F5 NGINX Server

nShield® HSM Integration Guide - PKCS #11

2026-04-01

Table of Contents

Chapter 1. Introduction

You can integrate the Entrust nShield HSMs with NGINX to generate 2048-bit RSA key pairs for SSL and protect the private keys within a FIPS 140 certified Hardware Security Module (HSM). This integration uses the PKCS #11 interface to integrate the HSM and NGINX Server.

The benefits of using an nShield HSM with the NGINX Server include:

- Secure storage of the private key.
- FIPS 140 Level 3 validated hardware.
- Improved server performance by offloading the cryptographic processing.
- Full life cycle management of the keys.
- Failover support.
- Load balancing between HSMs.

1.1. Product configurations

Entrust tested nShield HSM integration with the NGINX server in the following configurations:

Product	Version
Operating System	Red Hat Enterprise Linux 9
F5 NGINX Plus	nginx version: nginx/1.29.3 (nginx-plus-r36-p3)
NGINX Open Source	nginx version: nginx/1.20.1
Security World	13.6.15
OpenSSL	OpenSSL 3.5.1 1 Jul 2025
OpenSSL PKCS #11	openssl-pkcs11-0.4.11-9

1.1.1. Supported nShield features

Entrust tested nShield HSM integration with the following features:

Feature	Support
Softcards	Yes

Feature	Support
Module-only key	Yes
OCS cards	Yes
nSaaS	Yes

1.1.2. Supported nShield hardware and software versions

Entrust tested with the following nShield hardware and software versions:

HSM	Security World Software	Firmware	Image
Connect XC	13.6.15	12.72.4 (FIPS 140-2 certified)	13.6.15
nShield 5c	13.6.15	13.4.5 (FIPS 140-3 certified)	13.6.15

1.2. Requirements

Ensure that you have supported versions of the Entrust, NGINX, and third-party products.

Consult the security team in your organization for a suitable setting of the following:

- The SE Linux policy to allow the web server read access to the files in `/opt/nfast`.
- The firewall.

To perform the integration tasks, you must have:

- `root` access on the operating system.
- Access to `nfast`.

Before starting the integration process, familiarize yourself with:

- The documentation for the HSM.
- The documentation and setup process for the NGINX Server.

Before using the nShield software, you need to know:

- The number and quorum of Administrator cards in the Administrator Card Set (ACS) and the policy for managing these cards.
- Whether the application keys are protected by the module, an Operator Card Set (OCS) or a Softcard with or without a pass phrase.

- The number and quorum of Operator cards in the OCS and the policy for managing these cards.
- Whether the Security World should be compliant with FIPS 140 Level 3.

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

For more information, refer to the *User Guide* and *Installation Guide* for the HSM.

1.3. More information

For more information about OS support, contact your NGINX Server sales representative or Entrust nShield Support, https://nshieldsupport.entrust.com.

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

Chapter 2. Procedures

2.1. Install the NGINX Server

2.1.1. F5 NGINX Plus

See Installing NGINX Plus for detailed instructions on how to install NGINX Plus.

2.1.2. NGINX Open Source

See Setting up and Configuring NGINX for detailed instructions on how to install NGINX Open Source.

2.2. Configure the NGINX server

1. Open the firewall. An active firewall might prevent NGINX from loading.

```
% sudo firewall-cmd --zone=public --permanent --add-service=http
% sudo firewall-cmd --zone=public --permanent --add-service=https
% sudo firewall-cmd --reload
```

2. Switch off SE Linux. If SE Linux is active, this might prevent NGINX from loading.

```
% sudo setenforce 0
```

3. Enable the NGINX service to start at boot:

```
% sudo systemctl enable nginx.service
```

4. Install the OpenSSL packages. These packages are needed to configure OpenSSL and to use PKCS11 libraries.

```
% sudo yum install -y opensc openssl-pkcs11 gnutls-utils nano openssl-libs
```

5. Restart the NGINX service:

```
% sudo systemctl restart nginx
```

6. Check if NGINX is running by opening the browser at `http://<your-ip-address>`.

2.3. Install the HSM

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

Entrust recommends that you install the HSM before configuring the Security World software with your NGINX Server.

2.4. Install the Security World software and create a Security World

1. On the computer running the NGINX Server, install the latest version of the Security World software as described in the *Installation Guide* for the HSM.

   Entrust recommends that you uninstall any existing nShield software before installing the new nShield software.

2. Create the Security World as described in the *User Guide*, creating the ACS and OCS that you require.

2.5. Set up the PKCS #11 engine

To avoid problems associated with the Entrust-supplied OpenSSL, which is used internally by `generatekey` to make certificates, ensure that `/opt/nfast/bin` is not at the front of your `$PATH`.

You can confirm that the right binary is being run with the following command:

```
% which openssl

/usr/bin/openssl
```

If this command returns output inside `/opt/nfast`, check your `$PATH` variable.

2.5.1. Configure OpenSSL

1. Find out where your OpenSSL configuration file is located:

```
% openssl version -d

OPENSSLDIR: "/etc/pki/tls"
```

   On this integration we will use the default OpenSSL configuration file and we will also create a OpenSSL configuration file to be used when PKCS #11 operations are needed. Here is the minimum configuration for the OpenSSL PKCS #11 configuration file. Adjust it to your organization's values.

```
HOME = .

openssl_conf = openssl_def

[openssl_def]
engines = engine_section

[engine_section]
pkcs11 = pkcs11_section

[pkcs11_section]
engine_id = pkcs11
dynamic_path = /usr/lib64/engines-3/pkcs11.so
MODULE_PATH = /opt/nfast/toolkits/pkcs11/libcknfast.so
init = 0

[req]
distinguished_name = req_distinguished_name
req_extensions = v3_req
prompt = no

[req_distinguished_name]
C = US
ST = FL
L = Sunrise
O = Entrust
OU = nShield
CN = localhost

[v3_req]
subjectAltName = @alt_names
extendedKeyUsage = clientAuth, serverAuth

[alt_names]
DNS.1 = www.entrust.com
DNS.2 = entrust.com
IP.1 = xxx.xxx.xxx.xxx
IP.2 = xxx.xxx.xxx.xxx
```

   The `dynamic_path` may be different for different distributions.

2. Make sure the server's hostname matches the CN in the certificate.
3. Create the OpenSSL PKCS #11 configuration file and call it `openssl.pkcs11.cnf` with the settings above. Save it where your OpenSSL configuration settings are located.
4. Create or edit the file `/etc/pki/tls/openssl.pkcs11.cnf` and enter the settings above:

```
% sudo vi /etc/pki/tls/openssl.pkcs11.cnf
```

You only want to use this file when `pkcs11` operations are required.

2.5.2. Set up /opt/nfast/cknfastrc

1. Add the following variables to the `/opt/nfast/cknfastrc` file. These are referenced in this guide to address certain situations and their use will depend on your current environment.

```
CKNFAST_DEBUG=10
CKNFAST_DEBUGFILE=/path/to/debug/file
CKNFAST_FAKE_ACCELERATOR_LOGIN=1
CKNFAST_LOADSHARING=1
```

2. Turn debug off in a production environment.

2.5.3. Test the configuration

1. Update OpenSSL so that it uses the new configuration file that you created. Export the `OPENSSL_CONF` environment variable:

```
% export OPENSSL_CONF=/etc/pki/tls/openssl.pkcs11.cnf
```

2. Test the configuration. The output should be similar to this:

```
% openssl engine -tt -c -v

(rdrand) Intel RDRAND engine
 [RAND]
     [ available ]
(dynamic) Dynamic engine loading support
     [ unavailable ]
     SO_PATH, NO_VCHECK, ID, LIST_ADD, DIR_LOAD, DIR_ADD, LOAD
(pkcs11) pkcs11 engine
 [RSA, rsaEncryption, id-ecPublicKey]
     [ available ]
     SO_PATH, MODULE_PATH, PIN, VERBOSE, QUIET, INIT_ARGS, FORCE_LOGIN
```

During testing you will see that we set and unset the `OPENSSL_CONF` environment variable. An issue was found with OPENSSL version 3 on RedHat 9 where the certificate file being generated during testing was empty. That issue was not observed on RedHat 8 which uses OpenSSL version 1. To get around the problem, we unset the `OPENSSL_CONF` environment variable so the default OpenSSL configuration file is used instead, when pkcs11 operations are not required.

2.5.4. Debug notes

1. Security World permissions:

   The following message indicates that there is no Security World.

   ```
   Unable to load module /opt/nfast/toolkits/pkcs11/libcknfast.so
   ```

   Make sure you create a Security World first.

2. Debug variables:

   You can set the following debug variables in `/opt/nfast/cknfastrc` or as environment variables.

   ```
   CKNFAST_DEBUG=10
   CKNFAST_DEBUGFILE=/path
   ```

3. Missing PKCS11 engine in the output:

   If you don't see the PKCS11 engine in the output, check the `dynamic_path` line in the `openssl.pkcs11.cnf` configuration file. It may be different on other platforms and other operating system versions.

   ```
   dynamic_path = /usr/lib64/engines-3/pkcs11.so
   ```

2.6. Configure the NGINX Server to use the PKCS11 engine

You need to update the NGINX startup file to use the new Open SSL configuration file. Update the NGINX service startup file to pass the necessary environment variables. These environment variables allow PKCS11 engine to work.

1. Edit `/usr/lib/systemd/system/nginx.service` and add the environment variables under the `Service` section:

   ```
   [Service]
   Environment=LANG=C
   Environment="OPENSSL_CONF=/etc/pki/tls/openssl.pkcs11.cnf"
   Environment="NFAST_NFKM_TOKENSFILE=<path-to-preload-file>"
   ```

   Where `<path-to-preload-file>` is the location of the `preload` file.

   

   You must ensure that the location of the `preload` file has the appropriate read-access group permissions so that only the

> intended application and the permitted administrators, who will load the softcard, can access it. The location must not be world-readable, otherwise any user could access the softcard. It is not recommended to use `kmdata/local` for this, because you would need to restrict all of `kmdata` to protect it.

Notice the `OPENSSL_CONF` variable. It points to the OpenSSL pkcss11 config file.

2. With Softcard and OCS protection, the usual arrangement of spawning worker processes requires preloading the Softcard or the OCS card. You must specify a `preload` file and define its location in the environment to give the other processes access to the key. No pin value is used in the configuration file, but you can include a fake one to avoid typing one in on start-up. For the master process you must ensure the variable is set in the system or session from which the master process is launched. For worker processes, you must specify the variable in the NGINX config file.
3. Restart the daemon units:

```
% sudo systemctl daemon-reload
```

4. Edit `/etc/nginx/nginx.conf` so that it uses the PKCS11 engine.
   a. For Softcard or OCS protection, add the following line after the `pid` line to expose `tokensfile` to the worker processes:

   ```
   env NFAST_NFKM_TOKENSFILE=<path-to-preload-file>;
   ```

   b. Add the PKCS11 engine after the `Events` section:

   ```
   ssl_engine pkcs11;
   ```

   c. If it is not in the `http` section, before the end of the section, add the following line:

   ```
   include /etc/nginx/conf.d/*.conf;
   ```

   d. Example `nginx.conf` file:

   ```
   user  nginx;
   worker_processes  auto;

   error_log  /var/log/nginx/error.log notice;
   pid        /var/run/nginx.pid;
   env NFAST_NFKM_TOKENSFILE=<path-to-preload-file>;


   events {
       worker_connections  1024;
   ```

```
}

ssl_engine pkcs11;

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

    include /etc/nginx/conf.d/*.conf;
}
```

5. Create a `https.conf` file in `/etc/nginx/conf.d` folder. Include the following content with all lines commented out:

```
#server {
#    listen 443 ssl;
#
#    ssl_certificate     /etc/nginx/ssl/test.crt;
#    ssl_certificate_key /etc/nginx/ssl/test.key;
#
#    ssl_client_certificate /etc/pki/tls/misc/ca.crt;
#    ssl_verify_client on;
#
#
#    ssl_protocols       TLSv1 TLSv1.1 TLSv1.2;
#
#    location / {
#        root   /usr/share/nginx/html;
#        index  index.html index.htm;
#    }
#}
```

The `ssl_client_certificate` and `ssl_verify_client` lines should only be used if the server will be configured for mTLS. Comment out these lines if you are not using mTLS.

1. Restart the NGINX service:

```
% sudo systemctl restart nginx
```

2.7. Test the PKCS #11 integration with the NGINX Server and the HSM

Your organization can use the following scenarios, according to the security guidelines that you follow:

- Module-only protection.
- Softcard protection.
- OCS protection.

If mTLS is used in the configuration, create a CA certificate and a client certificate.

A self-signed certificate is used for tests in this guide. In a production environment exposed to the internet, create the certificate request and sign it by the Trusted Certificate Authority.

1. Generate the CA certificate key:

```
% openssl genpkey -algorithm RSA -out ./ca.key
```

2. Generate the CA certificate:

```
% openssl req -new -x509 -key ./ca.key -out ./ca.crt -subj
"/C=US/ST=Florida/L=Sunrise/O=Entrust/OU=nShield/CN=ROOT-CA"
```

3. Copy the `ca.crt` file to the location specified in the `https.conf` file on the `ssl_client_certificate` line.

```
% sudo cp ca.crt /etc/pki/tls/misc/ca.crt
```

4. Generate a client certificate key:

```
% openssl genpkey -algorithm RSA -out ./client.key
```

5. Generate the client certificate CSR:

```
% openssl req -new -key ./client.key -out ./client.csr -subj
"/C=US/ST=Florida/L=Sunrise/O=Entrust/OU=nShield/CN=CLIENT-CERT"
```

6. Generate the client certificate:

```
% openssl x509 -req -in ./client.csr -CA ./ca.crt -CAkey ./ca.key -CAcreateserial -out ./client.crt
```

7. Testing the connection:

Here is how you test the connection with the NGINX server after all is configured. Do not do this now. You have to do this once everything is setup for each type of protection in this guide.

With mTLS:

```
% openssl s_client -connect localhost:443 -CAfile ./ca.crt -key ./client.key -cert ./client.crt
```

You also can use the `curl` command to test the connection with mTLS:

```
% curl --cert ./client.crt --key ./client.key --cacert ./ca.crt https://localhost:443
```

Without mTLS:

```
% openssl s_client -crlf -connect localhost:443 -CAfile pkcs11localhost.crt
```

8. Connection output

The output should be something like this when connecting with OpenSSL.

```
CONNECTED(00000003)

Certificate chain
 0 s:CN=ocshsm1_110044
   i:CN=ocshsm1_110044
   a:PKEY: RSA, 2048 (bit); sigalg: sha256WithRSAEncryption
   v:NotBefore: Mar 26 15:00:50 2026 GMT; NotAfter: Mar 26 15:00:50 2027 GMT
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIDWzCCAkMCFAZbDVSvlbRy9ZVbWy123456789CMA0GCSqGSIb3DQEBCwUAMGox
....
AVh1aVUKvE2xFnt8rq7890123456IqMLDz7Ww0INt6yTrXlb9lcUsFdcOcvAnipk=
-----END CERTIFICATE-----
subject=CN=ocshsm1_110044
issuer=CN=ocshsm1_110044
---
No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: rsa_pss_rsae_sha256
Peer Temp Key: X25519, 253 bits
---
SSL handshake has read 1398 bytes and written 398 bytes
Verification: OK
---
New, TLSv1.2, Cipher is ECDHE-RSA-AES256-GCM-SHA384
Protocol: TLSv1.2
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES256-GCM-SHA384
```

```
    Session-ID: 6B6B916073E486FA1B96243445760CF4747417960BF2EA7872F2D3A7076BBF31
    Session-ID-ctx:
    Master-Key:
22CBE6E73D90312A2876CD504190D242C2428FB75E54167D7A9BCACE8DA4F293490A5C8085A976B7DA8E04600E901BCC
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:
    0000 - 61 f3 13 8c 48 89 e9 6e-e7 bf b1 02 5c 8e 91 fb   a...H..n....\...
    0010 - 01 88 53 33 bf 93 08 11-e1 77 7d 43 53 83 60 45   ..S3.....w}CS.`E
    0020 - d3 7e ef 76 68 32 ad 58-6d b2 e7 98 d9 98 50 85   .~.vh2.Xm.....P.
    0030 - 71 06 bc 66 0c 71 c0 ec-db 48 bd c1 f0 57 3d c1   q..f.q...H...W=.
    0040 - 1f ac 5d 37 47 c0 44 67-91 91 f9 80 f0 34 0c bd   ..]7G.Dg.....4..
    0050 - 7c 4c 39 61 80 ad cb ca-e7 95 f7 b7 37 7a 09 96   |L9a........7z..
    0060 - 61 1f f1 e4 ec 43 af a3-63 4b 19 e5 61 3d d1 5f   a....C..cK..a=._
    0070 - b1 ee 1d b1 71 2f 0c ce-f6 7c 6b 46 78 33 bd 59   ....q/...|kFx3.Y
    0080 - e5 90 68 48 f1 99 03 fa-81 e8 88 87 41 cf 21 9c   ..hH........A.!.
    0090 - 4c 88 ed c4 c8 c6 d8 71-ec 5d 97 20 6c f6 3d 9e   L......q.]. l.=.
    00a0 - ca aa 8a 39 81 e7 56 e7-8e 62 e3 10 44 65 1a 05   ...9..V..b..De..
    00b0 - ce 6b 26 8b f1 a9 89 b8-b3 1a 5b 43 6a 5a 8b 5c   .k&.......[CjZ.\

    Start Time: 1774537254
    Timeout   : 7200 (sec)
    Verify return code: 0 (ok)
    Extended master secret: yes
```

9. Check the following messages and fields in the output:
    - CONNECTED(00000003)
    - depth
    - Certificate chain information
    - Server certificate information
    - Session-ID
    - Master-Key
    - TLS session ticket:
    - Verify return code: 0 (ok)

2.7.1. Module protection

1. Remove the `preload` file if it exists:

```
% sudo rm -f <path-to-preload-file>
```

2. To allow module protection, set the `cknfast` library to allow access to the module (`CKNFAST_FAKE_ACCELERATOR_LOGIN`).

   Edit the `/opt/nfast/cknfastrc` file and add the following information before proceeding to set up module protection:

```
CKNFAST_FAKE_ACCELERATOR_LOGIN=1
```

3. Create a key:

```
% generatekey -b -g -m1 pkcs11 plainname=modulersa type=rsa protect=module size=2048

key generation parameters:
 operation    Operation to perform              generate
 application  Application                       pkcs11
 verify       Verify security of key            yes
 type         Key type                          rsa
 size         Key size                          2048
 pubexp       Public exponent for RSA key (hex)
 plainname    Key name                          modulersa
 nvram        Blob in NVRAM (needs ACS)         no

Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_uacf07dbd534d0b1973377585e07fe54c91d95b5f6
```

4. Set the environment variable so that OpenSSL commands use the PKCS11 engine:

```
% export OPENSSL_CONF=/etc/pki/tls/openssl.pkcs11.cnf
```

5. Get the certificate using this key:

   With mTLS:

```
% openssl req -new -engine pkcs11 -keyform engine -key "pkcs11:token=accelerator;object=modulersa" -out
modulersa.csr
% unset OPENSS_CONF
% openssl x509 -req -in modulersa.csr -CA ./ca.crt -CAkey ./ca.key -CAcreateserial -out modulersa.pem
```

   Without mTLS:

```
% openssl req -engine pkcs11 -x509 -out modulersa.pem -days 365 -key
"pkcs11:token=accelerator;object=modulersa" -keyform engine -subj "/CN=modulersa"
% unset OPENSS_CONF
```

   If you get the following error, you probably have `CKNFAST_LOADSHARING=1` set in `/opt/nfast/cknfastrc`. Comment it out and try again.

```
engine "pkcs11" set.
Specified object not found
Specified object not found
PKCS11_get_private_key returned NULL
cannot load Private Key from engine
140640559179584:error:80067065:pkcs11 engine:ctx_load_privkey:object not found:eng_back.c:975:
140640559179584:error:26096080:engine routines:ENGINE_load_private_key:failed loading private
key:crypto/engine/eng_pkey.c:78:
unable to load Private Key
```

6. Configure the NGINX Server for SSL:
    a. Copy the `.pem` file:

    ```
    % sudo cp modulersa.pem /etc/pki/tls/certs/.
    ```

    b. Edit `/etc/httpd/conf.d/https.conf` and change the following lines to use the new `.key` and `.pem` files.

    Enable the SSL settings by uncommenting the server section if it is still commented out.

    ```
    ssl_certificate /etc/pki/tls/certs/modulersa.pem;
    ssl_certificate_key "engine:pkcs11:pkcs11:object=modulersa;token=accelerator";
    ```

    c. If you are using mTLS in the configuration, add the `ssl_client_certificate` and `ssl_verify_client` lines to `/etc/httpd/conf.d/https.conf`. If you are not using mTLS, remove or comment out these lines.

    d. Restart the NGINX service:

    ```
    % sudo systemctl restart nginx
    ```

7. Test the connections:

    With mTLS:

    ```
    % openssl s_client -connect localhost:443 -CAfile ./ca.crt -key ./client.key -cert ./client.crt
    ```

    You also can use the `curl` command to test the connection with mTLS:

    ```
    % curl --cert ./client.crt --key ./client.key --cacert ./ca.crt https://localhost:443
    ```

    Without mTLS:

    ```
    % openssl s_client -crlf -connect localhost:443 -CAfile modulersa.pem
    ```

8. Check the following messages and fields in the output:
    - CONNECTED(00000003)
    - depth
    - Certificate chain information
    - Server certificate information

- Session-ID
- Master-Key
- TLS session ticket:
- Verify return code: 0 (ok)

2.7.2. Set up Softcard protection

1. Remove the `preload` file if it exists:

```
% sudo rm -f <path-to-preload-file>
```

2. To expose Softcards, set the `cknfast` library to load sharing mode (`CKNFAST_LOADSHARING`).

   Edit the `/opt/nfast/cknfastrc` file and add the following information before proceeding to set up Softcard protection:

```
CKNFAST_LOADSHARING=1
```

3. Create a Softcard:

```
% ppmk -n softcardhsm_1

Enter new pass phrase:
Enter new pass phrase again:
New softcard created: HKLTU 541c437751f2b296f5733bd326e5c116435cb814
```

   123456 is the passphrase for the Softcard in the example.

4. Create a key:

```
% generatekey -b -g -m1 pkcs11 plainname=softcardhsm1_170047 type=rsa protect=softcard recovery=no
size=2048 softcard=softcardhsm_1

key generation parameters:
 operation    Operation to perform              generate
 application  Application                       pkcs11
 protect      Protected by                      softcard
 softcard     Soft card to protect key          softcardhsm_1
 recovery     Key recovery                      no
 verify       Verify security of key            yes
 type         Key type                          rsa
 size         Key size                          2048
 pubexp       Public exponent for RSA key (hex)
 plainname    Key name                          softcardhsm1_170047
 nvram        Blob in NVRAM (needs ACS)         no
Please enter the pass phrase for softcard `softcardhsm_1':

Please wait........
```

```
Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_uc415a6f3e010e0a4a9a7f8869eb2ac70210a54f2b-
25143883fd360f7aa24bc7a750f7fab0ebb38160
```

5. Set the environment variable so that OpenSSL commands use the PKCS11 engine:

```
% export OPENSSL_CONF=/etc/pki/tls/openssl.pkcs11.cnf
```

6. Get the certificate using this key:

   With mTLS:

```
% openssl req -new -engine pkcs11 -keyform engine -key "pkcs11:model=;token=softcardhsm_1;pin-
value=123456;object=softcardhsm1_170047" -out softcardhsm1_170047.csr
% unset OPENSSL_CONF
% openssl x509 -req -in softcardhsm1_170047.csr -CA ./ca.crt -CAkey ./ca.key -CAcreateserial -out
softcardhsm1_170047.pem
```

   Without mTLS:

```
% openssl req -engine pkcs11 -x509 -out softcardhsm1_170047.pem -days 365 -key
pkcs11:model=;token=softcardhsm_1;pin-value=123456;object=softcardhsm1_170047 -keyform ENGINE -subj
/CN=softcardhsm1_170047
% unset OPENSSL_CONF
```

   If you get an `ENGINE_load_private_key` error:

```
engine "pkcs11" set.
Specified object not found
PKCS11_get_private_key returned NULL
cannot load Private Key from engine
139939575797568:error:80067065:pkcs11 engine:ctx_load_privkey:object not found:eng_back.c:975:
139939575797568:error:26096080:engine routines:ENGINE_load_private_key:failed loading private
key:crypto/engine/eng_pkey.c:78:
```

   Make sure you expose the Softcards as described in this section and run the command again.

7. Configure the NGINX Server for SSL.
   a. Copy the `.pem` file:

```
% sudo cp softcardhsm1_170047.pem /etc/pki/tls/certs/.
```

   b. Edit `/etc/httpd/conf.d/https.conf` and change the following lines to use the new `.key` and `pem` files.

      Enable the SSL settings by uncommenting the server section if it is it still commented out:

```
ssl_certificate /etc/pki/tls/certs/softcardhsm1_170047.pem;
ssl_certificate_key "engine:pkcs11:pkcs11:object=softcardhsm1_170047;token=softcardhsm_1;pin-
value=123456";
```

c. If you are using mTLS in the configuration, add the `ssl_client_certificate` and `ssl_verify_client` lines to `/etc/httpd/conf.d/https.conf`. If you are not using mTLS, remove or comment out these lines.

d. Restart the NGINX service:

```
% sudo systemctl stop nginx
% preload --preload-file <path-to-preload-file> softcardhsm_1 sudo systemctl start nginx
```

If you don't restart NGINX by executing `preload` first, you get an error like this and the certificate doesn't load:

```
CONNECTED(00000003)
Can't use SSL_get_servername
...
No client certificate CA names sent
...
```

8. With Softcard and OCS protection, the usual arrangement of spawning worker processes requires preloading the Softcard or the OCS card. Specify a `preload` file and define its location in the environment to give the other processes access to the key (see the note in Configure the NGINX Server to use the PKCS11 engine). No pin value is used in the configuration file, but you can include a fake one to avoid typing one in on start-up. For the master process you must ensure the variable is set in the system or session from which the master process is launched. For worker processes, specify the variable in the NGINX config file.

```
% grep NFAST_NFKM_TOKENSFILE /usr/lib/systemd/system/nginx.service

Environment="NFAST_NFKM_TOKENSFILE=<path-to-preload-file>"
```

```
% grep NFAST_NFKM_TOKENSFILE /etc/nginx/nginx.conf

env NFAST_NFKM_TOKENSFILE=<path-to-preload-file>;
```

```
% grep ssl_certificat_key /etc/nginx/conf.d/https.conf

ssl_certificate_key "engine:pkcs11:pkcs11:object=softcardhsm1_170047;token=softcardhsm_1;pin-value=123456";
```

9. Test the connections:

With mTLS:

```
% openssl s_client -connect localhost:443 -CAfile ./ca.crt -key ./client.key -cert ./client.crt
```

You also can use the `curl` command to test the connection with mTLS:

```
% curl --cert ./client.crt --key ./client.key --cacert ./ca.crt https://localhost:443
```

Without mTLS:

```
% openssl s_client -crlf -connect localhost:443 -CAfile softcardhsm1_170047.pem
```

10. Check the following messages and fields in the output:
    - CONNECTED(00000003)
    - depth
    - Certificate chain information
    - Server certificate information
    - Session-ID
    - Master-Key
    - TLS session ticket:
    - Verify return code: 0 (ok)

2.7.3. Set up OCS protection

1. Remove the `preload` file if it exists:

```
% sudo rm -f <path-to-preload-file>
```

2. Create an OCS:

```
% /opt/nfast/bin/createocs -m1 -s2 --persist -Q 1/1 -N ocscard

FIPS 140-2 level 3 auth obtained.

Creating Cardset:
 Module 1: 0 cards of 1 written
 Module 1 slot 0: Admin Card #2
 Module 1 slot 3: inappropriate Operator Card (TokenAuthFailed)
 Module 1 slot 2: unknown card
 Module 1 slot 2:- passphrase specified - overwriting card
Card writing complete.

cardset created; hkltu = 454e988e226b33fa94087c0ee6112e0975c1557f
```

123456 is the passphrase for the OCS in the example.

3. Create a key:

```
% /opt/nfast/bin/generatekey --cardset=ocscard pkcs11 protect=token type=rsa size=2048 pubexp=65537
plainname=ocskey nvram=no recovery=yes

slot: Slot to read cards from? (0-3) [0] > 2
key generation parameters:
 operation    Operation to perform              generate
 application  Application                        pkcs11
 protect      Protected by                       token
 slot         Slot to read cards from            2
 recovery     Key recovery                       yes
 verify       Verify security of key             yes
 type         Key type                           rsa
 size         Key size                           2048
 pubexp       Public exponent for RSA key (hex)  65537
 plainname    Key name                           ocskey
 nvram        Blob in NVRAM (needs ACS)          no

Loading `ocscard':
 Module 1: 0 cards of 1 read
 Module 1 slot 2: `ocscard' #1
 Module 1 slot 0: Admin Card #2
 Module 1 slot 3: inappropriate Operator Card (TokenAuthFailed)
 Module 1 slot 2:- passphrase supplied - reading card
Card reading complete.

Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_uc454e988e226b33fa94087c0ee6112e0975c1557f-
bf7b5f0412619a354f86f58c77d796f27bd3ee12
```

4. Set the environment variable so that OpenSSL commands use the PKCS11 engine:

```
% export OPENSSL_CONF=/etc/pki/tls/openssl.pkcs11.cnf
```

5. Get the certificate using this key:

   With mTLS:

```
% openssl req -new -engine pkcs11 -keyform engine -key
"pkcs11:token=ocscard;object=ocskey;type=private?pin-value=123456" -out ocskey.csr
% unset OPENSSL_CONF
% openssl x509 -req -in ocskey.csr -CA ./ca.crt -CAkey ./ca.key -CAcreateserial -out ocskey.pem
```

   Without mTLS:

```
% openssl req -engine pkcs11 -x509 -out ocskey.pem -days 365 -key
"pkcs11:token=ocscard;object=ocskey;type=private?pin-value=123456" -keyform engine -subj "/CN=ocskey"
% unset OPENSSL_CONF
```

6. Configure the NGINX Server for SSL.

   a. Copy the `.pem` file:

```
% sudo cp ocskey.pem /etc/pki/tls/certs/.
```

b. Edit `/etc/httpd/conf.d/https.conf` and change the following lines to use the new `.key` and `.pem` files.

   Enable the SSL settings by uncommenting the `server` section if it is still commented out:

```
ssl_certificate /etc/pki/tls/certs/ocskey.pem;
ssl_certificate_key "engine:pkcs11:pkcs11:object=ocskey;token=ocscard;pin-value=123456";
```

c. If you are using mTLS in the configuration, add the `ssl_client_certificate` and `ssl_verify_client` lines to `/etc/httpd/conf.d/https.conf`. If you are not using mTLS, remove or comment out these lines.

d. Restart the NGINX service:

```
% sudo systemctl stop nginx
% preload --preload-file <path-to-preload-file> -c ocscard sudo systemctl start nginx

Mar 26 11:34:23 f5nginx-redhat-9 systemd[1]: Starting NGINX Plus - high performance web server...
Mar 26 11:34:23 f5nginx-redhat-9 systemd[1]: Started NGINX Plus - high performance web server.
```

7. With Softcard and OCS protection, the usual arrangement of spawning worker processes requires preloading the Softcard or the OCS card. Specify a `preload` file and define its location in the environment to give the other processes access to the key (see the note in Configure the NGINX Server to use the PKCS11 engine). No pin value is used in the configuration file, but you can include a fake one to avoid typing one in on start-up. For the master process you must ensure the variable is set in the system or session from which the master process is launched. For worker processes, specify the variable in the NGINX config file.

```
% grep NFAST_NFKM_TOKENSFILE /usr/lib/systemd/system/nginx.service

Environment="NFAST_NFKM_TOKENSFILE=<path-to-preload-file>"
```

```
% grep NFAST_NFKM_TOKENSFILE /etc/nginx/nginx.conf

env NFAST_NFKM_TOKENSFILE=<path-to-preload-file>;
```

```
% grep ssl_certificat_key /etc/nginx/conf.d/https.conf

ssl_certificate_key "engine:pkcs11:pkcs11:object=ocskey;token=ocscard;pin-value=123456";
```

8. Test the connections:

With mTLS:

```
% openssl s_client -connect localhost:443 -CAfile ./ca.crt -key ./client.key -cert ./client.crt
```

You also can use the `curl` command to test the connection with mTLS:

```
% curl --cert ./client.crt --key ./client.key --cacert ./ca.crt https://localhost:443
```

Without mTLS:

```
% openssl s_client -crlf -connect localhost:443 -CAfile ocskey.pem
```

9. Check the following messages and fields in the output:
    - CONNECTED(00000003)
    - depth
    - Certificate chain information
    - Server certificate information
    - Session-ID
    - Master-Key
    - TLS session ticket:
    - Verify return code: 0 (ok)

Chapter 3. Additional resources and related products

3.1. nShield HSMs

3.2. nShield as a Service

3.3. nShield Container Option Pack

3.4. Entrust products

3.5. nShield product documentation