
# Mirantis Kubernetes Engine

## nShield® HSM Integration Guide

2024-02-12

# Table of Contents


1. Introduction . . . . . 1
    1.1. Product configurations . . . . . 1
    1.2. Supported nShield hardware and software versions . . . . . 1
    1.3. Supported nShield HSM functionality . . . . . 2
    1.4. Requirements . . . . . 2
    1.5. More information . . . . . 3
2. Procedures . . . . . 4
    2.1. Prerequisites . . . . . 4
    2.2. Push the nCOP container images to an internal Docker registry . . . . . 4
    2.3. Create the registry secrets . . . . . 5
3. Additional resources and related products . . . . . 17
    3.1. Video . . . . . 17
    3.2. nShield Connect . . . . . 17
    3.3. nShield as a Service . . . . . 17
    3.4. nShield Container Option Pack . . . . . 17
    3.5. Entrust digital security solutions . . . . . 17
    3.6. nShield product documentation . . . . . 17

# Chapter 1. Introduction

This guide describes the steps to integrate the nShield Container Option Pack (nCOP) with Mirantis Kubernetes Engine. The nCOP provides application developers, within a container-based Mirantis Kubernetes Engine environment, the ability to access the cryptographic functionality of an nShield Hardware Security Module (HSM).

## 1.1. Product configurations

We have successfully tested nShield HSM integration with Mirantis Kubernetes Engine in the following configurations:

| Software | Version |
|---|---|
| nCOP | 1.1.1 |
| Operating System | CentOS 8 |
| Mirantis Kubernetes Engine | 3.4.5 |
| Mirantis Container Runtime | 20.10.7 |

## 1.2. Supported nShield hardware and software versions

We have successfully tested with the following nShield hardware and software versions:

### 1.2.1. Connect XC

| Security World Software | Firmware | Image | OCS | Softcard | Module |
|---|---|---|---|---|---|
| 12.71.0 | 12.50.11 | 12.60.10 | ✓ | ✓ | ✓ |

### 1.2.2. Connect +

| Security World Software | Firmware | Image | OCS | Softcard | Module |
|---|---|---|---|---|---|
| 12.71.0 | 12.50.8 | 12.60.10 | ✓ | ✓ | ✓ |

## 1.3. Supported nShield HSM functionality

| Feature | Support |
|---|---|
| Module-only key | Yes |
| OCS cards | Yes |
| Softcards | Yes |
| nSaaS | Yes |
| FIPS 140 Level 3 | Yes |

## 1.4. Requirements

Before installing these products, read the associated documentation:

- For the nShield HSM: *Installation Guide* and *User Guide*.
- If nShield Remote Administration is to be used: *nShield Remote Administration User Guide*.
- *nShield Container Option Pack User Guide*.
- MCR documentation (https://docs.mirantis.com/mcr/20.10/install/mcr-linux.html)
- MKE documentation (https://docs.mirantis.com/mke/3.4/index.html).
- kubectl documentation (https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/)

Furthermore, the following design decisions have an impact on how the HSM is installed and configured:

- Whether your Security World must comply with FIPS 140 Level 3 standards.
    - If using FIPS 140 Level 3, it is advisable to create an OCS for FIPS authorization. The OCS can also provide key protection for the Vault

master key. For information about limitations on FIPS authorization, see the *Installation Guide* of the nShield HSM.

- Whether to instantiate the Security World as recoverable or not.

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

## 1.5. More information

For more information about OS support, contact your Mirantis 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. Prerequisites

Before you can use nCOP and pull the nCOP container images to the external registry, complete the following steps:

1. Install the Mirantis Container Runtime on the host machine. This can be a VM running CentOS 8 or other compatible Operating Systems.
2. Install the Mirantis Kubernetes Engine on the host machine.
3. Install kubectl on the host machine.
4. Set up the HSM. See the Installation Guide for your HSM.
5. Configure the HSM(s) to have the IP address of your container host machine as a client.
6. Load an existing Security World or create a new one on the HSM. Copy the Security World and module files to your container host machine at a directory of your choice. Instructions on how to copy these two files into a persistent volume accessible by the application containers are given when you create the persistent volume during the deployment of MKE.
7. Install nCOP and create the containers that contain your application. For the purpose of this guide you will need the nCOP hardserver container and your application container. In this guide they are referred to as the *nshield-hwsp* and *nshield-app* containers. For instructions, see the *nShield Container Option Pack User Guide*.

For more information on configuring and managing nShield HSMs, Security Worlds, and Remote File Systems, see the User Guide for your HSM(s).

## 2.2. Push the nCOP container images to an internal Docker registry

You will need to register the nCOP container images you created to a Docker registry so they can be used when you deploy the Kubernetes pods later. In this guide, the external registry is <docker-registry-address>. Distribution of the nCOP container image is not permitted because the software components are under strict export controls.

To deploy an nCOP container images for use with Mirantis Kubernetes Engine:

1. Log in to the container host machine server as root, and launch a terminal window. We assume that you have built the nCOP container images in this host and that they are available locally in Docker. They are: nshield-hwsp:12.71.0 and nshield-app:12.71.0.
2. Log in to the Docker registry.

```
% docker login -u YOURUSERID https://<docker-registry-address>
```

3. Register the images:
   a. Tag the images:

```
% sudo docker tag nshield-hwsp:12.71.0 <docker-registry-address>/nshield-hwsp
% sudo docker tag nshield-app:12.71.0 <docker-registry-address>/nshield-app
```

   b. Push the images to the registry:

```
% sudo docker push <docker-registry-address>/nshield-hwsp
% sudo docker push <docker-registry-address>/nshield-app
```

   c. Remove the local images:

```
% sudo docker rmi <docker-registry-address>/nshield-hwsp
% sudo docker rmi <docker-registry-address>/nshield-app
```

   d. List the images:

```
% sudo docker images
```

   e. Pull the images from the registry:

```
% sudo docker pull <docker-registry-address>/nshield-hwsp
% sudo docker pull <docker-registry-address>/nshield-app
```

   f. List the images:

```
% sudo docker images
```

## 2.3. Create the registry secrets

At the beginning of our process, we created nCOP Docker containers and we pushed them to our internal Docker registry. Now it is necessary to let MKE know

about how to authenticate to that registry.

1. Create the secret.

```
% kubectl create secret generic regcred --from-file= dockerconfigjson=/home/<YOUR USER
ID>/.docker/config.json --type=kubernetes.io/dockerconfigjson
```

2. Check if the secret was created.

```
% kubectl get secret regcred --output=yaml
```

### 2.3.1. Create the Configuration Map for the HSM details

We have created a `.yaml` file that can be modified according to the HSM you are using. Edit the file accordingly.

This integration was tested using kubectl commands for generating kubernetes objects with yaml files. The MKE web ui provides an alternative interface that can be used to generate these objects, and view them. See MKE documentation for more information.

For example:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: config
data:
  config: |
    syntax-version=1

    [nethsm_imports]
    local_module=1
    remote_esn=BD10-03E0-D947
    remote_ip=10.194.148.36
    remote_port=9004
    keyhash=2dd7c10c73a3c5346d1246e6a8cf6766a7088e41
    privileged=0
```

1. Create the Config Map.

```
% kubectl apply -f configmap.yaml

configmap/config created
```

2. Verify the config map was created successfully.

```
% kubectl describe configmap/config

Name:         config
Namespace:    default
Labels:       <none>
Annotations:
Data
====
config:

syntax-version=1

[nethsm_imports]
local_module=1
remote_esn=BD10-03E0-D947
remote_ip=10.194.148.36
remote_port=9004
keyhash=2dd7c10c73a3c5346d1246e6a8cf6766a7088e41
privileged=0

Events:  <none>
```

## 2.3.2. Create the MKE persistent Volumes

This section describes how the persistent volumes is created in MKE.

Before you proceed with the creation of the persistent volume, you must create the directory `/opt/nfast/kmdata/local` in your host machine and copy the Security World and module files to it.

The example YAML files below are used to create and claim the persistent volume.

- The persistent_volume_kmdata_definition.yaml file:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfast-kmdata
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 1G
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: /opt/nfast/kmdata
```

- The persistent_volume_kmdata_claim.yaml file:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
```

```
  name : nfast-kmdata
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: local-storage
  resources:
    requests:
      storage: 1G
  storageClassName: manual
```

1. Apply the definition file to MKE.

```
% kubectl apply -f persistent_volume_kmdata_definition.yaml

persistentvolume/nfast-kmdata created
```

2. Verify the persistent volume has been created.

```
% kubectl get pv

NAME CAPACITY   ACCESS MODES   RECLAIM   POLICY STATUS    CLAIM        STORAGECLASS   REASON   AGE
nfast-kmdata    1G             RWO       Retain           Available    manual                  43m
```

3. Create the claim.

```
% kubectl apply -f persistent_volume_kmdata_claim.yaml

persistentvolumeclaim/nfast-kmdata created
```

4. Verify the claim has been created.

```
% kubectl get pvc

NAME            STATUS   VOLUME          CAPACITY   ACCESS MODES   STORAGECLASS   AGE
nfast-kmdata    Bound    nfast-kmdata    1G         RWO            manual         61m

% kubectl get pv

NAME            CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM
STORAGECLASS   REASON   AGE
nfast-kmdata    1G         RWO            Retain           Bound    default/nfast-kmdata    manual
67m
```

## 2.3.3. Deploy the nCOP Pod with your application

You will need to create a `.yaml` file that defines how to launch the hardserver and your application container into MKE. The examples below were created to show how you can talk to the HSM from inside the Kubernetes pod. Each example shows how to execute the following commands: `enquiry` and `nfkminfo`.

#### 2.3.3.1. Populating the persistent volume with the world and module file

Before running any of the applications, /opt/nfast/kmdata/local in the persistent volume needs to be updated with the latest world and module files. To do this, create a yaml file to run a pod that gives access to the persistent volume so these files can be copied.

For example, the following persistent_volume_kmdata_populate.yaml file shows how to get access to the persistent volume:

```
kind: Pod
apiVersion: v1
metadata:
  name: ncop-populate-kmdata
  labels:
    app: nshield
spec:
  imagePullSecrets:
    - name: regcred
  containers:
    - name: ncop-kmdata
      command:
        - sh
        - '-c'
        - sleep 3600
      image: <docker-registry-address>/nshield-app
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      volumeMounts:
        - name: ncop-kmdata
          mountPath: /opt/nfast/kmdata
        - name: ncop-sockets
          mountPath: /opt/nfast/sockets
  securityContext: {}
  volumes:
    - name: ncop-config
      configMap:
        name: config
        defaultMode: 420
    - name: ncop-hardserver
      emptyDir: {}
    - name: ncop-kmdata
      persistentVolumeClaim:
        claimName: nfast-kmdata
    - name: ncop-sockets
      emptyDir: {}
```

- Deploy the pod

  ```
  % kubectl apply -f persistent_volume_kmdata_populate.yaml
  ```

- Check if the Pod is running

  ```
  % kubectl get pods
  ```

You should see the deployment taking place. Wait 10 seconds and run the command again until the status is Running. This will also let you know if there are any errors. If there are errors, run the following command:

```
% kubectl describe pod ncop-populate-kmdata
```

- Copy the module file to /opt/nfast/kmdata/local in the pod.

```
% kubectl cp /opt/nfast/kmdata/local/module_BD10-03E0-D947 ncop-populate-kmdata:/opt/nfast/kmdata/local/.
```

- Copy the world file to /opt/nfast/kmdata/local in the pod.

```
% kubectl cp /opt/nfast/kmdata/local/world ncop-populate-kmdata:/opt/nfast/kmdata/local/.
```

- Check if the files are in the persistent volume.

```
% kubectl exec ncop-populate-kmdata -- ls -al /opt/nfast/kmdata/local

total 68
drwxr-xr-x 2 root root  4096 Sep 20 18:40 .
drwxr-xr-x 3 root root  4096 Dec 16  2020 ..
-rwxrwxrwx 1 root 1001  3488 Sep 20 18:40 module_BD10-03E0-D947
-rwxrwxrwx 1 root 1001 39968 Sep 20 18:40 world
```

#### 2.3.3.2. Running the enquiry command

To run the `enquiry` command, which prints enquiry data from the module, use the following pod_enquiry_app.yaml file.

```
kind: Pod
apiVersion: v1
metadata:
  name: ncop-test-enquiry
  labels:
    app: nshield
spec:
  imagePullSecrets:
    - name: regcred
  containers:
    - name: ncop
      command:
        - sh
        - '-c'
        - /opt/nfast/bin/enquiry && sleep 3600
      image: <docker-registry-address>/nshield-app
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      volumeMounts:
        - name: ncop-kmdata
          mountPath: /opt/nfast/kmdata
```

```
      - name: ncop-sockets
        mountPath: /opt/nfast/sockets
  - name: ncop-hwsp
    image: <docker-registry-address>/nshield-hwsp
    ports:
      - containerPort: 8080
        protocol: TCP
    resources: {}
    volumeMounts:
      - name: ncop-config
        mountPath: /opt/nfast/kmdata/config
      - name: ncop-hardserver
        mountPath: /opt/nfast/kmdata/hardserver.d
      - name: ncop-sockets
        mountPath: /opt/nfast/sockets
volumes:
  - name: ncop-config
    configMap:
      name: config
      defaultMode: 420
  - name: ncop-hardserver
    emptyDir: {}
  - name: ncop-kmdata
    persistentVolumeClaim:
      claimName: nfast-kmdata
  - name: ncop-sockets
    emptyDir: {}
```

In this example, <docker_registry-address> is the address of your internal docker registry server.

- Deploy the pod.

  ```
  % kubectl apply -f pod_enquiry_app.yaml
  ```

- Check if the Pod is running.

  ```
  % kubectl get pods
  ```

  You should see the deployment taking place. Wait 10 seconds and run the command again until the status is Running. This will also let you know if there are any errors. If there are errors, run the following command:

  ```
  % kubectl describe pod ncop-test-enquiry
  ```

- Check if the `enquiry` command ran successfully.

  ```
  % kubectl logs pod/ncop-test-enquiry ncop

  Server:
   enquiry reply flags  none
   enquiry reply level  Six
   serial number        BD10-03E0-D947
   mode                 operational
  ```

```
 version              12.71.0
 speed index          478
 rec. queue           110..208
 level one flags      Hardware HasTokens SupportsCommandState
 version string       12.71.0-353-f63c551, 12.50.11-270-fb3b87dd465b6f6e53d9f829fc034f8be2dafd13 2019/05/16
22:02:33 BST, Bootloader: 1.2.3, Security Processor: 12.50.11 , 12.60.10-708-ea4dc41d
 checked in           000000006053229a Thu Mar 18 09:51:22 2021
 level two flags      none
 max. write size      8192
 level three flags    KeyStorage
 level four flags     OrderlyClearUnit HasRTC HasNVRAM HasNSOPermsCmd ServerHasPollCmds FastPollSlotList
HasSEE HasKLF HasShareACL HasFeatureEnable HasFileOp HasLongJobs ServerHasLongJobs AESModuleKeys NTokenCmds
JobFragmentation LongJobsPreferred Type2Smartcard
 module type code     0
 product name         nFast server
 device name
 EnquirySix version   4
 impath kx groups
 feature ctrl flags   none
 features enabled     none
 version serial       0
 level six flags      none
 remote server port   9004
 kneti hash           5ebd9844cd9896ed40829c3bafa91a5bbba7a886

Module #1:
 enquiry reply flags  UnprivOnly
 enquiry reply level  Six
 serial number        BD10-03E0-D947
 mode                 operational
 version              12.50.11
 speed index          478
 rec. queue           22..50
 level one flags      Hardware HasTokens SupportsCommandState
 version string       12.50.11-270-fb3b87dd465b6f6e53d9f829fc034f8be2dafd13 2019/05/16 22:02:33 BST,
Bootloader: 1.2.3, Security Processor: 12.50.11 , 12.60.10-708-ea4dc41d
 checked in           000000005cddcfe9 Thu May 16 21:02:33 2019
 level two flags      none
 max. write size      8192
 level three flags    KeyStorage
 level four flags     OrderlyClearUnit HasRTC HasNVRAM HasNSOPermsCmd ServerHasPollCmds FastPollSlotList
HasSEE HasKLF HasShareACL HasFeatureEnable HasFileOp HasLongJobs ServerHasLongJobs AESModuleKeys NTokenCmds
JobFragmentation LongJobsPreferred Type2Smartcard ServerHasCreateClient HasInitialiseUnitEx
AlwaysUseStrongPrimes Type3Smartcard HasKLF2
module type code     12
 product name         nC3025E/nC4035E/nC4335N
 device name          Rt1
 EnquirySix version   7
 impath kx groups     DHPrime1024 DHPrime3072 DHPrime3072Ex
 feature ctrl flags   LongTerm
 features enabled     StandardKM EllipticCurve ECCMQV AcceleratedECC HSMBaseSpeed
 version serial       37
 connection status    OK
 connection info      esn = BD10-03E0-D947; addr = INET/10.194.148.36/9004; ku hash =
383666ac8d0a8062519b9baa964d0af8014e5d8d, mech = Any
 image version        12.60.10-507-ea4dc41d
 level six flags      none
 max exported modules 100
 rec. LongJobs queue  21
 SEE machine type     PowerPCELF
 supported KML types  DSAp1024s160 DSAp3072s256
 using impath kx grp  DHPrime3072Ex
 active modes         UseFIPSApprovedInternalMechanisms AlwaysUseStrongPrimes FIPSLevel3Enforcedv2
 hardware status      OK
```

#### 2.3.3.3. nfkminfo

The following pod_nfkminfo_app.yaml file shows how to run the `nfkminfo` command which shows information about the current security world.

```
kind: Pod
apiVersion: v1
metadata:
  name: ncop-test-nfkminfo
  labels:
    app: nshield
spec:
  imagePullSecrets:
    - name: regcred
  containers:
    - name: ncop
      command:
        - sh
        - '-c'
        - /opt/nfast/bin/nfkminfo && sleep 3600
      image: <docker-registry-address>/nshield-app
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      volumeMounts:
        - name: ncop-kmdata
          mountPath: /opt/nfast/kmdata
        - name: ncop-sockets
          mountPath: /opt/nfast/sockets
    - name: ncop-hwsp
      image: <docker-registry-address>/nshield-hwsp
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      volumeMounts:
        - name: ncop-config
          mountPath: /opt/nfast/kmdata/config
        - name: ncop-hardserver
          mountPath: /opt/nfast/kmdata/hardserver.d
        - name: ncop-sockets
          mountPath: /opt/nfast/sockets
  volumes:
    - name: ncop-config
      configMap:
        name: config
        defaultMode: 420
    - name: ncop-hardserver
      emptyDir: {}
    - name: ncop-kmdata
      persistentVolumeClaim:
        claimName: nfast-kmdata
    - name: ncop-sockets
      emptyDir: {}
```

In this example, <docker_registry-address> is the address of your internal docker registry server.

- Deploy the pod.

```
% kubectl apply -f pod_nfkminfo_app.yaml
```

- Check if the Pod is running.

```
% kubectl get pods
```

You should see the deployment taking place. Wait 10 seconds and run the command again until the status is Running. This will also let you know if there are any errors. If there are errors, run the following command:

```
% kubectl describe pod ncop-test-nfkminfo
```

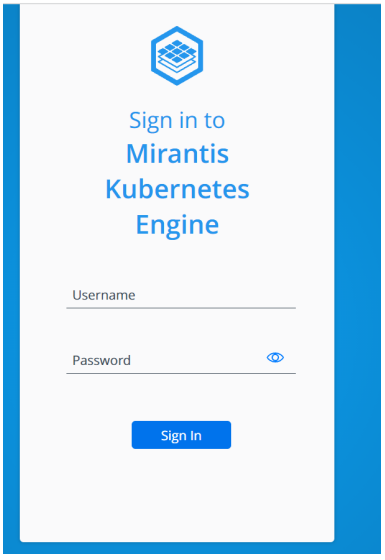
- Check if the `nfkminfo` command ran successfully.

```
% kubectl logs pod/ncop-test-nfkminfo ncop

World
 generation  2
 state       0x3737000c Initialised Usable Recovery !PINRecovery !ExistingClient RTC NVRAM FTO
AlwaysUseStrongPrimes !DisablePKCS1Padding !PpStrengthCheck !AuditLogging SEEDebug AdminAuthRequired
 n_modules   1
 hknso       d2ac1dba1d16223a1ec0b0084b318dd49886988c
 hkm         7f07f1feeccf930031c30be59fc8157954b90dbb (type Rijndael)
 hkmwk       c2be99fe1c77f1b75d48e2fd2df8dffc0c969bcb
 hkre        6e0bf62ed6b4d80acddf3a299d3f42c0bbb5ad95
 hkra        3c59d561b8ae95de9d5882bc036cfd93e3fc7b23
 hkfips      3e25ab9a86a304fb2f67943d343900d8a2ac5d13
 hkmc        bfc8fbe15696c172b7efa265b807ea891c958200
 hkrtc       c038c6e683f4c0c19b7a99b5398514828772afd9
 hknv        522ee0d2f37779984ef58674e3aa8e66ca9d726a
 hkdsee      3cca644c83678164d823d069b288ac28afa5a7a6
 hkfto       d03b7fa8cc1c67a9d934c38e966ee6321f5faffb
 hkmnull     0100000000000000000000000000000000000000
 ex.client   none
 k-out-of-n  1/1
 other quora m=1 r=1 nv=1 rtc=1 dsee=1 fto=1
 createtime  2021-09-21 14:37:40
 nso timeout 10 min
 ciphersuite DLf3072s256mAEScSP800131Ar1
 min pp      0 chars
 mode        fips1402level3

Module #1
 generation 2
 state      0x2 Usable
 flags      0x10000 ShareTarget
 n_slots    4
 esn        530E-02E0-D947
 hkml       016d74cd9d76a8f976dfb0b17f1f8d6de1b350e6

Module #1 Slot #0 IC 1
 generation    1
 phystype      SmartCard
 slotlistflags 0x2 SupportsAuthentication
 state         0x4 Admin
 flags         0x10000
 shareno       1
```

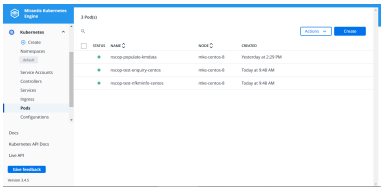
```
 shares        LTNSO(PIN) LTM(PIN) LTR(PIN) LTFIPS LTNV(PIN) LTRTC(PIN) LTDSEE(PIN) LTFTO(PIN)
 error         OK
No Cardset

Module #1 Slot #1 IC 0
 generation    1
 phystype      SoftToken
 slotlistflags 0x0
 state         0x2 Empty
 flags         0x0
 shareno       0
 shares
 error         OK
No Cardset

Module #1 Slot #2 IC 29
 generation    1
 phystype      SmartCard
 slotlistflags 0x180002 SupportsAuthentication DynamicSlot Associated
 state         0x6 Unidentified
 flags         0x0
 shareno       1
 shares
 error         OK
No Cardset

Module #1 Slot #3 IC 0
 generation    1
 phystype      SmartCard
 slotlistflags 0x80002 SupportsAuthentication DynamicSlot
 state         0x2 Empty
 flags         0x0
 shareno       0
 shares
 error         OK
No Cardset

No Pre-Loaded Objects
```

## 2.3.4. Test MKE Web Interface

- Open a web browser and go to https://<host-node-ip-address>



- Log in with the account created during MKE installation.
- Navigate on the left pane to Kubernetes > Pods.
- The pods created should be shown running on this page.



- The other kubernetes objects generated in this integration can be viewed under the Kubernetes tab.

# Chapter 3. Additional resources and related products

## 3.1. Video

## 3.2. nShield Connect

## 3.3. nShield as a Service

## 3.4. nShield Container Option Pack

## 3.5. Entrust digital security solutions

## 3.6. nShield product documentation