THALES

Hyperledger Fabric (Blockchain): Integration Guide

THALES LUNA HSM AND DPOD LUNA CLOUD HSM

Document Information

Document Part Number	007-000084-001
Revision	E
Release Date	25 February 2021

Trademarks, Copyrights, and Third-Party Software

CONTENTS

Overview

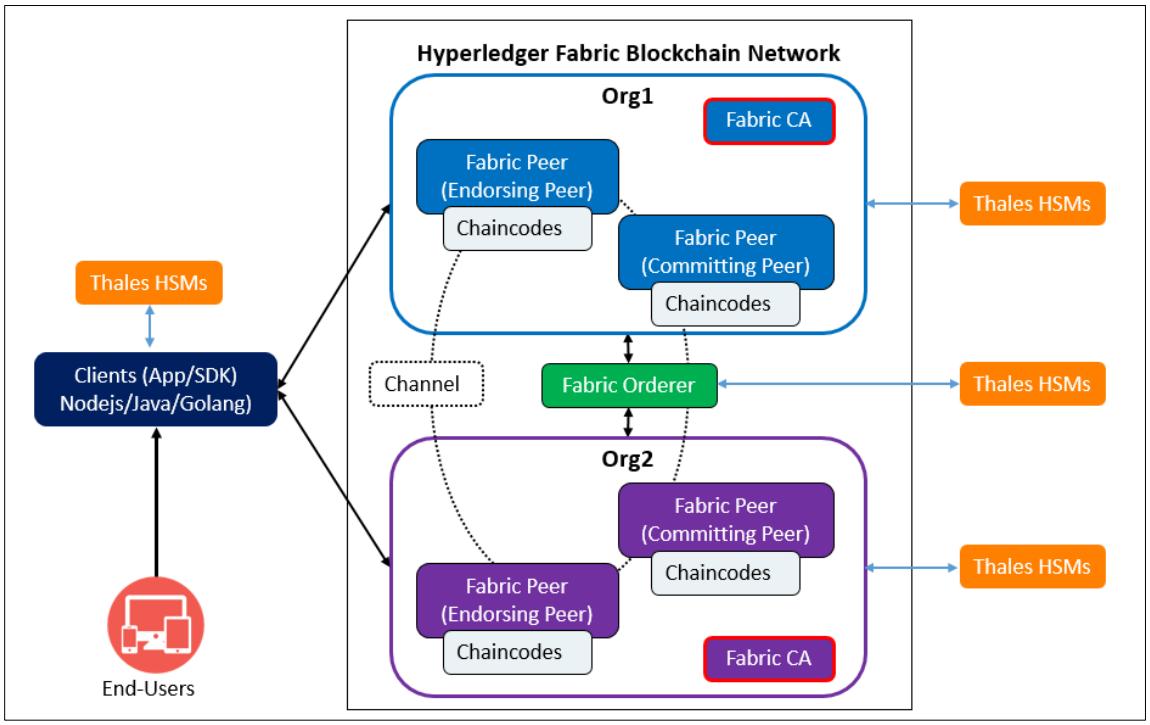
This guide explains how to use Luna HSM devices or Luna Cloud HSM services to securely store Hyperledger Fabric Admin User, Peer, and Orderer private keys. It guides administrator users through configuring the Blockchain Crypto Service Provider (BCCSP) to use Luna HSM devices or Luna Cloud HSM services for generating and securing the Blockchain Admin. Peer, and Orderer application security keys. Thales HSMs integrate with Hyperledger Fabric to generate 384-bit ECDSA signing key pairs for blockchain identities and provide security by protecting the identity private keys. Thales HSMs can also integrate with the Hyperledger Fabric SDK client for node.js and java. This allows users to secure the signing keys used by Hyperledger Fabric clients.

The benefits of integrating Hyperledger Fabric with Thales HSMs include:

- Secure generation, storage, and protection of the signing private key on FIPS 140-2 level 3 validated hardware.
- Full life cycle management of the keys.
- Access to the HSM audit trail*.
- The ability to use cloud services with confidence.

*Luna Cloud HSM services do not have access to the secure audit trail.

Understanding Hyperledger Fabric Blockchain Network



Hyperledger Fabric is one of the blockchain projects within Hyperledger. Like other blockchain technologies, it has a ledger, uses smart contracts, and is a system by which participants manage their transactions. Hyperledger Fabric is different from other blockchain systems as it is private and permissioned. Rather than an

open permission-less system that allows unknown identities to participate in the network, the members of a Hyperledger Fabric network enroll through a Membership Service Provider (MSP).

The default MSP implementation in Hyperledger Fabric uses X.509 certificates as identities, thus adopting a traditional Public Key Infrastructure (PKI) hierarchical model. Fabric CA is the certificate authority which issues the certificates to Peers, Orderers, and Users. Peers, Orderers and Users are different actors in a Blockchain network. Each of these actors has an identity that is encapsulated in an X.509 digital certificate. These identities matter because they determine the exact permissions over the resources that actors have in a Blockchain network. Most importantly, an identity must be verifiable (a real identity, in other words), and for this reason it must come from an authority trusted by the system. A membership service provider (MSP) is the means to achieve this in Hyperledger Fabric. Thales HSMs are used to generate the key pairs for these identities (Peers, Orderers, and Users).

Certified Platforms

Certified Platforms for Luna HSM

Certified Platforms for Luna Cloud HSM

Certified Platforms for Luna HSM

The following platforms are certified for integrating Hyperledger Fabric with Luna HSM:

HSM Type	Platforms	Hyperledger
Luna HSM	RHEL	Hyperledger Fabric release-1.4
Luna HSM	CentOS	Hyperledger Fabric 1.4 Client SDK Node
Luna HSM	CentOS	Hyperledger Fabric 1.4 Client SDK Java

Luna HSM: Luna HSM appliances are purposefully designed to provide a balance of security, high performance, and usability that makes them an ideal choice for enterprise, financial, and government organizations. Luna HSMs physically and logically secure cryptographic keys and accelerate cryptographic processing. Luna HSM on premise offerings include Luna Network HSM, Luna PCIe HSM, and Luna USB HSM. Luna HSMs are also available for access as an offering from cloud service providers such as IBM cloud HSM and AWS cloud HSM classic.

Certified Platforms for Luna Cloud HSM

The following platforms are certified for integrating Hyperledger Fabric with Luna Cloud HSM:

HSM Type	Platforms	Hyperledger
Luna Cloud HSM	RHEL	Hyperledger Fabric release-1.4
Luna Cloud HSM	CentOS	Hyperledger Fabricn1.4 Client SDK Node
Luna Cloud HSM	CentOS	Hyperledger Fabric 1.4 Client SDK Java

Luna Cloud HSM: Luna Cloud HSM services provide on-demand HSM and Key Management services through a simple graphical user interface. With Luna Cloud HSM, security is simple, cost effective and easy to manage because there is no hardware to buy, deploy and maintain. As an Application Owner, you click and deploy services, generate usage reports, and maintain just the services you need.

Prerequisites

Before you proceed with the integration, complete the following tasks:

> Configure Luna HSM

> Create additional partitions for Luna HSM

> Configure Luna Cloud HSM service

> Create additional partitions for Luna Cloud HSM

> Set up Hyperledger Fabric, Fabric CA, and Fabric-Samples

Configure Luna HSM

If you are using Luna HSM:

1. Verify that the HSM is set up, initialized, provisioned, and ready for deployment. Refer to Luna HSM documentation for more information.
2. Create a partition that will be later used by Hyperledger.
3. If using a Luna Network HSM, register a client for the system and assign the client to the partition to create an NTLS connection. Initialize the Crypto Officer and Crypto User roles for the registered partition.

   NOTE: If you are using PCIe or USB HSMs you don't need to create NTLS connection.

4. Ensure that the partition is successfully registered and configured. The command to see the registered partitions is:

```
/usr/safenet/lunaclient/bin/lunacm

lunacm (64-bit) v10.3.0-275. Copyright (c) 2020 SafeNet. All rights reserved.
```

```
Available HSMs:
Slot Id ->              0
Label ->                org.example.com
Serial Number ->        1238696044924
Model ->                LunaSA 7.4.0
Firmware Version ->     7.4.0
Configuration ->        Luna User Partition With SO (PW) Signing With
                        Cloning Mode
Slot Description ->     Net Token Slot
FM HW Status ->         Non-FM
```

5. For PED-authenticated HSM, enable partition policies 22 and 23 to allow activation and auto-activation.

> **NOTE:** Refer to Luna HSM documentation for detailed steps on creating NTLS connection, initializing the partitions, and assigning various user roles.

Create additional partitions for Luna HSM

Create additional partitions if you are using Hyperledger Fabric or Hyperledger Fabric Client using the steps given below:

> Create partitions for Hyperledger Fabric

> Create partitions for Hyperledger Fabric Client

Create partitions for Hyperledger Fabric

If you are using Hyperledger Fabric you need to create additional partitions using the following steps:

1. Create a separate partition for each Peer and Orderer organization in Hyperledger on the Luna HSM and label them as follows.
   - org1.example.com
   - org2.example.com
   - orderer.example.com

   > **NOTE:** For demonstration purposes, we have registered a single client for all the separate partitions with crypto officer password as *userpin* for the different organizations. For a production environment where each Identity (Peer, Orderer, and User) is running on separate systems, we recommend that you register each client Identity with their own HSM partition.

2. Ensure that the partition is successfully registered and configured. The command to see the registered partitions is:

```
/usr/safenet/lunaclient/bin/lunacm
lunacm (64-bit) v10.3.0-275. Copyright (c) 2020 SafeNet. All rights reserved.
```

```
Available HSMs:
Slot Id ->              0
Label ->                org1.example.com
Serial Number ->        1238696044924
Model ->                LunaSA 7.4.0
Firmware Version ->     7.4.0
Configuration ->        Luna User Partition With SO (PW) Signing With
                        Cloning Mode
Slot Description ->     Net Token Slot
FM HW Status ->         Non-FM

Slot Id ->              1
Label ->                org2.example.com
Serial Number ->        1238696044925
Model ->                LunaSA 7.4.0
Firmware Version ->     7.4.0
Configuration ->        Luna User Partition With SO (PW) Signing With
                        Cloning Mode
Slot Description ->     Net Token Slot
FM HW Status ->         Non-FM

Slot Id ->              2
Label ->                orderer.example.com
Serial Number ->        1238696044926
Model ->                LunaSA 7.4.0
Firmware Version ->     7.4.0
Configuration ->        Luna User Partition With SO (PW) Signing With
                        Cloning Mode
Slot Description ->     Net Token Slot
FM HW Status ->         Non-FM
```

Create partitions for Hyperledger Fabric Client

If you are using Hyperledger Fabric Client you need to create additional partitions using the following steps:

1. Create a partition on the HSM that will be later used by Hyperledger Fabric Client.

2. Ensure that the partition is successfully registered and configured. The command to see the registered partitions is:

```
# /usr/safenet/lunaclient/bin/lunacm
lunacm (64-bit) v7.3.0-165. Copyright (c) 2018 SafeNet. All rights reserved.


Available HSMs:


Slot Id ->              0
Label ->                fabric-sdk
Serial Number ->        1280780175900
Model ->                LunaSA 7.3.0
 Firmware Version ->     7.3.0
 Configuration ->        Luna User Partition With SO (PW) Key Export
                         With Cloning Mode
Slot Description ->     Net Token Slot
Current Slot Id: 0
```

> **NOTE:** The end-2-end execution example uses the label “fabric-sdk” and the password “userpin”. We recommend setting the password as per your organization’s security policy if implementing in a production environment.

Set up Luna HSM High-Availability

Refer to Luna HSM documentation for HA steps and details regarding configuring and setting up two or more HSM boxes on host systems. You must enable the HAOnly setting in HA for failover to work so that if the primary goes down due to any reason, all calls automatically route to the secondary until the primary recovers and restarts.

Configure Luna Cloud HSM service

You can configure Luna Cloud HSM service in the following ways:

- Configure standalone Cloud HSM service using minimum client package
- Configure standalone Cloud HSM service using full Luna client package
- Configure Luna HSM and Luna Cloud HSM service in hybrid mode

> **NOTE:** Luna Client v10.x or higher is required for configuring Luna HSM device and Luna Cloud HSM service in hybrid mode.

Configure standalone Cloud HSM service using minimum client package

To configure Luna Cloud HSM service using minimum client package:

1. Transfer the downloaded .zip file to your Client workstation using pscp, scp, or other secure means.
2. Extract the .zip file into a directory on your client workstation.

3. Extract or untar the appropriate client package for your operating system. Do not extract to a new subdirectory; place the files in the client install directory.

   ```
   [Linux]

   cvclient-min.tar

   # tar -xvf cvclient-min.tar
   ```

4. Run the setenv script to create a new configuration file containing information required by the Luna Cloud HSM service.

   ```
   [Linux]

   Source the setenv script.

   # source ./setenv
   ```

5. Run the LunaCM utility and verify the Cloud HSM service is listed.

Configure standalone Cloud HSM service using full Luna client package

To configure Luna Cloud HSM service using full Luna client package:

1. Transfer the downloaded .zip file to your Client workstation using pscp, scp, or other secure means.
2. Extract the .zip file into a directory on your client workstation.
3. Extract or untar the appropriate client package for your operating system. Do not extract to a new subdirectory; place the files in the client install directory.

   ```
   [Linux]

   cvclient-min.tar

   # tar -xvf cvclient-min.tar
   ```

4. Run the setenv script to create a new configuration file containing information required by the Luna Cloud HSM service.

   ```
   [Linux]

   Source the setenv script.

   # source ./setenv
   ```

5. Copy the server and partition certificates from the Cloud HSM service client directory to Luna client certificates directory:

   > **NOTE:** Skip this step for Luna Client v10.2 or higher.

   Cloud HSM Certificates:

   ```
   server-certificate.pem

   partition-ca-certificate.pem

   partition-certificate.pem
   ```

   LunaClient Certificate Directory:

   ```
   [Linux default location for Luna Client]

   /usr/safenet/lunaclient/cert/
   ```

6. Open the configuration file from the Cloud HSM service client directory and copy the XTC and REST section.

```
[Linux]

Chrystoki.conf
```

7. Edit the Luna Client configuration file and add the XTC and REST sections copied from Cloud HSM service client configuration file.
8. Change server and partition certificates path from step 5 in XTC and REST sections. Do not change any other entries provided in these sections.

> **NOTE:** Skip this step for Luna Client v10.2 or higher.

[XTC]

```
. . .
PartitionCAPath=<LunaClient_cert_directory>\partition-ca-certificate.pem
PartitionCertPath00=<LunaClient_cert_directory>\partition-certificate.pem
. . .

[REST]
. . .
SSLClientSideVerifyFile=<LunaClient_cert_directory>\server-certificate.pem
. . .
```

9. Edit the following entry from the Misc section and update the correct path for the plugins directory:

```
Misc = {
...
PluginModuleDir=<LunaClient_plugins_directory>;
...
}

[Linux Default]

/usr/safenet/lunaclient/plugins/
```

10. Save the configuration file and delete the extracted Cloud HSM service client directory.
11. Reset the ChrystokiConfigurationPath environment variable and point back to the location of the Luna Client configuration file.

```
# export ChrystokiConfigurationPath=/etc/
```

12. Run the LunaCM utility and verify that the Cloud HSM service is listed. In hybrid mode, both Luna and Cloud HSM service will be listed.

> **NOTE:** Follow the Luna Cloud HSM documentation for detailed steps for creating service, client, and initializing various user roles.

Configure Luna HSM and Luna Cloud HSM service in hybrid mode

To configure Luna HSM and Luna Cloud HSM service in hybrid mode, follow the steps mentioned under the Configure standalone Cloud HSM service using full Luna client package section above.

NOTE: Luna Client v10.x or higher is required for configuring Luna HSM device and Luna Cloud HSM service in hybrid mode.

Use Luna Cloud HSM service in FIPS mode

Cloud HSM service operates in both FIPS and non-FIPS mode. If your organization requires non-FIPS algorithms for your operations, enable the Allow non-FIPS approved algorithms check box when configuring your Cloud HSM service. The FIPS mode is enabled by default. Refer to the Mechanism List in the SDK Reference Guide for more information about available FIPS and non-FIPS algorithms.

Create additional partitions for Luna Cloud HSM

Create additional partitions if you are using Hyperledger Fabric or Hyperledger Fabric Client using the steps given below:

> Create partitions for Hyperledger Fabric

> Create partitions for Hyperledger Fabric Client

Create partitions for Hyperledger Fabric

If you are using Hyperledger Fabric you need to create additional partitions using the following steps:

1. Create the following Luna Cloud services in Luna Cloud HSM:
   - org1.example.com
   - org2.example.com
   - orderer.example.com

   NOTE: It is recommended to use minimal client package if you are using three partitions on the same host.
2. For each service, create a Linux service client and download the zip to the host system.
3. Make the following directories on the client machine:

   ```
   # mkdir -p /etc/hyperledger/fabric/dpod/org1.example.com
   # mkdir -p /etc/hyperledger/fabric/dpod/org2.example.com
   # mkdir -p /etc/hyperledger/fabric/dpod/orderer.example.com
   ```
4. Unzip the client zip files for org1.example.com, org2.example.com and orderer.example.com in to their respective directories.
5. Complete the following processes:
   - Initialize the partition, Crypto Officer, and Crypto User roles.
   - Set the ChrystokiConfigurationPath environment variable to point to the Chrystoki.conf file.
   - Set the partition password to "userpin" for partition labels org1.example.com, org2.example.com and orderer.example.com using LunaCM.

NOTE: It is recommended to use separate partitions for all Peers, Orderers, and Users. For this example make sure to initialize all partitions with the label provided above and that “userpin” is used as a partition password to successfully complete the demo using byfn.

You need to set the partition password as per your organization security policy for a production environment.

Create partitions for Hyperledger Fabric Client

If you are using Hyperledger Fabric Client you need to create additional partitions using the following steps:

1. Create the fabric-sdk Luna Cloud service in Luna Cloud HSM.
2. After creating the service, create a Linux service client and download the zip to the host system.
3. Make the following directory on the client machine:

   ```
   # mkdir -p /usr/safenet/dpod
   ```
4. Unzip the client zip files for fabric-sdk in to the directory.
5. Follow the instructions in the *Application Owner Quick Start Guid*e and complete the following tasks:
   - Initialize the partition, Security Officer, Crypto Officer, and Crypto User roles.
   - Set the ChrystokiConfigurationPath environment variable or create the soft link /etc/Chrystoki.conf to point to the Chrystoki.conf file.
   - Set the partition password to "userpin" for partition label fabric-sdk using LunaCM.

NOTE: The end-2-end execution example uses the label “fabric-sdk” and the password “userpin”. We recommend setting the password as per your organization security policy if implementing in a production environment.

Set up Hyperledger Fabric, Fabric CA, and Fabric-Samples

Both Hyperledger Fabric and the Fabric CA client must be installed to complete the integration with Luna HSM. Complete the following tasks to install both Hyperledger Fabric and the Fabric CA client on the Linux host system.

> Install Hyperledger Fabric and Fabric CA prerequisite libraries

> Install and set up Golang

> Install and set up Docker and Docker Compose

> Install and configure Hyperledger Fabric and Fabric CA

> Install and configure Hyperledger Fabric-Samples

Install Hyperledger Fabric and Fabric CA prerequisite libraries

To install Hyperledger Fabric and Fabric CA prerequisite libraries:

Ubuntu

```
# sudo apt-get install git curl alien python-pip libltdl-dev
```

RHEL/CentOS

```
# sudo yum install git curl alien python-pip libtool-ltdl-devel
```

Install and set up Golang

Hyperledger Fabric uses Go programming language for many of its components. Install **golang** using the following URLs:

> Installation steps: https://golang.org/doc/install

> Download binaries: https://golang.org/dl/

Install and set up Docker and Docker Compose

To install Docker and Docker Compose:

1. Install Docker and Docker Compose on the host system by following the instructions available in Docker documentation:
   - Docker Installation: https://docs.docker.com/engine/install/
   - Docker Compose Installation: https://docs.docker.com/compose/install/
2. Configure the docker so that sudo is not required to run further commands:

```
# sudo gpasswd -a $USER docker
# newgrp docker
```

3. Ensure that the go executable is in the PATH.

```
# export PATH=/usr/local/go/bin:$PATH
```

Install and configure Hyperledger Fabric and Fabric CA

To install and configure Hyperledger Fabric and Fabric CA:

1. Set the **GOPATH**, the value will be a directory tree child of your development workspace.

```
# export GOPATH=/opt/gopath
# mkdir -p $GOPATH/src/github.com/hyperledger
# cd $GOPATH/src/github.com/hyperledger
```

2. Create the Hyperledger Fabric repository by executing the following command.

```
# git clone https://github.com/hyperledger/fabric
# cd fabric
# git checkout -b release-1.4 origin/release-1.4  (Hypeledger Fabric)
```

   For Client SDK run:

```
# git checkout -b v1.4.0 v1.4.0      (Hypeledger Fabric Client SDK)
```

> **NOTE:** The instructions were developed against the tag release-1.4 for Hyperledger Fabric and v1.4.0 for Hyperledger Fabric Client SDK. It is advisable to use the repo checkout from these tags as the instructions may not be compatible with the latest check-in in the master branch of Fabric and Fabric Client SDK.

3. Build the docker images and executables.

```
# GO_TAGS=pkcs11 make peer orderer cryptogen configtxgen configtxlator
```

4. Clone the fabric-ca project and build the fabric-ca-client binary.

```
# cd $GOPATH/src/github.com/hyperledger
```

```
# git clone https://github.com/hyperledger/fabric-ca
# cd fabric-ca
# git checkout -b release-1.4 origin/release-1.4 (Hypeledger Fabric)
# make fabric-ca-client
```

For Client SDK run:

```
# git checkout -b v1.4.0 v1.4.0 (Hypeledger Fabric Client SDK)
```

> **NOTE:** The instructions were developed against the tag release-1.4 for Hyperledger Fabric and v1.4.0 for Hyperledger Fabric Client SDK. It is advisable to use the repo checkout from these tags as the instructions may not be compatible with the latest check-in in the master branch of Fabric and Fabric Client SDK.

Install and configure Hyperledger Fabric-Samples

To install and configure Hyperledger Fabric-Samples:

1. Clone the fabric-samples project.

```
# cd $GOPATH/src/github.com/hyperledger
# git clone https://github.com/hyperledger/fabric-samples/
# cd fabric-samples/
# git checkout -b release-1.4 origin/release-1.4
```

2. Create directory $GOPATH/src/github.com/hyperledger/fabric-samples/bin and $GOPATH/src/github.com/hyperledger/fabric-samples/first-network/bin

```
# mkdir $GOPATH/src/github.com/hyperledger/fabric-samples/bin
# mkdir $GOPATH/src/github.com/hyperledger/fabric-samples/first-network/bin
```

3. Copy the executables to $GOPATH/src/github.com/hyperledger/fabric-samples/bin directory.

```
# cp $GOPATH/src/github.com/hyperledger/fabric-ca/bin/*
$GOPATH/src/github.com/hyperledger/fabric-samples/bin/
# cp $GOPATH/src/github.com/hyperledger/fabric/.build/bin/*
$GOPATH/src/github.com/hyperledger/fabric-samples/bin
```

4. Copy the executables to $GOPATH/src/github.com/hyperledger/fabric-samples/first-network/bin directory.

```
# cp $GOPATH/src/github.com/hyperledger/fabric/.build/bin/*
$GOPATH/src/github.com/hyperledger/fabric-samples/first-network/bin
# cp $GOPATH/src/github.com/hyperledger/fabric-ca/bin/*
$GOPATH/src/github.com/hyperledger/fabric-samples/first-network/bin
```

Integrating Thales Luna HSM with Hyperledger Fabric

To integrate Thales Luna HSM with Hyperledger Fabric, you need to complete the following tasks:

> Generate a CSR using fabric-ca-client and PKCS11 BCCSP for an MSP directory

> Configure peer nodes

> Configure Orderer nodes

NOTE: Refer to Build Your First Network(BYFN) using Luna HSM section below for an example of end to end integration of Hyperledger Fabric with Luna HSM.

Generate a CSR using fabric-ca-client and PKCS11 BCCSP for an MSP directory

The fabric-ca-client utility can be used to generate certificate signing requests for Peers, Orderers, and Users in their respective MSP directories. The fabric-ca-client utility uses the BCCSP to generate crypto material. If the BCCSP is configured to use the PKCS11 implementation of the BCCSP then it can be used to generate keys on the Thales Luna HSM. The fabric-ca-client BCCSP can be configured in the ~/.fabric-ca-client/fabric-ca-client-config.yaml file. This involves following steps:

> Configure BCCSP to use the Luna PKCS#11 API

> Generate the cryptographic keys using the gencsr command

Configure BCCSP to use the Luna PKCS#11 API

1. Modify the BCCSP section in ~/.fabric-ca-client/fabric-ca-client-config.yaml:

   NOTE: If the ~/.fabric-ca-client/fabric-ca-client-config.yaml file is not present run the following command to generate it: `# fabric-ca-client gencsr`

```
bccsp:
    default: PKCS11
    sw:
        hash: SHA2
        security: 256
        filekeystore:
            keystore: msp/keystore
    pkcs11:
        hash: SHA2
        security: 384
        library: /usr/safenet/lunaclient/lib/libCryptoki2_64.so
        label: org1.example.com
        pin: userpin
        filekeystore:
            keystore: msp/keystore
```

2. Add a keyrequest setting to the csr section of ~/.fabric-ca-client/fabric-ca-client-config.yaml file to specify the key size as follows

```
csr:
  cn:
  keyrequest:
    algo: ecdsa
    size: 384
```

Generate the cryptographic keys using the gencsr command

To generate ECDSA private keys on the HSM using the PKCS#11 BCCSP and create a CSR in the msp/signcerts directory for the private key, you need to undertake the following process:

1. The PKCS11 values in the fabric-ca-client-config.yaml file can be left blank and specified using environment variables. Alternatively, the values in the file can be overridden using the following environment variables:

```
# export FABRIC_CA_CLIENT_BCCSP_DEFAULT=PKCS11

# export FABRIC_CA_CLIENT_BCCSP_PKCS11_LABEL=<HSM Partition Label>

# export FABRIC_CA_CLIENT_BCCSP_PKCS11_PIN=<Partition Password>

# export FABRIC_CA_CLIENT_BCCSP_PKCS11_LIBRARY=<HSM PKCS11 Library>
```

2. The command to generate CSRs is: `# ./fabric-ca-client gencsr [options]`

   Where:

```
--csr.cn string           The common name field of the certificate signing
   request

--mspdir string        Membership Service Provider directory (default "msp")

--csr.names stringSlice      A list of comma-separated CSR names of the form
                             <name>=<value> (e.g. C=CA,OU=peer)
```

> **NOTE:** When generating the CSR request you need to ensure that PKCS11 BCCSP settings are correct or set the correct values in environment variables for the Peer/Orderer. Also, make sure to specify the correct CN, MSP directory and Names mainly OU (peer/orderer/client) in fabric-ca-client options.

3. To generate the key for orderer.example.com, execute the following command:

```
# ./fabric-ca-client gencsr --csr.cn orderer.example.com --mspdir
ordererOrganizations/orderer.example.com/orderers/orderer.example.com/msp --
csr.names "C=US,ST=California,L=San Francisco,OU=orderer"
```

   Options and exported variables should be changed accordingly as per the requirement for generating the specific certificate signing request. Submit the CSR to your CA to obtain the signed certificate for the Peer/Orderer/User and place the signed certificate in their respective msp/signcerts directories.

4. To generate the CSR for the peer0 of org1.example.com, execute the following commands:

```
# export FABRIC_CA_CLIENT_BCCSP_DEFAULT=PKCS11

# export FABRIC_CA_CLIENT_BCCSP_PKCS11_LABEL=org1.example.com

# export FABRIC_CA_CLIENT_BCCSP_PKCS11_PIN=userpin

# export
FABRIC_CA_CLIENT_BCCSP_PKCS11_LIBRARY=/usr/safenet/lunaclient/lib/libCryptoki2_
64.so

# ./fabric-ca-client gencsr --csr.cn peer0.org1.example.com --mspdir ./crypto-
config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp --
csr.names "C=US,ST=California,L=San Francisco,OU=peer"
```

   The above command generates the key pair for peer0.org1.example.com on HSM partition org1.example.com and creates the CSR ./crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp/signcerts/peer0.org1.example.com.csr. Copy the CSR and send it to your CA to obtain a signed certificate and then place the certificate in same directory.

5. Similarly, use the following command for generating the certificate request for Admin User:

```
# ./fabric-ca-client gencsr --csr.cn Admin@org1.example.com --mspdir ./crypto-
config/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp --
csr.names "C=US,ST=California,L=San Francisco,OU=client"
```

Configure peer nodes

To configure BCCSP to use the Luna HSM for peer nodes:

1. Change the core.yaml file and specify PKCS11 as the default in the BCCSP section as follows:

```
BCCSP:
  Default: PKCS11
  PKCS11:
    Library: /usr/safenet/lunaclient/lib/libCryptoki2_64.so
    Label: org1.example.com
    Pin: userpin
    Hash: SHA2
    Security: 384
```

   Alternatively, the PKCS11 values in the core.yaml file can be left blank and specified using environment variables, or the values in the file can be overridden using these environment variables. The following environment variables can be used to change the configuration settings of the core.yaml BCCSP.

```
# export CORE_PEER_BCCSP_DEFAULT=PKCS11
# export CORE_PEER_BCCSP_PKCS11_LABEL=<HSM Partition Label>
# export CORE_PEER_BCCSP_PKCS11_PIN=<Partition Password>
# export CORE_PEER_BCCSP_PKCS11_LIBRARY=<HSM PKCS11 Library>
```

Configure Orderer nodes

To configure BCCSP to use the Luna HSM for orderer nodes:

1. Change the orderer.yaml file and specify PKCS11 as the default in the BCCSP section as follows:

```
BCCSP:
  Default: PKCS11
  PKCS11:
    Library: /usr/safenet/lunaclient/lib/libCryptoki2_64.so
    Label: orderer.example.com
    Pin: userpin
    Hash: SHA2
    Security: 384
```

   Alternatively, the PKCS11 values in the orderer.yaml file can be left blank and specified using environment variables or the values in the file can be overridden using these environment variables. The following environment variables can be used to change the configuration settings of the orderer.yaml BCCSP.

```
# export ORDERER_GENERAL_BCCSP_DEFAULT=PKCS11
# export ORDERER_GENERAL_BCCSP_PKCS11_LABEL=<HSM Partition Label>
# export ORDERER_GENERAL_BCCSP_PKCS11_PIN=<Partition Password>
# export ORDERER_GENERAL_BCCSP_PKCS11_LIBRARY=<HSM PKCS11 Library>
```

You have completed the integration and all the required components are installed on the platform(s) on which you'll be developing Blockchain applications and/or operating Hyperledger Fabric. All key materials have been generated on a Thales Luna HSM using the PKCS11 BCCSP. You have configured the core.yaml file and the orderer.yaml file to use the PKCS11 BCCSP and mounted them as a volume in Peer and Orderer configuration files.

Now start Fabric CA, Orderer and Peers to create channel artifacts and run the channel. Join the nodes in the channel to create permissioned Blockchain network. This is achieved by assigning identity certificates to the member orgs and their nodes which will be used to identify themselves and conduct transactions in the network. A library of X509 certificates (commonly termed as cryptographic material) gets created for associated Peer/Orderer nodes using the PKCS11 BCCSP which in turn generates all key pairs on the Thales Luna HSM.

> **NOTE:** Ensure to specify the required environment variable to use PKCS11 BCCSP in the script that is required to be executed for creating the Blockchain.

Build Your First Network(BYFN) using Luna HSM

To build your first network using a Luna HSM:

1. Open ~/.fabric-ca-client/fabric-ca-client-config.yaml file and change the bccsp section to:

```
bccsp:
    default: PKCS11
    sw:
        hash: SHA2
        security: 256
        filekeystore:
            keystore: msp/keystore
    pkcs11:
        hash: SHA2
        security: 384
        library: /usr/safenet/lunaclient/lib/libCryptoki2_64.so
        label:
        pin:
```

2. Add a keyrequest setting to the csr section of ~/.fabric-ca-client/fabric-ca-client-config.yaml file to specify the key size as follows:

```
csr:
  cn:
  keyrequest:
    algo: ecdsa
    size: 384
```

3. Go to first-network directory

```
# cd $GOPATH/src/github.com/hyperledger/fabric-samples/first-network
```

4. Copy the below script and save it as gencerts.sh to generate crypto material on Thales Luna HSM. It works in conjunction with the cryptogen tool. The script generates all of the Peer, Orderer and Admin User MSPs using "fabric-ca-client gencsr" and certificates are generated using openssl and the CAs that come from cryptogen.

```
#!/bin/bash
#################################################################
# This script generates certificates and keys to work with the cryptogen util
# to be used with the hyperledger fabric BYFN example.
# This allows the keys for the certificate to be generated with the
# PKCS11 BCCSP which in turn allows keys to be generated in an HSM.
#################################################################


CA_CLIENT=./bin/fabric-ca-client
CRYPTO_CONFIG=$PWD/crypto-config
```

```
ROOT=$PWD
BCCSP_DEFAULT=PKCS11
PIN=userpin
check_error() {
  if [ $? -ne 0 ]; then
    echo "ERROR: Something went wrong!"
    exit 1
  fi
}


signcsr() {
  MSP=$1
  CN=$2
  CA_DIR=$3
  CA_NAME=$4
  CA_CERT=$(find $CA_DIR -name "*.pem")
  CA_KEY=$(find $CA_DIR -name "*_sk")
  CSR=$MSP/signcerts/$CN.csr
  CERT=$MSP/signcerts/$CN-cert.pem
  openssl x509 -req -sha256 -days 3650 -in $CSR -CA $CA_CERT -CAkey $CA_KEY -
CAcreateserial -out $CERT
  check_error
}


genmsp() {
  ORG_DIR=$1
  ORG_NAME=$2
  NODE_DIR=$3
  NODE_NAME=$4
  NODE_OU=$6
  CN=${NODE_NAME}${ORG_NAME}
  CA_PATH=$CRYPTO_CONFIG/$ORG_DIR/$ORG_NAME
  NODE_PATH=$CA_PATH/$NODE_DIR/$CN
  MSP=$NODE_PATH/msp
  TLS=$NODE_PATH/tls
  LABEL=$5
  rm -rf $MSP/keystore/*
  rm -rf $MSP/signcerts/*
  echo $LABEL
  export FABRIC_CA_CLIENT_BCCSP_DEFAULT=$BCCSP_DEFAULT
  export FABRIC_CA_CLIENT_BCCSP_PKCS11_LABEL=$LABEL
  export FABRIC_CA_CLIENT_BCCSP_PKCS11_PIN=$PIN
  $CA_CLIENT gencsr --csr.cn $CN --mspdir $MSP --csr.names
"C=US,ST=California,L=San Francisco,OU=$NODE_OU"
  check_error
  signcsr $MSP $CN $CA_PATH/ca $ORG_NAME
 }


copy_admin_cert_node() {
  ORG_DIR=$1
  ORG_NAME=$2
  NODE_DIR=$3
```

```
  NODE_NAME=$4
  CN=$NODE_NAME.$ORG_NAME
  CA_PATH=$CRYPTO_CONFIG/$ORG_DIR/$ORG_NAME
  NODE_PATH=$CA_PATH/$NODE_DIR/$CN
  MSP=$NODE_PATH/msp
  ADMIN_CN=Admin@$ORG_NAME
  ADMIN_CERT=$CA_PATH/users/$ADMIN_CN/msp/signcerts/$ADMIN_CN-cert.pem
  cp $ADMIN_CERT $NODE_PATH/msp/admincerts
  check_error
 }


copy_admin_cert_ca() {
  ORG_DIR=$1
  ORG_NAME=$2
  CA_PATH=$CRYPTO_CONFIG/$ORG_DIR/$ORG_NAME
  ADMIN_CN=Admin@$ORG_NAME
  ADMIN_CERT=$CA_PATH/users/$ADMIN_CN/msp/signcerts/$ADMIN_CN-cert.pem
  cp $ADMIN_CERT $CA_PATH/msp/admincerts
  check_error
  cp $ADMIN_CERT $CA_PATH/users/$ADMIN_CN/msp/admincerts
  check_error
}


for org in 1 2; do
  for peer in 0 1; do
    genmsp peerOrganizations org${org}.example.com peers peer${peer}.
org${org}.example.com peer
  done
  genmsp peerOrganizations org${org}.example.com users Admin@
org${org}.example.com client
  for peer in 0 1; do
    copy_admin_cert_node peerOrganizations org${org}.example.com peers
peer${peer}
  done
  copy_admin_cert_ca peerOrganizations org${org}.example.com
done
genmsp ordererOrganizations example.com orderers orderer. orderer.example.com
orderer
genmsp ordererOrganizations example.com users Admin@ orderer.example.com client
copy_admin_cert_node ordererOrganizations example.com orderers orderer
orderer.example.com
copy_admin_cert_ca ordererOrganizations example.com
################################################################################
# End of generate.sh script
################################################################################
```

5. Copy $GOPATH/src/github.com/hyperledger/fabric/sampleconfig/orderer.yaml to $GOPATH/src/github.com/hyperledger/fabric-samples/first-network/orderer.yaml

```
# cp $GOPATH/src/github.com/hyperledger/fabric/sampleconfig/orderer.yaml
$GOPATH/src/github.com/hyperledger/fabric-samples/first-network/orderer.yaml
```

6. Copy $GOPATH/src/github.com/hyperledger/fabric/sampleconfig/core.yaml to $GOPATH/src/github.com/hyperledger/fabric-samples/first-network/core.yaml

```
# cp $GOPATH/src/github.com/hyperledger/fabric/sampleconfig/core.yaml
$GOPATH/src/github.com/hyperledger/fabric-samples/first-network/core.yaml
```

7. Open $GOPATH/src/github.com/hyperledger/fabric-sample/first-network/orderer.yaml and modify the bccsp section to:

```
    BCCSP:
        Default: PKCS11
        PKCS11:
            Library:
            Label:
            Pin:
            Hash: SHA2
            Security: 384
```

8. Open $GOPATH/src/github.com/hyperledger/fabric-samples/first-network/core.yaml file and add or modify the bccsp and system section to:

```
    BCCSP:
        Default: PKCS11
        PKCS11:
            Library:
            Label:
            Pin:
            Hash: SHA2
            Security: 384
    system:
        escc: enable
        vscc: enable
```

9. Open $GOPATH/src/github.com/hyperledger/fabric-samples/first-network/base/peer-base.yaml file and perform the following steps:

   i. Add or modify the following text in service peer-base:

```
 image: fabric-peer-pkcs11:${IMAGE_TAG}
 build:
     context: ..
     dockerfile: ../docker-files/Dockerfile.peer
     args:
       - IMAGE_TAG=${IMAGE_TAG}
```

   ii. Add the following text in service peer-base:

```
      -
CORE_PEER_BCCSP_PKCS11_LIBRARY=/usr/safenet/lunaclient/lib/libCryptoki2_64.s
o
      - CORE_PEER_BCCSP_PKCS11_PIN=userpin
```

   iii. Add volumes section in service peer-base:

```
    volumes:
      - /usr/safenet/lunaclient:/usr/safenet/lunaclient
      - /etc/Chrystoki.conf:/etc/Chrystoki.conf
      - ../core.yaml:/etc/hyperledger/fabric/core.yaml
```

   iv. In service orderer-base add or modify the following:

```
    image: fabric-orderer-pkcs11:${IMAGE_TAG}
```

```
    build:
      context: ..
      dockerfile: ../docker-files/Dockerfile.orderer
      args:
        - IMAGE_TAG=${IMAGE_TAG}
```

10. Open $GOPATH/src/github.com/hyperledger/fabric-samples/first-network/base/docker-compose-base.yaml file and perform the following tasks:

    i. Add the following text in service orderer.example.com under the volume section:

```
        - /usr/safenet/lunaclient:/usr/safenet/lunaclient
        - /etc/Chrystoki.conf:/etc/Chrystoki.conf
        - ../orderer.yaml:/etc/hyperledger/fabric/orderer.yaml
```

    ii. Add the environment section in service orderer.example.com:

```
    environment:
      - ORDERER_GENERAL_BCCSP_PKCS11_LABEL=orderer.example.com
      -
ORDERER_GENERAL_BCCSP_PKCS11_LIBRARY=/usr/safenet/lunaclient/lib/libCryptoki
2_64.so
      - ORDERER_GENERAL_BCCSP_PKCS11_PIN=userpin
```

    iii. Add the following text in service peer0.org1.example.com, under the environment section:

```
      - CORE_PEER_BCCSP_PKCS11_LABEL=org1.example.com
```

    iv. Add the following text in service peer1.org1.example.com under the environment section:

```
      - CORE_PEER_BCCSP_PKCS11_LABEL=org1.example.com
```

    v. Add the following text in service peer0.org2.example.com under the environment section:

```
      - CORE_PEER_BCCSP_PKCS11_LABEL=org2.example.com
```

    vi. Add the following text in service peer1.org2.example.com, under the environment section:

```
      - CORE_PEER_BCCSP_PKCS11_LABEL=org2.example.com
```

11. Open $GOPATH/src/github.com/hyperledger/fabric-samples/first-network/docker-compose-cli.yaml file and make the following changes in the cli section:

    i. Add or modify the following text:

```
    image: fabric-tools-pkcs11:${IMAGE_TAG}
    build:
      context: .
      dockerfile: ../docker-files/Dockerfile.tools
      args:
           - IMAGE_TAG=${IMAGE_TAG}
```

    ii. Add the following under the environment section:

```
      -
   CORE_PEER_BCCSP_PKCS11_LIBRARY=/usr/safenet/lunaclient/lib/libCryptoki2_6
   4.so
      - CORE_PEER_BCCSP_PKCS11_PIN=userpin
```

    iii. Add the following under the volumes section:

```
        - /usr/safenet/lunaclient:/usr/safenet/lunaclient
        - /etc/Chrystoki.conf:/etc/Chrystoki.conf
        - ./core.yaml:/etc/hyperledger/fabric/core.yaml
```

12. Open $GOPATH/src/github.com/hyperledger/fabric-samples/first-network/scripts/utils.sh file and make the following changes to the setGlobals function:

    i. After “if [$ORG -eq 1]” line add the following code:

    ```
        export CORE_PEER_BCCSP_PKCS11_LABEL=org1.example.com
    ```

    ii. After “elif [$ORG -eq 2]” line add the following code:

    ```
        export CORE_PEER_BCCSP_PKCS11_LABEL=org2.example.com
    ```

13. Create a directory $GOPATH/src/github.com/hyperledger/fabric-samples/docker-files as follows:

    ```
    # mkdir $GOPATH/src/github.com/hyperledger/fabric-samples/docker-files
    ```

14. Create a file $GOPATH/src/github.com/hyperledger/fabric-samples/docker-files/Dockerfile.orderer and add the following code:

    ```
    ARG IMAGE_TAG
    FROM hyperledger/fabric-orderer:$IMAGE_TAG
    RUN apt-get update && apt-get install -y libtool
    COPY ./bin/orderer /usr/local/bin
    ```

15. Create a file $GOPATH/src/github.com/hyperledger/fabric-samples/docker-files/Dockerfile.peer and add the following code:

    ```
    ARG IMAGE_TAG
    FROM hyperledger/fabric-peer:$IMAGE_TAG
    RUN apt-get update && apt-get install -y libtool
    COPY ./bin/peer /usr/local/bin
    ```

16. Create a file $GOPATH/src/github.com/hyperledger/fabric-samples/docker-files/Dockerfile.tools and add the following code:

    ```
    ARG IMAGE_TAG
    FROM hyperledger/fabric-tools:$IMAGE_TAG
    RUN apt-get update && apt-get install -y libtool
    COPY ./bin/peer /usr/local/bin
    COPY ./bin/fabric-ca-client /usr/local/bin
    ```

17. Open the $GOPATH/src/github.com/hyperledger/fabric-samples/first-network/byfn.sh file and make the following changes:

    i. In function networkDown comment the following code:

    ```
    rm -rf channel-artifacts/*.block channel-artifacts/*.tx crypto-config
    ./org3-artifacts/crypto-config/ channel-artifacts/org3.json
    ```

    ii. Add ./gencerts.sh between replacePrivateKey and generateChannelArtifacts in main function after line no.599. For example:

    ```
    elif [ "${MODE}" == "generate" ]; then
      generateCerts
      replacePrivateKey
      ./gencerts.sh
      generateChannelArtifacts
    ```

18. Go to $GOPATH/src/github.com/hyperledger/fabric-samples/first-network directory:

    ```
    # cd $GOPATH/src/github.com/hyperledger/fabric-samples/first-network
    ```

19. Generate the crypto material in the HSM partitions, create CSRs, and issue certificates.

    ```
    # ./byfn.sh generate
    ```

20. Run the first-network example:

```
# ./byfn.sh up -i 1.4.8
```

Upon successful completion of the above steps, the following message will appear on your screen:

```
========= All GOOD, BYFN execution completed ===========
```

Integrating Luna Cloud HSM with Hyperledger Fabric

To integrate Thales Luna Cloud HSM with Hyperledger Fabric, you need to complete the following tasks:

> Generate CSR using fabric-ca-client and PKCS11 BCCSP for MSP directory

> Configure Peer Nodes

> Configure the Orderer Nodes

NOTE: Refer to Build Your First Network(BYFN) using Luna Cloud HSM service section below for an example of end to end integration of Hyperledger Fabric with Luna Cloud HSM.

Generate CSR using fabric-ca-client and PKCS11 BCCSP for MSP directory

The fabric-ca-client utility is used to generate certificate signing requests for Peers, Orderers, and Users in their respective MSP directories. The fabric-ca-client utility uses the BCCSP to generate crypto material. If the BCCSP is configured to use the PKCS#11 implementation of BCCSP, then it can be used to generate keys on the Luna Cloud service. The fabric-ca-client BCCSP can be configured in the ~/.fabric-ca-client/fabric-ca-client-config.yaml file. This involves following steps:

> Configure BCCSP to use Luna Cloud HSM services PKCS#11 API

> Generate the cryptographic keys using the gencsr command

Configure BCCSP to use Luna Cloud HSM services PKCS#11 API

1. Modify the BCCSP section in ~/.fabric-ca-client/fabric-ca-client-config.yaml file:

NOTE: If the ~/.fabric-ca-client/fabric-ca-client-config.yaml file is not present, run the following command to generate it: `# fabric-ca-client gencsr`

```
bccsp:
    default: PKCS11
    sw:
        hash: SHA2
        security: 256
        filekeystore:
            keystore: msp/keystore
    pkcs11:
```

```
        hash: SHA2
        security: 384
        library:
/etc/hyperledger/fabric/dpod/org1.example.com/libs/64/libCryptoki2.so
        label: org1.example.com
        pin: userpin
        filekeystore:
            keystore: msp/keystore
```

2. Add a keyrequest setting to the csr section of ~/.fabric-ca-client/fabric-ca-client-config.yaml file to specify the key size as follows:

```
csr:
  cn:
  keyrequest:
    algo: ecdsa
    size: 384
```

Generate the cryptographic keys using the gencsr command

To generate ECDSA private keys on the HSM using the PKCS#11 BCCSP and create a CSR in the msp/signcerts directory for the private key, you need to perform the following steps:

1. The PKCS11 values in the fabric-ca-client-config.yaml file can be left blank and specified using environment variables. Alternatively, the values in the file can be overridden using the following environment variables:

```
# export FABRIC_CA_CLIENT_BCCSP_DEFAULT=PKCS11

# export FABRIC_CA_CLIENT_BCCSP_PKCS11_LABEL=<HSM Partition Label>

# export FABRIC_CA_CLIENT_BCCSP_PKCS11_PIN=<Partition Password>

# export FABRIC_CA_CLIENT_BCCSP_PKCS11_LIBRARY=<HSM PKCS11 Library>
```

2. The command to generate CSR is as follows:

```
# ./fabric-ca-client gencsr [options]
```

Where options are:

```
--csr.cn string           The common name field of the certificate signing
   request.

--mspdir string        Membership Service Provider directory (default "msp").

--csr.names stringSlice      A list of comma-separated CSR names of the form
                             <name>=<value> (e.g. C=CA,OU=peer)
```

> **NOTE:** When generating a CSR request, ensure that you have exported the correct partition label and that the ChrystokiConfigurationPath environment variable points to the path of the correct Chrystoki.conf file. Ensure you specify the correct CN, MSP directory, Names, and OU (the OU should be either peer, orderer, or client) in the fabric-ca-client options.

3. To generate the key for orderer.example.com, execute the following command:

```
# ./fabric-ca-client gencsr --csr.cn orderer.example.com –mspdir
ordererOrganizations/orderer.example.com/orderers/orderer.example.com/msp
--csr.names "C=US,ST=California,L=San Francisco,OU=orderer"
```

Options and exported variables should be changed as per the requirement for generating the specific certificate signing request. Submit the CSR to your CA to obtain the signed certificate for the Peer/Orderer/User and place the signed certificate in their respective msp/signcerts directories.

4. To generate the CSR for the peer0 of org1.example.com, execute the following command:

```
# export FABRIC_CA_CLIENT_BCCSP_DEFAULT=PKCS11

# export FABRIC_CA_CLIENT_BCCSP_PKCS11_LABEL=org1.example.com

# export FABRIC_CA_CLIENT_BCCSP_PKCS11_PIN=userpin

# export
   ChrystokiConfigurationPath=/etc/hyperledger/fabric/dpod/org1.example.com
```

5. The following command generates the key pair for peer0.org1.example.com on the HSM service org1.example.com and creates the CSR ./crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp/signcerts/peer0.org1.example.com.csr.

```
# export

FABRIC_CA_CLIENT_BCCSP_PKCS11_LIBRARY=/etc/hyperledger/fabric/dpod/org1.example
.com/libs/64/libCryptoki2.so

# ./fabric-ca-client gencsr --csr.cn peer0.org1.example.com --mspdir ./crypto
config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp
--csr.names "C=US,ST=California,L=San Francisco,OU=peer"
```

6. Copy the CSR and send it to your CA to obtain a signed certificate and then place the certificate in same directory.

7. Similarly, to generate the certificate request for Admin User for org1:

```
# ./fabric-ca-client gencsr --csr.cn Admin@org1.example.com --mspdir ./crypto-
config/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
--csr.names "C=US,ST=California,L=San Francisco,OU=client"
```

Configure Peer Nodes

To configure BCCSP to use Luna Cloud Service for peer nodes:

1. Change the core.yaml file and specify PKCS#11 as default in the BCSSP section as follows:

```
BCCSP:
  Default: PKCS11
  PKCS11:
    Library:
/etc/hyperledger/fabric/dpod/org1.example.com/libs/64/libCryptoki2.so
    Label: org1.example.com
    Pin: userpin
    Hash: SHA2
    Security: 384
```

Alternatively, the PKCS#11 values in the core.yaml file can be left blank and specified using environment variables. The following environment variables can be used to change the configuration settings of the core.yaml BCCSP.

```
# export CORE_PEER_BCCSP_DEFAULT=PKCS11

# export CORE_PEER_BCCSP_PKCS11_LABEL=<HSM Partition Label>
```

```
# export CORE_PEER_BCCSP_PKCS11_PIN=<Partition Password>

# export CORE_PEER_BCCSP_PKCS11_LIBRARY=<HSM PKCS11 Library>
```

2. Ensure that the Luna Cloud HSM client directory is available to the peer and correct partition is used. The ChrystokiConfigurationPath must point to the Luna Cloud HSM client directory where Chrystoki.conf is available.

```
# export ChrystokiConfigurationPath=<PATH of Chrystoki.conf>
```

Configure the Orderer Nodes

To configure BCCSP to use Luna Cloud Service for orderer nodes:

1. Change the orderer.yaml file and specify the PKCS11 to default in the BCCSP section as follows:

```
BCCSP:
  Default: PKCS11
  PKCS11:
    Library:
/etc/hyperledger/fabric/dpod/orderer.example.com/libs/64/libCryptoki2.so
    Label: orderer.example.com
    Pin: userpin
    Hash: SHA2
    Security: 384
```

   Alternatively, the PKCS11 values in the orderer.yaml file can be left blank and specified using the following environment variables, or the values in the file can be overridden using these environment variables.

```
# export ORDERER_GENERAL_BCCSP_DEFAULT=PKCS11

# export ORDERER_GENERAL_BCCSP_PKCS11_LABEL=<HSM Partition Label>

# export ORDERER_GENERAL_BCCSP_PKCS11_PIN=<Partition Password>

# export ORDERER_GENERAL_BCCSP_PKCS11_LIBRARY=<HSM PKCS11 Library>
```

2. Ensure that the Luna Cloud HSM client directory is available to the Orderer and correct partition is used for that Orderer. The ChrystokiConfigurationPath must point to the Luna Cloud HSM client directory of the Orderer where Chrystoki.conf is available.

```
# export ChrystokiConfigurationPath=<PATH of Chrystoki.conf>
```

You have completed the integration and all the required components are installed on the platform(s) on which you will be developing Blockchain applications and/or operating Hyperledger Fabric. All key materials have been generated on the Luna Cloud service using the PKCS11 BCCSP. You have configured the core.yaml and the orderer.yaml to use the PKCS11 BCCSP. If you are deploying your nodes using docker compose, after building your own images you can update your docker compose files to mount the hsm library and configuration file inside the container using volumes.

Now start the Fabric CA, Orderer, and Peers to create channel artifacts and run the channel. Join the nodes in the channel to create a permissioned Blockchain network. This is achieved by assigning identity certificates to the member orgs and their nodes which will be used to identify themselves and conduct transactions in the network. A library of X509 certificates (commonly termed as cryptographic material) gets created for the associated Peer/Orderer nodes using the PKCS11 BCCSP which in turn generates all key pairs on the Luna Cloud HSM.

NOTE: Ensure you include the PKCS#11 BCCSP in the script that will be used to create the Blockchain. Failure to include the PKCS#11 BCCSP will result in the Blockchain not using the Luna Cloud service.

Build Your First Network (BYFN) using Luna Cloud HSM service

This section of the guide demonstrates creating the Blockchain network, invoking transactions, and querying the state of the Blockchain using a byfn example.

Build your first network using Luna Cloud HSM service

To build your first network using Luna Cloud HSM service:

1. Open ~/.fabric-ca-client/fabric-ca-client-config.yaml file and change the bccsp section to:

```
bccsp:
    default: PKCS11
    sw:
        hash: SHA2
        security: 256
        filekeystore:
            keystore: msp/keystore
    pkcs11:
        hash: SHA2
        security: 384
        library:
/etc/hyperledger/fabric/dpod/org1.example.com/libs/64/libCryptoki2.so
        label:
        pin:
```

2. Add a keyrequest setting to the csr section of ~/.fabric-ca-client/fabric-ca-client-config.yaml file to specify the key size as follows:

```
   csr:
     cn:
     keyrequest:
       algo: ecdsa
       size: 384
```

3. Go to first-network directory:

```
# cd $GOPATH/src/github.com/hyperledger/fabric-samples/first-network
```

4. Copy the following script and save it as gencerts.sh to generate crypto material on Thales Luna Cloud HSM. It works in conjunction with the cryptogen tool. The script generates all of the Peer, Orderer and Admin User MSPs using "fabric-ca-client gencsr" and certificates are generated using openssl and the CAs that come from cryptogen.

```
#!/bin/bash
#############################################################################
# This script generates certificates and keys to work with the cryptogen util
# to be used with the hyperledger fabric BYFN example.
# This allows the keys for the certificate to be generated with the
# PKCS11 BCCSP which in turn allows keys to be generated in an HSM.
#############################################################################

CA_CLIENT=./bin/fabric-ca-client
```

```
CRYPTO_CONFIG=$PWD/crypto-config
ROOT=$PWD
BCCSP_DEFAULT=PKCS11
PIN=userpin
check_error() {
  if [ $? -ne 0 ]; then
    echo "ERROR: Something went wrong!"
    exit 1
  fi
}

signcsr() {
  MSP=$1
  CN=$2
  CA_DIR=$3
  CA_NAME=$4
  CA_CERT=$(find $CA_DIR -name "*.pem")
  CA_KEY=$(find $CA_DIR -name "*_sk")
  CSR=$MSP/signcerts/$CN.csr
  CERT=$MSP/signcerts/$CN-cert.pem
  openssl x509 -req -sha256 -days 3650 -in $CSR -CA $CA_CERT -CAkey $CA_KEY -
CAcreateserial -out $CERT
  check_error
}

genmsp() {
  ORG_DIR=$1
  ORG_NAME=$2
  NODE_DIR=$3
  NODE_NAME=$4
  NODE_OU=$6
  CN=${NODE_NAME}${ORG_NAME}
  CA_PATH=$CRYPTO_CONFIG/$ORG_DIR/$ORG_NAME
  NODE_PATH=$CA_PATH/$NODE_DIR/$CN
  MSP=$NODE_PATH/msp
  TLS=$NODE_PATH/tls
  LABEL=$5
  rm -rf $MSP/keystore/*
  rm -rf $MSP/signcerts/*
  echo $LABEL
  export FABRIC_CA_CLIENT_BCCSP_DEFAULT=$BCCSP_DEFAULT
  export FABRIC_CA_CLIENT_BCCSP_PKCS11_LABEL=$LABEL
  export FABRIC_CA_CLIENT_BCCSP_PKCS11_PIN=$PIN
  export ChrystokiConfigurationPath=/etc/hyperledger/fabric/dpod/$LABEL
  export
FABRIC_CA_CLIENT_BCCSP_PKCS11_LIBRARY=/etc/hyperledger/fabric/dpod/$LABEL/libs/
64/libCryptoki2.so
  $CA_CLIENT gencsr --csr.cn $CN --mspdir $MSP --csr.names
"C=US,ST=California,L=San Francisco,OU=$NODE_OU"
  check_error
  signcsr $MSP $CN $CA_PATH/ca $ORG_NAME
 }

copy_admin_cert_node() {
  ORG_DIR=$1
```

```
  ORG_NAME=$2
  NODE_DIR=$3
  NODE_NAME=$4
  CN=$NODE_NAME.$ORG_NAME
  CA_PATH=$CRYPTO_CONFIG/$ORG_DIR/$ORG_NAME
  NODE_PATH=$CA_PATH/$NODE_DIR/$CN
  MSP=$NODE_PATH/msp
  ADMIN_CN=Admin@$ORG_NAME
  ADMIN_CERT=$CA_PATH/users/$ADMIN_CN/msp/signcerts/$ADMIN_CN-cert.pem
  cp $ADMIN_CERT $NODE_PATH/msp/admincerts
  check_error
 }

copy_admin_cert_ca() {
  ORG_DIR=$1
  ORG_NAME=$2
  CA_PATH=$CRYPTO_CONFIG/$ORG_DIR/$ORG_NAME
  ADMIN_CN=Admin@$ORG_NAME
  ADMIN_CERT=$CA_PATH/users/$ADMIN_CN/msp/signcerts/$ADMIN_CN-cert.pem
  cp $ADMIN_CERT $CA_PATH/msp/admincerts
  check_error
  cp $ADMIN_CERT $CA_PATH/users/$ADMIN_CN/msp/admincerts
  check_error
}

for org in 1 2; do
  for peer in 0 1; do
    genmsp peerOrganizations org${org}.example.com peers peer${peer}.
org${org}.example.com peer
  done
  genmsp peerOrganizations org${org}.example.com users Admin@
org${org}.example.com client
  for peer in 0 1; do
    copy_admin_cert_node peerOrganizations org${org}.example.com peers
peer${peer}
  done
  copy_admin_cert_ca peerOrganizations org${org}.example.com
done
genmsp ordererOrganizations example.com orderers orderer. orderer.example.com
orderer
genmsp ordererOrganizations example.com users Admin@ orderer.example.com client
copy_admin_cert_node ordererOrganizations example.com orderers orderer
orderer.example.com
copy_admin_cert_ca ordererOrganizations example.com
################################################################################
# End of generate.sh script
################################################################################
```

5. Copy the $GOPATH/src/github.com/hyperledger/fabric/sampleconfig/orderer.yaml file to $GOPATH/src/github.com/hyperledger/fabric-samples/first-network/orderer.yaml file:

   ```
   # cp $GOPATH/src/github.com/hyperledger/fabric/sampleconfig/orderer.yaml
   $GOPATH/src/github.com/hyperledger/fabric-samples/first-network/orderer.yaml
   ```

6. Copy the $GOPATH/src/github.com/hyperledger/fabric/sampleconfig/core.yaml file to $GOPATH/src/github.com/hyperledger/fabric-samples/first-network/core.yaml file:

```
# cp $GOPATH/src/github.com/hyperledger/fabric/sampleconfig/core.yaml
$GOPATH/src/github.com/hyperledger/fabric-samples/first-network/core.yaml
```

7. Open the $GOPATH/src/github.com/hyperledger/fabric-sample/first-network/orderer.yaml file and modify the bccsp section as follows:

```
    BCCSP:
        Default: PKCS11
        PKCS11:
            Library:
            Label:
            Pin:
            Hash: SHA2
            Security: 384
```

8. Open the $GOPATH/src/github.com/hyperledger/fabric-samples/first-network/core.yaml file and add or modify the bccsp and system section to:

```
    BCCSP:
        Default: PKCS11
        PKCS11:
            Library:
            Label:
            Pin:
            Hash: SHA2
            Security: 384
    system:
        escc: enable
        vscc: enable
```

9. Open the $GOPATH/src/github.com/hyperledger/fabric-samples/first-network/base/peer-base.yaml file and make the following changes:

   i. In service peer-base, add or modify the following:

```
 image: fabric-peer-pkcs11:${IMAGE_TAG}
   build:
     context: ..
     dockerfile: ../docker-files/Dockerfile.peer
     args:
       - IMAGE_TAG=${IMAGE_TAG}
```

   ii. In service peer-base, add the following code under the environment section:

```
      - CORE_PEER_BCCSP_PKCS11_PIN=userpin
```

   iii. In service peer-base, add a volumes section:

```
 volumes:
  - ../core.yaml:/etc/hyperledger/fabric/core.yaml
```

   iv. In service orderer-base, add or modify the following code:

```
   image: fabric-orderer-pkcs11:${IMAGE_TAG}
   build:
     context: ..
     dockerfile: ../docker-files/Dockerfile.orderer
     args:
       - IMAGE_TAG=${IMAGE_TAG}
```

10. Open the $GOPATH/src/github.com/hyperledger/fabric-samples/first-network/base/docker-compose-base.yaml file and perform these steps:

    i. In service orderer.example.com, under volume section, add the following code:

```
        - ../orderer.yaml:/etc/hyperledger/fabric/orderer.yaml
        -
/etc/hyperledger/fabric/dpod/orderer.example.com:/etc/hyperledger/fabric/dpo
d/orderer.example.com
```

    ii. In service orderer.example.com, add an environment section:

```
    environment:
      - ORDERER_GENERAL_BCCSP_PKCS11_LABEL=orderer.example.com
      -
ORDERER_GENERAL_BCCSP_PKCS11_LIBRARY=/etc/hyperledger/fabric/dpod/orderer.ex
ample.com/libs/64/libCryptoki2.so
      -
ChrystokiConfigurationPath=/etc/hyperledger/fabric/dpod/orderer.example.com
      - ORDERER_GENERAL_BCCSP_PKCS11_PIN=userpin
```

    iii. In service peer0.org1.example.com, under the environment section, add:

```
      - CORE_PEER_BCCSP_PKCS11_LABEL=org1.example.com
      -
CORE_PEER_BCCSP_PKCS11_LIBRARY=/etc/hyperledger/fabric/dpod/org1.example.com
/libs/64/libCryptoki2.so
      -
ChrystokiConfigurationPath=/etc/hyperledger/fabric/dpod/org1.example.com
```

    iv. In service peer0.org1.example.com, under the volumes section, add:

```
    -
/etc/hyperledger/fabric/dpod/org1.example.com:/etc/hyperledger/fabric/dpod/o
rg1.example.com
```

    v. In service peer1.org1.example.com, under environment section, add:

```
      - CORE_PEER_BCCSP_PKCS11_LABEL=org1.example.com
      -
CORE_PEER_BCCSP_PKCS11_LIBRARY=/etc/hyperledger/fabric/dpod/org1.example.com
/libs/64/libCryptoki2.so
      -
ChrystokiConfigurationPath=/etc/hyperledger/fabric/dpod/org1.example.com
```

    vi. In service peer1.org1.example.com, under volumes section, add:

```
        -
/etc/hyperledger/fabric/dpod/org1.example.com:/etc/hyperledger/fabric/dpod/o
rg1.example.com
```

    vii. In service peer0.org2.example.com, under environment section, add:

```
   - CORE_PEER_BCCSP_PKCS11_LABEL=org2.example.com
   -
CORE_PEER_BCCSP_PKCS11_LIBRARY=/etc/hyperledger/fabric/dpod/org2.example.com
/libs/64/libCryptoki2.so
   - ChrystokiConfigurationPath=/etc/hyperledger/fabric/dpod/org2.example.com
```

viii. In service peer0.org2.example.com, under volumes section, add:

```
    -
/etc/hyperledger/fabric/dpod/org2.example.com:/etc/hyperledger/fabric/dpod/o
rg2.example.com
```

ix. In service peer1.org2.example.com, under environment section, add:

```
   - CORE_PEER_BCCSP_PKCS11_LABEL=org2.example.com
   -
CORE_PEER_BCCSP_PKCS11_LIBRARY=/etc/hyperledger/fabric/dpod/org2.example.com
/libs/64/libCryptoki2.so
   - ChrystokiConfigurationPath=/etc/hyperledger/fabric/dpod/org2.example.com
```

x. In service peer1.org2.example.com, under volumes section, add:

```
    -
/etc/hyperledger/fabric/dpod/org2.example.com:/etc/hyperledger/fabric/dpod/o
rg2.example.com
```

11. Open the $GOPATH/src/github.com/hyperledger/fabric-samples/first-network/docker-compose-cli.yaml file and make the following changes in the cli section:

i. Add or modify the following code:

```
    image: fabric-tools-pkcs11:${IMAGE_TAG}
    build:
      context: .
      dockerfile: ../docker-files/Dockerfile.tools
      args:
        - IMAGE_TAG=${IMAGE_TAG}
```

ii. Add the following code under the environment section:

```
      - CORE_PEER_BCCSP_PKCS11_PIN=userpin
```

iii. Add the following code under the volumes section:

```
        - /etc/hyperledger/fabric/dpod:/etc/hyperledger/fabric/dpod
        - ./core.yaml:/etc/hyperledger/fabric/core.yaml
```

12. Open the $GOPATH/src/github.com/hyperledger/fabric-samples/first-network/scripts/utils.sh file and make the following changes to the setGlobals function:

i. After “if [$ORG -eq 1]” line, add the following code:

```
    export CORE_PEER_BCCSP_PKCS11_LABEL=org1.example.com
    export
CORE_PEER_BCCSP_PKCS11_LIBRARY=/etc/hyperledger/fabric/dpod/org1.example.com
/libs/64/libCryptoki2.so
    export
ChrystokiConfigurationPath=/etc/hyperledger/fabric/dpod/org1.example.com
```

ii. After “elif [$ORG -eq 2]” line, add the following code:

```
    export CORE_PEER_BCCSP_PKCS11_LABEL=org2.example.com
    export
CORE_PEER_BCCSP_PKCS11_LIBRARY=/etc/hyperledger/fabric/dpod/org2.example.com
/libs/64/libCryptoki2.so
    export
ChrystokiConfigurationPath=/etc/hyperledger/fabric/dpod/org2.example.com
```

13. Create a directory $GOPATH/src/github.com/hyperledger/fabric-samples/docker-files:

```
# mkdir $GOPATH/src/github.com/hyperledger/fabric-samples/docker-files
```

14. Create a file $GOPATH/src/github.com/hyperledger/fabric-samples/docker-files/Dockerfile.orderer and add the following code:

```
ARG IMAGE_TAG
FROM hyperledger/fabric-orderer:$IMAGE_TAG
RUN apt-get update && apt-get install -y libtool
COPY ./bin/orderer /usr/local/bin
```

15. Create a file $GOPATH/src/github.com/hyperledger/fabric-samples/docker-files/Dockerfile.peer and add the following code:

```
ARG IMAGE_TAG
FROM hyperledger/fabric-peer:$IMAGE_TAG
RUN apt-get update && apt-get install -y libtool
COPY ./bin/peer /usr/local/bin
```

16. Create a file $GOPATH/src/github.com/hyperledger/fabric-samples/docker-files/Dockerfile.tools and add the following code:

```
ARG IMAGE_TAG
FROM hyperledger/fabric-tools:$IMAGE_TAG
RUN apt-get update && apt-get install -y libtool
COPY ./bin/peer /usr/local/bin
COPY ./bin/fabric-ca-client /usr/local/bin
```

17. Open the $GOPATH/src/github.com/hyperledger/fabric-samples/first-network/byfn.sh file and make the following changes:

    i. Comment the following line in function networkDown:

    ```
    rm -rf channel-artifacts/*.block channel-artifacts/*.tx crypto-config
    ./org3-artifacts/crypto-config/ channel-artifacts/org3.json
    ```

    ii. Add ./gencerts.sh between replacePrivateKey and generateChannelArtifacts in main function after line no.599.

    For example:

    ```
    elif [ "${MODE}" == "generate" ]; then
      generateCerts
      replacePrivateKey
      ./gencerts.sh
      generateChannelArtifacts
    ```

18. Go to $GOPATH/src/github.com/hyperledger/fabric-samples/first-network directory:

```
# cd $GOPATH/src/github.com/hyperledger/fabric-samples/first-network
```

19. Generate the crypto material in the HSM partitions, create CSRs, and issue certificates:

```
# ./byfn.sh generate
```

20. Run the first-network example:

```
# ./byfn.sh up -i 1.4.8
```

Upon successful completion of the above steps, the following message will appear on your screen:

```
========= All GOOD, BYFN execution completed ===========
```

Integrating Luna HSM or Luna Cloud HSM with Hyperledger Fabric Client

To integrate Luna HSM or Luna Cloud HSM with Hyperledger Fabric Client, you need to perform the following tasks:

> Integrate Hyperledger Fabric Client SDK for Node.js with Luna HSM or Luna Cloud HSM

> Integrate Hyperledger Fabric Client SDK for Java with a Luna HSM or Luna Cloud HSM

Integrate Hyperledger Fabric Client SDK for Node.js with Luna HSM or Luna Cloud HSM

The Hyperledger Fabric Client (HFC) SDK for Node.js provides a powerful and easy to use API to interact with a Hyperledger Fabric Blockchain. The HFC is designed to be used in the Node.js JavaScript runtime. To generate the keys on the Thales Luna HSM or Luna Cloud service and run end-2-end execution, complete the following steps:

1. Install the nodejs and npm using Linux package manager.
2. Add the fabric-ca-client and configtxgen binaries in the path.

   ```
   # export PATH=/opt/gopath/src/github.com/hyperledger/fabric-ca/bin:/opt/gopath/src/github.com/hyperledger/fabric/release/linux-amd64/bin:$PATH
   ```

3. Check out the fabric-sdk-node source code.

   ```
   # cd $GOPATH/src/github.com/hyperledger
   # git clone https://gerrit.hyperledger.org/r/fabric-sdk-node
   # cd fabric-sdk-node
   # git checkout -b v1.4.0 v1.4.0
   # cd ..
   ```

   NOTE: The instructions were developed against the tag v1.4.0 for Hyperledger Fabric and Hyperledger Fabric Client SDK. It is advisable to use the repo checkout from these tags as the instructions may not be compatible with the latest check-in in the master branch of Fabric and Client SDK.

4. Check out the Fabric SDK HSM Integration repo.

   ```
   # git clone https://github.com/gemalto/fabric-sdk-hsm
   ```

5. Generate the fabric-ca-client default configuration file.

```
# fabric-ca-client gencsr
```

6. Modify the bccsp section in ~/.fabric-ca-client/fabric-ca-client-config.yaml to add PKCS11 as default bccsp.

```
bccsp:
    default: PKCS11
    pkcs11:
        hash: SHA2
        security: 256
        library: /usr/safenet/lunaclient/lib/libCryptoki2_64.so
        label:
        pin:
        filekeystore:
            keystore: msp/keystore
```

NOTE: Use spaces not tabs and pay attention to the indentation. Ensure that the library path points to the correct Thales Cryptoki library.

7. Run the helper script to generate private keys in the HSM, create CSRs for the Peer and Orderer Admin users, and create certificates.

```
# PKCS11_LABEL=fabric-sdk PKCS11_PIN=userpin ./fabric-sdk-
hsm/node/genAdminPkcs11Node.sh
```

Where `fabric-sdk` is the partition label and `userpin` is partition CO password. The helper script executes configtxgen and generates the genesis block with new certificates.

8. Copy the required javascript files from fabric-sdk-hsm to fabric-sdk-node.

```
# cp fabric-sdk-hsm/node/e2eHSM.js fabric-sdk-node/test/integration/
# cp fabric-sdk-hsm/node/utilHSM.js fabric-sdk-node/test/unit/
```

9. Install gulp and the required dependencies.

```
# cd fabric-sdk-node
# sudo npm install -g gulp
# npm install
# gulp ca
```

10. Open a new terminal in the fabric-sdk-node directory and start the fabric docker containers.

```
# cd test/fixtures
# export DOCKER_IMG_TAG=:1.4.0
# docker-compose up
```

11. In the previous terminal, configure the constant values for the slot and partition password if required in the test/unit/utilHSM.js file.

```
const PKCS11_LIB = '/usr/safenet/lunaclient/lib/libCryptoki2_64.so';
const PKCS11_SLOT = 0;
const PKCS11_PIN = 'userpin';
const PKCS11_USER_TYPE = 1;
```

NOTE: Ensure that the PKCS11_LIB path points to the correct Thales Cryptoki library when using Thales Luna HSM or Luna Cloud Service.

12. Run the end-2-end HSM integration test.

```
# node test/integration/e2eHSM.js
```

The following snippet will appear on your screen when test gets completed successfully.

```
***** TransientMap Support in Proposals *****
ok 207 Successfully retrieved TLS certificate
ok 208 Successfully loaded member from persistence
ok 209 Successfully enrolled user 'admin' (e2eUtil 4)
ok 210 Checking the result has the transientMap value returned by the chaincode
ok 211 Checking the result has the transientMap value returned by the chaincode
ok 212 Successfully verified transient map values
1..212
# tests 212
# pass  212
# ok
```

13. To clean up the docker containers in the docker-compose terminal, press CTRL-C and run the following commands:

```
# docker rm -f $(docker ps -aq)
# docker-compose up
```

14. Perform step 12 to execute end-2-end HSM integration test again.

Integrate Hyperledger Fabric Client SDK for Java with a Luna HSM or Luna Cloud HSM

The Hyperledger Fabric Client (HFC) SDK for Java provides a powerful and easy to use API to interact with a Hyperledger Fabric Blockchain. The SDK facilitates Java applications to manage the lifecycle of Hyperledger channels and user chaincode. The SDK also provides a means to execute user chaincode, query blocks, and transactions on the channel, and to monitor events on the channel.

Integrate Hyperledger Fabric Client SDK Java

1. Install java and maven using Linux package manager.
2. Add the fabric-ca-client and configtxgen binaries in the path.

```
# export PATH=/opt/gopath/src/github.com/hyperledger/fabric-ca/bin:/opt/gopath/src/github.com/hyperledger/fabric/release/linux-amd64/bin:$PATH
```

3. Copy the LunaProvider.jar and libLunaAPI.so in the <Java installation path>/jre/lib/ext directory.

```
# cp /usr/safenet/lunaclient/jsp/lib/LunaProvider.jar /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.201.b09-2.el7_6.x86_64/jre/lib/ext
# cp /usr/safenet/lunaclient/jsp/lib/libLunaAPI.so /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.201.b09-2.el7_6.x86_64/jre/lib/ext
```

> **NOTE:** LunaProvider.jar and libLunaAPI.so for Luna Cloud are available at <Luna Cloud HSM Installation Directory>/jsp/LunaProvider.jar and <Luna Cloud HSM Installation Directory>/jsp/64/ libLunaAPI.so.

4. Checkout the fabric-sdk-java source code.

```
# git clone https://gerrit.hyperledger.org/r/fabric-sdk-java
# cd fabric-sdk-java
# git checkout -b v1.4.0 v1.4.0
# cd ..
```

> **NOTE:** The instructions were developed against the tag v1.4.0 for Hyperledger Fabric and Hyperledger Fabric Client SDK. It is advisable to use the repo checkout from these tags as the instructions may not be compatible with the latest check-in in the master branch of Fabric and Fabric Client SDK.

5. Checkout the Fabric SDK HSM Integration repo.

```
# git clone https://github.com/gemalto/fabric-sdk-hsm
```

6. Generate the fabric-ca-client default configuration file.

```
# fabric-ca-client gencsr
```

7. Modify the bccsp section in ~/.fabric-ca-client/fabric-ca-client-config.yaml to add PKCS11 as the default bccsp.

```
bccsp:
    default: PKCS11
    pkcs11:
        hash: SHA2
        security: 256
        library: /usr/safenet/lunaclient/lib/libCryptoki2_64.so
        label:
        pin:
        filekeystore:
            keystore: msp/keystore
```

> **NOTE:** Use spaces not tabs and pay attention to the indentation. Ensure that the library path points to the correct Thales Cryptoki library when using Thales Luna HSM or Luna Cloud Service.

8. Run the helper script to generate private keys in the HSM, create CSRs for the Peer and Orderer Admin users, and create certificates.

```
# PKCS11_LABEL=fabric-sdk PKCS11_PIN=userpin ./fabric-sdk-
hsm/java/genAdminPkcs11Java.sh
```

> **NOTE:** Where “fabric-sdk” is partition label and “userpin” is partition CO password. The helper script executes configtxgen and generates the genesis block with new certificates.

9. Copy the required java files from fabric-sdk-hsm to fabric-sdk-java.

```
# cp fabric-sdk-hsm/java/End2endHSMIT.java fabric-sdk-
java/src/test/java/org/hyperledger/fabric/sdkintegration/
```

```
# cp fabric-sdk-hsm/java/SampleHSMStore.java fabric-sdk-
java/src/test/java/org/hyperledger/fabric/sdkintegration/
```

10. Open a new terminal in the fabric-sdk-java directory and start the fabric docker containers.

```
# cd ./fabric-sdk-java/src/test/fixture/sdkintegration

# export DOCKER_IMG_TAG=:1.4.0

# docker-compose up
```

11. In the previous terminal, configure the constant values for the slot and partition password if required in the fabric-sdk-java/src/test/java/org/hyperledger/fabric/sdkintegration/End2endHSMIT.java file.

```
private static final String TOKEN_LABEL = "fabric-sdk";

private static final String PARTITION_PASSWORD = "userpin";
```

> **NOTE:** Where “fabric-sdk” is partition label and “userpin” is partition CO password.

12. Run the end-2-end HSM integration test.

```
# cd fabric-sdk-java

# mvn failsafe:integration-test -Dtest=End2endHSMIT test
```

The following snippet will appear on your screen when test gets completed successfully:

```
-------------------------------------------------------------------------
That's all folks!

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 40.431 sec - in
org.hyperledger.fabric.sdkintegration.End2endHSMIT

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO]

[INFO] --- jacoco-maven-plugin:0.7.9:report (post-unit-test) @ fabric-sdk-java
---

[INFO] Loading execution data file
/opt/gopath/src/github.com/hyperledger/fabric-sdk-java/target/coverage-
reports/jacoco-ut.exec

[INFO] Analyzed bundle 'fabric-java-sdk' with 231 classes

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 1:42.468s

[INFO] Finished at: Mon Apr 08 13:04:06 IST 2019

[INFO] Final Memory: 30M/188M

[INFO] ------------------------------------------------------------------------
```

13. To clean up the docker containers in the docker-compose terminal, press CTRL-C and run the following commands.

```
# docker rm -f $(docker ps -aq)
```

```
# docker-compose up
```

14. Perform step 12 to execute end-2-end HSM integration test again.

    This completes the integration of Luna HSM or Luna Cloud HSM with Hyperledger Fabric Client.

Contacting Customer Support

If you encounter a problem during this integration, contact your supplier or Thales Customer Support. Thales Customer Support operates 24 hours a day, 7 days a week. Your level of access to this service is governed by the support plan arrangements made between Thales and your organization. Please consult this support plan for further information about your entitlements, including the hours when telephone support is available to you.

Customer Support Portal

The Customer Support Portal, at https://supportportal.thalesgroup.com, is a database where you can find solutions for most common problems. The Customer Support Portal is a comprehensive, fully searchable repository of support resources, including software and firmware downloads, release notes listing known problems and workarounds, a knowledge base, FAQs, product documentation, technical notes, and more. You can also use the portal to create and manage support cases.

> **NOTE:** You require an account to access the Customer Support Portal. To create a new account, go to the portal and click on the **REGISTER** link.

Telephone Support

If you have an urgent problem, or cannot access the Customer Support Portal, you can contact Thales Customer Support by telephone at +1 410-931-7520. Additional local telephone support numbers are listed on the support portal.

Email Support

You can also contact technical support by email at technical.support.DIS@thalesgroup.com.