ENTRUST

SECURING A WORLD IN MOTION

NGINX Server

nShield® HSM Integration Guide - PKCS #11

Version: 1.0

Date: Friday, July 30, 2021

Information in this document is subject to change without notice.

nCipher Security Limited makes no warranty of any kind with regard to this information, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. nCipher Security Limited shall not be liable for errors contained herein or for incidental or consequential damages concerned with the furnishing, performance or use of this material.

Where translations have been made in this document English is the canonical language.

nCipher Security Limited
Registered Office: One Station Square
Cambridge, UK CB1 2GA
Registered in England No. 11673268

nCipher is an Entrust company.

Contents

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 140-2 certified hardware security module. This integration uses the PKCS #11 interface to integrate the HSM and NGINX Server.

The benefits of using an nShield Hardware Security Module (HSM) with the NGINX Server include:

- Secure storage of the private key.
- FIPS 140-2 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 X86-64
NGINX	nginx/1.14.1
F5 NGINX Plus	nginx/1.19.10 (nginx-plus-r24-p1)
OpenSSL	openssl-libs-1:1.1.1g-12
OpenSSL PKCS #11	openssl-pkcs11-0.4.10-2

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
12.60.11	12.50.11	12.60.10	✓	✓	✓

1.1.2.2. Connect +

Security World Software	Firmware	Image	OCS	Softcard	Module
12.60.11	12.50.8	12.60.10	✓	✓	✓

1.2. Requirements

Ensure that you have supported versions of the nShield, 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 140-2 level 3.

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.

2. Procedures

Integration procedures include:

- Installing the NGINX Server.
- Configuring the NGINX Server.
- Installing the HSM.
- Installing the Security World software and creating the Security World.
- Setting up the PKCS11 engine.
- Configuring the NGINX Server to use the PKCS11 engine.
- Testing the PKCS #11 integration with the NGINX Server and the HSM.

2.1. Install the NGINX Server - open-source NGINX

See Installing NGINX Plus for detailed instructions on how to install NGINX Plus. The installation instructions vary between the open-source version of F5 NGINX and NGINX Plus.

```
% sudo yum install -y nginx
```

2.2. Install the NGINX Server - F5 NGINX Plus

1. If you already have old NGINX Plus packages installed, back up your configuration and log files:

   ```
   % sudo cp -a /etc/nginx /etc/nginx-plus-backup
   % sudo cp -a /var/log/nginx /var/log/nginx-plus-backup
   ```

2. Create the `/etc/ssl/nginx directory`:

   ```
   % sudo mkdir -p /etc/ssl/nginx
   ```

3. Log in to MyF5 Customer Portal and download your `nginx-repo.crt` and `nginx-repo.key` files.
4. Copy the `.crt` and `.key files` to the `/etc/ssl/nginx` directory:

   ```
   % sudo cp nginx-repo.crt /etc/ssl/nginx/
   % sudo cp nginx-repo.key /etc/ssl/nginx/
   ```

5. Install the required `ca-certificates` dependency:

```
% sudo yum install ca-certificates
```

6. Add the NGINX Plus repository by downloading the `nginx-plus-8.repo` file to `/etc/yum.repos.d`:

   ```
   % sudo wget -P /etc/yum.repos.d https://cs.nginx.com/static/files/nginx-plus-8.repo
   ```

7. If you have NGINX ModSecurity subscription, add the NGINX ModSecurity repository by downloading the `modsecurity-8.repo` file to `/etc/yum.repos.d`:

   ```
   % sudo wget -P /etc/yum.repos.d https://cs.nginx.com/static/files/modsecurity-8.repo
   ```

8. Install the `nginx-plus` package. An older NGINX Plus package is automatically replaced.

   ```
   % sudo yum install nginx-plus
   ```

9. If you have NGINX ModSecurity subscription, install the ModSecurity module:

   ```
   % sudo yum install nginx-plus nginx-plus-module-modsecurity
   ```

10. Check the `nginx` binary version to ensure that you have NGINX Plus installed correctly:

    ```
    % nginx -v

    nginx version: nginx/1.19.10 (nginx-plus-r24-p1)
    ```

2.3. 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.

   To make sure NGINX is up and running after a reboot, enable the service.

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

4. Install the OpenSSL packages.

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

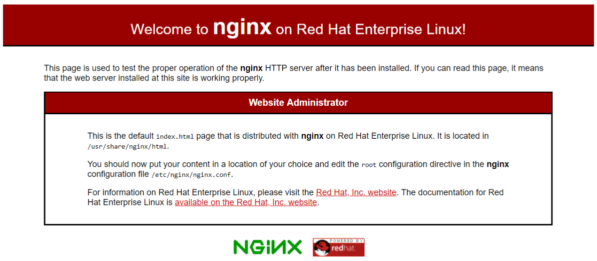
   ```
   % sudo yum install -y opensc openssl-pkcs11 gnutls-utils nano openssl-libs
   ```

5. Restart the NGINX service.

   ```
   % sudo systemctl restart nginx
   ```

6. Check if NGINX is running.
   a. Open the browser on the URL: *http://<your-ip-address>*.
   b. You should see something similar to this:

NGINX



Welcome to **nginx** on Red Hat Enterprise Linux!

This page is used to test the proper operation of the **nginx** HTTP server after it has been installed. If you can read this page, it means that the web server installed at this site is working properly.

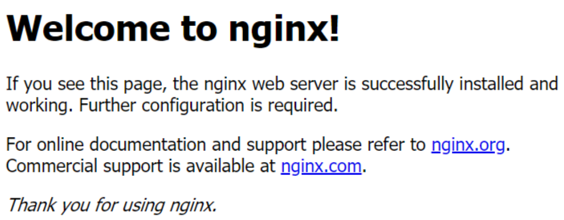
Website Administrator

This is the default `index.html` page that is distributed with **nginx** on Red Hat Enterprise Linux. It is located in `/usr/share/nginx/html`.

You should now put your content in a location of your choice and edit the `root` configuration directive in the **nginx** configuration file `/etc/nginx/nginx.conf`.

For information on Red Hat Enterprise Linux, please visit the Red Hat, Inc. website. The documentation for Red Hat Enterprise Linux is available on the Red Hat, Inc. website.

NGINX Plus



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

2.4. Install the HSM

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

We recommend that you install the HSM before configuring the Security World software with your NGINX Server.

2.5. 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.

2.6. 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 something inside `/opt/nfast`, check your `$PATH` variable.

2.6.1. Configure OpenSSL

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

   ```
   % openssl version -d

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

   The minimum configuration is something like this:

```
#
# 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 = www.entrust.com
[v3_req]
subjectAltName = @alt_names
[alt_names]
DNS.1 = www.entrust.com
DNS.2 = entrust.com
```

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
   ```

2.6.2. Set up /opt/nfast/cknfastrc

1. You might have to add the following variables to the /opt/nfast/cknfastrc file.

   They 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.

2.6.3. 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
   ```

2.6.4. Debug notes

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.

Debug variables

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

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

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
```

2.6.5. List available slots

Generate and insert your OCS as usual.

```
% pkcs11-tool --module /opt/nfast/toolkits/pkcs11/libcknfast.so -L

Available slots:
Slot 0 (0x1d622495): 6308-03E0-D947 Rt1
  token label        : accelerator
  token manufacturer : nCipher Corp. Ltd
  token model        :
  token flags        : rng, token initialized, PIN initialized, other flags=0x200
  hardware version   : 0.0
  firmware version   : 12.50
  serial num         : 6308-03E0-D947
  pin min/max        : 0/256
Slot 1 (0x1d622496): 6308-03E0-D947 Rt1 slot 0
  (empty)
Slot 2 (0x1d622497): 6308-03E0-D947 Rt1 slot 2
  (token not recognized)
Slot 3 (0x1d622498): 6308-03E0-D947 Rt1 slot 3
  (empty)
```

2.7. Configure the NGINX Server to use the PKCS11 engine

You need to update the NGINX Startup file to tell it to use the new Open SSL configuration file. Update the NGINX service start-up 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=/opt/nfast/kmdata/local/preload"
```

2. With Softcard and OCS protection, the usual arrangement of spawning worker processes requires preloading the Softcard or the OCS card. You have to 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 something in on start-up. For the master process you have to set the variable is set in the system or session from which the master process is launched. For worker processes, you have to 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.

For Softcard or OCS protection, add the following line after the pid line to expose tokensfile to the worker processes:

```
env NFAST_NFKM_TOKENSFILE=/opt/nfast/kmdata/local/preload;
```

1. Add the PKCS11 engine. Put it after the Events section

```
ssl_engine pkcs11;
```

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

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

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=/opt/nfast/kmdata/local/preload;


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;
}
```

2. Create a `https.conf` file in `/etc/nginx/conf.d` folder with the following content, and 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_protocols       TLSv1 TLSv1.1 TLSv1.2;
#
#    location / {
#        root   /usr/share/nginx/html;
#        index  index.html index.htm;
#    }
#}
```

3. Restart the NGINX service:

```
% sudo systemctl restart nginx
```

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

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

2.8. 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.

2.8.1. 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 /opt/nfast/kmdata/local/preload
   ```

2. Create a directory to hold the keys.

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

3. Create a private key:

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

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

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

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. Restart the NGINX service:

```
% sudo systemctl restart nginx
```

6. Test the connection:

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

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-----
MIIDWzCCAkMCFB/U1WNhDP+Vh5xHhs9n0KxA1FSqMA0GCSqGSIb3DQEBCwUAMGox
CzAJBgNVBAYTAlVTMQswCQYDVQQIDAJGTDEQMA4GA1UEBwwHU3VucmlzZTEQMA4G
A1UECgwHRW50cnVzdDEQMA4GA1UECwwHblNoaWVsZDEYMBYGA1UEAwwPd3d3LmVu
dHJ1c3QuY29tMB4XDTIxMDcwOTE5MjAxMloXDTIyMDcwOTE5MjAxMlowajELMAkG
A1UEBhMCVVMxCzAJBgNVBAgMAkZMMRAwDgYDVQQHDAdTdW5yaXNlMRAwDgYDVQQK
DAdFbnRydXN0MRAwDgYDVQQLDAduU2hpZWxkMRgwFgYDVQQDDA93d3cuZW50cnVz
dC5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDcI2m2GyOJ3AEA
irWfX75YHVIiR/fDjaYALA8niv2G/j3WY4ac2v1s6UeKzXcRZobLzPXwfjTya+hR
bcKdIT+4LTZ6zXeMX4JXw+tZTwatdVeQ1/wSKtXdN+5FbzS0mPubCeR61ilTzIVl
cUqLD1B55f3gRcGFIbop7sQb2U1poVwBM+fT09YAqKTUhtqI8xEx0WbUVW/TGUkt
Xus1R/X88iyoNb2tJsFONb/PCxl8B913lQHwXyHM+WEa854TMqkwu4ZZXcP7Cqhy
wwgBMy1nt6cbtkH5jCCzVJAdkeWA+JaeGFU1dMGY1knI+1WeEbiMYbZ+m4ZN8k3g
pUHs7xgJAgMBAAEwDQYJKoZIhvcNAQELBQADggEBAKJbLXeniQE61Ll3/Nol9L0r
Dj24QxFN0PcemPyrpxRSgkPX02/ZYH7pds4odYlJGvqu940PYPkmiXc0wmjoX9FM
Sazt3ZmRqjpVWv4CKNdKFi95ZT5izbroad1z24l2xjjuydPir1d6+uMdbBjQa6UX
4qxfWSn0zyDNzySBpvfvoBaoiTUzx9DHaka22BndCC1wO+TVu+QogyAGsDQJl3EC
GqLt5AcxdLBbjjNzsbO7ZOtdjfGxUrJiRnnj/i7Iwr7Vy/hOlstkf4dnHUmcKaPE
z9ilo/rYCXr19n0p9slskBIsn2INNXbdzss0b++bXf8bJtJ2A9qhxX9a3/dbndA=
-----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: 7145FC06C460819C5568C1CAFE024D7051792DB1C7B9B4233C5FA1AFE3369FAB
    Session-ID-ctx:
    Master-Key: B33DAC747716606E535DB94115E5795C90A4015E67B11BDC28F1A515866876759902D39F7A7D29981EFFFFAC9C0DB22E
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:
    0000 - e0 d0 68 54 81 96 f0 cd-f1 f3 2c b6 c8 71 2f 24   ..hT......,..q/$
    0010 - 05 a5 dc 98 dc 0d e9 32-a8 a3 8c 74 e4 71 58 00   .......2...t.qX.
    0020 - f3 29 5b ea 82 71 ca 81-65 fb dc 73 36 16 f2 3f   .)[..q..e..s6..?
    0030 - 22 b1 1d 47 59 da dc ef-76 ec 5f 39 19 5d 9b e8   "..GY...v._9.]..
    0040 - 3c a0 49 aa d1 ac 54 da-31 bf c1 2c 3c 62 a0 0f   <.I...T.1..,<b..
    0050 - 91 19 85 7e 8d ca 0d 06-30 8e 77 2b 57 b9 e3 9a   ...~....0.w+W...
    0060 - 23 0b 0d 24 e5 de f0 0d-7d 64 ff a6 1a 52 96 d1   #..$....}d...R..
    0070 - 98 8a a0 b6 8b 48 1f 07-bf 5b b4 cf b5 1f 39 ce   .....H...[....9.
    0080 - 39 b7 3e 50 0f 08 c0 cb-f5 ca a0 61 9d 25 38 76   9.>P.......a.%8v
    0090 - 6a 63 30 e4 cc e9 18 99-f6 5d 8c f6 9b 84 50 79   jc0......]....Py
    00a0 - 02 e8 3e 50 c5 6d 50 cb-61 df 2e 1d ac bb 99 cd   ..>P.mP.a.......

    Start Time: 1625858466
    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)

2.8.2. Module protection

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

```
% sudo rm -f /opt/nfast/kmdata/local/preload
```

2. To allow module protection, the `cknfast` library has to be set so it allows login 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_ua06d0ee167cd56f3423a3cb91c9cbd04a83599e31
```

4. Get the certificate using this key:

```
% 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. Restart the NGINX service:

```
% sudo systemctl restart nginx
```

6. Test the connections:

```
% 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)

2.8.3. Set up Softcard protection

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

```
% sudo rm -f /opt/nfast/kmdata/local/preload
```

2. To expose Softcards, the `cknfast` library has to be in 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 mysoftcard

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=softcardkey type=rsa protect=softcard recovery=no size=2048
softcard=mysoftcard

key generation parameters:
 operation    Operation to perform               generate
 application  Application                        pkcs11
 protect      Protected by                       softcard
 softcard     Soft card to protect key           mysoftcard
 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                           softcardkey
 nvram        Blob in NVRAM (needs ACS)          no
Please enter the pass phrase for softcard `mysoftcard':

Please wait........
Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_uc541c437751f2b296f5733bd326e5c116435cb814-
2080cf356215b42c73e85a1a58190ea933fb6f4c
```

5. Get the certificate using this key:

```
% openssl req -engine pkcs11 -x509 -out softcard.crt -days 365 -key "pkcs11:model=;token=mysoftcard;pin-
value=123456;object=softcardkey" -keyform ENGINE -subj "/CN=softcardkey"

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 `.crt` file:

   ```
   % sudo cp softcard.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 it still commented out.

```
ssl_certificate /etc/pki/tls/certs/softcard.crt
ssl_certificate_key "engine:pkcs11:pkcs11:object=softcardkey;token=mysoftcard;pin-value=123456"
```

c. Restart the NGINX service:

```
% ppmk --preload --preload-file /opt/nfast/kmdata/local/preload mysoftcard sudo systemctl restart nginx
```

If you don't restart NGINX by executing `ppm --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. You have to 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 something in on start-up. For the master process you have to set the variable is set in the system or session from which the master process is launched. For worker processes, you have to specify the variable in the NGINX config file.

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

Environment="NFAST_NFKM_TOKENSFILE=/opt/nfast/kmdata/local/preload"
```

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

env NFAST_NFKM_TOKENSFILE=/opt/nfast/kmdata/local/preload;
```

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

ssl_certificate_key "engine:pkcs11:pkcs11:object=softcardkey;token=mysoftcard;pin-value=123456";
```

8. Test the connections:

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

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)

2.8.4. Set up OCS protection

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

   ```
   % sudo rm -f /opt/nfast/kmdata/local/preload
   ```

2. Create an OCS:

   ```
   % /opt/nfast/bin/createocs -m1 -s0 --persist -Q 1/1 -N ocscard


   Creating Cardset:
    Module 1: 0 cards of 1 written
    Module 1 slot 0: Admin Card #1
    Module 1 slot 2: blank card
    Module 1 slot 3: empty
    Module 1 slot 2:- passphrase specified - writing card
   Card writing complete.

   cardset created; hkltu = 53513d8094e907099a2ddbe2b00e15cd99158bd2
   ```

   123456 is the passphrase for the OCS in the example.

3. Create a key:

```
% 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 #1
 Module 1 slot 3: empty
 Module 1 slot 2:- passphrase supplied - reading card
Card reading complete.

Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_uc53513d8094e907099a2ddbe2b00e15cd99158bd2-
6d696040526f1b24a58fa633ec6c90e033c9a11a
```

4. Get the certificate using this key:

```
% 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. Restart the NGINX service:

```
% preload --preload-file /opt/nfast/kmdata/local/preload -c ocscard sudo systemctl restart nginx

 Preload running with: --preload-file /opt/nfast/kmdata/local/preload -c ocscard sudo systemctl restart nginx
2021-07-12 14:55:06: [7367]: INFO: Created a (new) connection to Hardserver
2021-07-12 14:55:06: [7367]: INFO: Modules newly usable: [1].
2021-07-12 14:55:06: [7367]: INFO: Found a change in the system: an update pass is needed.
2021-07-12 14:55:06: [7367]: INFO: Loading cardset: ocscard in modules: [1]

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

2021-07-12 14:55:10: [7367]: INFO: Loading cardset: Cardset: ocscard (5351...) in module: 1
2021-07-12 14:55:10: [7367]: INFO: Stored key Cardset: ocscard (5351...) in module #1
2021-07-12 14:55:10: [7367]: INFO: Maintaining the cardset ocscard protected
key(s)=['pkcs11:uc53513d8094e907099a2ddbe2b00e15cd99158bd2-6d696040526f1b24a58fa633ec6c9\
0e033c9a11a'].
2021-07-12 14:55:10: [7367]: INFO: The private/symmetric key
pkcs11/uc53513d8094e907099a2ddbe2b00e15cd99158bd2-6d696040526f1b24a58fa633ec6c90e033c9a11a is loaded in \
module(s): [1].
2021-07-12 14:55:10: [7367]: 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. You have to 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 something in on start-up. For the master process you have to set the variable is set in the system or session from which the master process is launched. For worker processes, you have to specify the variable in the NGINX config file.

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

Environment="NFAST_NFKM_TOKENSFILE=/opt/nfast/kmdata/local/preload"
```

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

env NFAST_NFKM_TOKENSFILE=/opt/nfast/kmdata/local/preload;
```

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

ssl_certificate_key "engine:pkcs11:pkcs11:object=softcardkey;token=mysoftcard;pin-value=123456";
```

7. Test the connections:

```
% 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)

Contact Us

Web site	https://www.entrust.com
Support	https://nshieldsupport.entrust.com
Email Support	nShield.support@entrust.com
Online documentation:	Available from the Support site listed above.

You can also contact our Support teams by telephone, using the following numbers:

Europe, Middle East, and Africa

United Kingdom:	+44 1223 622444 One Station Square Cambridge, UK CB1 2GA

Americas

Toll Free:	+1 833 425 1990
Fort Lauderdale:	+1 954 953 5229 Sawgrass Commerce Center – A Suite 130 13800 NW 14 Street Sunrise, FL 33323 USA

Asia Pacific

Australia:	+61 8 9126 9070 World Trade Centre Northbank Wharf Siddeley St Melbourne VIC 3005 Australia
Japan:	+81 50 3196 4994
Hong Kong:	+852 3008 3188 31/F, Hysan Place, 500 Hennessy Road, Causeway Bay

To get help with
Entrust nShield HSMs

nShield.support@entrust.com

nshieldsupport.entrust.com

ABOUT ENTRUST CORPORATION

Entrust keeps the world moving safely by enabling trusted identities, payments, and data protection. Today more than ever, people demand seamless, secure experiences, whether they're crossing borders, making a purchase, accessing e-government services, or logging into corporate networks. Entrust offers an unmatched breadth of digital security and credential issuance solutions at the very heart of all these interactions. With more than 2,500 colleagues, a network of global partners, and customers in over 150 countries, it's no wonder the world's most entrusted organizations trust us.

ENTRUST
SECURING A WORLD IN MOTION