

nShield Security World

CodeSafe 5 v13.9.6 Developer Guide

03 September 2026

© Entrust Corporation. All rights reserved.

Table of Contents

1. Introduction . . . 1
    1.1. CodeSafe basic concepts . . . 1
2. New features in CodeSafe 5 . . . 2
    2.1. Management of CodeSafe 5 . . . 2
    2.2. Compatibility with CodeSafe on Solo+ and Solo XC . . . 3
3. CodeSafe 5 features . . . 4
    3.1. Applications as container images . . . 4
    3.2. Easy and fast network connectivity . . . 4
    3.3. 'Secure by default' client communication . . . 4
    3.4. Better language support . . . 5
    3.5. Developer authentication . . . 5
4. CodeSafe 5 Application Authentication . . . 6
    4.1. Introduction . . . 6
        4.1.1. CodeSafe 5 signing keys . . . 6
    4.2. Application Authentication chain of trust . . . 6
        4.2.1. Application signatures and ACLs . . . 8
        4.2.2. Handling of Developer ID certificates . . . 8
        4.2.3. Offline signing using the Developer ID key . . . 9
        4.2.4. Validation of CodeSafe 5 images . . . 9
5. Install the CodeSafe 5 SDK on Linux . . . 11
6. Install the CodeSafe 5 SDK on Windows . . . 12
    6.1. Prerequisites . . . 12
    6.2. Install the Security World Software . . . 12
    6.3. Install CodeSafe 5 . . . 12
7. Create and sign CodeSafe 5 applications with csadmin . . . 13
    7.1. Running csadmin . . . 13
    7.2. Developer Identity management . . . 13
        7.2.1. Create a Developer ID key and CSR . . . 13
        7.2.2. Upload and view Developer ID certificates . . . 15
        7.2.3. Check a Developer ID certificate . . . 15
        7.2.4. Retrieve a Developer ID certificate . . . 16
        7.2.5. Remove a Developer ID certificate . . . 16
        7.2.6. Manage Developer ID certificates on an nShield 5c appliance . . . 16
    7.3. Image creation and signing . . . 17
        7.3.1. Create an image file . . . 17
        7.3.2. Inspect a CodeSafe 5 image . . . 18
        7.3.3. Create an Application Signing Key . . . 18

7.3.4. Sign with an Application Signing Key . . . 18
7.3.5. Add extra signatures to a CodeSafe 5 image . . . 20
7.3.6. Offline signing commands . . . 21
7.4. Encrypt a CodeSafe 5 image . . . 22
7.4.1. Create an image encryption key . . . 23
7.4.2. Encrypt an image with csadmin image encrypt . . . 24
7.4.3. Sign and encrypt in a single command . . . 24
7.5. Load an encrypted CodeSafe 5 image . . . 26
7.5.1. Errors when loading an encrypted image . . . 27
8. Building CodeSafe 5 applications . . . 28
8.1. General SDK use . . . 28
8.2. Prerequisites . . . 28
8.3. SDK file structure overview . . . 28
8.3.1. SDK location . . . 28
8.3.2. Container root file system . . . 29
8.3.3. Image creation options . . . 29
8.3.4. CMake . . . 32
8.3.5. Include directories . . . 32
8.3.6. nShield-specific CodeSafe libraries . . . 33
8.3.7. Compatibility . . . 36
8.4. Building new CodeSafe applications . . . 37
8.4.1. Developer authentication . . . 37
8.4.2. Deploying CodeSafe applications . . . 37
8.4.3. CodeSafe application initialization requirements . . . 38
8.4.4. SEElib Functions . . . 38
8.4.5. Host/CodeSafe application communication . . . 40
9. nShield 5c CodeSafe 5 Configuration . . . 41
10. Auto-Load Configuration of CodeSafe 5 Applications via the nShield 5c . . . 42
10.1. Installation and Setup . . . 42
10.2. Configuring CodeSafe 5 Auto-Load . . . 43
10.2.1. SEEJobs Configuration . . . 43
10.2.2. Optimized SEEJobs Configuration . . . 43
10.2.3. Non-SEEJobs Configuration . . . 43
10.2.4. Encrypted Image Configuration . . . 44
10.2.5. Copying the Configuration and CodeSafe 5 Application to the Appliance . . . 44
10.3. Starting the CodeSafe 5 Application . . . 45
11. Automatic Configuration of CodeSafe 5 Applications via the Host Machine . . . 47
11.1. CodeSafe 5 Configuration Section . . . 47

11.2. CodeSafe 5 configuration section documentation: 47
11.3. Automatic Startup 49
11.4. Interactive `hsc_codesafe` execution 50
11.5. Monitoring Connection Status 50
11.6. Troubleshooting 51
12. Build and sign example CodeSafe applications on Linux 53
12.1. Build module-side C examples 53
12.2. Building Host Side C Examples 53
12.3. Build CS5 Images for Python Examples 54
12.4. Sign CodeSafe Images 54
12.5. Run NetSEE examples 56
12.5.1. helloworld_tcp 56
12.5.2. helloworld_udp 57
12.6. Run NetSEE examples via SSH tunnel 59
12.6.1. helloworld_tcp via SSH Tunnel 60
12.7. Run CSEE examples via automatic configuration 63
12.7.1. hello CSEE via automatic configuration 63
12.7.2. tickets CSEE via automatic configuration 65
12.7.3. benchmark CSEE via automatic configuration 66
13. Build and sign example CodeSafe applications on Windows 68
13.1. Prerequisites 68
13.2. Building Windows CodeSafe C, CSEE, and NETSEE examples 68
13.2.1. Host-side examples 69
13.2.2. Module-side examples 69
13.3. CS5 images for Python examples 69
13.4. Sign CodeSafe images 70
14. Build and run Java examples 73
14.1. Prerequisites 73
14.2. The Java interface 73
14.3. Build the examples 74
14.4. Run the examples 74
14.4.1. BenchMark5 75
14.4.2. Echo5 76
14.4.3. HelloWorld5 76
14.4.4. HostTickets5 77
15. Debugging CodeSafe 5 applications 78
15.1. CodeSafe application logging 78
15.1.1. CodeSafe IPC Daemon logging 78
15.2. nShield 5c Configured for Auto-Loading CodeSafe 5 applications 78

15.3. Host Configured for Automatic CodeSafe 5 Application Loading . . . 79
15.3.1. config log set enabled . . . 79
15.3.2. config log set disabled . . . 80
15.3.3. log get . . . 80
15.3.4. log clear . . . 80
15.4. Crash Reporter . . . 81
16. Uninstall the CodeSafe 5 SDK . . . 82
17. Porting existing CodeSafe applications to CodeSafe 5 . . . 83
17.1. Communication with the host . . . 83
17.2. SEElib library . . . 84
17.3. Deployment differences . . . 84
17.4. User Data . . . 85
18. CodeSafe 5 FAQ . . . 86
18.1. Signing keys . . . 86
18.1.1. What signing keys do I need for CodeSafe 5 application development? . . . 86
18.1.2. Can I use my existing CodeSafe signing key? . . . 86
18.1.3. Why do I see VerifyFailed messages after adding an extra signature? . . . 86
18.1.4. Can I use separate Security Worlds? . . . 87
18.2. Developer ID keys and Certificates . . . 87
18.2.1. What is a Developer ID certificate, and how do I get one? . . . 87
18.2.2. Do Developer ID certificates need to be renewed? . . . 87
18.2.3. What happens when a Developer ID certificate expires? . . . 87
18.2.4. Why might I not renew a Developer ID certificate? . . . 88
18.3. Version and feature compatibility . . . 88
19. SEE API documentation . . . 92
19.1. SEElib functions . . . 92
19.1.1. SEElib_init . . . 92
19.1.2. SEElib_ReadUserData . . . 92
19.1.3. SEElib_ReleaseUserData . . . 93
19.1.4. SEElib_InitComplete . . . 93
19.1.5. SEElib_StartTransactListener . . . 93
19.1.6. SEElib_Transact . . . 93
19.1.7. SEElib_MarshalSendCommand . . . 94
19.1.8. SEElib_GetUnmarshalResponse . . . 94
19.1.9. SEElib_FreeCommand . . . 94
19.1.10. SEElib_FreeReply . . . 95
19.1.11. SEElib_SetPort . . . 95
19.1.12. SEElib_SubmitCoreJob . . . 95
19.1.13. SEElib_GetCoreJob . . . 95

19.1.14. SEElib_GetUserDataLen . . . 96
19.1.15. SEElib_Submit . . . 96
19.1.16. SEElib_Query . . . 96
19.1.17. SEElib_AwaitJob . . . 97
19.1.18. SEElib_AwaitJobEx . . . 97
19.1.19. SEElib_ReturnJob . . . 98
19.1.20. SEElib_StartProcessorThreads . . . 98
19.1.21. SEElib_StartSEEJobListener . . . 99
19.1.22. SEElib_QuerySEEJob . . . 99
19.1.23. SEElib_ReleaseSEEJob . . . 99
19.2. Host-side SEEJobs . . . 100
20. System calls allowed by CodeSafe 5 applications . . . 101
20.1. Denied System Calls . . . 101
20.2. Allowed System Calls . . . 101

1. Introduction

CodeSafe 5 evolves Entrust's established Secure Execution Engine (SEE) technology, which allows third-party code to run within the HSM. The CodeSafe 5 product comprises an SDK for application development, and tooling for managing the deployment of those applications on HSMs.

CodeSafe 5 is available on the 5s, 5c, and 5c 10G HSM models. Running a CodeSafe 5 application is a licensable feature.

1.1. CodeSafe basic concepts

A CodeSafe 5 application is built using the SDK and assembled into a CodeSafe 5 image (.cs5) file. The developer then applies one or more signatures to the image binary, typically using keys held in a Security World, and the resulting signed image is loaded onto one or more nShield HSMs. These signatures are verified before application execution begins.

The application runs and receives commands (in some application-specific form) from an external client. It may perform its own computations, and has access to the nCore API for cryptographic operations. The keys which signed the application can be referenced in nCore command certificates, which in turn allows the creation of Security World keys which can only be used by that application.

All commands, replies, temporary keys and other nCore API objects created by a running CodeSafe 5 application are private to that instance. A running instance is sometimes referred to as a CodeSafe *machine*, and its data is not visible outside the HSM or to other nCore API clients.

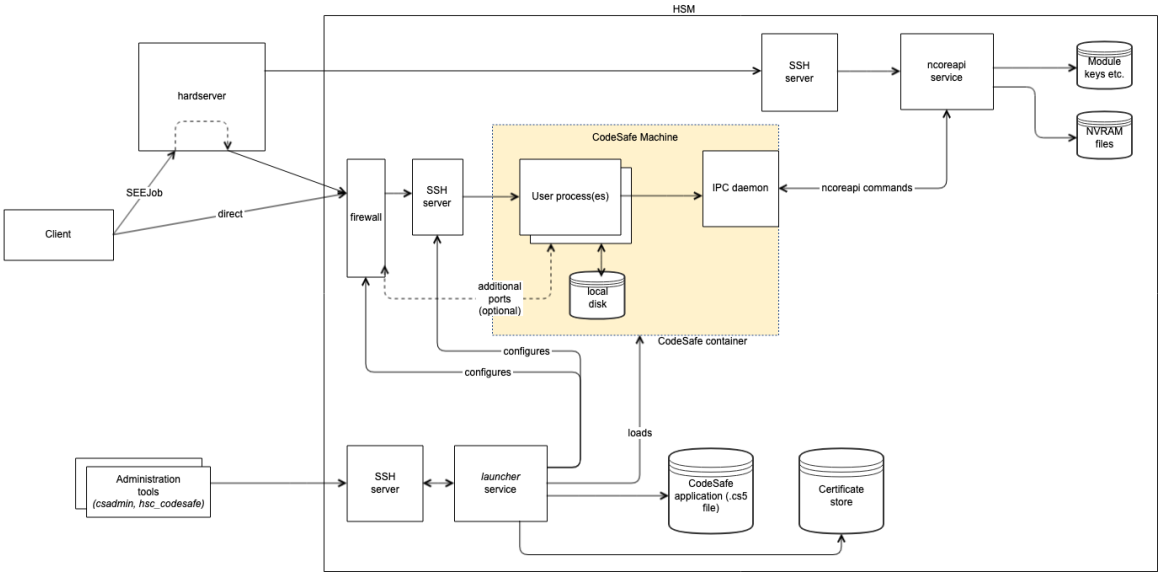
The CodeSafe 5 machine runs in a protected, logically separate environment to the nCore API service; this means that the FIPS 140 validation of the HSM is not affected by the use of a CodeSafe application. Key data held by the nCore API is still subject to the restrictions expressed in its ACL; this remains true even if a CodeSafe machine itself is compromised.

So, CodeSafe applications can implement functionality such as:

- decrypting messages with one key then re-encrypting with another
- adding business-specific validation of message data before it is signed
- securely generating personalisation data for injection into embedded devices
- implementing custom cryptographic algorithms, communication protocols, etc.

2. New features in CodeSafe 5

The CodeSafe 5 execution environment is shown below:



The CodeSafe 5 architecture provides significant enhancements not available on CodeSafe for older Solo+ and XC HSMs, taking advantage of the capabilities of the HSM's Linux kernel. These include:

- The application is now a container image, rather than a single executable. It may include multiple executables, and has its own private nonvolatile storage.
- The container has access to native IPv4 and IPv6 networking. It may accept incoming network connections and make outgoing ones.
- An SSH server is provided for use by the application, to allow end-to-end secure transport of application data between the HSM and a client.
- Most commonly-used Linux system calls are supported natively, allowing a wider range of languages to be supported.
- Each CodeSafe 5 application is linked to the identity of its developer, which is verified by Entrust. The administrator can select which developer identities are permissible on each HSM.

CodeSafe 5 features gives more details.

2.1. Management of CodeSafe 5

CodeSafe 5 is managed by a new *launcher* service, which has its own network port, SSH server, and client credentials. Unlike earlier HSMs, administrators do not install or configure

of CodeSafe machines using nCore API commands. This allows separation of the user role for managing CodeSafe 5 machines from the role(s) used for regular cryptographic operations.

Three options are currently supported as *launcher* service clients:

- The *csadmin* tool. This allows the user to issue individual commands to load, start and stop CodeSafe 5 machines from the command line, or from user-written scripts.
- The *hsc_codesafe* utility. This provides a more automated way of running and configuring CodeSafe 5 machines, driven from a configuration file and triggered when the state of an HSM changes.
- KeySafe 5. This is described in the KeySafe 5 User Guide.

2.2. Compatibility with CodeSafe on Solo+ and Solo XC

A CodeSafe application for earlier HSMs will need to be recompiled in order to run on a CodeSafe 5 system. The SDK's *seelib* library provides a compatible set of API calls to allow dispatch of nCore API commands, and processing of requests from the client. This allows existing sources to be rebuilt with minimal changes.

Also, the SDK now offers a Generic Stub (*nfstub*) library, which is API compatible with the host-side Generic Stub.

Note that the `Cmd_SEEJob` nCore API command is no longer implemented by the *ncoreapi* service. Instead, the client sends `SEEJob` commands directly to the CodeSafe 5 machine via its dedicated SSH tunnel. A client can establish its own SSH connection to send commands directly to the HSM; alternatively the *hardserver* process can manage the SSH connection, and the client can send `SEEJob` commands to that hardserver. This is achieved using *hsc_codesafe* configuration.

It is no longer *required* that communication with a client application is framed into `SEEJob` commands. The SSH service (and other network connections) provides a general-purpose pipe which can be used for other protocols. If the CodeSafe 5 application does not start up the compatibility code, it can use other arrangements to accept network connections.

3. CodeSafe 5 features

3.1. Applications as container images

In CodeSafe 5, the application (also known as a SEE machine) 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 in the `/home/codesafe` directory.
- 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.

3.2. Easy and fast network connectivity

nShield 5 HSMs and CodeSafe 5 containers are logically connected via TCP and UDP networking. The container can receive incoming connections from the host-side application, establishing two-way communication between the host-side application and the container. 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.

3.3. '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 the SEE-Jobs protocol) can be configured automatically by the nShield software.

This means that applications, including applications ported from older versions of CodeSafe for Solo XC or Solo+, can benefit from strong authentication of their clients and protection from unauthorized network traffic without additional code.

3.4. Better language support

The CodeSafe 5 SDK supports:

- C11
- C++17
- Python 3

  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.

3.5. 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.

4. CodeSafe 5 Application Authentication

4.1. 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.

4.1.1. 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)

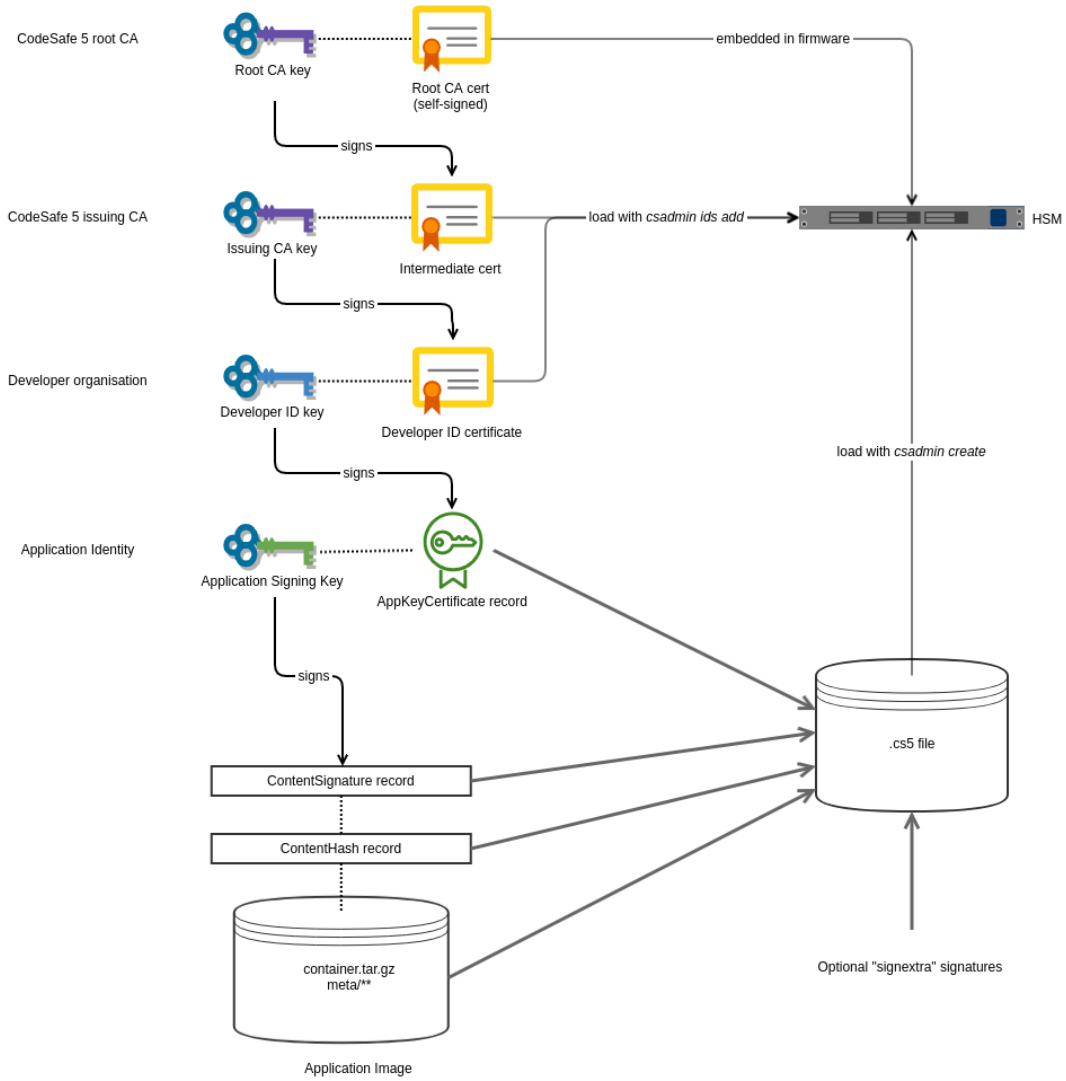
The Developer ID key belongs to a company or other organization, and that organization has an active nShield support contract or similar relationship with Entrust. Entrust issues a *Developer ID Certificate*, which is an X.509 certificate binding the Developer ID key to an organization name, location,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

4.2. 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.

4.2.1. 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 command, as 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 name can be used for the `seeintegname` parameter, or its hash passed as the `--trusted-certifier` value. The ASK 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.

4.2.2. 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 from 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.

4.2.3. 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.

4.2.4. Validation of CodeSafe 5 images

The contents of a *.cs5* file are validated when it is first loaded onto the HSM (using `csadmin load`), and when the machine is started (using `csadmin start`, or when the HSM boots if configured to start automatically).

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.

5. 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`.

6. Install the CodeSafe 5 SDK on Windows

6.1. 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.

6.2. 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.

6.3. 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`.

7. 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 applications. It replaces a number of other utilities such as `tct2`, `elftool` and `loadmache`; these are not used with CodeSafe 5.



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

7.1. 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 v13.9.6 Management Guide and sub-pages) to perform these operations.

Subcommands for management of CodeSafe 5 applications 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.

7.2. 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.

7.2.1. Create a Developer ID key and CSR

Developer ID keys, and their associated certificate requests (CSRs), are created with `csad-`

`min 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
```

In this command:

- `KEYNAME` 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, otherwise this command generates a fresh CSR for the existing key. The Developer ID key should be OCS-protected.
- `ORGANIZATION` and `COMMON_NAME` should match the name and DNS name associated with your organization, as understood by Entrust Technical Support.

7.2.1.1. 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 sent 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.

7.2.2. 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.

7.2.3. 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.

7.2.4. 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-----
```

7.2.5. 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
```

7.2.6. Manage Developer ID certificates on an nShield 5c appliance

When CodeSafe 5 applications are auto-loaded from the `[codesafe]` section of the nShield 5c (Connect) appliance's own `config` file, the required Developer ID certificates are loaded automatically (see the tip under Upload and view Developer ID certificates), and no client has direct access to the `csadmin` utility on the appliance. In that case, use the `appliance-cli cs5` commands to list or remove the loaded certificates on the appliance if required (replace `-m1` with the module number of the 5c):

- `appliance-cli -m1 cs5 certlist` lists the certificates currently loaded on the appliance.
- `appliance-cli -m1 cs5 certremove <serial-number>` removes the certificate with the given serial number.



In the current release these commands are refused while auto-load is enabled (`enabled=yes` in the `[codesafe]` section), reporting `Command '<name>' unavailable as Codesafe 5 autoload currently configured on Connect 5c`. To list or remove certificates, temporarily set `enabled=no` in the `[codesafe]` section and apply the configuration, then run the com

> mand. A CodeSafe 5 application that is already running is not stopped by this change.

See Auto-Load Configuration of CodeSafe 5 Applications via the nShield 5c for appliance auto-load configuration.

7.3. Image creation and signing

7.3.1. Create an image file

The `csadmin image generate` command takes a directory containing a CodeSafe 5 application container contents, and packs it into a 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
```

In this command:

- `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.

7.3.1.1. 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!
```

7.3.2. 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
```

After an image has been signed, and optionally encrypted, `csadmin image inspect` reports additional fields, including `Signed`, `SigningKeys`, `SHA512Hash`, and (for an encrypted image) `Encrypted` and `EncryptKey`. See Encrypt a CodeSafe 5 image for an example.

7.3.3. 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 recommended 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
```

7.3.4. Sign with an Application Signing Key

A CodeSafe 5 image 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]
                        [--enckey ENCKEY] [--encapp ENCAPP]
                        [--encauthkey ENCAUTHKEY] [--encauthapp ENCAUTHAPP]
                        [--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
  --enckey ENCKEY       Optional name (ident) of key for image encryption
  --encapp ENCAPP       Appname for ENCKEY (default 'seeconf')
  --encauthkey ENCAUTHKEY
                        Optional name (ident) of the key to authorize use of
                        ENCKEY. If --enckey is but --encauthkey is not
                        specified, the ASK is used for authorization.
  --encauthapp ENCAUTHAPP
                        Optional appname for ENCAUTHKEY (default 'simple';
                        ignored if --encauthkey is not specified)
  --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 (that is, 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, you can use the `DEVKEYNAME` and `DEVAPP` parameters to identify this key should be specified. In this case the `--startdate` 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.



You must specify the `--devcert` option to supply information about the Developer ID key. However, the Developer ID certificate is not copied into the *.cs5* file itself; it is uploaded separately (see Upload and view

> Developer ID certificates) when the CodeSafe 5 application is deployed on an HSM. This allows the certificate to be renewed without requiring a new *.cs5* file.

The `csadmin image sign` command can also encrypt the image in the same step, using the `--enckey`, `--encapp`, `--encauthkey` and `--encauthapp` options. See Encrypt a CodeSafe 5 image.

7.3.5. 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 an existing `seeinteg` key signs the application 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
```



> If a CodeSafe 5 machine has already been loaded on the HSM, adding an extra signature to its image using `csadmin image signextra` does **not** automatically cause the machine to be reloaded.
>
> If the machine continues to run with the previously loaded image, verifi-

> cation of the newly added signature will fail, and `VerifyFailed` messages may be reported in the logs.
>
> To ensure that the updated image is used, configure the hardserver to force a reload of the CodeSafe 5 machine when the image changes. This can be done by enabling the `force_reload` setting in the hardserver configuration.

7.3.6. 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.

7.3.6.1. 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
```

In this command:

- `ASKAPP` and `ASKEYNAME` specify the key which will be placed in the certificate message. You should run this command 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 application. This must match the `PACKAGENAME` which will be given later to `csadmin image generate`, when creating images to be signed with this ASK.

- You should ensure the `DEVCERT` file is available (to supply information about the Developer ID key), but the Developer ID key itself is not needed.

7.3.6.2. Sign an ASK Certificate message

The `csadmin image signask` command uses the Developer ID key to sign the output from the `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
```

7.4. Encrypt a CodeSafe 5 image

CodeSafe 5 images can optionally be encrypted so that the image contents are confidential at rest. An encrypted image is decrypted only inside the HSM, at load time; the plaintext image is never exposed on the host.

Encryption uses AES-256-GCM. The image signatures are computed over the plaintext con tent, whose hashes are stored separately in the image. Because of this:

- An image can be signed first and encrypted afterwards, or signed and encrypted in a single `csadmin image sign` command (see Sign and encrypt in a single command).
- Additional signatures can still be added with `csadmin image signextra` after the image has been encrypted.

Signing and encryption are separate operations, so they can be performed in different Secu rity Worlds. When you run `csadmin image encrypt` as its own step, the `seeconf` encryption key (and the `seeinteg` key that certifies it) can belong to a different Security World from the Application Signing Key (ASK) and the Developer ID key. The only requirement is that the `seeconf` key and its certifying `seeinteg` key belong to the same Security World, so that they can be loaded together during encryption. Signing and encryption can only be com bined in one `csadmin image sign` command when all the keys involved are in the same Secu

rity World.

Encrypting an image raises its file format version to 2. An encrypted image can only be loaded on an HSM whose firmware supports CodeSafe 5 image encryption. On older firmware, loading fails with an error such as `File version was 2; this tool only supports up to version 1`.

7.4.1. Create an image encryption key

The encryption key is an AES-256 `seeconf` key created with the generatekey utility. It should be generated module-protected to support automatic loading. The `seeintegname` parameter must be specified to designate an existing `seeinteg` key that must be presented to authorize use of the encryption key for the encrypt operation. This certifier key must belong to the same Security World as the `seeconf` encryption key, because the two keys are loaded together when the image is encrypted. We recommend using the Application Signing Key (ASK) as the certifier if you intend to sign and encrypt in one command (see Sign and encrypt in a single command), although this is not mandatory. Using the ASK as the certifier in that case avoids the need to create another `seeinteg` key. If encryption is done as a separate step from signing, any other `seeinteg` key may be used as the certifier. The certifying `seeinteg` key does not need to be the ASK or Developer ID key. If encryption is performed as a separate step, the key does not even need to belong to the same Security World. The certifier key (and its protection credential) must be loaded when encrypting, but not when loading the encrypted application, as the HSM protects the decryption operation internally (the ACL on the `seeconf` key generated by the `generatekey` utility does not allow general decryption by a client-side application).

```
$ generatekey seeconf
module: Module to use? (1, 2) [1] > 2
protect: Protected by? (token, module) [module] >
size: Key size? (bits, 128-256) [256] >
logkeyusage: Log key usage? (yes/no) [no] > yes
plainname: Key name? [] > aes256seeconf
OPTIONAL: seeintegname: SEE integrity key(s)? (devidseeinteg) []
> devidseeinteg
nvram: Blob in NVRAM (needs ACS)? (yes/no) [no] >
OPTIONAL: trusted-certifier: implicitly trusted seeinteg key hashes? [] >
key generation parameters:
 operation          Operation to perform                    generate
 application        Application                             seeconf
 module             Module to use                           2
 protect            Protected by                            module
 verify             Verify security of key                  yes
 type               Key type                                AES
 size               Key size                                256
 logkeyusage        Log key usage                           yes
 plainname          Key name                                aes256seeconf
 seeintegname       SEE integrity key(s)                    devidseeinteg
 nvram              Blob in NVRAM (needs ACS)               no
 trusted-certifier  implicitly trusted seeinteg key hashes
Key successfully generated.
```

```
Path to key: /opt/nfast/kmdata/local/key_seeconf_aes256seeconf
```

7.4.2. Encrypt an image with csadmin image encrypt

The `csadmin image encrypt` command encrypts an existing *.cs5* image:

```
$ csadmin image encrypt --help
usage: csadmin image encrypt [-h] [--out OUT] --enckey ENCKEY
                             [--encapp ENCAPP] --encauthkey ENCAUTHKEY
                             [--encauthapp ENCAUTHAPP] [--verbose]
                             CS5FILE

positional arguments:
  CS5FILE               The cs5 file to be encrypted

options:
  -h, --help            show this help message and exit
  --out OUT             Name of the output file. If not specified, the cs5
                        file is overwritten
  --enckey ENCKEY       Name (ident) of the encryption key
  --encapp ENCAPP       Appname for ENCKEY (default 'seeconf')
  --encauthkey ENCAUTHKEY
                        Name (ident) of the key to authorize use of ENCKEY
  --encauthapp ENCAUTHAPP
                        Appname for ENCAUTHKEY (default 'simple')
  --verbose             Print verbose logs
```

In this command:

- `--enckey` (and `--encapp`, default `seeconf`) identify the image encryption key created above.
- `--encauthkey` (and `--encauthapp`) identify the `seeinteg` key that authorizes use of the encryption key. This must be the key named as `seeintegname` when the encryption key was generated.

The following example encrypts a previously signed image. Here the encryption key is protected by an Operator Card Set (OCS), so the card is requested when the key is loaded:

```
$ csadmin image encrypt --enckey aes256seeconf --encauthapp seeinteg --encauthkey devidseeinteg --out encrypted-
echo.cs5 ./echo.cs5

Loading `exampleocs':
 Module 2 slot 2: `exampleocs' #1
 Module 2 slot 2:- passphrase supplied - reading card
Card reading complete.

* Wrote CS5 file encrypted-echo.cs5
```

7.4.3. Sign and encrypt in a single command

`csadmin image sign` accepts the same `--enckey`, `--encapp`, `--encauthkey` and `--encauthapp`

options as `csadmin image encrypt`, so an image can be signed and encrypted in one command. When `--enckey` is given but `--encauthkey` is not, the Application Signing Key (ASK) is used to authorize the encryption.

Such a command uses several keys: the Developer ID key (OCS-protected), the ASK (OCS- or softcard-protected, and may be the same key as the Developer ID key), the key that authorizes the encryption (by default the same key as the ASK), and the `seeconf` encryption key (module-protected). Where the same card-protected or softcard-protected key is used for more than one of these operations, you may be prompted to present its credentials several times.

To avoid this, run the command under the preload utility, which loads the required credentials once, up front, and makes them available for the whole operation. Naming a module with `--module` also confines credential loading to that one module, which avoids the extra work of loading an OCS on every module of a multi-module system, which is the `preload` default. The same benefit applies to single-operation commands such as `csadmin image signextra` and `csadmin image encrypt`. `preload` can also present credentials out of band, for example in an automated build where a shared `preload` session is set up in advance. See preload for details.

You do not need `preload` for correctness. The `csadmin image` commands automatically load keys that must reside on the same module together, so the `seeconf` encryption key is loaded on the same module as the key that certifies its use. Where a key is OCS- or softcard-protected, any usable module can be selected. `preload` is a convenience for caching credentials and for choosing the module up front.

The following example signs (with an ASK and a Developer ID key) and encrypts in a single command. Here the same `seeinteg` key is used as the ASK, as the Developer ID key, and as the key that authorizes the encryption, so one key makes both signatures and certifies use of the encryption key; in general these can be different keys.

```
$ preload --module=2 --cardset=exampleocs csadmin image sign --askapp seeinteg --askeyname devidseeinteg --devapp
seeinteg --devkeyname devidseeinteg --devcert developerid_cert.pem --enckey aes256seeconf --out signed-encrypted-
echo.cs5 ./echo.cs5
...
INFO: Loading complete. Executing subprocess csadmin image sign --askapp seeinteg --askeyname devidseeinteg
--devapp seeinteg --devkeyname devidseeinteg --devcert developerid_cert.pem --enckey aes256seeconf --out signed-
encrypted-echo.cs5 ./echo.cs5
* Wrote CS5 file signed-encrypted-echo.cs5
```

Stand-alone signing with the ASK (with the Developer ID key delegated up front, as described in CodeSafe 5 Application Authentication) and stand-alone encryption with `csadmin image encrypt` remain fully supported; this combined command is a convenience.

`csadmin image inspect` reports whether an image is encrypted, and the name of the encryp

tion key:

```
$ csadmin image inspect signed-encrypted-echo.cs5
Type               codesafe-container
Platform           nShield5
Format             1
PackageName        echo
Version            1.0
EntryPoint         /usr/bin/entrypoint
Signed             True
SigningKeys        seeinteg-devidseeinteg
SHA512Hash
8deeb5fa1294ab70c278977e5665bce7c68d5fe71b14251b1d64d128ed02869a83833f68e93dcd08f96a92d105e46e60734c1174fbeff3e70
003b47e7d69181a
Encrypted          True
EncryptKey         aes256seeconf
```

7.5. Load an encrypted CodeSafe 5 image

To load an encrypted image, the HSM must have the image encryption key installed as a SEE Loader key. The image is then decrypted inside the HSM. The `seeconf` encryption key must therefore be present in the Security World in which the HSM is enrolled when the image is loaded, even if the image was encrypted in a different Security World. Only the encryption key itself is needed at load time; the `seeinteg` key that certified it during encryp tion is not needed to load the image. The decrypted form is retained (including across reboots) so the application can be restarted without presenting the key again, unless the machine is destroyed and reloaded. The signatures are verified over the plaintext image every time the application starts, exactly as for an unencrypted image.

If the encryption key has not been installed, loading fails with an error such as `Required decryption key '<keyname>' has not been installed`.

There are three ways to load an encrypted image; each installs the encryption key for you when you identify it:

Manual load with `csadmin load`

Pass the encryption key with `--enckey` (and `--encapp`, default `seeconf`):

```
$ csadmin load --esn E866-0C27-CF25 --enckey aes256seeconf signed-encrypted-echo.cs5
E866-0C27-CF25        UUID: de131c2a-0da7-483b-9ccd-26233fe4d600
```

Interactive load with `hsc_codesafe`

Specify `encryption_key=<key ident>`. The `hsc_codesafe` name=value arguments correspond to the `[codesafe]` configuration fields.

```
$ hsc_codesafe -m2 image_file=signed-encrypted-echo.cs5 worldid_pubname=echosee encryption_key=aes256seeconf
-- pause
```

```
hsc_codesafe:INFO: Installing SEE Loader key: aes256seeconf
...
hsc_codesafe:INFO: READY: signed-encrypted-echo.cs5 running on E866-0C27-CF25 (#2); SEE World ID published
as echosee
```

Automatic load from the `[codesafe]` configuration section

Set the `encryption_key` field in the `[codesafe]` section of the `config` file. See Automatic Configuration of CodeSafe 5 Applications via the Host Machine and Auto-Load Configuration of CodeSafe 5 Applications via the nShield 5c.

7.5.1. Errors when loading an encrypted image

Loading an encrypted image fails in two common cases. The following example uses `csadmin load` without an encryption key against two modules: one whose firmware does not support image encryption, and one that supports it but does not yet have the encryption key installed:

```
$ csadmin load signed-encrypted-echo.cs5
8ED1-2C9A-9331        ERROR: Failed to create machine: launcher create: File version was 2; this tool only
supports up to version 1.
E866-0C27-CF25        ERROR: Failed to create machine: launcher create: LXCError Required decryption key
'aes256seeconf' has not been installed
Failed to load the machine for module(s): ['E866-0C27-CF25', '8ED1-2C9A-9331']
```

`File version was 2; this tool only supports up to version 1`

The target module's firmware does not support CodeSafe 5 image encryption. Load the image on a module whose firmware supports image encryption, or upgrade the module firmware.

`Required decryption key '<keyname>' has not been installed`

The module supports image encryption but the encryption key was not supplied, so it could not be installed. Re-run the load, identifying the encryption key: pass `--enckey` to `csadmin load`, `encryption_key=` to `hsc_codesafe`, or set the `encryption_key` field in the `[codesafe]` configuration section. The same error appears in the client or 5c hardserver log if an encrypted image is auto-loaded from the `[codesafe]` section without `encryption_key` set.

8. Building CodeSafe 5 applications

8.1. General SDK use

The CodeSafe 5 SDK provides the tools necessary to build and run CodeSafe applications on nShield 5 HSMs. The CodeSafe 5 applications (also known as 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 CodeSafe application and the host. The SDK provides headers and libraries for building applications and a root file system for the container containing runtime libraries, binary tools such as `touch`, `cat`, `grep` and optionally a Python distribution.

8.2. Prerequisites

- Access to an nShield 5 HSM, with an installed Security World, for managing signing keys.
- CodeSafe 5 SDK installation, as described at Install the CodeSafe 5 SDK on Linux or Install the CodeSafe 5 SDK on Windows
- `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 Developer Identity management.

8.3. SDK file structure overview

8.3.1. 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 which are required for operations on CodeSafe applications (also known as SEE machines) might be found elsewhere in the main installation. For example, `csadmin`, which enables loading, starting, and stopping CodeSafe applications, 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.

8.3.2. 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.

8.3.2.1. 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.

8.3.2.2. 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.

8.3.3. 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 CodeSafe application'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.

8.3.3.1. Example extra-packages-conf.json file

This example shows how to specify extra packages to be included in the container image, which will be copied from the host into the container at build time.

The `binaries` package is required for most applications, as it includes the IPC daemon which is required to mediate between the CodeSafe application and the nCore API service.

The other packages are needed for all applications: * The `python` packages can be omitted if Python is not being used. * The `kmlocal` and `cardlist` packages support use of the NFKM library as they pull in Security World files to their usual `/opt/nfast/kmdata` locations within the container.

It is also possible to specify custom packages here, which can be copied from the host into the container at build time, and then used by the application. These may be relative paths to the nShield installation location, like the `python` and `binaries` packages in the example; absolute paths to any location on the host file system; or paths referenced via environment variables, as in the `kmlocal` and `cardlist` examples.

```
{
    "packages": [{
            "package": "python",
            "description": "python 3.11 binaries",
            "host_path": "python3/csd5/ppc64/usr/bin",
            "machine_path": "usr/bin",
            "exclude": ""
        },
        {
            "package": "python",
            "description": "python 3.11 libraries",
            "host_path": "python3/csd5/ppc64/usr/lib/python3.11",
            "machine_path": "usr/lib/python3.11",
            "exclude": ""
        },
        {
            "package": "binaries",
            "description": "binaries for script support 1.0.0",
            "host_path": "c/csd5/rootfs",
            "machine_path": "/",
            "exclude": ""
        },
        {
            "package": "kmlocal",
            "description": "Security World files",
            "host_path": "${NFAST_KMLOCAL}",
            "machine_path": "opt/nfast/kmdata/local",
            "exclude": ""
        },
        {
            "package": "cardlist",
            "description": "cardlist (smartcard allow-list for NFKM library)",
            "host_path": "${NFAST_CARDLIST}",
            "machine_path": "opt/nfast/kmdata/config/cardlist",
            "exclude": ""
        }
    ]
}
```

8.3.3.2. Example network-conf.json file

This example shows port 8888 (the default port for the nShield SEEJobs protocol) accessible securely through the SSH tunnel. TCP ports 8000 and 8001 are exposed directly on the HSM network interface for other application services, and no outgoing ports are allowed.

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

8.3.3.3. Example entry point script

```
#!/bin/sh
# Export any environment variables needed by the application here, for example:
# export MY_VAR=value
# Then run the application itself, usually installed in /usr/bin:
/usr/bin/myapplication
# The script may contain additional commands for diagnostics or error reporting.
# Any output from the application or from this script will be captured in the CodeSafe application logs.
```

8.3.4. CMake

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

- **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.

8.3.5. Include directories

The SDK provides two directories with header files that can be included along with their respective libraries to provide additional functionality in CodeSafe applications. 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\*`

8.3.6. nShield-specific CodeSafe libraries

8.3.6.1. C/C++ libraries

The nShield-specific C libraries are located in:

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

Applications that wish to communicate with the nCore API service, or expose the SEEJobs protocol, from a C or C++ application, must link against either of two main groups of libraries:

- **Traditional SEElib (including SEEJobs)**:
  - `seelib.a` and `librtusr.a` libraries.
  - This exposes the `SEElib` API, which includes support for SEEJobs and communication with the nCore API.
  - `seelib.a` is not compatible with `libnfstub.a` / `libnfkm.a`.
  - This restricts to the traditional APIs that were historically referred to as CSEE APIs.
- **Standard host-side APIs**:
  - These APIs are the same as those used by host applications communicating with nShield HSMs, and are not specific to CodeSafe applications.
  - The `hellonshield` example CodeSafe application in the SDK (`/opt/nfast/c/csd5/examples/netsee/hellonshield` on Linux and `C:\Program Files\nCipher\nfast\c\csd5\examples\netsee\hellonshield` on Windows) demonstrates use of the C Generic Stub API and NFKM API in a CodeSafe application, and can be used as a reference for this use-case. It doesn't have a client-side component, and the example's output is simply written to the application log.
  - **C Generic Stub (the NFastApp API for nCore)**
  - `libnfstub.a` library, and additionally add the `hilibs` include subdirectory, such as `/opt/nfast/c/csd5/include-see/hilibs`, for headers such as `nfastapp.h`.

- `libnfstub.a` is not compatible with `seelib.a` but it is compatible with `libnfkm.a`.
- `libnfstub.a` includes all of the SEElib API in a form that is compatible with the C Generic Stub, enabling SEElib (including SEEJobs) to be combined with use of the standard host-side APIs if desired.
- **NFKM (the Security World API)**
- `libnfkm.a`, and additionally add the `sworld` include subdirectory, such as `/opt/nfast/c/csd5/include-see/sworld`, for headers such as `nfkm.h`.
- Effective use of the NFKM library will usually require the `kmlocal` and (if interacting with smartcards) `cardlist` packages to be included in the image as described in Example extra-packages-conf.json file.

8.3.6.2. Combining SEElib with C Generic Stub (NFastApp) or NFKM



When combining SEElib with NFastApp or NFKM, always link against `libnfstub.a` instead of `seelib.a`.

The traditional CodeSafe SEElib library exists as the stand-alone library `seelib.a`. This static library is not compatible with `libnfstub.a` or `libnfkm.a`.

As a convenience for users combining use of SEElib (`SEElib_*` functions) with the `NFastApp_*` functions in the C Generic Stub and/or the NFKM APIs, the C Generic stub static library `libnfstub.a` in CodeSafe 5 includes the full SEElib API.

If the goal is to migrate use of `SEElib_Transact` to `NFastApp_Transact` for consistency with client-side application code whilst still retaining support for SEEJobs, or if SEElib is being used primarily but NFKM library must also be used in the same application, then `libnfstub.a` must be used instead. This provides all the SEElib symbols in a form compatible with `libnfstub.a` and `libnfkm.a`. `seelib.a` must not be linked against in this case. The usual `seelib.h` header should continue to be used to obtain the declarations, but if also including C Generic Stub (such as `nfastapp.h`) or NFKM headers (such as `nfkm.h`), they should be included before `seelib.h`.

If `SEElib_init()` is called in the `libnfstub.a` implementation, it will initialize an `NFast_AppHandle` internally which is configured to marshal `M_Bignum` objects in the form defined by `struct NFast_Bignum` in the `seelib.h` header, which is an `M_ByteBlock` containing a little-endian representation of the value. This makes the default marshalling behaviour and `struct` representation of objects when migrating existing SEElib code from `seelib.a` to `libnfstub.a` identical.

In order to support inter-operation with other `NFastApp` use-cases, new APIs `SEElib_init_NFastApp()` and `SEElib_init_NFastApp_default()` have been added in `seelib.h` that can be

called instead of `SEElib_init()` if required. These new functions are declared if `nfastapp.h` is included before `seelib.h`. `SEElib_init_NFastApp()` allows initialization of SEElib with the caller's own `NFast_AppHandle`, allowing the caller to precisely control the NFastApp initialization. `SEElib_init_NFastApp_default()` initializes the SEElib library with the default NFastApp Bignum implementation defined in `nfastapp-bignum.h`. This provides maximum compatibility with typical default NFastApp initialization that would be used in most client-side application code that is being migrated to CodeSafe 5. If `nfastapp-bignum.h` is being used, it should be included before `seelib.h`.

If "manual" marshalling or unmarshalling of nCore types is being performed using functions like `NF_Marshal_CertificateList` or `NF_Unmarshal_CertificateList`, the `struct NF_UserData *` context set in the `NF_Marshal_Context.u` or `NF_Unmarshal_Context.u` field was set to `NULL` in traditional `seelib.a` applications. If using marshalling or unmarshalling functions when linking against `libnfstub.a`, the `struct NF_UserData * u;` field must now be set to a valid Generic Stub object. This should either be obtained in the usual manner using the `NFastApp` APIs, or if using the `SEElib` initialization extensions, it can be obtained using the new `SEElib_GetUserData()` function. When using `seelib.a`, `SEElib_GetUserData()` will always return `NULL` as it is not applicable for the implementation of the marshalling and unmarshalling functions in that version of the library.

8.3.6.3. Python modules

The nShield-specific Python modules are `nfpython` (for nCore interfaces) and `nfkm` (for NFKM interfaces, which is a superset of the `nfpython` module and so can be used instead of `nfpython` if using both APIs).

If using the Python runtime from the extra packages (see Example extra-packages-conf.json file), these modules are available as `import nfpython` or `import nfkm` without any additional configuration, and behave in the same way as they do on the host side, except that they communicate with the nCore API service via the IPC daemon within the container instead of communicating via the hardserver service.

As with the C `libnfkm.a` API, effective use of the Python `nfkm` API for Security World functionality will usually require the `kmlocal` and (if interacting with smartcards) `cardlist` packages to be included in the image as described in Example extra-packages-conf.json file.

The Python `webserver` example in the SDK (`/opt/nfast/python3/csd5/examples/webserver` on Linux and `C:\Program Files\nCipher\nfast\python3\csd5\examples\webserver` on Windows) demonstrates use of the `nfpython` and `nfkm` modules in a CodeSafe application, and can be used as a reference for this use-case. It can be used in conjunction with the supplied `hostside` client, or from a web browser.

8.3.6.4. C++ applications

CodeSafe 5 supports C++17.

You can link the `seelib.a` or the `libnfstub.a` and `libnfkm.a` libraries with C++ applications to use the nShield APIs. The relevant header files contain `extern "C"` guards so that they can be included in C++ code.

The `helloplus` example in the SDK (`/opt/nfast/c/csd5/examples/netsee/helloplus` on Linux and `C:\Program Files\nCipher\nfast\c\csd5\examples\netsee\helloplus` on Windows) shows how to build a simple CodeSafe 5 application with C++ 17 and demonstrates various C++ language and library features working with the cross-compiler. It doesn't have a client-side component, and the example's output is simply written to the application log.

If not using CMake to build your CodeSafe 5 application, the C++ compiler and linker flags defined in the SDK's `codesafe-toolchain-nshield5-csee.cmake` file should be consulted to obtain the correct flags for your own build files. Note that the `-static` linker flag is required to statically link the C and C++ runtime libraries in this case.

The reserved C++ keyword `export` is used as a field name in `M_Command.args.export` and `M_Reply.reply.export` (defined in both `seelib.h` and `nfastapp.h` headers). By default, the field is renamed using the pre-processor to `exportcmd` when compiling with a C++ compiler. If an application has already worked around this issue by defining `export` before including relevant nShield headers, the pre-processor renaming will not be applied, and the user's existing work-around will continue to behave as before. If `STDMARSHAL_KEEP_EXPORT_RENAME` is defined before including relevant headers, `export` will not be `#undef`-ed after the affected headers, so that code can still reference `M_Command.args.export` and `M_Reply.reply.export` for backwards-compatibility with code migrated from C to C++; this should only be done if `export` is not otherwise being used in the codebase. To completely disable these workarounds, define the `STDMARSHAL_NO_EXPORT_RENAME` macro before including relevant headers.

8.3.7. 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.

8.4. Building new CodeSafe applications

A CodeSafe application is a container image with a complete filesystem which can be loaded onto a CodeSafe 5-enabled HSM as part of a container. The SEElib, Generic Stub, or `nfpython` library enables CodeSafe applications to interface with the nCore API via the IPC daemon.

Source code is compiled using one of the GCC (or G++) 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 CodeSafe applications on Linux and Build and sign example CodeSafe applications on Windows.

The container image (the `.cs5` file) must be signed using the `csadmin` utility.

8.4.1. Developer authentication

CodeSafe 5 requires a signed CodeSafe image to run CodeSafe applications 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`.

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 Code Safe application images with the `csadmin` tool.

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

8.4.2. Deploying CodeSafe applications

After the code has been compiled, built, and signed, the `csadmin` utility may be used to deploy the CodeSafe application, on a 5s or remotely on a 5c, although in most deployments this is done indirectly via the automatic-loading configuration in the host-side `config` file or 5c (Connect) `config` file, or interactively via the `hsc_codesafe` utility, in order to simplify deployment.

For more information on the `csadmin` utility, see Create and sign CodeSafe 5 applications with csadmin. For more information on automatic-loading configuration, see Automatic Configuration of CodeSafe 5 Applications via the Host Machine.

When the CodeSafe application starts, the IPC daemon is started automatically in the background, and then the `entrypoint` script is executed, which then runs the primary process in the container and its `main()` function.

8.4.3. CodeSafe application initialization requirements

If using `SEElib`, a CodeSafe application must initialize `SEElib` before making use of any of its functionality. This is done by calling `SEElib_init()`, or by using one of the Generic Stub (`NFastApp`) `SEElib` extension functions for initialization described in Combining SEElib with C Generic Stub (NFastApp) or NFKM if linking against the Generic Stub library rather than traditional `seelib.a`. It is recommended that this call is made immediately within the `main()` function of a CodeSafe application.

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()`. If multiple processes are running within the container that need to use SEElib, only one of them can listen for SEEJobs, and so all but one of the processes must call `SEElib_SetPort(0)` before `SEElib_init()` to avoid multiple initializations of the SEEJobs listener.

If using the Generic Stub (`NFastApp`) library, the same initialization requirements as for host-side usage apply, and if using the `NFastApp_Init()` or `NFastApp_InitEx()` functions, rather than the `SEElib` extensions for `NFastApp`, this will not initialize the SEEJobs listener, and the usual functions such as `NFastApp_Connect()` and `NFastApp_Transact()` will be available for issuing nCore commands via the IPC daemon. If the `SEElib` extensions for `NFastApp` are used for initialization, the SEEJobs listener will be initialized by default as described in the previous paragraph, and the usual `SEElib` functions will be available for issuing nCore commands via the IPC daemon, including support for SEEJobs, in addition to the `NFastApp` functions.

8.4.4. 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.

8.4.4.1. 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`.

8.4.4.2. 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 unmar shalled 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.

8.4.4.3. 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()`.

8.4.4.4. Other SEElib methods

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

8.4.5. Host/CodeSafe application communication

Host/CodeSafe application 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 (or optionally using the 5c "Connect" `config` file if using a 5c). Alternatively (or additionally) communication between the host and the CodeSafe application can be done via TCP and UDP IPv6 networking.

Multiple processes within the container can connect to the `ncoreapi` service concurrently (this is multiplexed automatically by the IPC daemon), but only one process may bind to the SEEJobs port, or to any other port for implementing a custom protocol.

8.4.5.1. IPv6 vs IPv4 networking

The 5s HSM itself only exposes an IPv6 network interface, so only IPv6 can be used for direct TCP or UDP communication between the host and the CodeSafe application. However, if using a 5c (Connect form factor), then the communication is relayed automatically to the 5c's network interface, which supports both IPv4 and IPv6, so the host can communicate with the CodeSafe application via IPv4 if desired in that case. See Working with CodeSafe for more information regarding configuration of CodeSafe on the 5c.

9. nShield 5c CodeSafe 5 Configuration

The nShield 5c (Connect) can be configured for two different modes of automatic CodeSafe 5 SEE machine loading. These two configurations are mutually exclusive:

1. Automatic loading from the `[codesafe]` configuration section in the nShield 5c (Connect) appliance `config` file. See: Auto-Load Configuration of CodeSafe 5 Applications via the nShield 5c. If using SEEJobs, this enables multiple remote clients to communicate with the CodeSafe 5 application via its published SEE World ID. It also allows the CodeSafe application to be automatically loaded on appliance startup so it is ready for SEEJobs or other TCP communication as implemented by the CodeSafe application shortly after boot. SEEJobs communication externally on the network is secured via nCipher Secure Transport/Impath rather than SSH in this case, and the internal communication between the 5c hardserver and the CodeSafe application inside the HSM may optionally be configured to be plaintext for performance reasons if this is acceptable for the security model (although SSH is used internally in the appliance by default).
2. A single remote client machine has direct control over the CodeSafe `launcher` service inside the HSM, and has direct access to configure the CodeSafe application via the `csadmin` utility run on the client machine, but doesn't allow multiple remote clients access to the CodeSafe 5 application via SEEJobs, although any non-SEEJobs TCP services exposed in the CodeSafe application's firewall rules will still be accessible via the 5c's network interface. This setup uses the `[codesafe]` configuration section in the `config` file of the machine that is the client of the nShield 5c, in which case both `launcher` and (if applicable) SEEJobs SSH protocol keys are securely exchanged automatically to enable secure communication directly to the HSM inside the 5c over the network. See: Automatic Configuration of CodeSafe 5 Applications via the Host Machine. This configuration is more convenient for debugging purposes (including via interactive `hsc_codesafe` execution) due to the direct `csadmin` and `launcher` service access, and also enables direct communication to the HSM for SEEJobs which has security and performance benefits.

It is also possible to manually configure the nShield 5c for CodeSafe 5. See CodeSafe 5 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 a Connect image prior to v13.7 when automatic configuration was introduced).

10. Auto-Load Configuration of CodeSafe 5 Applications via the nShield 5c

The nShield 5c (Connect) with this configuration supports the following CodeSafe 5 functionality:

- CodeSafe 5 applications can be started directly via the nShield 5c (Connect) config file, with the CodeSafe 5 image, Developer ID certificates, and encryption key (if specified) pulled automatically from the RFS.
- Applications specified in the nShield 5c config are started automatically when the nShield 5c is rebooted or module cleared.
- CodeSafe 5 applications loaded directly from the nShield 5c can make their SEE World ID available as a published object, enabling multiple remote client machines to communicate with the same CodeSafe 5 application via the SEEJobs protocol.

10.1. Installation and Setup

Install the latest nShield host-side software and run the `rfs-setup` utility to allow the nShield 5c to retrieve the required Developer ID certificate(s) from the RFS's `cscerts` volume. If you configured your RFS before v13.9.5, run `rfs-setup` again to add the permission for the new volume.

Build and sign the CodeSafe 5 image as described in Building CodeSafe 5 applications.

Permit the nShield 5c to have a configuration file pushed from a remote machine. By default, the configuration file can be pushed from the RFS, provided it is authenticated. To allow another remote client to push configuration, use the 5c's `Client config push` menu from its front panel UI (menu 1-1-6-2). For more information, see Connect configuration.

Configure the nShield 5c to append its logs (hardserver and system logs) to the RFS once per minute in order to get timely feedback on CodeSafe 5 configuration operations. For more information, see Connect logging configuration.

Copy the Developer ID certificate(s) required by the CodeSafe 5 application to this directory on the RFS `/opt/nfast/kmdata/cscerts` (Linux) or `C:\ProgramData\nCipher\Key Management Data\cscerts` (Windows).

Copy the CodeSafe 5 image into `/opt/nfast/custom-seemachines` (Linux) or `C:\Program Files\nCipher\nfast\custom-seemachines` (Windows).

If the CodeSafe 5 image is encrypted, make sure the `seeconf` key that protects it is present

on the RFS in the Security World key data directory `/opt/nfast/kmdata/local` (Linux) or `C:\ProgramData\nCipher\Key Management Data\local` (Windows), so that the key is pulled to the appliance along with the image.

10.2. Configuring CodeSafe 5 Auto-Load

Copy the original nShield 5c configuration file `/opt/nfast/kmdata/hsm-<ESN>/config/config` (Linux) or `C:\ProgramData\nCipher\Key Management Data\hsm-<ESN>\config\config` (Windows) to `config.new` in the same directory.

10.2.1. SEEJobs Configuration

Edit the nShield 5c `config.new` file's `[codesafe]` section on the RFS to have the following contents (replace `seemachine.cs5` with the actual filename - this will be pulled from the RFS volume for `custom-seemachines` so just the filename and *not* the path must be used here):

```
[codesafe]
image_file=seemachine.cs5
worldid_pubname=mysee
pull_rfs=yes
```

10.2.2. Optimized SEEJobs Configuration

The `seejobs_transport=plain` option may be used to allow plaintext communication internally inside the appliance, between the 5c (Connect) hardserver and the CodeSafe 5 application running on the HSM inside the appliance. This improves SEEJobs latency and throughput and may be appropriate if plaintext internal communication is acceptable for your security model. Remote communication for SEEJobs is secured via nCipher Secure Transport/Impath regardless of this setting, so communication is always secured on the external network.

```
[codesafe]
image_file=seemachine.cs5
seejobs_transport=plain
worldid_pubname=mysee
pull_rfs=yes
```

10.2.3. Non-SEEJobs Configuration

If SEEJobs are not being used, the `seejobs_port` option can be set to `0`. The `worldid_pubname` option may still be specified, but may not be relevant unless the SEE World ID is

needed for ticketing objects. Non-SEEJobs TCP protocols that are allowed in the CodeSafe 5 application's firewall rules will be exposed on the 5c's external network interface in this case.

```
[codesafe]
image_file=seemachine.cs5
seejobs_port=0
pull_rfs=yes
```

10.2.4. Encrypted Image Configuration

If the CodeSafe 5 image is encrypted, add the `encryption_key` field and set it to the ident of the `seeconf` key that protects the image. The key must be module-protected, and must be present in the Security World used by the appliance. For an unencrypted image, omit this field or leave it blank.

```
[codesafe]
image_file=seemachine.cs5
worldid_pubname=mysee
encryption_key=aes256seeconf
pull_rfs=yes
```

For information about creating encrypted images and their `seeconf` keys, see Encrypt a CodeSafe 5 image.

10.2.5. Copying the Configuration and CodeSafe 5 Application to the Appliance

For more information on the `[codesafe]` section, see CodeSafe 5 configuration section.

Permit the nShield 5c to have a configuration file pushed from a remote machine. By default, the configuration file can be pushed from the RFS machine, provided it is authenticated.

To allow another remote machine to push configuration, use the 5c's `Client config push` menu from its front panel UI (menu 1-1-6-2).

Configure the nShield 5c to append its logs (hardserver and system logs) to the RFS once per minute in order to get timely feedback on CodeSafe 5 configuration operations. Alternatively, you can configure a remote UDP syslog server to receive streamed logs, or you can obtain the most recent logs that have not yet been pushed to the RFS on demand from a privileged client by using the following commands (replace `-m1` with the module number of the 5c):

```
appliance-cli -m1 getservlog --log hardserver
appliance-cli -m1 getservlog --log system
```

Push the configuration to the nShield 5c using the `cfg-pushnethsm` utility.

On Linux:

```
cfg-pushnethsm -a <nShield 5c IP address> -m <module number> /opt/nfast/kmdata/hsm-<ESN>/config/config.new
```

On Windows:

```
cfg-pushnethsm -a <nShield 5c IP address> -m <module number> "C:\ProgramData\nCipher\Key Management Data\hsm-
<ESN>\config\config.new"
```

Because the configuration is set with `pull_rfs=yes`, pushing it makes the nShield 5c to auto matically fetch the CodeSafe 5 image, Developer ID certificates, and the encryption key (if configured) from the RFS into the appliance's storage within a few seconds. If any of these files change on the RFS, push the configuration again with `pull_rfs=yes` to re-fetch them. This will not immediately apply the configuration to the HSM itself or stop any currently run ning CodeSafe 5 application, but will allow the new configuration to be applied on the next reboot or clear of the module, at which point the new CodeSafe 5 application will be loaded and started automatically.

10.3. Starting the CodeSafe 5 Application

Clear the nShield 5c to apply the new configuration and start the CodeSafe 5 application:

```
nopclearfail -m <module number> --clear --wait
```

Wait for the CodeSafe 5 application to be automatically loaded and started. Check the nShield 5c's hardserver log (using one of the logging options explained in the previous sec- tion) for the following message, which indicates that the application has been started:

```
Startup: hsc_codesafe:INFO: READY: /config/see/machine/seemachine.cs5 running on <ESN> (#1); SEE World ID
published as mysee; appliance-cli -m <module number> cs5 getlog <UUID> to retrieve logs
```

Note that `seemachine.cs5` mentioned in the log message is the internal filename when it has been copied from the RFS. The actual file on the RFS with the original filename specified in the `config` file is what will have been copied.

Clients can now communicate with the CodeSafe 5 application.

See the Debugging CodeSafe 5 applications section for information about debugging a CodeSafe 5 application.

11. Automatic Configuration of CodeSafe 5 Applications via the Host Machine

11.1. 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.

11.2. 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
#
# The transport used for SEEJobs (default="ssh"). Set to "plain" for plaintext
# communication. Plaintext communication is permitted from a client-side
```

```
# hardserver to a local HSM or for internal communication between a 5c
# (Connect) hardserver and its HSM (in which case the communication on the
# external network will be secured by Impath). Only "ssh" is permitted as the
# transport between a client-side hardserver and a remote HSM.
#  seejobs_transport=TRANSPORT
#
# 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 identities) 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
#
# Set to "yes" to pull the SEE machine, and certificates from the RFS before
# loading on the remote module. (default=no)
#  pull_rfs=ENUM
#
# The ident of the seeconf key that protects the SEE machine. Only
# module-protected keys can be used here. If the machine is not encrypted then
# leave this field blank.
#  encryption_key=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.

The `pull_rfs` field is relevant only in an nShield 5c (Connect) configuration file, when the configuration is being applied directly in the appliance rather than from a remote client machine; see Auto-Load Configuration of CodeSafe 5 Applications via the nShield 5c for further details.

If the `image_file` is an encrypted CodeSafe 5 image, set the `encryption_key` field to the

ident of the seeconf key that protects it. The key must be module-protected. Leave the field blank for an unencrypted image. See Encrypt a CodeSafe 5 image for information on how encrypted images and their associated keys are created.

11.3. 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 (if the module is a 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_CreateSEEConnection 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.



The SSH connection uses a hybrid post-quantum key exchange (`mlkem768x25519-sha256`) by default to mitigate "harvest now, decrypt later" attacks. The key exchange, ciphers, and MAC algorithms can be tuned with the `NCSSH_KEY_EXCHANGES`, `NCSSH_CIPHERS` and `NCSSH_MACS` environment variables; see Environment variables.



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.

11.4. 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_CreateSEEConnection` 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
```

11.5. 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 com-

mand-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 HSM firmware version and status information that is reported in the `enquiry` output for the "normal" module, such as module 1.

11.6. 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).

12. Build and sign example CodeSafe applications on Linux

12.1. 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`.

12.2. Building Host Side C Examples

1. Create an empty directory to build the host-side clients for the CodeSafe applications, 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`.

12.3. 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`.

12.4. 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'}

12.5. Run NetSEE examples

NetSEE examples communicate between the client and CodeSafe application 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.

12.5.1. 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`).

12.5.1.1. 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!
```

12.5.2. 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`).

12.5.2.1. 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!
```

12.6. Run NetSEE examples via SSH tunnel

`NetSEE` examples communicate between the client and CodeSafe application 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.

12.6.1. 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!
```

12.7. 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.

12.7.1. 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). This means that 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`.

12.7.2. 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.
```

12.7.3. 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
   ```

13. Build and sign example CodeSafe applications on Windows

13.1. Prerequisites

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

13.2. 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:

13.2.1. 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
```

13.2.2. 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
```

13.3. 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
```

13.4. 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 -a0
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.
```

14. Build and run Java examples

The following Java examples are included:

- BenchMark5
- Echo5
- HelloWorld5
- HostTickets5

14.1. Prerequisites

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

- Java 8 (or Java 1.8x)
- Java 17
- Java 21
- Java 25

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.

14.2. 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 CodeSafe applications on Linux and Build and sign example CodeSafe applications 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.

14.3. 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"
```

14.4. 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
```

14.4.1. 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!
```

14.4.2. 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
```

14.4.3. 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
```

14.4.4. 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
```

15. Debugging CodeSafe 5 applications

15.1. CodeSafe application logging

CodeSafe application 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 CodeSafe application's own logging framework.

15.1.1. CodeSafe IPC Daemon logging

The IPC daemon will log to the CodeSafe application log if it encounters a fatal error during startup, if it loses connection to the nCore API service (after previously being connected), or if it encounters other fatal runtime issues. It will also warn if the CodeSafe application has connected to the IPC daemon but the nCore API service connection is not available for an extended period, which may indicate that the host-side startup process is taking a long time to complete or has not been configured correctly.

To enable more verbose logging, edit the `ncoreipcdaemon` script in the rootfs to set the `NFAST_IPCD_LOGLEVEL` environment variable (this is present in `/opt/nfast/c/csd5/rootfs/bin/ncoreipcdaemon` on Linux and `C:\Program Files\nCipher\nfast\c\csd5\rootfs\bin\ncoreipcdaemon` on Windows). Note that editing this will affect all CodeSafe 5 applications built with this modified rootfs.

If `export NFAST_IPCD_LOGLEVEL=1` is set in the environment, INFO-level messages are also emitted (startup, and connections accepted or closed) in addition to ERROR-level messages.

If `export NFAST_IPCD_LOGLEVEL=2` is set, logging is further extended with DEBUG-level messages.

Log messages indicate that they are from IPCD, and include a millisecond timestamp for clarity.

15.2. nShield 5c Configured for Auto-Loading CodeSafe 5 applications

If the nShield 5c is used with auto-loading of CodeSafe 5 applications via the nShield 5c's configuration, logging in the CodeSafe application is enabled by default (unless explicitly disabled). Retrieving and clearing the CodeSafe application's logs can be done via the `appliance-cli` command:

- To retrieve the CodeSafe application's logs:

```
appliance-cli -m <module number> cs5 getlog
```

- To clear the CodeSafe application's logs:

```
appliance-cli -m <module number> cs5 clearlog
```



The `csadmin` utility, or interactive use of `hsc_codesafe`, cannot be used with an nShield 5c that's configured to auto-load CodeSafe 5 applications via the appliance configuration file (instead of the host machine's configuration file). For debugging purposes, it may be convenient to remove the auto-loading configuration from the nShield 5c and use `csadmin` or `hsc_codesafe` to load and start the CodeSafe application remotely from the host machine, to simplify the workflow of testing changes to the application, and only deploy to the 5c configuration file once the application is ready for production deployment.

15.3. Host Configured for Automatic CodeSafe 5 Application Loading

If automatic configuration is used, logging in the CodeSafe application is enabled by default (unless explicitly disabled), so this does not need to be configured manually.

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

15.3.1. 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 <UUID> --esn <ESN>
```

- `<UUID>` is the UUID of the CodeSafe application created by the load command.
- `<ESN>` is the ESN of the HSM hosting the CodeSafe application.

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.

15.3.2. config log set disabled

The `config log set disabled` command should be issued while the CodeSafe application is not running. It uses the following format:

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

- `<UUID>` is the UUID of the CodeSafe application created by the load command.
- `<ESN>` is the ESN of the HSM hosting the CodeSafe application.

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.

15.3.3. log get

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

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

`<UUID>` is the UUID of the CodeSafe application 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 application
```

15.3.4. log clear

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

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

`<UUID>` is the UUID of the CodeSafe application 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
```

15.4. 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 CodeSafe application log, as shown 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 CodeSafe application should be built with debugging symbols at least at `-g1` level and not have the symbols stripped from the binary.

16. 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.

17. Porting existing CodeSafe applications to CodeSafe 5

Follow the steps in this chapter if you need to port an existing CodeSafe application (also known as a SEE machine) to run on CodeSafe 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 CodeSafe applications 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.

17.1. 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.

17.2. 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 (`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.

17.3. Deployment differences

See Automatic Configuration of CodeSafe 5 Applications via the Host Machine 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 CodeSafe applications 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.

17.4. 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.

18. CodeSafe 5 FAQ

18.1. Signing keys

See also: CodeSafe 5 Application Authentication

18.1.1. 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.

18.1.2. 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. The key should be protected by an Operator Card Set.

18.1.3. Why do I see VerifyFailed messages after adding an extra signature?

If a CodeSafe 5 machine is already loaded on the HSM, adding an extra signature to its image using the `csadmin image signextra` command does not automatically reload the running machine.

If the machine continues to run with the previously loaded image, the newly added signature cannot be verified, and `VerifyFailed` messages may appear in the logs.

To avoid this, configure the hardserver to force a reload of the CodeSafe 5 machine when

the image changes, for example, by enabling the `force_reload` setting in the hardserver configuration.

18.1.4. 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 are contained within the *.cs5* application file itself, except for the Developer ID certificate which is loaded separately into the certificate database.

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

18.2. Developer ID keys and Certificates

18.2.1. 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), which is a short text 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).

18.2.2. 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.

18.2.3. 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 load`), and when it is started (using `csadmin start`, or if auto-start 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. Any data in its non-volatile storage, such as `/home/codesafe` in the container or data stored using the nCore NVMem APIs, 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 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.

18.2.4. 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.

18.3. Version and feature compatibility

CodeSafe 5 features are delivered across components that are versioned and signed independently of each other:

- The **HSM firmware** (the PCIe module firmware). This is the only component that carries FIPS approval. The latest FIPS-approved firmware is v13.4.5. Firmware v13.8.4 is a FIPS-pending release.
- The **host-side software**, consisting of the Security World software, the CodeSafe SDK, and, for the nShield 5c, the appliance image (a separately signed artifact). These components can be updated independently of the FIPS-approved firmware.

Because the firmware and the host-side software are versioned separately, a feature can require a minimum version of either component, or both. Customers who must remain on FIPS-approved firmware (v13.4.5) can still update the host-side software and SDK to gain any feature that does not require newer firmware.

The IPC daemon, the SEElib library, and the `libnfstub.a` and `libnfkm.a` libraries work on v13.4.5 firmware. This enables applications that use these libraries to be built for and run on the FIPS-approved firmware.

Feature	Minimum firmware	Minimum host-side (SecWorld / CodeSafe SDK / 5c image)	Notes
CodeSafe 5 (baseline)	v13.4.5	v13.4.3	First CodeSafe 5 release. v13.4.5 firmware is the latest FIPS-approved firmware.
Expanded system-call set for general-purpose development	v13.7.1	v13.4.3	A small number of additional system calls were allowed in v13.5.1 firmware, and the majority added in v13.7.1 firmware. This is a firmware change, so it takes effect with newer firmware regardless of the host-side version. See System calls allowed by CodeSafe 5 applications for the full allowed list, including the firmware version in which each call was first allowed.
Python 3.11 applications	v13.5.1	v13.6.3	Note that `nfpython` / `nfkm` modules for Python were not introduced in the SDK until v13.9.5 (see below). The Python 3.11 interpreter, local file I/O, and the `nfpython` / `nfkm` modules work on v13.4.5 firmware provided a v13.9.5 or later Code Safe SDK is used. Sockets are not supported on v13.4.5 firmware with any SDK later than v13.4.3. Original v13.4.3 shipped Python 3.8 in its SDK, but lacked `nfpython` / `nfkm`.

Feature	Minimum firmware	Minimum host-side (SecWorld / CodeSafe SDK / 5c image)	Notes
Turnkey auto-load (`[codesafe]` config / `hsc_codesafe`)	v13.4.5	v13.7.3	Core auto-load works against v13.4.5 firmware. On a 5c, initial launcher client-key enrollment on firmware earlier than v13.8.0 has a Maintenance-mode caveat; see the *5c launcher client-key enrollment without Maintenance mode* entry below. See Automatic Configuration of CodeSafe 5 Applications via the Host Machine.
Extra image signatures beyond the ASK (`csadmin image signextra`)	v13.7.1	v13.7.3	See Create and sign CodeSafe 5 applications with csadmin.
Offline signing (Developer ID and ASK sign separately)	v13.4.5	v13.9.4	The Developer ID key signs a delegation for the Application Signing Key up front (`csadmin image presignask`, then `csadmin image signask`), after which images can be signed with the ASK alone (`csadmin image sign --signedask`). This is a host-side split of the signing workflow and works with v13.4.5 firmware; the minimum host-side version shown is the one that provides the tooling. See Offline signing using the Developer ID key.
ASK may be a `seeinteg` key	v13.4.5	v13.9.4	Works from v13.4.5 firmware onwards: the change is in host-side metadata, not the underlying nCore key representation.
Deterministic CodeSafe 5 builds	Not applicable	v13.9.4	Host-side / SDK tooling only.
5c launcher client-key enrollment without Maintenance mode	v13.8.0	v13.9.4	The appliance-cli client-key enrollment commands (`cs5 getclientinfo`, `cs5 setclientinfo`, and `cs5 clearclientinfo`) access the launcher client key. On firmware versions earlier than v13.8.0, these commands temporarily switch the module into Maintenance mode and are therefore not recommended for live-deployed systems. From v13.8.0 they run without Maintenance mode. See nShield 5c CodeSafe 5 Configuration.

Feature	Minimum firmware	Minimum host-side (SecWorld / CodeSafe SDK / 5c image)	Notes
Generic Stub (NFastApp) and NFKM C APIs, and their Python equivalents	v13.4.5	v13.9.5	These intentionally support v13.4.5 firmware, so that applications that use them can be built for and run on the FIPS-approved firmware. The Python equivalents (nfpython, including `nfkm`) also work on v13.4.5 firmware, because they reach the HSM through the in-container IPC daemon rather than through TCP/IP sockets.
C++ support (formalized)	v13.4.5	v13.9.5	C applications run on v13.4.5 and later firmware. C17 applications built with the current SDK are supported on v13.4.5 firmware, including language features such as exceptions, RTTI, threads and synchronization primitives, filesystem and streams, chrono and async.
Auto-load from the 5c RFS	v13.4.5	v13.9.5	Moves the auto-load function into the appliance itself, so it is not firmware-dependent. See Auto-Load Configuration of CodeSafe 5 Applications via the nShield 5c.
CodeSafe 5 image encryption	v13.8.6	v13.9.6	Encrypted images cannot be loaded on earlier firmware. See Encrypt a CodeSafe 5 image.
SSH hybrid post-quantum key exchange	v13.8.6	v13.9.6	Negotiated automatically; falls back to the classical key exchange against earlier firmware. See Environment variables for the `NCSSH_KEY_EXCHANGES` variable.

19. SEE API documentation

SEElib is an API that enables a CodeSafe application (also known as a SEE machine) to both execute nCore API commands and accept messages from the host machine via the SEE-Jobs protocol. It is supported both in CodeSafe 5 and in CodeSafe for previous HSM models.



The traditional SEElib API is provided as a library seelib.a. Its install location is /opt/nfast/c/csd5/lib-ppc64-linux-musl/seelib.a on Linux. This API is also available in libnfstub.a so that it can be used in conjunction with the Generic Stub (from libnfstub.a) and NFKM (from libnfkm.a) APIs for maximum compatibility with host-side application code. See Combining SEElib with C Generic Stub (NFastApp) or NFKM for more information about which libraries to link against and the SEElib-in-nfstub compatibility feature.

19.1. SEElib functions

19.1.1. SEElib_init

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

This function initializes the SEElib library.



This function does not return on error.



If the version of SEElib in libnfstub.a is used, this function is still available, in which case it will also initialize NFastApp automatically, but see Combining SEElib with C Generic Stub (NFastApp) or NFKM for more information about the SEElib-in-nfstub compatibility feature and its alternative initialization functions to support interoperability with the NFastApp APIs.

19.1.2. 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/userdata.codesafe`) which must be added at that path when the container image is constructed.



This is not to be confused with the concept of the `struct NF_UserData` which is used for user context to nCore marshalling and unmarshalling functions, and also in the Generic Stub (NFastApp) APIs, which is not related to the legacy CodeSafe `UserData` block concept.

19.1.3. SEElib_ReleaseUserData

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

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

19.1.4. 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.

19.1.5. 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`.

19.1.6. 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.

19.1.7. 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 CodeSafe application 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`
- `MergeKeyIDs`.

If the CodeSafe application 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.

19.1.8. 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.

19.1.9. 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*).

19.1.10. 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*).

19.1.11. 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()`.

19.1.12. 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).

19.1.13. 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).

19.1.14. 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`.

19.1.15. 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.

19.1.16. 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.

19.1.17. 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 application. 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 application.

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.

19.1.18. SEElib_AwaitJobEx

```
extern int 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.

19.1.19. 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.

19.1.20. 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 con-

venient 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 leaves a reply at pC→iobuf. The return value from the function indicates the number of bytes to be returned from this buffer.

19.1.21. 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_ReleaseSEEJob to free the buffer.

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

19.1.22. 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.

19.1.23. 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.

19.2. 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 CodeSafe application in question. The hardserver will automatically dispatch such commands over the mutually authenticated SSH tunnel to the CodeSafe application where applicable.

SEEJobs can be sent as an nCore `M_Command` with the `cmd` field set to either `Cmd_SEEJob` or `Cmd_FastSEEJob`. `Cmd_SEEJob` does not time out unless communication fails. `Cmd_FastSEEJob` times out after 2 minutes if the CodeSafe application does not reply in that time; in that case, the status is set to `Status_HardwareFailed`, but this is not fatal unless communication as a whole has failed.

Java clients may alternatively use the `seeJob()` helper method of the `PublishedSEEWorld` or `SEEWorld5` classes.

20. System calls allowed by CodeSafe 5 applications

CodeSafe 5 applications are restricted to a subset of Linux system calls (syscalls) which they are permitted to execute.

20.1. Denied System Calls

Any denied syscalls fail by returning `-1` and setting `errno` to the value `38`, which denotes `ENOSYS` (Function not implemented). This behavior applies in v13.7 firmware and later; before this, the `SIGSYS` signal was raised instead when a denied syscall was attempted.

Denying with `ENOSYS` (rather than raising a signal) allows applications to fall back gracefully to allowed alternative syscalls.

Library code (including the C library itself) often supports several alternative syscalls for an operation to maintain compatibility across platforms and kernel versions.

This enables security restrictions to be enforced while making it more likely that general-purpose library or application code can work with minimal modification in the restricted CodeSafe environment.

For example, the `getrandom` syscall is not supported in CodeSafe 5 and will fail with `ENOSYS`. This prevents applications from accidentally using the Linux kernel's non-HSM RNG.

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. Many libraries fall back to `/dev/random` or `/dev/urandom` if the `getrandom` syscall (or the `getentropy` C library wrapper, which uses `getrandom`) fails with `ENOSYS`, allowing them to continue working without modification while still using the HSM-backed RNG.

If a library does not automatically fall back, configure it explicitly to use `/dev/random`. The `/dev/random` device is always seeded and fully initialized before the CodeSafe 5 application container starts.

When cross-compiling recent versions of OpenSSL, pass `--with-rand-seed=devrandom` to the `Configure` script to ensure `/dev/random` is used. OpenSSL no longer falls back automatically when `getentropy` returns `ENOSYS`, so this option is required to avoid runtime failures.

20.2. Allowed System Calls

The table below lists all system calls allowed in CodeSafe 5 applications, including each syscall number, name, a link to the Linux documentation page, a broad operation category, and the firmware version in which it was first allowed in CodeSafe 5.

Syscall Number	Syscall Name	Category	Comment
1	exit	Process Management: Process Creation & Termination	Allowed in CodeSafe 5 since v13.4
2	fork	Process Management: Process Creation & Termination	Allowed in CodeSafe 5 since v13.4
3	read	File System: Read/Write Operations	Allowed in CodeSafe 5 since v13.4
4	write	File System: Read/Write Operations	Allowed in CodeSafe 5 since v13.4
5	open	File System: File Descriptor & Control Operations	Allowed in CodeSafe 5 since v13.4
6	close	File System: File Descriptor & Control Operations	Allowed in CodeSafe 5 since v13.4
7	waitpid	Process Management: Process Creation & Termination	Allowed in CodeSafe 5 since v13.4
8	creat	File System: File Descriptor & Control Operations	Allowed in CodeSafe 5 since v13.4
9	link	File System: File Links and Symlinks	Allowed in CodeSafe 5 since v13.4
10	unlink	File System: File Links and Symlinks	Allowed in CodeSafe 5 since v13.4
11	execve	Process Management: Process Creation & Termination	Allowed in CodeSafe 5 since v13.4
12	chdir	File System: Directory Operations	Allowed in CodeSafe 5 since v13.4
13	time	Time & Timers: System Time Access & Adjustment	Allowed in CodeSafe 5 since v13.4
15	chmod	File System: Access Control and Permissions	Allowed in CodeSafe 5 since v13.4

Syscall Number	Syscall Name	Category	Comment
16	lchown	File System: Access Control and Permissions	Allowed in CodeSafe 5 since v13.7
19	lseek	File System: File Descriptor & Control Operations	Allowed in CodeSafe 5 since v13.4
20	getpid	Process Management: Process Lifecycle & Waiting	Allowed in CodeSafe 5 since v13.4
24	getuid	Resource Control & Permissions: User & Group Identifiers	Allowed in CodeSafe 5 since v13.4
27	alarm	Time & Timers: Timers & Sleep Functions	Allowed in CodeSafe 5 since v13.7
29	pause	Process Management: Process Lifecycle & Waiting	Allowed in CodeSafe 5 since v13.4
30	utime	File System: Metadata & Filesystem Attributes	Allowed in CodeSafe 5 since v13.7
33	access	File System: Access Control and Permissions	Allowed in CodeSafe 5 since v13.4
34	nice	Process Management: Scheduling & Prioritization	Allowed in CodeSafe 5 since v13.7 (to reduce priority only)
36	sync	File System: Filesystem Synchronization	Allowed in CodeSafe 5 since v13.4
37	kill	Process Management: Signal Handling	Allowed in CodeSafe 5 since v13.4
38	rename	File System: Directory Operations	Allowed in CodeSafe 5 since v13.4
39	mkdir	File System: Directory Operations	Allowed in CodeSafe 5 since v13.4
40	rmdir	File System: Directory Operations	Allowed in CodeSafe 5 since v13.4
41	dup	File System: File Descriptor & Control Operations	Allowed in CodeSafe 5 since v13.4
42	pipe	Inter-Process Communication: Shared Memory, Pipes & Events	Allowed in CodeSafe 5 since v13.4

Syscall Number	Syscall Name	Category	Comment
43	times	Resource Control & Permissions: Resource Limits	Allowed in CodeSafe 5 since v13.7
45	brk	Memory Management: Memory Management Operations	Allowed in CodeSafe 5 since v13.4
47	getgid	Resource Control & Permissions: User & Group Identifiers	Allowed in CodeSafe 5 since v13.4
49	geteuid	Resource Control & Permissions: User & Group Identifiers	Allowed in CodeSafe 5 since v13.4
50	getegid	Resource Control & Permissions: User & Group Identifiers	Allowed in CodeSafe 5 since v13.4
54	ioctl	Device Control: Character Device Operations	Allowed in CodeSafe 5 since v13.4
55	fcntl	File System: File Descriptor & Control Operations	Allowed in CodeSafe 5 since v13.4
57	setpgid	Resource Control & Permissions: Session & Group Management	Allowed in CodeSafe 5 since v13.7
60	umask	File System: Access Control and Permissions	Allowed in CodeSafe 5 since v13.4
63	dup2	File System: File Descriptor & Control Operations	Allowed in CodeSafe 5 since v13.4
64	getppid	Resource Control & Permissions: Session & Group Management	Allowed in CodeSafe 5 since v13.4
65	getpgrp	Resource Control & Permissions: Session & Group Management	Allowed in CodeSafe 5 since v13.4
66	setsid	Resource Control & Permissions: Session & Group Management	Allowed in CodeSafe 5 since v13.4
75	setrlimit	Resource Control & Permissions: Resource Limits	Allowed in CodeSafe 5 since v13.7 (to reduce resource limits only)

Syscall Number	Syscall Name	Category	Comment
77	getrusage	Resource Control & Permissions: Resource Limits	Allowed in CodeSafe 5 since v13.7
78	gettimeofday	Time & Timers: System Time Access & Adjustment	Allowed in CodeSafe 5 since v13.4
80	getgroups	Resource Control & Permissions: Session & Group Management	Allowed in CodeSafe 5 since v13.5
83	symlink	File System: File Links and Symlinks	Allowed in CodeSafe 5 since v13.4
85	readlink	File System: File Links and Symlinks	Allowed in CodeSafe 5 since v13.4
88	reboot	Process Management: Process Control	Allowed in CodeSafe 5 since v13.4
90	mmap	Memory Management: Memory Mapping	Allowed in CodeSafe 5 since v13.4
91	munmap	Memory Management: Memory Mapping	Allowed in CodeSafe 5 since v13.4
92	truncate	File System: Metadata & Filesystem Attributes	Allowed in CodeSafe 5 since v13.7
93	ftruncate	File System: Metadata & Filesystem Attributes	Allowed in CodeSafe 5 since v13.7
94	fchmod	File System: Access Control and Permissions	Allowed in CodeSafe 5 since v13.4
95	fchown	File System: Access Control and Permissions	Allowed in CodeSafe 5 since v13.7
96	getpriority	Process Management: Scheduling & Prioritization	Allowed in CodeSafe 5 since v13.5
97	setpriority	Process Management: Scheduling & Prioritization	Allowed in CodeSafe 5 since v13.7 (to reduce priority only)
99	statfs	File System: Metadata & Filesystem Attributes	Allowed in CodeSafe 5 since v13.4
100	fstatfs	File System: Metadata & Filesystem Attributes	Allowed in CodeSafe 5 since v13.7

Syscall Number	Syscall Name	Category	Comment
102	socketcall	Networking & Socket Operations: Socket Management & Data Transfer	Allowed in CodeSafe 5 since v13.4
104	setitimer	Time & Timers: Timers & Sleep Functions	Allowed in CodeSafe 5 since v13.7
105	getitimer	Time & Timers: Timers & Sleep Functions	Allowed in CodeSafe 5 since v13.7
106	stat	File System: Metadata & Filesystem Attributes	Allowed in CodeSafe 5 since v13.4
107	lstat	File System: Metadata & Filesystem Attributes	Allowed in CodeSafe 5 since v13.4
108	fstat	File System: Metadata & Filesystem Attributes	Allowed in CodeSafe 5 since v13.4
114	wait4	Process Management: Process Creation & Termination	Allowed in CodeSafe 5 since v13.4
117	ipc	Inter-Process Communication: Compatibility Interfaces	Allowed in CodeSafe 5 since v13.5
118	fsync	File System: Filesystem Synchronization	Allowed in CodeSafe 5 since v13.7
120	clone	Process Management: Process Creation & Termination	Allowed in CodeSafe 5 since v13.4
122	uname	Miscellaneous System Functions: Utility & System Info	Allowed in CodeSafe 5 since v13.4
125	mprotect	Memory Management: Memory Mapping	Allowed in CodeSafe 5 since v13.4
132	getpgid	Resource Control & Permissions: Session & Group Management	Allowed in CodeSafe 5 since v13.7
133	fchdir	File System: Directory Operations	Allowed in CodeSafe 5 since v13.7
140	_llseek	File System: File Descriptor & Control Operations	Allowed in CodeSafe 5 since v13.4

Syscall Number	Syscall Name	Category	Comment
141	getdents	File System: Directory Operations	Allowed in CodeSafe 5 since v13.4
142	_newselect	Networking & Socket Operations: I/O Multiplexing & Readiness	Allowed in CodeSafe 5 since v13.7
143	flock	File System: File Descriptor & Control Operations	Allowed in CodeSafe 5 since v13.7
144	msync	Memory Management: Memory Mapping	Allowed in CodeSafe 5 since v13.7
145	readv	File System: Read/Write Operations	Allowed in CodeSafe 5 since v13.4
146	writev	File System: Read/Write Operations	Allowed in CodeSafe 5 since v13.4
147	getsid	Resource Control & Permissions: Session & Group Management	Allowed in CodeSafe 5 since v13.7
148	fdatasync	File System: Filesystem Synchronization	Allowed in CodeSafe 5 since v13.7
158	sched_yield	Process Management: Scheduling & Prioritization	Allowed in CodeSafe 5 since v13.7
162	nanosleep	Time & Timers: Timers & Sleep Functions	Allowed in CodeSafe 5 since v13.4
163	mremap	Memory Management: Memory Mapping	Allowed in CodeSafe 5 since v13.4
167	poll	Networking & Socket Operations: I/O Multiplexing & Readiness	Allowed in CodeSafe 5 since v13.4
172	rt_sigreturn	Process Management: Signal Handling	Allowed in CodeSafe 5 since v13.4
173	rt_sigaction	Process Management: Signal Handling	Allowed in CodeSafe 5 since v13.4
174	rt_sigprocmask	Process Management: Signal Handling	Allowed in CodeSafe 5 since v13.4
175	rt_sigpending	Process Management: Signal Handling	Allowed in CodeSafe 5 since v13.4
176	rt_sigtimedwait	Process Management: Signal Handling	Allowed in CodeSafe 5 since v13.4

Syscall Number	Syscall Name	Category	Comment
177	rt_sigqueueinfo	Process Management: Signal Handling	Allowed in CodeSafe 5 since v13.4
178	rt_sigsuspend	Process Management: Signal Handling	Allowed in CodeSafe 5 since v13.4
179	pread64	File System: Read/Write Operations	Allowed in CodeSafe 5 since v13.4
180	pwrite64	File System: Read/Write Operations	Allowed in CodeSafe 5 since v13.7
181	chown	File System: Access Control and Permissions	Allowed in CodeSafe 5 since v13.4
182	getcwd	File System: Directory Operations	Allowed in CodeSafe 5 since v13.4
185	sigaltstack	Process Management: Signal Handling	Allowed in CodeSafe 5 since v13.4
190	ugetrlimit	Resource Control & Permissions: Resource Limits	Allowed in CodeSafe 5 since v13.4
202	getdents64	File System: Directory Operations	Allowed in CodeSafe 5 since v13.4
205	madvise	Memory Management: Memory Mapping	Allowed in CodeSafe 5 since v13.4
207	gettid	Process Management: Process Lifecycle & Waiting	Allowed in CodeSafe 5 since v13.4
208	tkill	Process Management: Signal Handling	Allowed in CodeSafe 5 since v13.5
221	futex	Process Management: Synchronization	Allowed in CodeSafe 5 since v13.4
232	set_tid_address	Process Management: Thread/Process Context Operations	Allowed in CodeSafe 5 since v13.4
234	exit_group	Process Management: Process Creation & Termination	Allowed in CodeSafe 5 since v13.4
236	epoll_create	File System: File Descriptor & Control Operations	Allowed in CodeSafe 5 since v13.7
237	epoll_ctl	File System: File Descriptor & Control Operations	Allowed in CodeSafe 5 since v13.7

Syscall Number	Syscall Name	Category	Comment
238	epoll_wait	File System: File Descriptor & Control Operations	Allowed in CodeSafe 5 since v13.7
246	clock_gettime	Time & Timers: Clock Management & Resolution	Allowed in CodeSafe 5 since v13.4
247	clock_getres	Time & Timers: Clock Management & Resolution	Allowed in CodeSafe 5 since v13.7
248	clock_nanosleep	Time & Timers: Clock Management & Resolution	Allowed in CodeSafe 5 since v13.7
250	tgkill	Process Management: Signal Handling	Allowed in CodeSafe 5 since v13.4
251	utimes	File System: Metadata & Filesystem Attributes	Allowed in CodeSafe 5 since v13.7
252	statfs64	File System: Metadata & Filesystem Attributes	Allowed in CodeSafe 5 since v13.4
253	fstatfs64	File System: Metadata & Filesystem Attributes	Allowed in CodeSafe 5 since v13.7
272	waitid	Process Management: Process Creation & Termination	Allowed in CodeSafe 5 since v13.7
280	pselect6	Networking & Socket Operations: I/O Multiplexing & Readiness	Allowed in CodeSafe 5 since v13.7
281	ppoll	Networking & Socket Operations: I/O Multiplexing & Readiness	Allowed in CodeSafe 5 since v13.4
286	openat	File System: Metadata & Filesystem Attributes	Allowed in CodeSafe 5 since v13.4
287	mkdirat	File System: Metadata & Filesystem Attributes	Allowed in CodeSafe 5 since v13.7
289	fchownat	File System: Metadata & Filesystem Attributes	Allowed in CodeSafe 5 since v13.7
291	newfstatat	File System: Metadata & Filesystem Attributes	Allowed in CodeSafe 5 since v13.7
292	unlinkat	File System: Metadata & Filesystem Attributes	Allowed in CodeSafe 5 since v13.7
293	renameat	File System: Metadata & Filesystem Attributes	Allowed in CodeSafe 5 since v13.7

Syscall Number	Syscall Name	Category	Comment
294	linkat	File System: Metadata & Filesystem Attributes	Allowed in CodeSafe 5 since v13.7
295	symlinkat	File System: Metadata & Filesystem Attributes	Allowed in CodeSafe 5 since v13.7
296	readlinkat	File System: Metadata & Filesystem Attributes	Allowed in CodeSafe 5 since v13.7
297	fchmodat	File System: Metadata & Filesystem Attributes	Allowed in CodeSafe 5 since v13.7
298	faccessat	File System: File Descriptor & Control Operations	Allowed in CodeSafe 5 since v13.7
303	epoll_pwait	File System: File Descriptor & Control Operations	Allowed in CodeSafe 5 since v13.7
304	utimensat	File System: Metadata & Filesystem Attributes	Allowed in CodeSafe 5 since v13.5
307	eventfd	Inter-Process Communication: Shared Memory, Pipes & Events	Allowed in CodeSafe 5 since v13.7
309	fallocate	File System: File Descriptor & Control Operations	Allowed in CodeSafe 5 since v13.7
315	epoll_create1	File System: File Descriptor & Control Operations	Allowed in CodeSafe 5 since v13.5
316	dup3	File System: File Descriptor & Control Operations	Allowed in CodeSafe 5 since v13.7
317	pipe2	Inter-Process Communication: Shared Memory, Pipes & Events	Allowed in CodeSafe 5 since v13.7
320	preadv	File System: Read/Write Operations	Allowed in CodeSafe 5 since v13.7
321	pwritev	File System: Read/Write Operations	Allowed in CodeSafe 5 since v13.7
322	rt_tgsigqueueinfo	Process Management: Signal Handling	Allowed in CodeSafe 5 since v13.7
325	prlimit64	Resource Control & Permissions: Resource Limits	Allowed in CodeSafe 5 since v13.7 (to reduce resource limits only)

Syscall Number	Syscall Name	Category	Comment
326	socket	Networking & Socket Operations: Socket Management & Data Transfer	Allowed in CodeSafe 5 since v13.4
327	bind	Networking & Socket Operations: Socket Management & Data Transfer	Allowed in CodeSafe 5 since v13.4
328	connect	Networking & Socket Operations: Socket Management & Data Transfer	Allowed in CodeSafe 5 since v13.4
329	listen	Networking & Socket Operations: Socket Management & Data Transfer	Allowed in CodeSafe 5 since v13.4
330	accept	Networking & Socket Operations: Socket Management & Data Transfer	Allowed in CodeSafe 5 since v13.4
331	getsockname	Networking & Socket Operations: Socket Management & Data Transfer	Allowed in CodeSafe 5 since v13.4
332	getpeername	Networking & Socket Operations: Socket Management & Data Transfer	Allowed in CodeSafe 5 since v13.4
333	socketpair	Networking & Socket Operations: Socket Management & Data Transfer	Allowed in CodeSafe 5 since v13.4
334	send	Networking & Socket Operations: Socket Management & Data Transfer	Allowed in CodeSafe 5 since v13.4
335	sendto	Networking & Socket Operations: Socket Management & Data Transfer	Allowed in CodeSafe 5 since v13.4
336	recv	Networking & Socket Operations: Socket Management & Data Transfer	Allowed in CodeSafe 5 since v13.4
337	recvfrom	Networking & Socket Operations: Socket Management & Data Transfer	Allowed in CodeSafe 5 since v13.4
338	shutdown	Networking & Socket Operations: Socket Management & Data Transfer	Allowed in CodeSafe 5 since v13.4

Syscall Number	Syscall Name	Category	Comment
339	setsockopt	Networking & Socket Operations: Socket Management & Data Transfer	Allowed in CodeSafe 5 since v13.4
340	getsockopt	Networking & Socket Operations: Socket Management & Data Transfer	Allowed in CodeSafe 5 since v13.4
341	sendmsg	Networking & Socket Operations: Socket Management & Data Transfer	Allowed in CodeSafe 5 since v13.4
342	recvmsg	Networking & Socket Operations: Socket Management & Data Transfer	Allowed in CodeSafe 5 since v13.4
343	recvmmsg	Networking & Socket Operations: Socket Management & Data Transfer	Allowed in CodeSafe 5 since v13.4
344	accept4	Networking & Socket Operations: Socket Management & Data Transfer	Allowed in CodeSafe 5 since v13.4
348	syncfs	File System: Filesystem Synchronization	Allowed in CodeSafe 5 since v13.7
349	sendmmsg	Networking & Socket Operations: Socket Management & Data Transfer	Allowed in CodeSafe 5 since v13.4
357	renameat2	File System: Metadata & Filesystem Attributes	Allowed in CodeSafe 5 since v13.7
362	execveat	Process Management: Process Creation & Termination	Allowed in CodeSafe 5 since v13.7
365	membarrier	Process Management: Synchronization	Allowed in CodeSafe 5 since v13.4
380	preadv2	File System: Read/Write Operations	Allowed in CodeSafe 5 since v13.7
381	pwritev2	File System: Read/Write Operations	Allowed in CodeSafe 5 since v13.7
383	statx	File System: Metadata & Filesystem Attributes	Allowed in CodeSafe 5 since v13.7