/
INFO: Creating uncompressed tar ball content.tar out of cs5_build
INFO: creating cs5 file myapp.cs5
INFO: adding content hash to the package

INFO: File myapp.cs5 was created successfully!
```

6.3.2. Inspect a CodeSafe 5 image

Use `csadmin image inspect` to view the information in a *.cs5* file header:

```
$ csadmin image inspect webserver_mod.cs5
Type               codesafe-container
Platform           nShield5
Format             1
PackageName        webserver_mod
Version            1.0
EntryPoint         /usr/bin/entrypoint
```

6.3.3. Create an Application Signing Key

An Application Signing Key can be created using the generatekey utility. It should use the `seeinteg` app name, and be an ECDSA key using the NIST P521 curve. It is strongly recom-

mended that this uses OCS ('token') protection. For example:

```
/opt/nfast/bin/generatekey --batch --module=1 --cardset=some-ocs seeinteg type=ECDSA curve=NISTP521 ident=ask-
myapp plainname=ask-myapp protect=token
```

6.3.4. Sign with an Application Signing Key

A CodeSafe 5 machine must be signed with an ASK before it can be run. This is achieved with the `csadmin image sign` command:

```
usage: csadmin image sign [-h] --askeyname ASKEYNAME [--askapp ASKAPP] [--devkeyname DEVKEYNAME] [--devapp
DEVAPP]
                          [--devcert DEVCERT] [--startdate STARTDATE] [--expirydate EXPIRYDATE] [--signedask
SIGNEDASK] [--out OUT]
                          [--verbose]
                          CS5FILE

positional arguments:
  CS5FILE               The cs5 file to be signed

options:
  -h, --help            show this help message and exit
  --askeyname ASKEYNAME
                        Name (ident) of the application signing key
  --askapp ASKAPP       appname (e.g. simple, seeinteg) of the application signing key
  --devkeyname DEVKEYNAME
                        Name (ident) of the developer signing key
  --devapp DEVAPP       appname (e.g. simple, seeinteg) of the developer signing key
  --devcert DEVCERT     The signed developer certificate PEM file
  --startdate STARTDATE
                        Start of validity period for the signed ASK cert in Unix time (default: no start date)
  --expirydate EXPIRYDATE
                        End of validity period for the signed ASK cert in Unix time (default: no expiration date)
  --signedask SIGNEDASK
                        Signed ASK certificate file. Developer key, start, and expiry date params are ignored
when present
  --out OUT             Name of the output file. If not specified, the cs5 file is overwritten
  --verbose             Print verbose logs
```

This must be run in an environment where the **ASKEYNAME** key is available (i.e. with an attached HSM in the ASK's Security World). There are two different workflows for signing:

- "Online" workflow: if the Developer ID key is also available in this environment, the **DEVKEYNAME** and **DEVAPP** parameters identifying this key should be specified. In this case the `--startdata` and/or `--expirydate` options may be specified.
- "Offline" workflow: this avoids the need to use the Developer ID key when signing an application. In this workflow you use the `--signedask` option instead of `--devapp` and `--devcert`. The offline signing workflow is described here, and the associated commands are documented at Offline signing commands.

The `--devcert` option must be given, to supply information about the Developer ID key. However, this certificate is not copied into the *.cs5*

> file itself; it is uploaded separately (see Upload and view Developer ID certificates) when the CodeSafe 5 machine is deployed on an HSM.

6.3.5. Add extra signatures to a CodeSafe 5 image

The `csadmin image signextra` command allows a CodeSafe 5 image to be signed using additional keys which are not ASKs. This enables interoperation with CodeSafe deployments on XC and earlier HSMs, where a existing `seeinteg` key signs the machine image, and Security World application keys have been bound to that `seeinteg` key (see Application signatures and ACLs).

```
$ csadmin image signextra --help
usage: csadmin image signextra [-h] --appname APPNAME --key KEY
                               [--keymech
{SHA256Hash,SHA384Hash,SHA3b256Hash,SHA3b384Hash,SHA3b512Hash,SHA512Hash}] [--out OUT]
                               [--verbose]
                               CS5FILE

positional arguments:
  CS5FILE               The cs5 file to be signed

options:
  -h, --help            show this help message and exit
  --appname APPNAME     appname (e.g. simple, seeinteg) of the signing key
  --key KEY             Name (ident) of the signing key
  --keymech {SHA256Hash,SHA384Hash,SHA3b256Hash,SHA3b384Hash,SHA3b512Hash,SHA512Hash}
                        Key hashing mechanism (default SHA512Hash).
  --out OUT             Name of the output file. If not specified, the cs5 file is overwritten
  --verbose             Print verbose logs
```

The **APPNAME** and **KEY** parameters specify the signing key. The current firmware supports RSA, DSA, ECDSA and KCDSA (if enabled) key types here.

Example usage

```
$ csadmin image signextra --appname seeinteg --key seeintkeyname --keymech SHA512Hash --out ~/hello-signed-
extra.cs5 ~/hello.cs5
```

6.3.6. Offline signing commands

The Developer ID key can be kept in an offline system, or use a different HSM or Security World to the ASK. The offline signing workflow uses two operations described below.

6.3.6.1. Generate an ASK certificate message

When an ASK is created, it is bound to a CodeSafe 5 application identity (the 'package' name) by a signature using the Developer ID key.

The `csadmin image presignask` command generates a message to be signed:

```
$ csadmin image presignask --help
usage: csadmin image presignask [-h] --packagename PACKAGENAME --askeyname ASKEYNAME [--askapp ASKAPP] --devcert
DEVCERT
                                [--startdate STARTDATE] [--expirydate EXPIRYDATE] --out OUT [--verbose]

options:
  -h, --help            show this help message and exit
  --packagename PACKAGENAME
                        Name of the package
  --askeyname ASKEYNAME
                        Name (ident) of the application signing key
  --askapp ASKAPP       appname (e.g. simple, seeinteg) of the application signing key
  --devcert DEVCERT     The signed developer certificate PEM file
  --startdate STARTDATE
                        Start of validity period for the signed ASK cert in Unix time (default: no start date)
  --expirydate EXPIRYDATE
                        End of validity period for the signed ASK cert in Unix time (default: no expiration date)
  --out OUT             Name of the output file
  --verbose, -d         Print verbose logs
```

The **ASKAPP** and **ASKEYNAME** specify the key which will be placed in the certificate message. This command must be run in a Security World environment where at least the public key value for this key is available.

PACKAGENAME is the Developer-chosen package name for that key. This must match the **PACKAGENAME** which will be given to `csadmin image create` when creating images to be signed with this ASK.

The **DEVCERT** file must be available (to supply information about the Developer ID key), but the Developer ID key itself is not needed.

6.3.6.2. Sign an ASK Certificate message

The `csadmin image signask` command uses the Developer ID key to sign the output from `presignask` operation above. It creates a file containing an `AppKeyCertificate` record, suitable for passing as the `--signedask` parameter to `csadmin image sign`.

```
$ csadmin image signask --help
usage: csadmin image signask [-h] --devkeyname DEVKEYNAME [--devapp DEVAPP] --devcert DEVCERT --out OUT [--
verbose] ASKFILE

positional arguments:
  ASKFILE               The unsigned ASK message file being signed

options:
  -h, --help            show this help message and exit
  --devkeyname DEVKEYNAME
                        Name (ident) of the developer signing key
  --devapp DEVAPP       appname (e.g. simple, seeinteg) of the developer signing key
  --devcert DEVCERT     The signed developer certificate PEM file
  --out OUT             Name of the signed output ASK cert file
  --verbose, -d         Print verbose logs
```

7. Building CodeSafe 5 applications

7.1. General SDK use

The CodeSafe 5 SDK provides the tools necessary to build and run SEE machines on nShield 5 HSMs. The CodeSafe 5 SEE machines are containerized. The SDK provides the structure of the container, including a root file system, libraries required for communication with the nCore API, and libraries to enable communication between the SEE machine and the host. The SDK provides headers and libraries for building applications and a root file sys tem for the container containing runtime libraries, binary tools such as `touch`, `cat`, `grep` and optionally a Python distribution.

7.2. Prerequisites

- Access to an nShield HSM, with an installed Security World, for managing signing keys.
- CodeSafe 5 SDK installation, as described at xref:install-codesafe5-linux.adoc or xref:install-codesafe5-windows.doc
- `gcc` 8.x or later (Linux), or Visual Studio 2022 or later (Windows) is required for building host-side code.
- You must create a Developer ID key for your organization, and obtain a certificate for it, as described at xref:deploy-codesafe5-applications.adoc#csadmin-ids-subcmd.

7.3. SDK file structure overview

7.3.1. SDK location

The default installation location of the CodeSafe 5 SDK is:

- **Linux**: `/opt/nfast/c/csd5/`
- **Windows**: `C:\Program Files\nCipher\nfast\c\csd5\`

Some tools required for SEE machine operations might be found elsewhere in the main install. For example, `csadmin`, which enables loading, starting, and stopping SEE machines, is installed in the following default locations:

- **Linux**: `/opt/nfast/bin/csadmin`
- **Windows**: `C:\Program Files\nCipher\nfast\bin\csadmin` (Windows)

These cases are described in the following sections as required.

7.3.2. Container root file system

The container root file system is located in:

- **Linux**: `/opt/nfast/c/csd5/rootfs/`
- **Windows**: `C:\Program Files\nCipher\nfast\c\csd5\rootfs\`

This root file system contains two main parts: binary files and libraries.

7.3.2.1. Binaries

`rootfs/bin/` (Linux) or `rootfs\bin\` (Windows) contains many useful common Linux binaries that you might need within the container such as `cat`, `grep`, and `touch`.

`rootfs/sbin/` (Linux) or `rootfs\sbin\` (Windows) contains the `init` script for the container.

7.3.2.2. Libraries

`rootfs/lib/` and `rootfs/usr/lib/` (Linux) or `rootfs\lib\` and `rootfs\usr\lib\` (Windows) contain runtime libraries that may be needed by applications built with the SDK.

7.3.3. Image creation options

CS5 images are generated with `csadmin image generate` (see the csadmin image subcommand documentation.)

Generating an image requires the name of the CS5 file and the use of the following mandatory command-line arguments:

- `--package-name`
- `--version-str`
- `--entry-point`
- `--network-conf`
- `--packages-conf`
- `--rootdir`

The following items are also required:

- A container directory (not necessarily named "container") that points to what would be the SEE machine's root directory.

  This directory must include any files used by the application, including the entry point

program, for example:

```
container/
├── home
└── usr
    └── bin
        └── entrypoint
```

The container directory can be located anywhere in the host file system. Ensure you pass the full path to the `generate` command via the `--rootdir` argument, as specified in the command usage.

- An entry point program.

  This is the program that runs when the SEE container is started (on launcher start). It must be made executable so it can be launched accordingly. In the previous example, the entry point program is in `container/usr/bin/entrypoint`.

- A network configuration file. (See Example network-conf.json file.)

  The valid range for `container_port` is 1024 - 65535.

- A file with extra packages information. (See Example extra-packages-conf.json file)

`--entry-point` points to the full path of the executable program relative to the container's root.

7.3.3.1. Example extra-packages-conf.json file

```
{
    "packages": [{
            "package": "python",
            "description": "python 3.8 binaries",
            "host_path": "python3/csd5/ppc64/usr/bin",
            "machine_path": "usr/bin",
            "exclude": ""
        },
        {
            "package": "python",
            "description": "python 3.8 libraries",
            "host_path": "python3/csd5/ppc64/usr/lib/python3.8",
            "machine_path": "python3",
            "exclude": ""
        },
        {
            "package": "binaries",
            "description": "binaries for script support 1.0.0",
            "host_path": "c/csd5/rootfs/bin",
            "machine_path": "bin",
            "exclude": ""
        }
    ]
}
```

7.3.3.2. Example network-conf.json file

```
{
    "incoming" : {
        "tcp" : {
            "protos" : [ "ipv6" ], "ports" : [ 8000, 8001, 8888 ]
        }
    },
    "outgoing" : {
        "udp" : {
            "protos" : [ "ipv4" ], "ports" : [ 53 ]
        }
    },
    "ssh_tunnel" : {
             "container_port" : 8000
    }
}
```

7.3.3.3. Example entry point script

```
#!/bin/sh
export PYTHONHOME=/usr/bin
export PYTHONPATH=/usr/lib/python3.8/:/usr/lib/python3.8/lib-dynload:/usr/lib/python3.8/site-packages
python -m http.server --directory / --bind :: 8888
```

7.3.4. CMake

The SDK installs a directory which includes CMake toolchains used for building example

SEE machines:

- **Linux**: `/opt/nfast/c/csd5/cmake`
- **Windows**: `C:\Program Files\nCipher\nfast\c\csd5\cmake`

These toolchains can serve as examples themselves for creating custom toolchains.

7.3.5. Include directories

The SDK provides two directories with header files that can be included along with their respective libraries to provide additional functionality in SEE machines. These headers are stored in:

- **Linux**:
  - `/opt/nfast/c/csd5/gcc/*`
  - `/opt/nfast/c/csd5/include-see/*`
- **Windows**:
  - `C:\Program Files\nCipher\nfast\c\csd5\gcc\*`
  - `C:\Program Files\nCipher\nfast\c\csd5\include-see\*`

7.3.6. SEE specific libraries

The C libraries which are specific to SEE machines, including `seelib.a` and `librtusr.a`, are located in:

- **Linux**: `/opt/nfast/c/csd5/lib-ppc64-linux-musl/\*`
- **Windows**: `C:\Program Files\nCipher\nfast\c\csd5\lib-ppc64-linux-musl\*`

These libraries must be included to enable critical SEE machine functionality such as communication with the nCore API.

The Python module specific to SEE machines is `seeapi.py`. This module is located under Python site packages in `nshield.ipcdaemon.seeapi`. This must be imported as `SEEAPI` to enable critical SEE machine functionality such as communication with the nCore API.

7.3.7. Compatibility

The CodeSafe 5 SDK and nShield 5 HSMs are sufficiently different from previous implementations that applications for HSM models cannot run on the nShield 5.

Applications must be rebuilt using the CodeSafe 5 SDK to run on the nShield 5.

Where possible, APIs such as SEElib and SEEJobs behave as they did with previous HSM models, but signing and deployment details for CodeSafe 5 applications differ.

7.4. Building new SEE machines with SEElib

An SEE machine is a container image with a complete filesystem which can be loaded onto an CodeSafe 5-enabled HSM as part of a container. The SEElib library enables SEE machines to interface with the nCore API via the IPC daemon.

Source code is compiled using one of the GCC cross-compilers supplied with the CodeSafe SDK. For details of required compiler options, toolchains, makefiles and so on, see the CMake files supplied with the examples, as well as Build and sign example SEE machines on Linux and Build and sign example SEE machines on Windows.

The container image must be signed using the `csadmin` utility tool.

7.4.1. Developer authentication

CodeSafe 5 requires a signed CodeSafe image to run SEE machines on the HSM.

The CodeSafe developer needs to request a developer ID certificate by sending a Certificate Signing Request (CSR) to Entrust support. The tool used to create the CSR is integrated into the HSM software as a subcommand of `csadmin` utility.

For security purposes, a developer keypair must be created and stored within the HSM. In addition, the keypair must be OCS protected to provide authorization control on its use. The developer keypair will be created by `csadmin` if it does not already exist.

After the certificates are received, they are installed on the HSM and are used to sign CodeSafe application images with the `csadmin` tool.

The implementation of this is described in more detail in Create and sign CodeSafe 5 applications with csadmin.

7.4.2. Deploying SEE machines

After the code has been compiled, built, and signed, the `csadmin` utility tool is used to deploy the SEE machine. It is used to load the signed CodeSafe application image and then to start the SEE machine. The SEE machine then runs the `entrypoint` including the `main()` function.

For more information on the `csadmin` utility, see Create and sign CodeSafe 5 applications

with csadmin.

7.4.3. SEE machine initialization requirements

An SEE machine must initialize the `SEElib` before making use of any of the `SEElib` functionality. This is done by calling `SEElib_init()`. It is recommended that this call is made immediately within the `main()` function of an SEE machine.

By default, `SEElib_init()` will also enable SEEJobs support for communication from the host via `Cmd_SEEJob` and `Cmd_FastSEEJob` sent via a client hardserver. This uses port `8888` within the container, and this port should be set as the `ssh_tunnel` port in the container configuration as shown in later examples. It is possible to enable a non-default port for SEEJobs by calling `SEElib_SetPort(8000)` (for example) before `SEElib_init()`. To disable SEEJobs listening altogether, run `SEElib_SetPort(0)` before `SEElib_init()`

7.4.4. SEElib Functions

After initialization, SEElib functions can be used to communicate with the nCore API via the IPC daemon that is included as part of the container image. These functions behave the same as in previous CodeSafe versions although the underlying implementation has changed.

7.4.4.1. SEElib_Transact()

To send a command to the nCore API and block waiting for a reply:

```
int SEElib_Transact(struct M_Command *cmd, struct M_Reply *reply)
```

This sends the `cmd` command to the nCore API and waits for the reply to be written to `reply`.

7.4.4.2. SEElib_Submit() / SEElib_Query()

To send a non-blocking command to the nCore API:

```
int SEElib_Submit(M_Command *cmd, M_Reply *reply, PEVENT ev, SEElib_ContextHandle tctx)
```

The `cmd` command is submitted to the nCore API. The transaction listener thread will call EventSet `ev`, if `ev` is non-NULL when the reply returns for this command. The reply is unmarshalled into `reply` and `tctx` is returned to the caller with `SEElib_Query(M_Reply **replyp,`

`SEElib_ContextHandle *tctx_r)`.

Before using the `SEElib_Submit()` method, `SEElib_StartTransactListener()` must have been called to start the transaction listener.

Unlike `SEElib_SubmitCoreJob()`, `SEElib_Submit()` does not block and wait for all other calls to `SEElib_Transact()` to complete.

7.4.4.3. SEElib_SubmitCoreJob / SEElib_GetCoreJobEx()

To submit a job to the nCore API:

```
extern int SEElib_SubmitCoreJob(const unsigned char *data, unsigned int len)
```

To receive a job from the nCore API:

```
extern int SEElib_GetCoreJobEx(unsigned char *buf, M_Word *len_io, unsigned flags)
```

`SEElib_SubmitCoreJob()` is blocking. It waits for the job to be submitted, which includes waiting for existing calls made to `SEElib_Transact()` to be completed. The same is true for `SEElib_GetCoreJobEx()`.

For non-blocking calls, consider using `SEElib_Submit()`.

7.4.4.4. Other SEElib methods

For a comprehensive list of all functionality provided via the SEElib, see: SEE API documentation.

7.4.5. Host/SEE machine communication

Host/SEE machine communication can be done using SEEJobs via the hardserver using an automatically configured SSH tunnel set up by using the `[codesafe]` configuration section of the client-side `config` file. Alternatively (or additionally) communication between the host and the SEE machine can be done via TCP and UDP IPv6 networking.

The `ncoreapi` service can only connect to one CodeSafe container at a time.

7.4.5.1. Update Connects running in an IPv4 context

The host side of the CodeSafe 5 examples will only be able to communicate over IPv6. Connects running in an IPv4 context will not be able to run examples without changing how CodeSafe 5 is configured on the Connect. See Working with CodeSafe for more information.

8. Automatic Configuration of CodeSafe 5 Applications

8.1. CodeSafe 5 Configuration Section

The configuration section `[codesafe]` is supported in the client-side hardserver `config` file which allows specification of:

- the `esn` of the nShield 5s or nShield 5c module onto which to load the application
- the CodeSafe 5 `image_file` to load and run; this must be readable by the `nfast` user, which is a member of `nfast` group and `nfastadmin` group by default (the `/opt/nfast/custom-seemachines` (Linux) or `C:\Program Files\nCipher\nfast\custom-seemachines` (Windows) is a recommended location)
- the `worldid_pubname` to assign the published object that the SEE world ID is registered with

Run `cfg-mkdefault -f config.example` to create a new example config file showing this section with documentation of all its fields.

You can manually add it to your existing config file before the `[load_seemachine]` section if it is not already present.

CodeSafe 5 configuration section documentation:

```
[codesafe]
# Start of the codesafe section
# The CodeSafe 5 applications to load and start on an nShield 5. Use
# [load_seemachine] for previous HSM types.
# Each entry has the following fields:
#
# ESN of the module to load the CodeSafe application onto
#  esn=ESN
#
# Whether this configuration entry is enabled (default="yes"). This is a
# convenience for disabling CodeSafe auto-loading on a given module
# temporarily without removing the configuration.
#  enabled=ENUM
#
# The filename of the CodeSafe application for this module to host.
#  image_file=STRING
#
# Port in the CodeSafe application that is configured for SEEJobs
# communication. An SSH tunnel is automatically created for SEEJobs
# communication between the hardserver and this port. Set to 0 to explicitly
# disable. (default=8888)
#  seejobs_port=PORT
#
# Enable ncoreapi access for the CodeSafe application with
# Cmd_CreateSEEConnection. The identities in the image_file are passed
# automatically. (default="yes")
#  enable_ncoreapi=ENUM
```

```
#
# The PublishedObject name to use for publishing the KeyID of the started
# CodeSafe application.
#  worldid_pubname=STRING
#
# Enable auto-enrollment of the client with an nShield 5c. Auto-enrollment is
# enabled by default. This setting is ignored if the HSM is an nShield 5s or
# if this machine is an unprivileged client of a 5c.
#  auto_enroll=ENUM
#
# Unless set to "no", logging will be enabled for the CodeSafe application
# (default="yes"). Logs can be retrieved with csadmin log get -u UUID
#  logging=ENUM
#
# If set to "yes", the CodeSafe application is forcibly re-loaded even if a
# matching image appears to be loaded already (default="no")
#  force_reload=ENUM
#
# Key names of identities of CodeSafe application to pass to
# Cmd_CreateSEEConnection. This is only applicable if enable_ncoreapi has not
# been disabled. If only signed with the ASK, this is passed, if signed with
# the ASK and one extra signature (csadmin image signextra) then the default
# is to pass only the extra identity, and if there are more signers than this
# then all the identities are supplied. To override the defaults (e.g. to
# supply both the ASK and signextra identies) specify the key names as a
# space-separated list. Note, this controls what is reported by
# Cmd_GetWorldSigners instead of the CodeSafe application.
#  identities=STRING
```

Entries for additional modules may be added separated by `----` lines.

A new or modified configuration or CodeSafe app can be applied to a module dynamically by running `nopclearfail -c -m MODULE_NO`. The new configuration or changed application will be loaded automatically after the module returns from the clear operation.

8.2. Automatic Startup

The configuration-helper program `hsc_codesafe` is run automatically by the hardserver during module startup (or after clear) to process the `[codesafe]` configuration, including the following steps:

- Automatically configures CodeSafe 5 on the 5c and enrolls it with the local system (requires 13.7 or later Connect image, and will be skipped if the client is not privileged, in which case it will have to be configured up-front manually as per nShield 5c Codesafe 5 Configuration to allow access to the unprivileged client).
- Stops all existing CodeSafe applications on the module.
- Destroys any existing CodeSafe applications on the module if they do not match the hash of the one specified in configuration.
- Loads any required developer ID certificates from the `/opt/nfast/kmdata/cscerts` (Linux) or `C:\ProgramData\nCipher\Key Management Data\cscerts` (Windows) if not already loaded.

- Loads the CodeSafe application specified in config (if not already loaded).
- Enables CodeSafe logging in the application configuration.
- Enables the SSHD in the application configuration, gets the SSHD server public key, generates an ephemeral client key and registers it with the SSHD server (This is done for SEEJobs support unless `seejobs_port` is explicitly disabled).
- Starts the CodeSafe application.
- Runs the internal nShield `ncssh` tool to set up a mutually authenticated tunnel between a local UNIX domain socket that is accessible to the `hardserver` and the port (by default `8888`) for the SEEJobs protocol in the CodeSafe application.
- Runs `Cmd_CreateSEEConnection` with the identity information for the CodeSafe application (unless `enable_ncoreapi` is disabled).
- If `worldid_pubname` is set, runs `Cmd_SetPublishedObject` to register the SEE World ID returned by `Cmd_CreateSEEConection` so that applications can make use of the CodeSafe application (this ID should be used with calls to `Cmd_SEEJob` / `Cmd_FastSEEJob`).

`hsc_codesafe` will run until the module is cleared or the `hardserver` is shut down.

If the module is cleared, `hsc_codesafe` will be re-run automatically when the module returns from clear to negotiate a fresh mutually authenticated SSH tunnel and start the CodeSafe application again.

It takes a couple of minutes for the CodeSafe application to be available after the module is cleared. Loading performance will be improved in a future firmware release.

8.3. Interactive `hsc_codesafe` execution

In order to support build, run, debug cycles when the user is developing the CodeSafe 5 application, `hsc_codesafe` can also be run interactively without configuration in `config` and without needing to clear the module.

It can be passed the same name-value pairs of config entries on the command-line as would normally be read from the `config` file.

For example, to load a CSEE (SEEJobs) CodeSafe application, and then when it is ready, run a host-side command which uses the application:

```
hsc_codesafe -m1 image_file=/opt/nfast/custom-seemachines/echo.cs5 worldid_pubname=echosee -- perfcheck
"misc:SEEJob" -s -t 10
```

To load a non-SEEJobs application pass `seejobs_port=0`, and to prevent `Cmd_CreateSEECon-`

nection being run automatically pass enable_ncoreapi=no. The example below shows this, and just runs csadmin list to fetch the container address when loading has completed instead of running an application directly. In this example, the module to use is specified via ESN rather than via module number (either option is supported regardless of other parameters passed).

```
hsc_codesafe esn=8ED1-2C9A-9331 image_file=/opt/nfast/custom-seemachines/seetickets_netsee.cs5 seejobs_port=0
enable_ncoreapi=no -- csadmin list
```

8.4. Monitoring Connection Status

The log messages from hsc_codesafe are currently included in the hardserver log /opt/nfast/log/hardserver.log (or Event Log on Windows, can be retrieved on the command-line using nshieldeventlog -c 10 for the last 10 lines, for example).

A "pseudo-module" is registered in the hardserver for the CodeSafe application SEE Jobs endpoint and is visible when running enquiry -m 1001 as the counterpart for module 1, enquiry -m 1002 as the counterpart for module 2, and so on. This will return InvalidModule if the hsc_codesafe processing has not run to completion and is only visible when the SSH tunnel is set up and Cmd_CreateSEEConnection has been run successfully to trigger the enrollment of the CodeSafe application for SEEJobs via the hardserver. The version details reported by the pseudo-module reflect the version of the CodeSafe SDK whose SEELib library code the CodeSafe application is linked against (not the HSM firmware version).

CodeSafe 5 pseudo-module enquiry example:

```
$ enquiry -m1001
Module #1001:
 enquiry reply flags  none
 enquiry reply level  Five
 serial number
 mode                 operational
 version              13.7.2
 speed index          20000
 rec. queue           120..250
 level one flags      none
 version string       13.7.2-85-90e580d1
 checked in           0000000067ca1f41 Thu Mar  6 22:18:41 2025
 level two flags      none
 max. write size      262152
 level three flags    none
 level four flags     ServerHasPollCmds HasLongJobs ServerHasLongJobs ServerHasCreateClient
 module type code     17
 module type          nShield CodeSafe 5 on nShield 5
 product name         CodeSafe 5
 device name          #1001 CodeSafe 5 SEE Jobs Endpoint for 8ED1-2C9A-9331
 hardware status      OK
```

Apart from running enquiry for the status reporting of SEEJobs communication, the

pseudo-module should not currently be used for any other client operations.

The information reported by this "pseudo-module" is from the SEE machine itself, and reflects the version number of the CodeSafe SDK that the `seelib.a` it is linked against is from. This is distinct from the the HSM firmware version and status information that is reported in the `enquiry` output for the "normal" module like module 1.

8.5. Troubleshooting

You can monitor the progress in loading the CodeSafe application with `tail -f /opt/nfast/log/hardserver.log` (or monitoring the Event Log on Windows)

Output from CodeSafe 5 loading will appear as log lines containing `Module #1 Startup` (or whichever module number), so `grep` can be used to filter to just these messages.

When the CodeSafe application has been loaded successfully a log line like the below will be reported, meaning that the application is now available for client communication (and the message provides some example commands for further diagnostics information):

```
2025-03-13 12:33:24 t00e774076f7f0000: Hardserver [FP]: Notice: Module #1 Startup: hsc_codesafe:INFO: READY:
/opt/nfast/custom-seemachines/echo.cs5 running on 8ED1-2C9A-9331 (#1); SEE World ID published as echosee; csadmin
log get -u da5c30a8-a302-4dff-bda6-d424d5749273 to retrieve logs; enquiry -m 1001 to check SEEJobs endpoint
status
```

If loading fails, retry can be attempted by clearing the module (that is the main module, not the pseudo-module in the 1001+ range) or by restarting the client `hardserver` (`nFast Server` service on Windows).

9. nShield 5c Codesafe 5 Configuration

To use CodeSafe 5 with nShield 5c, launcher service keys must be exchanged between the client machine and the nShield 5c. These keys are essential for secure communication and access to the launcher service on the module.

If the automatic configuration and loading is used from a privileged client of the nShield 5c, this key exchange is done automatically with no extra user intervention required, provided that the nShield 5c Connect image is at least v13.7.

See CodeSafe setup for the nShield 5c for more information about manual setup where required (for unprivileged clients, where not using the automatic configuration, or when using an older Connect image)

10. Build and sign example SEE machines on Linux

10.1. Build module-side C examples

1. Create an empty directory to build the module side examples into, for example:

```
mkdir ~/buildmodule/
```

2. Navigate to the empty directory:

```
cd ~/buildmodule/
```

3. Build the module side examples with `cmake` using the following commands:

```
cmake -DCMAKE_TOOLCHAIN_FILE=/opt/nfast/c/csd5/cmake/codesafe-toolchain-nshield5-csee.cmake
/opt/nfast/c/csd5/examples/

cmake --build .
```

Successful builds create `.cs5` images for each example. For example, the classic SEE `Hello` example has a `.cs5` image at `~/buildmodule/n5/csee/hello/module/hello.cs5`.

10.2. Building Host Side C Examples

1. Create an empty directory to build the host-side clients for the SEE machines, for example:

```
mkdir ~/buildhost/
```

2. Navigate to the directory where the host-side examples will be built:

```
cd ~/buildhost/
```

3. Build the host-side examples with `cmake` using the following commands:

```
cmake /opt/nfast/c/csd5/examples/

cmake --build .
```

Successful builds create executable host-side clients for each example. For example, the

classic SEE `Hello` example has an executable program at `~/build-host/n5/csee/hello/host/hello`.

10.3. Build CS5 Images for Python Examples

1. Create an empty directory to build the Python examples into, for example:

```
mkdir ~/build_python
```

2. Navigate to the empty directory:

```
cd ~/build_python/
```

3. Build the examples with `cmake` using the following commands:

```
cmake /opt/nfast/python3/csd5/examples

cmake --build .
```

Successful builds create `.cs5` images and executable host-side clients for each example. For example, the `hello_tcp` example has a `.cs5` image at `~/build_python/n5/netsee/helloworld_tcp/module/helloworld_mod_tcp.cs5` and the executable program is located at `~/build_python/n5/netsee/helloworld_tcp/hostside/helloworld_host_tcp.py`.

10.4. Sign CodeSafe Images

1. Use `csadmin ids create` to generate the developer ID key, if it does not already exist, as well as the CSR file in a single step. If the key already exists, it only generates the CSR.

```
csadmin ids create --keyname developerid --x509cname yourcompany.com --x509org "yourorganization"
--x509orgunit "CodeSafe App Development"

Generate key 'developerid' ...

Loading 'TestOCS':
 Module 1: 0 cards of 1 read
 Module 1 slot 0: empty
Card reading complete.

OK
Generate a CSR in 'developerid.csr' ...
OK
Created CSR file 'developerid.csr'. Please send it to Entrust Support
```

This creates the CSR file in the location where the command was

> run. This developer ID creation was done with `TestOCS`, quorum of 1/1. Exact output might vary slightly with different OCS quorums.

2. Send the CSR to customer support to be signed by Entrust. You must obtain the signed developer ID certificate in order to sign and load an application.

> For more detailed information on Developer IDs and CSRs, see Create and sign CodeSafe 5 applications with csadmin.

3. Use `nfast generatekey` to generate a simple ECDSA NIST521P application signing key (ASK). The following example specifies the key to be protected by the module. However, end users are encouraged to protect the key with an OCS.

```
/opt/nfast/bin/generatekey --batch --module=1 simple type=ECDSA curve=NISTP521 ident=ask plainname=ask
protect=module
```

4. Sign the CodeSafe image, for example:

```
csadmin image sign --askeyname ask --devkeyname developerid --devcert ~/ca/developerid.pem --out
/tmp/hello-signed.cs5 ~/ca/hello.cs5
```

   Additional examples are provided later in this chapter.

5. Where applicable, sign the CodeSafe image with non-ASK keys, for example:

```
csadmin image signextra --appname seeinteg --key seeintkeyname --out /tmp/hello-signed-extra.cs5
/tmp/hello-signed.cs5
```

6. Use `csadmin ids add` to install the developer ID certificate chain from Entrust.

   You can use `csadmin ids list` to view the loaded certificate.

> The developer ID certificate you receive back from Entrust Support will be a chain of certificates. The first certificate in the chain will be the issuance certificate for Entrust and you will see thus see `Entrust` in the `Organization` field. Your certificate will be the second certificate in the chain with your organization name in the `Organization` field. If you load subsequent developer ID certificates the Entrust issuance certificate will only appear once in the list.

```
$ csadmin ids add entrust_developerid.pem
FEDC-BA09-8765        SUCCESS
$ csadmin ids list
FEDC-BA09-8765        SUCCESS
Certificates:
```

{'serialNumber': '705640309799691626348045512084088225642474535983', 'subject': 'Common Name: CodeSafe Development, Organizational Unit: nCipher, Organization: Entrust, Country: GB', 'keyid': '406386a0f3de84618ce0ee4022f67aa86768e6a1', 'authKeyid': 'd81e9f34b89c76b5c20cddd489b6f2d8d66cafd5', 'notBefore': '2023-03-03 09:25:25+00:00', 'notAfter': '2033-03-13 09:25:25+00:00'}
{'serialNumber': '116622002464887670342028053697306624797', 'subject': 'Common Name: yourcompany.com, Organization: yourorganization, Organizational Unit: CodeSafe App Development', 'keyid': 'b6687020bb527f56ef426d174eee61ac16baa16d', 'authKeyid': '406386a0f3de84618ce0ee4022f67aa86768e6a1', 'notBefore': '2025-10-30 16:28:00+00:00', 'notAfter': '2028-10-30 16:27:59+00:00'}

10.5. Run NetSEE examples

NetSEE examples communicate between the client and SEE machine directly via TCP or UDP IPv6 networking to the container, unlike legacy applications, such as for Solo XC or Solo+, which required using an emulation layer on top of SEEJobs to support networking.

10.5.1. helloworld_tcp

To execute the helloworld TCP example that opens a socket within the container and uses the connection to transact a "helloworld" message:

1. Sign the `.cs5` image using `devcert` and `askeys`:

```
csadmin image sign --askeyname ask --devkeyname developerid --devcert ~/ca/developerid.pem --out
~/buildmodule/n5/netsee/helloworld_tcp/module/helloworld_mod_tcp-signed.cs5
~/buildmodule/n5/netsee/helloworld_tcp/module/helloworld_mod_tcp.cs5
```

2. Load the signed `.cs5` image using `csadmin load`:

```
sudo /opt/nfast/bin/csadmin load ~/buildmodule/n5/netsee/helloworld_tcp/module/helloworld_mod_tcp-
signed.cs5
```

The output of `csadmin load` contains the UUID of the loaded container. This UUID will be required for starting the container. The UUID can always be retrieved from the output of `csadmin list`.

3. Start the container using `csadmin start`:

```
sudo /opt/nfast/bin/csadmin start --uuid fedcba09-8765-4321-1234-567890abcdef
```

`csadmin list` lists the UUIDs of all containers. The IPv6 address of the started container appears in the output of the `csadmin start` command. It can also be found in the output of `csadmin list` and `csadmin stats`.

4. Run the host-side application.

   The host-side application takes three positional arguments, the IPv6 address of the con tainer, the port number, and the message to send to the container. The port number used by this example is 8000 by default. The message can be any string of valid charac ters.

   ```
   ~/buildhost/n5/netsee/helloworld_tcp/hostside/helloworld_host_tcp ffff::fff:ffff:ffff:ffff%nshield0 8000
   hello_module
   ```

   Expected output:

   ```
   nseeContainerMachineIP=ffff::fff:ffff:ffff:ffff%nshield0
   nseeContainerMachinePort=8000
   mesg=hello_module
   Successful Connection to Socket...
   Host>Sending TCP Message-->hello_module
   Host>Hello World From HSM!
   ```

   The IPv6 address is link-local and requires the zone index to be appended (typically `%nshield0`).

10.5.1.1. helloworld_tcp for nShield 5c

The process is the same as the 5s example, but the host-side application command will dif-fer. Instead of IPv6, you can use the Connect's IPv4 address:

The examples for the nShield 5c work similarly to the 5s module, but the IP addresses and ports refer to the 5c Connect network. Similarly, for the TCP example, you can use the Con-nect's IPv4 address:

```
~/buildhost/n5/netsee/helloworld_tcp/hostside/helloworld_host_tcp 192.168.1.100 8000 hello_module
```

Example output:

```
nseeContainerMachineIP=192.168.1.100
nseeContainerMachinePort=8000
mesg=hello_module
Successful Connection to Socket...
Host>Sending TCP Message-->hello_module
Host>Hello World From HSM!
```

10.5.2. helloworld_udp

To execute the helloworld UDP example that opens a socket within the container and uses

the connection to transact a "helloworld" message:

1. Sign the `.cs5` image using `devcert` and `askeys`:

```
csadmin image sign --askeyname ask --devkeyname developerid --devcert ~/ca/developerid.pem --out
~/buildmodule/n5/netsee/helloworld_udp/module/helloworld_mod_udp-signed.cs5
~/buildmodule/n5/netsee/helloworld_udp/module/helloworld_mod_udp.cs5
```

2. Load the signed container using `csadmin load`:

```
sudo /opt/nfast/bin/csadmin load ~/buildmodule/n5/netsee/helloworld_udp/module/helloworld_mod_udp-
signed.cs5
```

   Example output:

```
FEDC-BA09-8765: Uploading ~/buildmodule/n5/netsee/helloworld_udp/module/helloworld_mod_udp-signed.cs5
FEDC-BA09-8765: creating machine
FEDC-BA09-8765        SUCCESS
UUID: fedcba09-8765-4321-1234-567890abcdef
```

   The output of `csadmin load` contains the UUID of the loaded container. This UUID will be required for starting the container. The UUID can always be retrieved from the output of `csadmin list`.

3. Start the container using `csadmin start`:

```
sudo /opt/nfast/bin/csadmin start --uuid fedcba09-8765-4321-1234-567890abcdef
```

   Example output:

```
FEDC-BA09-8765        SUCCESS
IP ADDRESS: ffff::fff:ffff:ffff:ffff
```

   `csadmin list` will list the UUIDs of all containers. The IPv6 address of the started container appears in the output of the `csadmin start` command. It can also be found in the output of `csadmin list` and `csadmin stats`.

4. Run the host-side application.

   The host-side application takes three positional arguments, the IPv6 address of the container, the port number, and the message to send to the container. The port number used by this example is 8000 by default. The message can be any string of valid characters.

```
~/buildhost/n5/netsee/helloworld_udp/hostside/helloworld_host_udp ffff::fff:ffff:ffff:ffff%nshield0 8000
hello_module
```

Example output:

```
nseeContainerMachineIP=ffff::fff:ffff:ffff:ffff%nshield0
nseeContainerMachinePort=8000
mesg=hello_module
Successful Connection to Socket...
Host>Sending UDP Message-->hello_module
Host>Hello World From HSM!
```

The IPv6 address is link-local and requires the zone index to be appended (typically `%nshield0`).

10.5.2.1. helloworld_udp for 5c

The process is the same as the 5s example, but the host-side application command will differ. Instead of IPv6, you can use the Connect's IPv4 address:

The examples for the nShield 5c work similarly to the 5s module, but the IP addresses and ports refer to the 5c Connect network.

```
~/buildhost/n5/netsee/helloworld_udp/hostside/helloworld_host_udp 192.168.1.100 8000 hello_module
```

Example output:

```
nseeContainerMachineIP=192.168.1.100
nseeContainerMachinePort=8000
mesg=hello_module
Successful Connection to Socket...
Host>Sending UDP Message-->hello_module
Host>Hello World From HSM!
```

10.6. Run NetSEE examples via SSH tunnel

`NetSEE` examples communicate between the client and SEE machine directly through a TCP/IPv6 network connection to the container, unlike legacy applications, such as for Solo XC or Solo+, which communicate through the hardserver to the nCore API.

On the nShield 5c network, the `SSHD listening address` may be an `IPv4` address instead of `IPv6`. Adjustments to the steps below may be needed to accommodate this.

10.6.1. helloworld_tcp via SSH Tunnel

To execute the helloworld TCP example via an SSH Tunnel that opens a socket within the container and uses the connection to transact a "helloworld" message:

1. Create an SSHD key for the hello example:

```
mkdir ~/examplekeys/
ssh-keygen -t ecdsa -f ~/examplekeys/helloworld_tcp_ecdsa_key
```

2. Modify the `network-conf.json` of the `helloworld_tcp` example to support SSH tunneling, for example:

```
cat ~/buildmodule/n5/netsee/helloworld_tcp/module/network-conf.json
{
        "incoming": {
                "tcp":
                {
                        "protos": ["ipv6"],
                        "ports": []
                }
        },
        "outgoing" : {
                "tcp" :
                {
                       "protos": ["ipv6"],
                       "ports": []
                }
        },
        "ssh_tunnel" : {
                 "container_port" : 8000
        }
}
```

+ When the container server app accepts a client connection on the specified incoming port (for example `8000`), it designates and responds to the client on an ephemeral port in the range `[32768-60999]` as the outgoing port. This port does not have to be defined in the `network-conf.json`. Only one port is supported for the `ssh_tunnel` / `container_port` in this version. The `ssh_tunnel` port must be the only port specified for communication that is restricted to be made over SSH. To only send communication in the plain, the port should instead be specified in the `incoming ports` list.

1. Rebuild the `.cs5` image with the updated `network-conf.json` so the loaded container will allow SSH tunneling:

```
sudo /opt/nfast/bin/csadmin image generate --package-name "helloworld_tcp" --entry-point
/usr/bin/entrypoint --network-conf ~/buildmodule/n5/netsee/helloworld_tcp/module/network-conf.json
--packages-conf ~/buildmodule/n5/netsee/helloworld_tcp/module/extra-packages-conf.json --version-str 1.0
--rootdir ~/buildmodule/n5/netsee/helloworld_tcp/module/container/
~/buildmodule/n5/netsee/helloworld_tcp/module/helloworld_mod_tcp.cs5
```

Most paths used in generating the new image are paths to the file locations on the host that is building the image However, the `--entry-point` path is the absolute path to the `entrypoint` file within the container and should be `/usr/bin/entrypoint`, not `~/build-module/n5/netsee/helloworld_tcp/module/container/usr/bin/entrypoint`.

2. Sign the new `.cs5` image using `devcert` and `askeys`:

```
sudo /opt/nfast/bin/csadmin image sign --askeyname ask --devkeyname developerid --devcert
~/ca/developerid_cert.pem --out ~/buildmodule/n5/netsee/helloworld_tcp/module/helloworld_mod_tcp-signed.cs5
~/buildmodule/n5/netsee/helloworld_tcp/module/helloworld_mod_tcp.cs5
```

3. Load the signed container using `csadmin load`:

```
sudo /opt/nfast/bin/csadmin load ~/buildmodule/n5/netsee/helloworld_tcp/module/helloworld_mod_tcp-
signed.cs5
```

The output of `csadmin load` contains the UUID of the loaded container. This UUID will be required for starting the container and managing the SSHD keys of the container. The UUID can always be retrieved from the output of `csadmin list`.

4. Load the public key created earlier (`helloworld_tcp_ecdsa_key`) to the container using `csadmin sshd setclient`:

```
sudo /opt/nfast/bin/csadmin sshd keys setclient --uuid fedcba09-8765-4321-1234-567890abcdef --keyfile
~/examplekeys/helloworld_tcp_ecdsa_key.pub
```

5. Enable SSH tunneling on the container:

```
sudo /opt/nfast/bin/csadmin sshd state enable --uuid fedcba09-8765-4321-1234-567890abcdef
```

Example output:

```
FEDC-BA09-8765        SUCCESS
SSHD PORT: 6789
LISTENING ADDRESS: aaaa::aa:aaaa:aaaa:aaaa
```

The output of `sshd state enable` contains the SSHD Port number and the listening address of the container SSHD.

6. Start the container using `csadmin start`:

```
sudo /opt/nfast/bin/csadmin start --uuid fedcba09-8765-4321-1234-567890abcdef
```

`csadmin list` lists the UUIDs of all containers. The IPv6 address of the started con-

tainer appears in the output of the `csadmin start` command. It can also be found in the output of `csadmin list` and `csadmin stats`.

7. Setup the SSH tunnel on the host:

   Run `csadmin sshd state get` and collect the following information:

   - Container tunnel address (`ffff::fff:ffff:ffff:ffff`)
   - Container port (`8000`)
   - SSHD port (`6789`)
   - SSHD listening address (`aaaa::aa:aaaa:aaaa:aaaa`)

   On nShield Connect the `SSHD listening address` may be an `IPv4` or `IPv6` address

   Next, choose a local IP address and port number through which to access the tunnel. Typically localhost is chosen as the local IP address (`127.0.0.1` or `[::1]`)

   Check the HSM's SSHD public key by running `csadmin sshd keys getserver -u UUID` for comparison against the output from `ssh` on first use, or this could be added to your `known_hosts` file directly.

   The SSH tunnel command is formatted as follows:

   ```
   ssh -i ~/examplekeys/helloworld_tcp_ecdsa_key -L LOCAL_IP:LOCAL_PORT:[TUNNEL_ADDRESS%lxcbr0]:CONTAINER_PORT
   -f -N -p SSHD_PORT launcher@LISTENING_ADDRESS
   ```

   Using the example data:

   ```
   ssh -i ~/examplekeys/helloworld_tcp_ecdsa_key -L [::1]:8000:[ffff::fff:ffff:ffff:ffff%lxcbr0]:8000 -f -N -p
   6789 launcher@aaaa::aa:aaaa:aaaa:aaaa%nshield0
   ```

   For nShield 5s HSMs, the IPv6 address is link-local and requires the zone index to be appended (typically `%nshield0`). If you are working with a 5c network, replace the IPv6 address with the appropriate nShield 5c network address (IPv4 or IPv6) for your configuration.

8. Run the host-side application.

   The host-side application takes three positional arguments, the IPv6 address set up in the forwarding step `[::1]`, the port number, and the message to send to the container. The port number used by this example is 8000 by default. The message can be any string of valid characters.

```
~/buildhost/n5/netsee/helloworld_tcp/hostside/helloworld_host_tcp ::1 8000 hello_module
```

Expected Output:

```
nseeContainerMachineIP=::1
nseeContainerMachinePort=8000
mesg=hello_module
Successful Connection to Socket...
Host>Sending TCP Message-->hello_module
Host>Hello World From HSM!
```

10.7. Run CSEE examples via automatic configuration

The Classic SEE (CSEE) examples use the SEElib library's SEEJobs functionality for communication with the host (using port `8888` in the container). These examples are identical to examples provided with previous iterations of nShield HSMs and CodeSafe. CSEE applications must be loaded using the `[codesafe]` section of the client-side `hardserver config` file, or by running the `hsc_codesafe` tool directly. Using this configuration support simplifies the deployment of CSEE applications and automates the `csadmin` operations (except for signing) and securely setting up the SSH tunnel.

10.7.1. hello CSEE via automatic configuration

This section describes executing the `hello` CSEE example. The hello example operates functionally identically to the CSEE hello example for Solo XC and Solo+.

The hello example sends a string from the host to the module. The module converts the string to uppercase and returns the string to the host.

1. Generate an input file containing a character string to be sent to the module.

   ```
   echo UPPERCASElowercase > ~/inputfile
   ```

   This input file has both uppercase and lowercase characters.

2. In all CSEE examples, the `network-conf.json` permits only secure communication via the SSH tunnel (and no plaintext communication via `incoming` is allowed by specifying an empty `ports` list), i.e. the `network-conf.json` in the SDK matches the below:

   ```
   cat ~/buildmodule/n5/csee/hello/module/network-conf.json
   {
          "incoming": {
                 "tcp":
                 {
   ```

```
                        "protos": ["ipv6"],
                        "ports": []
                }
        },
        "outgoing" : {
                "tcp" :
                {
                       "protos": ["ipv6"],
                       "ports": []
                }
        },
        "ssh_tunnel" : {
                 "container_port" : 8888
        }
}
```

3. Sign the `.cs5` image using `devcert` and `askeys`:

```
sudo /opt/nfast/bin/csadmin image sign --askeyname ask --devkeyname developerid --devcert
~/ca/developerid.pem --out /opt/nfast/custom-seemachines/hello-signed.cs5
~/buildmodule/n5/csee/hello/module/hello.cs5
```

4. Load the signed container using the `[codesafe]` of the client-side `config` file (replacing the `esn` field contents with the actual ESN of the module on which to load the application) and then clearing the module e.g. with `nopclearfail -c -m1` (replacing `1` with the module number in question):

```
[codesafe]
esn=5CB5-41F2-F235
image_file=/opt/nfast/custom-seemachines/hello-signed.cs5
worldid_pubname=hellosee
```

5. Run the host-side application.

   The host-side application takes one required positional argument for the input file containing a string to convert to uppercase on the module. Module number is assumed to be `1` by default, use `-m2` to specify module 2 etc. The published object name of the SEE World ID is assumed to be `hellosee` by default (as in the `config` example above); if a dif ferent name was used in the `[codesafe]` section of the `config` file, use `-p` parameter to specify it.

   ```
   ~/buildhost/n5/csee/hello/hostside/hello ~/inputfile
   ```

   Example output:

   ```
   Worldid: 0x1234abcd
   UPPERCASELOWERCASE
   ```

   The module has received the input string `UPPERCASElowercase` and has converted and

returned it as a fully uppercase string `UPPERCASELOWERCASE`.

10.7.2. tickets CSEE via automatic configuration

This section describes executing the `tickets` CSEE example. The tickets example operates functionally identically to the tickets example for Solo XC and Solo+. The tickets example serves to demonstrate cryptographic functionality by encrypting and having the module decrypt a user-provided string.

1. Generate a simple RSA key to encrypt with:

```
sudo /opt/nfast/bin/generatekey --module=1 simple type=RSA pubexp=3 ident=encryptionkeytickets
plainname=encryptionkeytickets protect=module nvram=no size=2048
```

2. Sign the `.cs5` image using `devcert` and `askeys`:

```
sudo /opt/nfast/bin/csadmin image sign --askeyname ask --devkeyname developerid --devcert
~/ca/developerid.pem --out /opt/nfast/custom-seemachines/seetickets-signed.cs5
~/buildmodule/n5/csee/tickets/module/seetickets.cs5
```

3. Load the signed container using the `[codesafe]` of the client-side `config` file (replacing the `esn` field contents with the actual ESN of the module on which to load the application) and then clearing the module e.g. with `nopclearfail -c -m1` (replacing `1` with the module number in question):

```
[codesafe]
esn=5CB5-41F2-F235
image_file=/opt/nfast/custom-seemachines/seetickets-signed.cs5
worldid_pubname=ticketsee
```

4. Run the host-side application.

   The host-side application accepts the encryption key created earlier as an optional argument (`--key`). Module number is assumed to be `1` by default, use `-m2` to specify module 2 etc. The published object name of the SEE World ID is assumed to be `ticketsee` by default (as in the `config` example above); if a different name was used in the `[codesafe]` section of the `config` file, use `-p` parameter to specify it.

```
~/buildhost/n5/csee/tickets/hostside/hosttickets --key simple,encryptionkeytickets
```

5. When prompted, enter a string to encrypt (for example, `testencryption`) and press **Return**:

```
Enter string to be encrypted (256 characters maximum): testencryption
```

The host encrypts the message then the module decrypts it and returns it in plain text format.

Example output:

```
HostSide> Loading security world key (simple,encryptionkeytickets)
HostSide> Creating World: init status was 0 (OK)
HostSide> Sending ticket for private RSA key to module
HostSide> Generating AES session key and creating blob under public RSA key
HostSide> Sending key blob to module
HostSide> Sending cipher-text to module
HostSide> decrypted cipher text received from SEE machine:
"testencryption"
HostSide> Thank you for watching. The end.
```

10.7.3. benchmark CSEE via automatic configuration

This section describes executing the `benchmark` CSEE example. The benchmark example operates functionally identically to the benchmark example for Solo XC and Solo+. The benchmark example will transact asynchronously with the module running multiple threads processing transactions. The benchmark example will output transactions/second data every second.

1. Generate a simple key for signing a ticket in the bm-machine on the module:

```
sudo /opt/nfast/bin/generatekey --module=1 simple type=RSA pubexp=3 ident=signingkeybenchmark
plainname=signingkeybenchmark protect=module nvram=no size=2048
```

2. Sign the `.cs5` image using `devcert` and `askeys`:

```
sudo /opt/nfast/bin/csadmin image sign --askeyname ask --devkeyname developerid --devcert
~/ca/developerid.pem --out /opt/nfast/custom-seemachines/bm-machine-signed.cs5
~/buildmodule/n5/csee/benchmark/module/bm-machine.cs5
```

3. Load the signed container using the `[codesafe]` of the client-side `config` file (replacing the `esn` field contents with the actual ESN of the module on which to load the application) and then clearing the module e.g. with `nopclearfail -c -m1` (replacing `1` with the module number in question):

```
[codesafe]
esn=5CB5-41F2-F235
image_file=/opt/nfast/custom-seemachines/bm-machine-signed.cs5
worldid_pubname=bmsee
```

4. Run the host-side application.

   The host-side application takes two positional arguments: the appname and the key name of the signing key created earlier. Module number is assumed to be `1` by default, use `-m2` to specify module 2 etc. The published object name of the SEE World ID is assumed to be `bmsee` by default (as in the `config` example above); if a different name was used in the `[codesafe]` section of the `config` file, use `-p` parameter to specify it.

   ```
   ~/buildhost/n5/csee/benchmark/hostside/bm-test simple signingkeybenchmark
   ```

   Example output:

   ```
   Worldid: 0x1234abcd
   1 759 759.00
   2 1522 761.00
   3 2361 787.00
   4 3324 831.00
   5 4238 847.60
   6 5124 854.00
   7 5948 849.71
   8 6723 840.38
   9 7579 842.11
   10 8408 840.80
   ```

11. Build and sign example SEE machines on Windows

11.1. Prerequisites

- Visual Studio 2022 buildtools
- CMAKE version 3.9 or newer
- Ninja build system latest version
- Visual Studio 2022 workload-vctools

11.2. Building Windows CodeSafe C, CSEE, and NETSEE examples

1. Start the Developer Command Prompt for VS 2022 as Administrator from the **Start** menu.
2. Navigate to the following directory:

```
cd "c:\Program Files (x86)\Microsoft Visual Studio\2022\BuildTools\Common7\Tools"
```

3. Install the MSVC C and C++ compiler `cl.exe`.
4. Execute `VsDevCmd.bat`:

```
VsDevCmd.bat
```

5. Run cl:

```
cl
```

6. Because the default is 32bit mode, the version displayed will show x86. Change to 64bit `cl` Compiler:

```
cd "c:\Program Files (x86)\Microsoft Visual Studio\2022\BuildTools\VC\Auxiliary\Build"
```

7. Execute `vcvars64.bat`:

```
vcvars64.bat
```

8. Run `cl` and verify that the x64 version is displayed:

```
cl
```

you can build the following examples in the same VS2022 Command window:

11.2.1. Host-side examples

```
c:\>mkdir examples\host

c:\>cd c:\examples\host\

c:\examples\host>cmake -G Ninja -DCMAKE_C_COMPILER=cl -DCMAKE_CXX_COMPILER=cl "c:\Program
Files\nCipher\nfast\c\csd5\examples"

c:\examples\host>ninja
```

11.2.2. Module-side examples

```
c:\>mkdir examples\module

c:\>cd c:\examples\module\

c:\examples\module>cmake -G "Ninja" -DCMAKE_TOOLCHAIN_FILE="c:\Program Files\nCipher\nfast\c\csd5\cmake\codesafe-
toolchain-nshield5-csee.cmake" "c:\Program Files\nCipher\nfast\c\csd5\examples"

c:\examples\module>ninja
```

11.3. CS5 images for Python examples

Build the following images in the VS2022 Command window configured in Building Windows CodeSafe C, CSEE, and NETSEE examples. You do not need to build host-side and module-side Python examples separately. They are both built into `examples\python\n5\netsee\<example>\`.

```
c:\>mkdir examples\python

c:\>cd c:\examples\python\

c:\examples\python>cmake -G "Ninja" "c:\Program Files\nCipher\nfast\python3\csd5\examples"

c:\examples\python>ninja
```

For example:

```
c:\examples\python\n5\netsee\tickets>dir
Volume in drive C is OS
```

```
Volume Serial Number is 582A-CFB6 Directory of c:\examples\python\n5\netsee\tickets 03/21/2023  12:32 PM    <DIR>
.
03/21/2023  12:32 PM    <DIR>          ..
03/21/2023  12:32 PM    <DIR>          hostside
03/21/2023  12:32 PM    <DIR>          module
               0 File(s)              0 bytes
               4 Dir(s)  906,165,829,632 bytes free
```

11.4. Sign CodeSafe images

Signing CodeSafe Images requires a Security World and Operator Card Set (OCS).

1. Insert the OCS card.
2. Create a certificate signing request (CSR) that should be sent to Entrust to be signed:

```
c:\ca_ids\>csadmin ids create --keyname developerid --x509cname yourcompany.com --x509org yourorganization
--x509orgunit "CodeSafe App development"
Generate key 'developerid' ...

Loading 'TestOCS':
 Module 1: 0 cards of 1 read
 Module 1 slot 0: empty
Card reading complete.

OK
Generate a CSR in 'developerid.csr' ...
OK
Created CSR file 'developerid.csr'. Please send it to Entrust Support
```

The developer ID creation in this example was done with `TestOCS`, quorum of 1/1. Exact output may vary slightly with different OCS quorums.

3. Send the resulting CSR to customer support to be signed by Entrust. You must obtain the signed developer ID certificate in order to sign and load an application.

   For more detailed information on Developer IDs and CSRs, see Create and sign CodeSafe 5 applications with csadmin.

4. Create the ASK on the HSM (the name of the key in this example is `test-ask`). The following example specifies the key to be protected by the module. However, end users are encouraged to protect the key with an OCS:

```
c:\ca_ids>C:\Progra~1\nCipher\nfast\bin\generatekey.exe --module=1 simple type=ECDSA curve=NISTP521
ident=test-ask plainname=test-ask
protect: Protected by? (token, module) [token] > module
nvram: Blob in NVRAM (needs ACS)? (yes/no) [no] >
key generation parameters:
 operation    Operation to perform       generate
```

```
 application  Application               simple
 protect      Protected by              module
 verify       Verify security of key    yes
 type         Key type                  ECDSA
 ident        Key identifier            test-ask
 plainname    Key name                  test-ask
 nvram        Blob in NVRAM (needs ACS) no
 curve        Elliptic curve            NISTP521
Key successfully generated.
Path to key: C:\ProgramData\nCipher\Key Management Data\local\key_simple_test-ask
```

5. Confirm that the keys were created in the previous step:

```
c:\ca_ids>nfkminfo -k
Key list - 2 keys
 AppName simple               Ident test-ask
 AppName simple               Ident developerid
```

6. Sign the `netsee\tickets` example. You need the signed `cert.pem` from customer support for this step and the OCS card must be inserted for signing.

```
c:\examples\module\n5\netsee\tickets_netsee\module>csadmin image sign --askeyname test-ask --devkeyname
developerid --devcert c:\ca_ids\developerid.pem --out seetickets_netsee-signed-with-hsm.cs5
seetickets_netsee.cs5
INFO: Reading CS5 file contents...
INFO: Getting key handle from HSM...

INFO: Signing the Application Signing Key...
INFO: hashing contents using 'SHA512Hash'
INFO: Obtaining public key data from HSM...
INFO: Storing public key data on CS5 file...
INFO: Getting key handle from HSM...
INFO: Requesting signature from HSM...
INFO: Saving CS5 file to disk...
INFO: file 'seetickets_netsee.cs5' was signed successfully!

Directory of c:\examples\module\n5\netsee\tickets_netsee\module

02/16/2023  03:53 PM        27,167,860 seetickets_netsee-signed-with-hsm.cs5
               1 File(s)     27,167,860 bytes
               0 Dir(s)  775,613,321,216 bytes free
```

7. Where applicable, sign the CodeSafe image with non-ASK keys, for example:

```
c:\examples\module\n5\netsee\tickets_netsee\module> csadmin image signextra --appname seeinteg --key
seeintkeyname --out seetickets_netsee-signed-with-hsm-extra.cs5 seetickets_netsee-signed-with-hsm.cs5
INFO: file 'seetickets_netsee-signed-with-hsm.cs5' was signed successfully!
```

8. Install the developer ID certificate chain from Entrust using `csadmin ids add` You can use `csadmin ids list` to view the loaded certificate.

The developer ID certificate you receive back from Entrust Support will be a chain of certificates. The first certificate in the chain will be the issuance certificate for Entrust and you will see thus see

> `Entrust` in the `Organization` field. Your certificate will be the second certificate in the chain with your organization name in the `Organization` field. If you load subsequent developer ID certificates the Entrust issuance certificate will only appear once in the list.

```
csadmin ids add entrust_developerid.pem
FEDC-BA09-8765        SUCCESS

csadmin ids list
FEDC-BA09-8765        SUCCESS
Certificates:
{'serialNumber': '705640309799691626348045512084088225642474535983', 'subject': 'Common Name: CodeSafe
Development, Organizational Unit: nCipher, Organization: Entrust, Country: GB', 'keyid':
'406386a0f3de84618ce0ee4022f67aa86768e6a1', 'authKeyid': 'd81e9f34b89c76b5c20cddd489b6f2d8d66cafd5',
'notBefore': '2023-03-03 09:25:25+00:00', 'notAfter': '2033-03-13 09:25:25+00:00'}
{'serialNumber': '116622002464887670342028053697306624797', 'subject': 'Common Name: yourcompany.com,
Organization: yourorganization, Organizational Unit: CodeSafe App Development', 'keyid':
'b6687020bb527f56ef426d174eee61ac16baa16d', 'authKeyid': '406386a0f3de84618ce0ee4022f67aa86768e6a1',
'notBefore': '2025-10-30 16:28:00+00:00', 'notAfter': '2028-10-30 16:27:59+00:00'}
```

9. Execute `netsee\tickets`:

```
c:\examples\module\n5\netsee\tickets_netsee\module>csadmin load seetickets_netsee-signed-with-hsm.cs5
FEDC-BA09-8765: Uploading seetickets_netsee-signed-with-hsm.cs5
FEDC-BA09-8765: creating machine
FEDC-BA09-8765        SUCCESS
UUID: fedcba09-8765-4321-1234-567890abcdef

c:\examples\module\n5\netsee\tickets_netsee\module>cd c:\examples\host\n5\netsee\tickets_netsee\hostside

c:\examples\host\n5\netsee\tickets_netsee\hostside>nopclearfail -aO
Module 1, command ClearUnitEx: OK

c:\examples\host\n5\netsee\tickets_netsee\hostside>csadmin start -u fedcba09-8765-4321-1234-567890abcdef
FEDC-BA09-8765        SUCCESS
IP ADDRESS: ffff::fff:ffff:ffff:ffff

c:\examples\host\n5\netsee\tickets_netsee\hostside>csadmin list
FEDC-BA09-8765
UUID                                  State     Name               IP Address
------------------------------------------------------------------------------------------
fedcba09-8765-4321-1234-567890abcdef    RUNNING   seetickets_netsee   ffff::fff:ffff:ffff:ffff


c:\examples\host\n5\netsee\tickets_netsee\hostside>hosttickets_netsee.exe -p 8000 -U fedcba09-8765-4321-
1234-567890abcdef -i ffff::fff:ffff:ffff:ffff%10 -c
c:\examples\module\n5\netsee\tickets_netsee\module\seetickets_netsee-signed-with-hsm.cs5
WSAStartup() Success.
HostSide>Enter string to be encrypted (8 characters maximum): hello
HostSide>Reading Identities from container
HostSide>Generating RSA keypair
HostSide>Creating World: init status was 0 (OK)
HostSide>Sending ticket for private RSA key to module
HostSide>Sending key blob to module
HostSide>Sending cipher-text to module
HostSide>decrypted cipher text received from SEE machine:
"hello"
HostSide>Thank you for watching. The end.
```

12. Build and run Java examples

The following Java examples are included:

- BenchMark5
- Echo5
- HelloWorld5
- HostTickets5

12.1. Prerequisites

The following versions of Java have been tested to work with, and are supported by, your nShield Security World Software:

- Java8 (or Java 1.8x)
- Java11
- Java17
- Java21

Ensure that Java is installed before you install the Security World software. The Java executable must be on your system path.

If you can do so, please use the latest Java version currently supported by Entrust that is compatible with your requirements. Java versions before those shown are no longer supported.

12.2. The Java interface

The Java host-side examples work with the same SEE machines as the C host-side examples, with build and signing instructions in Build and sign example SEE machines on Linux and Build and sign example SEE machines on Windows.

Java applications using SEEJobs to communicate with the SEE machine can use the `PublishedSEEWorld` class to talk to the CSEE application that has been auto-loaded via the `[codesafe]` section in the `config` file or run directly using `hsc_codesafe`. Existing Java applications that used `PublishedSEEWorld` class to send SEEJobs to previous HSM models can work unchanged with CodeSafe 5 as well.

Applications that wish to create the SEE World ID directly from the client application (rather than leaving it to the automatic configuration code to register it as a published

object) are recommended to use the `SEEWorld5` class to simplify that initialization. Existing applications that used `SEEWorld` to do this with previous HSM models should now use `SEEWorld5` instead.

12.3. Build the examples

The Java example files are in the `nCipherKM-SEE-Examples.jar` in `opt/nfast/java/examples` (**Linux**) or `%NFAST_HOME%\java\examples` (**Windows**).

Extract and compile the examples:

Linux

```
cd /opt/nfast/java/examples
jar xf nCipherKM-SEE-Examples.jar
javac -cp /opt/nfast/java/classes/nCipherKM.jar com/ncipher/see/hostside/*.java
javac -cp .:/opt/nfast/java/classes/nCipherKM.jar
com/ncipher/see/hostside/examples/benchmark5/BenchMark5.java
javac -cp .:/opt/nfast/java/classes/nCipherKM.jar com/ncipher/see/hostside/examples/echo5/Echo5.java
javac -cp .:/opt/nfast/java/classes/nCipherKM.jar
com/ncipher/see/hostside/examples/helloworld5/HelloWorld5.java
javac -cp .:/opt/nfast/java/classes/nCipherKM.jar
com/ncipher/see/hostside/examples/hosttickets5/HostTickets5.java
```

Windows

```
cd %NFAST_HOME%\java\examples
jar xf nCipherKM-SEE-Examples.jar
javac -cp "%NFAST_HOME%\java\classes\nCipherKM.jar com\ncipher\see\hostside\*.java"
javac -cp "%NFAST_HOME%\java\classes\nCipherKM.jar ^
com\ncipher\see\hostside\examples\benchmark5\BenchMark5.java"
javac -cp "%NFAST_HOME%\java\classes\nCipherKM.jar ^ com\ncipher\see\hostside\examples\echo5\Echo5.java"
javac -cp "%NFAST_HOME%\java\classes\nCipherKM.jar ^
com\ncipher\see\hostside\examples\helloworld5\HelloWorld5.java"
javac -cp "%NFAST_HOME%\java\classes\nCipherKM.jar ^
com\ncipher\see\hostside\examples\hosttickets5\HostTickets5.java"
```

12.4. Run the examples

All examples can be run with the following help options:

- `-m`, `--module=MODULE`: Use module MODULE.
  Default: 1
- `-n`, `--pubname=PUBNAME`: Use the specified published object name (should match `world-did_pubname` in `config`) if different from the default.
- `-h`, `--help`: Displays the help message.
- `-v`, `--version`: Displays the version number of this program.

- `-u`, `--usage`: Displays a brief usage summary.

Before running the examples, ensure that you are in the `examples` directory:

Linux

```
cd /opt/nfast/java/examples
```

Windows

```
cd %NFAST_HOME%\java\examples
```

12.4.1. BenchMark5

BenchMark5 is a simple demonstration of a Java hostside app for benchmarking end-to-end communication and crypto processing time.

Linux

```
java -cp .:/opt/nfast/java/classes/nCipherKM.jar com.ncipher.see.hostside.examples.benchmark5.BenchMark5
[options] <key-app>
```

Windows

```
java -cp "%NFAST_HOME%\java\classes\nCipherKM.jar ^
com\ncipher\see\hostside\examples\benchmark5\BenchMark5 [options] <key-app>"
```

Options (in addition to common options):

- `-s`, `--slot=SLOT`: Use slot SLOT for operator cards.
  Default: 0
- `-t`, `--threads=THREADS`: Use THREADS threads.
  Default: 32
- `-i`, `--iterations=ITERATIONS`: Each thread will perform ITERATIONS iterations.
  Default: 100
- `-l`, `--logfile=LOGFILE`: Record public key and timestamps in file LOGFILE.

Arguments:

- `key-app`: The key to be used to encrypt the data. This must be an RSA key, for example `simple rsa2k`. See generatekey for more information.

For example (this assumes that the SEE machine was loaded with the default `bmsee` as the published object name):

```
$ java -cp .:/opt/nfast/java/classes/nCipherKM.jar com.ncipher.see.hostside.examples.benchmark5.BenchMark5 -i 10
simple rsa2k
Generating 320 Timestamps Using 32 threads
Sending ticket...
Threads started...
Finished!
```

12.4.2. Echo5

Echo5 is a simple demonstration of a Java hostside app for performance testing.

Linux

```
java -cp .:/opt/nfast/java/classes/nCipherKM.jar com.ncipher.see.hostside.examples.echo5.Echo5 [options]
```

Windows

```
java -cp "%NFAST_HOME%\java\classes\nCipherKM.jar com\ncipher\see\hostside\examples\echo5\Echo5 [options]
```

Options (in addition to common options):

- `-t`, `--threads=THREADS`: Use THREADS threads.
  Default: 32
- `-p`, `--payload=PAYLOAD`: Send PAYLOAD bytes.
  Default: 32
- `-i`, `--iterations=ITERATIONS`: Each thread will perform ITERATIONS iterations.
  Default: 100
- `-e`, `--verify`: Verify that the returned value matches the value sent.

For example (this assumes that the SEE machine was loaded with the default `echosee` as the published object name):

```
$ java -cp .:/opt/nfast/java/classes/nCipherKM.jar com.ncipher.see.hostside.examples.echo5.Echo5 -i 10
Threads = 32 Payload = 32 bytes Iterations = 10 Verify replies = OFF
320 jobs in 0.04 seconds = 9006.18 jobs/Second,  288198 bytes/second
```

12.4.3. HelloWorld5

HelloWorld5 is a simple demonstration of a Java hostside app talking to a C SEE machine. It takes the text from an `<inputFile>` and outputs it with all the lower-case text converted to upper-case text.

The `HelloWorld5.java` example is not intended for use as the basis for real world applications.

Linux

```
java -cp .:/opt/nfast/java/classes/nCipherKM.jar com.ncipher.see.hostside.examples.helloworld5.HelloWorld5
[options] <inputFile>
```

Windows

```
java -cp "%NFAST_HOME%\java\classes\nCipherKM.jar ^
com\ncipher\see\hostside\examples\helloworld5\HelloWorld5 [options] <inputFile>"
```

For example (this assumes that the SEE machine was loaded with the default `hellosee` as the published object name):

```
$ cat /tmp/testfile.txt
lowercase
$ java -cp .:/opt/nfast/java/classes/nCipherKM.jar com.ncipher.see.hostside.examples.helloworld5.HelloWorld5
/tmp/testfile.txt
LOWERCASE
```

12.4.4. HostTickets5

HostTickets5 is a simple demonstration of a Java hostside app using tickets.

Linux

```
java -cp .:/opt/nfast/java/classes/nCipherKM.jar com.ncipher.see.hostside.examples.hosttickets5.HostTickets5
[options]
```

Windows

```
java -cp "%NFAST_HOME%\java\classes\nCipherKM.jar ^
com\ncipher\see\hostside\examples\hosttickets5\HostTickets5 [options]"
```

Options (in addition to common options):

- `-s`, `--string=STRING`: String to be encrypted.
  If you do not use this option, you will be prompted for a string when you execute the command.

For example (this assumes that the SEE machine was loaded with the default `ticketsee` as the published object name):

```
$ java -cp .:/opt/nfast/java/classes/nCipherKM.jar com.ncipher.see.hostside.examples.hosttickets5.HostTickets5
-s encryptme
String to be encrypted = encryptme
Sending ticket...
Sending blobbed key...
Sending encrypted text...
Decrypted text = encryptme
```

13. Debugging CodeSafe 5 SEE machines

13.1. SEE machine logging

SEE machine logs capture any stdout and stderr from the container. This can be used to capture log messages from the IPC daemon or SEElib library, or from the SEE machine's own logging framework.

If automatic configuration is used via `[codesafe]` section in `config` or via the `hsc_codesafe`, logging in the SEE machine is enabled by default (unless explicitly disabled), in which case retrieving and clearing the logs are the only applicable commands below.

The following SEE logging-related commands are supported by the `csadmin` utility.

13.1.1. config log set enabled

The `config log set enabled` command should be issued before the `start` command. It uses the following format:

```
/opt/nfast/bin/csadmin config set log enabled -u <SEE-machine-UUID> --esn <host-ESN>
```

- `<SEE-machine-UUID>` is the UUID of the SEE machine created by the load command.
- `<host-ESN>` is the ESN of the HSM hosting the SEE Machine.

For example:

```
/opt/nfast/bin/csadmin config set log enabled -u fedcba09-8765-4321-1234-567890abcdef --esn FEDC-BA09-8765
```

When successful, the command returns with no error.

13.1.2. config log set disabled

The `config log set disabled` command should be issued while the SEE machine is not run ning. It uses the following format:

```
/opt/nfast/bin/csadmin config set log disabled -u <SEE-machine-UUID> --esn <host-ESN>
```

- `<SEE-machine-UUID>` is the UUID of the SEE machine created by the load command.
- `<host-ESN>` is the ESN of the HSM hosting the SEE Machine.

For example:

```
/opt/nfast/bin/csadmin config set log disabled -u fedcba09-8765-4321-1234-567890abcdef --esn FEDC-BA09-8765
```

When successful, the command returns with no error.

13.1.3. log get

The `get` command returns the current SEE log contents, if any. It uses the following format:

```
/opt/nfast/bin/csadmin log get -u <SEE-machine-UUID>
```

`<SEE-machine-UUID>` is the UUID of the SEE machine created by the load command.

For example:

```
/opt/nfast/bin/csadmin log get -u fedcba09-8765-4321-1234-567890abcdef
FEDC-BA09-8765        SUCCESS
Success: Started ipcdaemon
```

13.1.4. log clear

The `clear` command deletes the current SEE log file if present. It uses the following format:

```
/opt/nfast/bin/csadmin log clear -u <SEE-machine-UUID>
```

`<SEE-machine-UUID>` is the UUID of the SEE machine created by the load command.

For example:

```
/opt/nfast/bin/csadmin log clear -u fedcba09-8765-4321-1234-567890abcdef
FEDC-BA09-8765        SUCCESS
log: log cleared
```

13.2. Crash Reporter

If the CodeSafe 5 application links against the `seelib.a` library, an in-process crash reporter will be registered for various termination and crash signals before `main()`.

If the application receives one of the handled signals, a crash report will be written to the SEE log, like the below:

```
SEE MACHINE CRASH: SIGSEGV (Invalid memory reference)
        Sending process PID: 0
        Sending process UID: 0
        Fault address: 0xcafecafecafecafe
        SEGV_MAPERR: Address not mapped to object.
  crashing.c: 298 --  -- some_crashing_function
  main.c: 56 --  -- main
Segmentation fault
```

In order to obtain a useful crash backtrace, the SEE machine should be built with debugging symbols at least at `-g1` level and not have the symbols stripped from the binary.

14. Uninstall the CodeSafe 5 SDK

Do not uninstall Security World or CodeSafe 5 software unless you are certain it is no longer required or you are going to upgrade it.
If you are using CodeSafe 5 with an nShield 5s HSM, you must back up its `sshadmin` keys by running `hsmadmin keys backup` before you uninstall Security World or CodeSafe 5.

The uninstaller only removes files that were created during the installation. To remove key data or Security World data, navigate to the installation directory and delete the files in the *%NFAST_KMDATA%* folder.

If you intend to remove your Security World before uninstalling the Security World Software, Entrust recommends that you erase the OCS before you erase the Security World or uninstall the Security World Software. Except where Remote Administration cards are used, after you have erased a Security World, you can no longer erase any cards that belonged to it.

1. Log in to the host computer as Administrator or as a user with local administrator rights.
2. Run the following command to erase the OCS:

   ```
   createocs -m# -s0 --erase
   ```

   Where # is the module number.

3. Uninstall the Security World and CodeSafe software:

   - **Linux**:

     Run the following command:

     ```
     /opt/nfast/sbin/install -u
     ```

   - **Windows**:

     1. Navigate to the Windows Control Panel, and select **Programs and Features**.
     2. Select the Security World Software entry, then select **Uninstall** to remove the software.

If required, you can safely remove the nShield module after shutting down all connected hardware.

15. Porting existing CodeSafe applications to CodeSafe 5

Follow the steps in this chapter if you need to port an existing SEE machine to run on Code Safe 5.

This chapter assumes the perspective of a CodeSafe application developer.

CodeSafe users using third-party CodeSafe applications for HSM models prior to nShield 5 should contact the developer of those applications to obtain a CodeSafe 5 version of the application. Ensure that the third-party CodeSafe developer is a trusted party, and can provide a signed CodeSafe 5 application and associated developer ID certificate issued by Entrust.

Examples of Classic SEE "CSEE" machines that have been ported to CodeSafe 5 can be found in Build and sign example SEE machines on Linux. These are examples from previous HSM models that have been modified to run with CodeSafe 5. In all other ways, these examples are identical to examples provided with previous iterations of nShield HSMs and CodeSafe.

It is assumed that an ASK and developer ID key have already been generated, and that required certificates have already been obtained from Entrust and installed into the target HSM.

15.1. Communication with the host

Legacy CodeSafe transacted data between host application and module SEE machines by sending SEEJobs via the harderver to the HSM's nCore API service, onto the CodeSafe application, and back again. TCP communication with the host was supported with a networking emulation layer on top of this protocol.

CodeSafe 5 continues to support SEEJobs via the hardserver, but this protocol is now implemented via a securely negotiated SSH tunnel automatically set up between the client hardserver and the CodeSafe application directly without the nCore API service being required to forward the jobs.

Additionally, the TCP network emulation layer (provided by `see-sock-serv` and related configuration support for BSDSEE/GLIBSEE applications) on top of the SEEJobs protocol has been removed and replaced with a first-class networking implementation which supports TCP and UDP between the host and the CodeSafe application.

The module-side SEE machine must be rebuilt and re-signed to function

> on CodeSafe 5 (CodeSafe applications from previous HSM models cannot run on nShield 5). The host-side client application communicating with the SEE machine may or may not require updates depending on how it interacts with the SEE machine, but details for loading and running the CodeSafe application from the client differ from previous HSM models and will need to be changed.

15.2. SEElib library

The module-side SEElib library behaves the same as before for both communication with the nCore API service on the HSM to execute `M_Command` commands and also for communication with the host via SEEJobs. Assuming the use of the default port of `8888` for SEEJobs, no source changes should be required, though it will be necessary to include a `network-conf.json` as per the CSEE examples to permit port `8888` as the `ssh_tunnel` port (and not to allow `incoming` plaintext access to this port).

Host-side applications continue to send SEEJobs (i.e. `Cmd_SEEJob` and `Cmd_FastSEEJob` commands) via an nCore connection to the hardserver, using the SEE World ID as the identifier for the SEE machine.

15.3. Deployment differences

See Automatic Configuration of CodeSafe 5 Applications for information about automatic loading of CodeSafe 5 applications using the `[codesafe]` section of the `config` file or using the `hsc_codesafe` tool. The `[codesafe]` section replaces the `[load_seemachine]` section used by previous HSM models.

> The "CodeSafe Direct" feature of loading a SEE machine from the nShield Connect `config` file is not supported by nShield 5, but the new `[codesafe]` section can be used to automatically load a SEE machine from a client hardserver `config` file instead. If using this with nShield 5, one client of the nShield 5c should have that `[codesafe]` configuration as only one CodeSafe application may be run at a time.

See the examples in Build and sign example SEE machines on Linux for information about running both TCP/UDP applications and CSEE applications with CodeSafe 5.

The NetSEE examples documentation explains how to configure networking; this replaces the network emulation that legacy `see-sock-serv` and related configuration provided.

The CSEE examples documentation show how to deploy using published objects for the

SEE World ID with the SEE machine automatically loaded via the `config` file, which is the simplest way to deploy, and was also a supported approach for interacting with SEE machines in legacy CodeSafe. Applications that were using published objects will require the fewest updates.

If the SEE World ID was previously loaded directly by the host-side application using `Cmd_CreateSEEWorld` within a hardserver connection, that should now be replaced with `Cmd_CreateSEEConnection`. If it was loaded using the `seeworld_auth_setup()` or `seeworld_setup()` helper functions from the `seeworld.h` / `seeworld.c` example utility code, `seeworld_connect()` should now be used instead. If using the `SEEWorld` class to load the SEE World ID in a Java application, `SEEWorld5` should now be used instead.

A SEE World ID is required if communicating with the SEE machine via SEEJobs as it is the handle for those jobs, but it may also be required even when not using SEEJobs if the SEE machine talks (locally) to the nCore API service on the HSM (for example to transact cryptographic `M_Command` commands such as signing or decryption) as the act of doing the `Cmd_CreateSEEConnection` from the host creates the permission for the CodeSafe application to use the nCore API service.

15.4. User Data

CodeSafe 5 does not retain the concept of `UserData` as a separate cpio or SAR file input alongside the CodeSafe application.

A developer can instead include any files, using any directory structure, in the container image that is installed in the HSM.

16. CodeSafe 5 FAQ

16.1. Signing keys

See also: CodeSafe 5 Application Authentication

16.1.1. What signing keys do I need for CodeSafe 5 application development?

Development requires the generation of the following keys:

- A *Developer ID* key, which identifies your company or organization.
- One *Application Signing Key* (ASK) for each application you intend to develop.

These keys should be kept in a Security World on an nShield HSM.

16.1.2. Can I use my existing CodeSafe signing key?

An existing `seeinteg` key can be used as an Application Signing Key if it is an ECDSA key using the NIST P521 curve. If not, a new ASK must be generated.

However, the `csadmin image signextra` command allows an existing `seeinteg` key to add additional signatures to a CodeSafe 5 application, which will permit that application to use existing working keys which are bound to that `seeinteg` key.

We strongly recommend that the Developer ID key be a newly-created Security World key, that has not been used to create signatures for other purposes. This must also be an ECDSA key using the NIST P521 curve.

16.1.3. Can I use separate Security Worlds?

Yes. The Developer ID Key and the ASK do not need to share a Security World, and these do not need to be the same world deployed on the HSM which is running the application itself. All the signatures and the information required to verify them is contained within the *.cs5* application file itself.

See Offline signing using the Developer ID key for more details.

16.2. Developer ID keys and Certificates

16.2.1. What is a Developer ID certificate, and how do I get one?

A Developer ID certificate is issued by Entrust nShield Technical Support to certify a Developer ID key as belonging to a particular organization.

The `csadmin ids create` command can create a new Developer ID key within a Security World, then generate a CSR (Certificate Signing Request) file. You should then contact Technical Support and upload the CSR file.

Entrust will then send a signed Developer ID certificate (another short text file), which is needed during application development and deployment. This does not contain secret data; it identifies your organization but note we do not normally accept CSRs which include Personally Identifiable Information (PII).

16.2.2. Do Developer ID certificates need to be renewed?

Yes. Developer ID certificates issued by Entrust have a validity period of 3 years from date of issue, and will need to be renewed before this period expires.

This expiry period is to provide protection for the whole CodeSafe 5 ecosystem in the event that a Developer ID key is misused, and to allow organizations to exercise control over the lifespan of their keys and CodeSafe 5 applications.

16.2.3. What happens when a Developer ID certificate expires?

The Developer ID certificate's validity is checked by the HSM when a CodeSafe 5 machine is loaded (using `csadmin create`), and when it is started (using `csadmin start`, or if autostart is enabled). It is not checked at other times, so:

- A CodeSafe 5 machine that is running will continue to run; it is not forcibly stopped when the certificate expires.
- The machine itself is not deleted, and any data in its nonvolatile storage is preserved.
- Once stopped (either manually or through an HSM reboot), it cannot be restarted until a replacement certificate has been uploaded.

If the Developer ID key itself is not changed, there is no need to re-sign the CodeSafe machine image or re-load its *.cs5* file. The replacement certificate will identify the same signing key and developer identity, just with a later expiry date.

Loading an updated Developer ID certificate is done via the `csadmin ids add` command. This can be done at any time (including while the machine is running). The HSM allows multi

ple certificates for the same Developer ID key in its database, and will accept whichever cer tificate which is currently valid.

The expected usage is that, at any time before the expiry of the current Developer ID certifi cate, you run `csadmin ids create` to create a new CSR for the existing key, and obtain a new certificate from Entrust Support. This can be uploaded at a convenient time, and when this is done the old certificate can be removed. There is no system downtime required to complete this process.

16.2.4. Why might I not renew a Developer ID certificate?

You may wish to allow a Developer ID certificate to expire if:

- You suspect that the Developer ID key could have been misused, for instance to sign an ASK which is not under your control.
- You have improved the security (e.g. process or physical security) around handling of such keys, and wish to use a new key which uses these arrangements.
- You wish to ensure old versions of your CodeSafe 5 applications can no longer be run.

17. SEE API documentation

SEElib is an API that enables an SEE machine to both execute nCore API commands and to accept messages from the host machine via the SEEJobs protocol. It is supported both in CodeSafe 5 and in CodeSafe for previous HSM models.

The SEElib API is provided as a library seelib.a that can be found in the rootfs after install. Its install location is /opt/nfast/c/csd5/lib-ppc64-linux-musl/seelib.a on Linux.

17.1. SEElib functions

17.1.1. SEElib_init

```
extern void SEElib_init(void);
```

This function initializes the SEElib library.

This function does not return on error.

17.1.2. SEElib_ReadUserData

```
extern int SEElib_ReadUserData ( M_Word offset, unsigned char *buf, M_Word len );
```

This function reads selected bytes from the UserData block, starting at offset bytes in and continuing for len bytes. It returns an M_Status value.

As the concept of a separate UserData input does not exist in CodeSafe 5, for backwards compatibility this function is supported as reading a file located inside the container (/etc/codesafe.userdata) which must be added at that path when the container image is constructed.

17.1.3. SEElib_ReleaseUserData

```
extern void SEElib_ReleaseUserData(void);
```

In CodeSafe 5 this function does not do anything. It is only present for backwards compatibilty.

17.1.4. SEElib_InitComplete

```
extern void SEElib_InitComplete( M_Word status );
```

In CodeSafe 5 this function does not do anything. It is only present for backwards compatibility.

17.1.5. SEElib_StartTransactListener

```
extern void SEElib_StartTransactListener(void);
```

This function starts the thread that listens for `SEElib_Transact` calls and dispatches them. This function must be called before any use is made of `SEElib_Transact`.

17.1.6. SEElib_Transact

```
extern int SEElib_Transact(struct M_Command *cmd, struct M_Reply *buf);
```

This function marshals a command, submits it, waits for the response, and unmarshals it into a reply structure.

17.1.7. SEElib_MarshalSendCommand

```
extern int SEElib_MarshalSendCommand(M_Command *cmd);
```

This function marshals a command and places it on the input queue for processing by the nShield core.

The command takes a reference to an `M_Command` structure, as described in the *nCore CodeSafe API Documentation*.

The SEE machine can submit any of the nCore API commands listed in the *Basic commands* and *Key-Management commands* sections of the *nCore CodeSafe API Documentation* except:

- `RetryFailedModule`
- `GetWhichModule`
- `MergeKeyIDs`.

If the SEE machine attempts to submit one of these commands, the nShield core returns a response with the status code `NotAvailable`.

The `SEElib_MarshalSendCommand` function returns an `M_Status` value. This value is `OK` if the command was marshalled and transferred to the nShield core correctly.

Do not mix calls to `SEE_Transact()` and `SEElib_MarshalSendCommand()` and `SEElib_GetUnmarshalResponse()`, because the replies may be misdirected.

17.1.8. SEElib_GetUnmarshalResponse

```
extern int SEElib_GetUnmarshalResponse(M_Reply *buf);
```

If there is a reply in the input queue for this SEE world, this function returns the first job in the queue. Otherwise, it blocks and waits for the nShield core to return a job.

On return, `M_Reply` contains the unmarshalled reply.

The `SEElib_GetUnmarshalResponse` function returns an `M_Status` value. This value is `OK` if the reply was unmarshalled successfully. The return of this value does not necessarily mean that the command was completed successfully, only that the reply was unmarshalled. You must also check the `M_Status` within the reply.

17.1.9. SEElib_FreeCommand

```
extern int SEElib_FreeCommand(struct M_Command *cmd);
```

This function frees a command structure and is equivalent to the generic stub function `NFastApp_FreeCommand` (described in the *nCore CodeSafe API Documentation*).

17.1.10. SEElib_FreeReply

```
extern int SEElib_FreeReply(struct M_Reply *reply);
```

This function frees a reply structure and is equivalent to the generic stub function `NFastApp_FreeReply` (described in the *nCore CodeSafe API Documentation*).

17.1.11. SEElib_SetPort

```
extern void SEElib_SetPort(unsigned short port);
```

CodeSafe 5 for nShield 5 only: Sets a custom port to bind to for receiving SEEJobs from the host.

By default, port `8888` is used, and that is also the default port used by client-side configuration support. If using the default port, there is no need to call this function. For a custom port to take effect, this must be called before `SEElib_init()`.

To disable listening for SEEJobs altogether, call `SEElib_SetPort(0)` before `SEElib_init()`.

17.1.12. SEElib_SubmitCoreJob

```
extern int SEElib_SubmitCoreJob( const unsigned char *data, unsigned int len );
```

This function puts a job on the input queue for processing by the core. The byte block is passed in `data` and `len`. It should be a full marshalled `M_Command` with a valid tag at the start.

This function returns an `M_Status`, which is typically `OK` or `BufferFull` (if `len` is too big).

17.1.13. SEElib_GetCoreJob

```
extern int SEElib_GetCoreJob ( unsigned char *buf, M_Word *len_io );
```

This function blocks and waits for a job submitted to the core to be returned. On entry, `buf` points to a buffer of length `(*len_io)` `max`. On exit, if successful, `*len_io` is the length of bytes returned.

This function returns an `M_Status`, which is typically `OK` or `BufferFull` (if `len_io` is too big).

17.1.14. SEElib_GetUserDataLen

```
extern M_Word SEElib_GetUserDataLen ( void );
```

In CodeSafe 5, this function gets the length in bytes of the `/etc/userdata.codesafe` file in the filesystem of the container.

If this data has been discarded because `SEElib_ReleaseUserData()` has been called, this function returns `0`.

17.1.15. SEElib_Submit

```
extern int SEElib_Submit(M_Command *cmd, M_Reply *reply, PEVENT ev, SEElib_ContextHandle tctx);
```

This function submits the command specified in `cmd`. The transaction listener thread calls EventSet `ev`, if `ev` is non-NULL, when the reply returns for this command. The reply is unmar shalled into `reply` and `tctx` is returned to the caller in `SEElib_Query`.

Unlike `SEElib_SubmitCoreJob` this function can be called at the same time as another thread is blocking in `SEElib_Transact`.

`SEElib_StartTransactListener` must have been called before this function is called.

17.1.16. SEElib_Query

```
extern int SEElib_Query(M_Reply **replyp, SEElib_ContextHandle *tctx_r);
```

This function is called to receive a reply that is being held by the transaction listener thread. It is typically called after having been woken from `EventWait` as a result of the transaction lis tener thread posting to the event passed in to `SEElib_Submit`.

If `*replyp` is NULL, `SEElib_Query` accepts any returned reply, and `*replyp` is changed to point to that reply. If `*replyp` is not NULL, the function accepts the reply specified; other replies are queued internally.

`tctx_r` can be NULL. If it is not, the `tctx` used when submitting the reply is stored in `*tctx_r`. `SEElib_Query` can return, in addition to the usual return values, `TransactionNotYet-Complete` if the reply (or any reply if `*replyp` was NULL) has not come back from the core yet.

`SEElib_StartTransactListener` must have been called before this function is called.

17.1.17. SEElib_AwaitJob

```
extern int SEElib_AwaitJob( M_Word *tag_out, unsigned char *buf, M_Word *len_io );
```

This function blocks and waits for the next `SEEJob` to come in from the host-side applica-tion. On entry, `*buf` and `*len_io` give the base and length of a buffer area to receive the job. On return, `*len_io` is set to the length delivered (if the job is received successfully). This buffer is a copy of the `seeargs` field of the `SEEJob` that was sent by the host-side applica-tion.

The *tag_out value is the tag for this command. Each transaction must have a unique tag when sent from the host-side application to ensure transactions are returned to their required caller. The generation of unique tags is handled by the host-side compatibility layer. The tag must be returned in the SEElib_ReturnJob so that the host-side compatibility layer associates the reply with this transaction.

The SEElib_AwaitJob function returns an M_Status, which is OK on success and normally, but not always, BufferFull on failure.

If you use SEElib_StartProcessorThreads(), these function calls are done automatically and you should not call this function yourself.

17.1.18. SEElib_AwaitJobEx

```
extern void SEElib_AwaitJobEx( M_Word *tag_out, unsigned char *buf, M_Word *len_io, unsigned flags );
```

Block on the socket waiting for a SEEJob command from the host.

The output parameters are filled with information obtained from the message itself. On entry, *buf and *len_io give the base and length of a buffer area to receive the job. On return, *len_io is set to the length delivered (if the job is received successfully). This buffer is a copy of the seeargs field of the SEEJob command. The *tag_out value is the tag for this command.

17.1.19. SEElib_ReturnJob

```
extern void SEElib_ReturnJob( M_Word tag, const unsigned char *data, unsigned int len );
```

This function returns an SEEJob reply to the host-side application. It is sent in a way that the host-side compatibility layer can interpret and write into the corresponding reply struct on the host-side.

If you use the SEElib_StartProcessorThreads() function, it calls SEElib_ReturnJob() for you.

The tag field must match the tag supplied in the SEElib_AwaitJob() call that created the job.

The given data is copied away and forms the seereply field of the SEEJob reply on the host-side application.

17.1.20. SEElib_StartProcessorThreads

```
struct ProcessThreadCtx; /* User-defined */
typedef struct SEElib_ProcessContext
{
  struct ProcessThreadCtx *uc;

  unsigned char *iobuf;
  int iobuf_maxlen;
}
  SEElib_ProcessContext;

typedef struct ProcessThreadCtx * (*SEEJobInitFn) (SEElib_ProcessContext *pC);

/* Function called during thread initialisation */
typedef int (*SEEJobFn) ( SEElib_ProcessContext *pC, M_Word tag, int in_len );

/* Function to process an SEEJob; data is sent in & out via pC->iobuf.
Returns length being returned.
*/
extern int SEElib_StartProcessorThreads(int nthreads, int stacksize, SEEJobInitFn
pfnInit, SEEJobFn pfnProcess);
```

This function causes the SEE compatibility layer to start a number of processing threads. Each thread has its own `SEElib_ProcessContext` allocated, which remains constant through out the life of the thread.

A working buffer for a given thread is allocated; the `iobuf` member points to this buffer and `iobuf_maxlen` is set to the size. Data for the `SEEJob` is passed in and out through this buffer.

For each thread, the supplied `SEEJobInitFn` is called first, and the `ProcessThreadCtx` pointer it returns is stored in the `SEElib_ProcessContext` structure. This structure is typically a convenient thread-local storage. The pointer may be NULL if it is not required.

When a job arrives for the given thread, the supplied `SEEJobFn` is called. It is passed the `SEElib_ProcessContext` pointer `pC`, a tag, and a length (`in_len`). The `SEEJob` data is at `pC→iobuf`, length `in_len`. The tag is for information only. The function processes the data and leave a reply at `pC→iobuf`. The return value from the function indicates the number of bytes to be returned from this buffer.

17.1.21. SEElib_StartSEEJobListener

```
extern int SEElib_StartSEEJobListener(PEVENT ev);
```

This function starts the `SEEJob` listener thread which blocks calling `SEElib_AwaitJob`, caches the new job and then sets the event `ev` if `ev` is non-NULL.

Use `SEElib_QuerySEEJob` to receive any `SEEJobs` that have been cached by this listener thread, followed by `SEElib_ReturnJob` to reply to the `SEEJob`, then followed by `SEElib_Re-`

leaseSEEJob to free the buffer.

It is safe to call this function multiple times, however calls after the first call have no effect.

17.1.22. SEElib_QuerySEEJob

```
extern M_Status SEElib_QuerySEEJob( M_Word *tag_out, unsigned char **buf, M_Word *len );
```

This function is called to receive a SEEJob that is being held by the SEEJob listener thread. It is typically called after having been woken from EventWait as a result of the SEEJob listener thread setting the event passed in to SEElib_StartSEEJobListener.

buf is set to the buffer containing the SEEJob, len is set to the length of the data contained in buf.

This function returns TransactionNotYetComplete if there were no outstanding SEEJobs.

17.1.23. SEElib_ReleaseSEEJob

```
extern void SEElib_ReleaseSEEJob( unsigned char **buf );
```

This function is called to release a buffer which was returned from SEElib_QuerySEEJob. It must be called after the buffer specified by buf in a call to SEElib_QuerySEEJob has been fin ished with. This function is safe to call even if *buf is NULL. In addition, it sets *buf to NULL on completion.

17.2. Host-side SEEJobs

Host-side applications using SEEJobs can send them as normal nCore commands over a hardserver connection, referencing the SEE World ID of the SEE machine in question. The hardserver will automatically dispatch such commands over the mutually authenticated SSH tunnel to the SEE machine where applicable.

SEEJobs can be sent as M_Command commands with the cmd set as either Cmd_SEEJob (which does not time out, except due to communication failure) and Cmd_FastSEEJob (which times out after 2 minutes if the SEE machine does not reply to that job in that timeframe, in which case the status is set to Status_HardwareFailed, but this is not a fatal error unless the com- munication as a whole has failed).

Java clients may alternatively use the seeJob() helper method of the PublishedSEEWorld or

`SEEWorld5` classes.

18. System calls allowed by CodeSafe 5 SEE machines

SEE machines are restricted to a subset of Linux system calls they can execute.

Attempting to execute any other system call will return `-1` and set `errno` to `ENOSYS`.

Allowed system calls	
1 __NR_exit	2 __NR_fork
3 __NR_read	4 __NR_write
5 __NR_open	6 __NR_close
7 __NR_waitpid	8 __NR_creat
9 __NR_link	10 __NR_unlink
11 __NR_execve	12 __NR_chdir
13 __NR_time	15 __NR_chmod
16 __NR_lchown	19 __NR_lseek
20 __NR_getpid	24 __NR_getuid
27 __NR_alarm	29 __NR_pause
30 __NR_utime	33 __NR_access
34 __NR_nice	36 __NR_sync
37 __NR_kill	38 __NR_rename
39 __NR_mkdir	40 __NR_rmdir
41 __NR_dup	42 __NR_pipe
43 __NR_times	45 __NR_brk
47 __NR_getgid	49 __NR_geteuid
50 __NR_getegid	54 __NR_ioctl
55 __NR_fcntl	57 __NR_setpgid
60 __NR_umask	63 __NR_dup2
64 __NR_getppid	65 __NR_getpgrp
66 __NR_setsid	75 __NR_setrlimit
77 __NR_getrusage	78 __NR_gettimeofday
80 __NR_getgroups	83 __NR_symlink

Allowed system calls	
85 __NR_readlink	88 __NR_reboot
90 __NR_mmap	91 __NR_munmap
92 __NR_truncate	93 __NR_ftruncate
94 __NR_fchmod	95 __NR_fchown
96 __NR_getpriority	97 __NR_setpriority
99 __NR_statfs	100 __NR_fstatfs
102 __NR_socketcall	104 __NR_setitimer
105 __NR_getitimer	106 __NR_stat
107 __NR_lstat	108 __NR_fstat
114 __NR_wait4	117 __NR_ipc
118 __NR_fsync	120 __NR_clone
122 __NR_uname	125 __NR_mprotect
132 __NR_getpgid	133 __NR_fchdir
140 __NR__llseek	141 __NR_getdents
142 __NR__newselect	143 __NR_flock
144 __NR_msync	145 __NR_readv
146 __NR_writev	147 __NR_getsid
148 __NR_fdatasync	158 __NR_sched_yield
162 __NR_nanosleep	163 __NR_mremap
167 __NR_poll	172 __NR_rt_sigreturn
173 __NR_rt_sigaction	174 __NR_rt_sigprocmask
175 __NR_rt_sigpending	176 __NR_rt_sigtimedwait
177 __NR_rt_sigqueueinfo	178 __NR_rt_sigsuspend
179 __NR_pread64	180 __NR_pwrite64
181 __NR_chown	182 __NR_getcwd
185 __NR_sigaltstack	186 __NR_sendfile
190 __NR_ugetrlimit	202 __NR_getdents64
205 __NR_madvise	207 __NR_gettid
208 __NR_tkill	221 __NR_futex

Allowed system calls	
232 __NR_set_tid_address	234 __NR_exit_group
236 __NR_epoll_create	237 __NR_epoll_ctl
238 __NR_epoll_wait	246 __NR_clock_gettime
247 __NR_clock_getres	248 __NR_clock_nanosleep
250 __NR_tgkill	251 __NR_utimes
252 __NR_statfs64	253 __NR_fstatfs64
272 __NR_waitid	280 __NR_pselect6
281 __NR_ppoll	286 __NR_openat
287 __NR_mkdirat	289 __NR_fchownat
291 __NR_newfstatat	292 __NR_unlinkat
293 __NR_renameat	294 __NR_linkat
295 __NR_symlinkat	296 __NR_readlinkat
297 __NR_fchmodat	298 __NR_faccessat
303 __NR_epoll_pwait	304 __NR_utimensat
307 __NR_eventfd	309 __NR_fallocate
315 __NR_epoll_create1	316 __NR_dup3
317 __NR_pipe2	320 __NR_preadv
321 __NR_pwritev	322 __NR_rt_tgsigqueueinfo
325 __NR_prlimit64	326 __NR_socket
327 __NR_bind	328 __NR_connect
329 __NR_listen	330 __NR_accept
331 __NR_getsockname	332 __NR_getpeername
333 __NR_socketpair	334 __NR_send
335 __NR_sendto	336 __NR_recv
337 __NR_recvfrom	338 __NR_shutdown
339 __NR_setsockopt	340 __NR_getsockopt
341 __NR_sendmsg	342 __NR_recvmsg
343 __NR_recvmmsg	344 __NR_accept4
348 __NR_syncfs	349 __NR_sendmmsg

Allowed system calls	
357 __NR_renameat2	362 __NR_execveat
365 __NR_membarrier	380 __NR_preadv2
381 __NR_pwritev2	383 __NR_statx

The `getrandom` syscall is not supported in CodeSafe 5 and will set `ENOSYS`. Use either the `Cmd_GenerateRandom` nCore command, or `/dev/random` or `/dev/urandom` within the CodeSafe 5 application in order to obtain HSM RNG instead.