F5 NGINX Server

nShield® HSM Integration Guide - PKCS #11

2024-07-24

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 8.9 X86-64
F5 NGINX Plus	nginx/1.25.1 (nginx-plus-r30-p1)
OpenSSL	openssl-libs-1:1.1.1k-9
OpenSSL PKCS #11	openssl-pkcs11-0.4.10-3

1.1.1. Supported nShield features

Entrust tested nShield HSM integration with the following features:

Feature	Support
Softcards	Yes
Module-only key	Yes

Feature	Support
OCS cards	Yes
nSaaS	Yes

1.1.2. Supported nShield hardware and software versions

Entrust tested with the following nShield hardware and software versions:

1.1.2.1. Connect XC

Security World Software	Firmware	Image	OCS	Softcard	Module
13.4.4	12.50.11 (FIPS Certified)	12.80.4	✓	✓	✓
13.4.4	12.72.1 (FIPS Certified)	12.80.5	✓	✓	✓

1.1.2.2. nShield 5c

Security World Software	Firmware	Image	OCS	Softcard	Module
13.4.4	13.2.2 (FIPS Pending)	13.2.2	✓	✓	✓

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

Integration procedures include:

- Install the NGINX Server using F5 NGINX Plus.
- Configure the NGINX server.
- Install the HSM.
- Install the Security World software and create a Security World.
- Set up the PKCS11 engine.
- Configure the NGINX Server to use the PKCS11 engine.
- Test the PKCS #11 integration with the NGINX Server and the HSM.

2.1. Install the NGINX Server using F5 NGINX Plus

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

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 on the URL: 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 PKCS11 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"
   ```

   Example minimum configuration:

   ```
   #
   # OpenSSL example configuration file.
   # This is mostly being used for generation of certificate requests.
   #

   # Note that you can include other files from the main configuration
   # file using the .include directive.
   #.include filename

   # This definition stops the following lines generating an error if HOME isn't
   # defined.
   HOME = .
   RANDFILE = $ENV::HOME/.rnd

   # nShield PKCS11
   openssl_conf = openssl_def
   [openssl_def]
   engines = engine_section
   [engine_section]
   pkcs11 = pkcs11_section
   [pkcs11_section]
   engine_id = pkcs11
   dynamic_path = /usr/lib64/engines-1.1/pkcs11.so
   MODULE_PATH = /opt/nfast/toolkits/pkcs11/libcknfast.so
   init = 0
   #!
   ```

   The `dynamic_path` may be different for different distributions.

2. If you see this message when creating certificates, you need to update your OpenSSL configuration:

   ```
   unable to find 'distinguished_name' in config
   problems making Certificate Request
   140493626791824:error:0E06D06C:configuration file routines:NCONF_get_string:no
   value:conf_lib.c:324:group=req name=distinguished_name
   ```

   Add the following to your OpenSSL configuration, adjusted to your

organization's values:

```
[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
```

3. Make sure the server's hostname matches the CN in the certificate.
4. Create a file called `openssl.pkcs11.cnf` with the settings above and save it where your OpenSSL configuration settings are located.
5. Create or edit the file `/etc/pki/tls/openssl.pkcs11.cnf` and enter the settings above:

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

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
```

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-1.1/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.

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

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

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:

- Functionality test with non-HSM keys.
- Module-only protection.
- Softcard protection.
- OCS protection.

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

2.7.1. mTLS

If mTLS is used in the configuration, create a CA certificate and a client certificate. In a production environment exposed to the internet, create the certificate request and sign it by the Trusted Certificate Authority.

For the test in this guide, self-signed certificates are used:

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.rt` 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
```

2.7.2. Functionality test with non-HSM keys

To make sure the NGINX Server installation is operational and capable of serving https content, create a software-based key and certificate before trying HSM-protected keys.

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

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

2. Create a directory to hold the keys:

```
% mkdir keys; cd keys
```

3. Create a private key:

```
% openssl genrsa -engine pkcs11 2048 > pkcs11localhost.key

engine "pkcs11" set.
Generating RSA private key, 2048 bit long modulus (2 primes)
.........................+++++
............+++++
e is 65537 (0x010001)
```

4. Create a self-signed certificate using this private key:

   With mTLS:

```
% openssl req -new -engine pkcs11 -keyform engine -key pkcs11localhost.key -out pkcs11localhost.csr

% openssl x509 -req -in pkcs11localhost.csr -CA ./ca.crt -CAkey ./ca.key -CAcreateserial -out
pkcs11localhost.crt
```

   Without mTLS:

```
% openssl req -engine pkcs11 -new -x509 -days 365 -key pkcs11localhost.key -out pkcs11localhost.crt

engine "pkcs11" set.
```

5. Configure the NGINX Server for SSL.
   a. Copy the `.key` and `.crt` files:

```
% sudo cp pkcs11localhost.key /etc/pki/tls/private/.
% sudo cp pkcs11localhost.crt /etc/pki/tls/certs/.
```

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

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

```
ssl_certificate /etc/pki/tls/certs/pkcs11localhost.crt;
ssl_certificate_key /etc/pki/tls/private/pkcs11localhost.key;
```

   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
```

6. Test the connection:

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
```

The output should be something like this:

```
CONNECTED(00000003)
Can't use SSL_get_servername
depth=0 C = US, ST = FL, L = Sunrise, O = Entrust, OU = nShield, CN = www.entrust.com
verify return:1
---
Certificate chain
 0 s:C = US, ST = FL, L = Sunrise, O = Entrust, OU = nShield, CN = www.entrust.com
   i:C = US, ST = FL, L = Sunrise, O = Entrust, OU = nShield, CN = www.entrust.com
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIDWzCCAkMCFAZbDVSvlbRy9ZVbWyC0rPrhYCdCMA0GCSqGSIb3DQEBCwUAMGox
CzAJBgNVBAYTAlVTMQswCQYDVQQIDAJGTDEQMA4GA1UEBwwHU3VucmlzZTEQMA4G
A1UECgwHRW50cnVzdDEQMA4GA1UECwwHblNoaWVsZDEYMBYGA1UEAwwPd3d3LmVu
dHJ1c3QuY29tMB4XDTIyMDkyMDIwNDMwOFoXDTIzMDkyMDIwNDMwOFowajELMAkG
A1UEBhMCVVMxCzAJBgNVBAgMAkZMMRAwDgYDVQQHDAdTdW5yaXNlMRAwDgYDVQQK
DAdFbnRydXN0MRAwDgYDVQQLDAduU2hpZWxkMRgwFgYDVQQDDA93d3cuZW50cnVz
dC5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQC7BG0KmYrdjaay
v34hFEARa2Ke2uM83CIkXsc/VxNbgnApShw3H+Wz3e/G33H0EBoFK8DBAHefxOVG
MihdoanQTQj4ve/9LxS64kd1U39bh+cNghC70rsd/PVFlmuJ1g3GfDZ9OLPt6pFW
7N+/EyB5/xQaKNUGpV1NFlVvPPM4tLPUiRRRyHAG2Zs1zMlZS5Lb0vCmHJHceXzu
Ek7IwI+xTWPxgT1QiljSWekznZ3XnFb2beyTGPqmfTDJ0yqT8M8pCxVZduUZ4gGi
qF+LQmONFYrA6tpRZh8ceRF0NhSfHaVm28MaBbbEYnCKu+W34evbCcIpLCfRS16b
uTbfN1+5AgMBAAEwDQYJKoZIhvcNAQELBQADggEBAGAQdC27WsbCILAx5tm4mQpF
f1kpyJ9/+O5UU27aRUULAo1S0U+dokZFPkPiT7PN4KqeNeSsbxK95SG00WVaWhY7
Q/0lrTyOyS2hOhByK2bDQJ0+k5gcJqdUXeqZUyBXeMsGK/KpGoPGzPTvCP277mv1
RXyaa+FrEjvSzEORxBoysLk885CBZMIfsWNQs3kFuGUJ7RBjiXc/HmdLAA1J3Rqq
KQI3AI8yk65z824JyJqUaj/lHRJK5yirZrXIIGwjpSw5poAN+ZqOQlrVEf5qv0kq
AVh1aVUKvE2xFnt8rq+W5zJy8ycIqMLDz7Ww0INt6yTrXlb9lcUsFdcOcvAnipk=
-----END CERTIFICATE-----
subject=C = US, ST = FL, L = Sunrise, O = Entrust, OU = nShield, CN = www.entrust.com

issuer=C = US, ST = FL, L = Sunrise, O = Entrust, OU = nShield, CN = www.entrust.com

---
No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: RSA-PSS
Server Temp Key: X25519, 253 bits
---
SSL handshake has read 1504 bytes and written 394 bytes
```

```
Verification: OK
---
New, TLSv1.2, Cipher is ECDHE-RSA-AES256-GCM-SHA384
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: CA7B4725395C94DFCF51AC312E39713F313920AA9E3DA278B321F71AFA60F792
    Session-ID-ctx:
    Master-Key:
1ABCCCB5A64FC49C84D7DF3EBF0FAF94EBB807F2DC4EA7606C8713A5CC907FB82F3A086D0D3BB45AE0F4490A38ED2398
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:
    0000 - 90 a4 24 fc db f8 f3 63-b4 55 47 d1 1c ea e8 34   ..$....c.UG....4
    0010 - c8 0d 7e ae 19 9e 9d 50-78 05 b2 1b a4 7d ec d6   ..~....Px....}..
    0020 - a4 c4 e5 35 86 f3 c6 7c-a1 bb 5f 0c e9 d9 e7 1d   ...5...|.._.....
    0030 - 83 52 13 d5 5c 2a 53 01-a3 b8 03 23 6e b2 61 4d   .R..\*S....#n.aM
    0040 - d1 5c 2d b6 d7 f4 51 69-b6 3d db 2f 4c 39 4d a8   .\-...Qi.=./L9M.
    0050 - b3 69 eb 80 0e 24 85 37-14 24 3c 15 1f 2b 86 1b   .i...$.7.$<..+..
    0060 - f5 47 f6 51 00 e4 ff 5d-a9 65 7f a3 d8 ef 91 e9   .G.Q...].e......
    0070 - de 81 e2 fe 66 9c fe 1d-6f ca 8f c9 ec c9 06 f5   ....f...o.......
    0080 - 74 6f 84 5d 7f 59 1e 24-d7 ab 4e 4f 9b 03 df e6   to.].Y.$..NO....
    0090 - 92 c7 ee 85 c8 20 8c ad-dd 7b 04 44 0f 60 53 7e   ..... ...{.D.`S~
    00a0 - 49 d0 28 27 48 52 7b f2-ee 38 93 5a 1f fc 66 d5   I.('HR{..8.Z..f.

    Start Time: 1663706976
    Timeout   : 7200 (sec)
    Verify return code: 0 (ok)
    Extended master secret: yes
---
closed
```

7. 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. 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. Get the certificate using this key:

   With mTLS:

```
% openssl req -new -engine pkcs11 -keyform engine -key "pkcs11:token=accelerator;object=modulersa" -out
modulersa.csr

% 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"

engine "pkcs11" set.
```

   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
```

5. 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
      ```

6. 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
   ```

7. 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)

2.7.4. 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. 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

   % 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

   engine "pkcs11" set.
   ```

   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.

6. 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:

```
% preload --preload-file <path-to-preload-file> softcardhsm_1 sudo systemctl restart 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
...
```

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=softcardhsm1_170047;token=softcardhsm_1;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 softcardhsm1_170047.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)

2.7.5. 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. 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

% 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"
```

5. 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:

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

2022-09-21 10:12:53: [160923]: INFO: Preload running with: --preload-file <path-to-preload-file> -c
ocscard sudo systemctl restart nginx
2022-09-21 10:12:58: [160923]: INFO: Created a (new) connection to Hardserver
2022-09-21 10:12:58: [160923]: INFO: Modules newly usable: [1].
2022-09-21 10:12:58: [160923]: INFO: Found a change in the system: an update pass is needed.
2022-09-21 10:12:58: [160923]: INFO: Loading cardset: ocscard in modules: [1]

Loading `ocscard':
 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.

2022-09-21 10:13:01: [160923]: INFO: Stored Admin key: kfips (5ab6...) in module #1
2022-09-21 10:13:01: [160923]: INFO: Loading cardset: Cardset: ocscard (454e...) in module: 1
2022-09-21 10:13:01: [160923]: INFO: Stored Cardset: ocscard (454e...) in module #1
2022-09-21 10:13:01: [160923]: INFO: Maintaining the cardset ocscard protected
key(s)=['pkcs11:uc454e988e226b33fa94087c0ee6112e0975c1557f-
bf7b5f0412619a354f86f58c77d796f27bd3ee12'].
2022-09-21 10:13:01: [160923]: INFO: The private/symmetric key
pkcs11/uc454e988e226b33fa94087c0ee6112e0975c1557f-bf7b5f0412619a354f86f58c77d796f27bd3ee12 is loaded
in module(s): [1].
2022-09-21 10:13:01: [160923]: INFO: Loading complete. Executing subprocess sudo systemctl restart
nginx
```

6. 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";
```

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 ocskey.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)

Chapter 3. Additional resources and related products