Hyperledger Fabric (Blockchain)

INTEGRATION GUIDE

SAFENET LUNA HSM

SAFENET DATA PROTECTION ON DEMAND

Document Information

Document Part Number	007-000084-001
Release Date	April 2019

Revision History

Revision	Date	Reason
D	April 2019	Update

Trademarks, Copyrights, and Third-Party Software

Disclaimer

All information herein is either public information or is the property of and owned solely by Gemalto NV. and/or its subsidiaries who shall have and keep the sole right to file patent applications or any other kind of intellectual property protection in connection with such information.

Nothing herein shall be construed as implying or granting to you any rights, by license, grant or otherwise, under any intellectual and/or industrial property rights of or concerning any of Gemalto’s information.

This document can be used for informational, non-commercial, internal and personal use only provided that:

- The copyright notice below, the confidentiality and proprietary legend and this full warning notice appear in all copies.
- This document shall not be posted on any network computer or broadcast in any media and no modification of any part of this document shall be made.

Use for any other purpose is expressly prohibited and may result in severe civil and criminal liabilities.

The information contained in this document is provided “AS IS” without any warranty of any kind. Unless otherwise expressly agreed in writing, Gemalto makes no warranty as to the value or accuracy of information contained herein.

The document could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Furthermore, Gemalto reserves the right to make any change or improvement in the specifications data, information, and the like described herein, at any time.

Gemalto hereby disclaims all warranties and conditions with regard to the information contained herein, including all implied warranties of merchantability, fitness for a particular purpose, title and non-infringement. In no event shall Gemalto be liable, whether in contract, tort or otherwise, for any indirect, special or consequential damages or any damages whatsoever including but not limited to damages resulting from loss of use, data, profits, revenues, or customers, arising out of or in connection with the use or performance of information contained in this document.

Gemalto does not and shall not warrant that this product will be resistant to all possible attacks and shall not incur, and disclaims, any liability in this respect. Even if each product is compliant with current security standards in force on the date of their design, security mechanisms' resistance necessarily evolves according to the state of the art in security and notably under the emergence of new attacks. Under no circumstances, shall Gemalto be held liable for any third party actions and in particular in case of any successful attack against systems or equipment incorporating Gemalto products. Gemalto disclaims any liability with respect to security for direct, indirect, incidental or consequential damages that result from any use of its products. It is further stressed that independent testing and verification by the person using the product is particularly encouraged, especially in any application in which defective, incorrect or insecure functioning could result in damage to persons or property, denial of service or loss of privacy.

CONTENTS

PREFACE ... 5

Scope ... 5
Document Conventions ... 5
Command Syntax and Typeface Conventions ... 6
Support Contacts ... 7
Customer Support Portal ... 7
Telephone Support ... 7
Email Support ... 7

CHAPTER 1: Introduction ... 8

Overview ... 8
Understanding Hyperledger Fabric Blockchain Network ... 8
Third Party Application Details ... 9
Supported Platforms ... 10
Prerequisites ... 11
Configuring the SafeNet HSM ... 11
Installing Hyperledger Fabric and Fabric CA ... 11

CHAPTER 2: Integrating Hyperledger Fabric with an HSM on Demand service ... 14

Provisioning your HSMoD service ... 14
Integrating Hyperledger Fabric with a HSMoD service ... 15
Generating a CSR Using fabric-ca-client and PKCS11 BCCSP for an MSP Directory ... 15
Configuring the Peer Nodes ... 18
Configuring the Orderer Nodes ... 19
Example: Running e2e_cli end-2-end execution using an HSMoD service ... 20

CHAPTER 3: Integrating Hyperledger Fabric with a SafeNet Luna Network HSM ... 30

Configuring the SafeNet Luna HSM ... 30
Integrating Hyperledger Fabric with a SafeNet Luna HSM ... 31
Generating a CSR Using fabric-ca-client and PKCS11 BCCSP for an MSP Directory ... 32
Configuring Peer Nodes ... 34
Configuring Orderer Nodes ... 35
Example: Running e2e_cli end-2-end execution using SafeNet Luna HSM ... 36

CHAPTER 4: Integrating Hyperledger Fabric Client with a SafeNet Luna Network HSM or HSM on Demand Service ... 45

Configuring the SafeNet Luna HSM ... 45
Provisioning the HSM on Demand service ... 46
Integrating the Hyperledger Fabric Client SDK for Node.js with a SafeNet HSM ... 47
Integrating Hyperledger Fabric Client SDK for Java with a SafeNet HSM ... 49

PREFACE

This document is intended to guide administrators through the steps of supporting Blockchain technology using Hyperledger Fabric and Fabric SDK with SafeNet HSMs including installation, configuration, and integration.

Scope

This guide is intended to show SafeNet HSMs protecting the Blockchain identities (Peer, Orderer and Users) and the transactions that they perform. This guide will demonstrate the use of Peers, Orderer, and User identities protected by SafeNet HSMs in a Hyperledger Blockchain performing transactions.

Document Conventions

This section provides information on the conventions used in this template.

Notes

Notes are used to alert you to important or helpful information. These elements use the following format:

> **NOTE:** Take note. Notes contain important or helpful information.

Cautions

Cautions are used to alert you to important information that may help prevent unexpected results or data loss. These elements use the following format:

> **CAUTION!** Exercise caution. Caution alerts contain important information that may help prevent unexpected results or data loss.

Warnings

Warnings are used to alert you to the potential for catastrophic data loss or personal injury. These elements use the following format:

> ****WARNING** Be extremely careful and obey all safety and security measures. In this situation you might do something that could result in catastrophic data loss or personal injury**

Command Syntax and Typeface Conventions

Convention	Description
bold	The bold attribute is used to indicate the following: > Command-line commands and options (Type **dir /p**.) > Button names (Click **Save As**.) > Check box and radio button names (Select the **Print Duplex** check box.) > Window titles (On the **Protect Document** window, click **Yes**.) > Field names (**User Name:** Enter the name of the user.) > Menu names (On the **File** menu, click **Save**.) (Click **Menu** > **Go To** > **Folders**.) > User input (In the **Date** box, type **April 1**.)
italic	The italic attribute is used for emphasis or to indicate a related document. (See the *Installation Guide* for more information.)
Double quote marks	Double quote marks enclose references to other sections within the document.
<variable>	In command descriptions, angle brackets represent variables. You must substitute a value for command line arguments that are enclosed in angle brackets.
[optional] [<optional>]	Square brackets enclose optional keywords or <variables> in a command line description. Optionally enter the keyword or <variable> that is enclosed in square brackets, if it is necessary or desirable to complete the task.
[a | b | c] [<a> | <b> | <c>]	Square brackets enclose optional alternate keywords or variables in a command line description. Choose one command line argument enclosed within the braces, if desired. Choices are separated by vertical (OR) bars.

{ a | b | c }
{ <a> | <b> | <c> }
	Braces enclose required alternate keywords or <variables> in a command line description. You must choose one command line argument enclosed within the braces. Choices are separated by vertical (OR) bars.

Support Contacts

If you encounter a problem while installing, registering, or operating this product, refer to the documentation. If you cannot resolve the issue, contact your supplier or Gemalto Customer Support.

Gemalto 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 Gemalto 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.gemalto.com, is a where you can find solutions for most common problems. The Customer Support Portal is a comprehensive, fully searchable database 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 Gemalto 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@gemalto.com.

CHAPTER 1: Introduction

Overview

This guide demonstrates using a SafeNet Luna HSM or HSM on Demand (HSMoD) service's PKCS#11 API to securely store a Hyperledger Admin User, Peer, and Orderer private keys. It guides administrator users through configuring the Blockchain Crypto Service Provider BCCSP to use the SafeNet Luna HSM or HSMoD service to generate and secure the Blockchain Admin. Peer, and Orderer application security keys.

This integration between SafeNet HSMs and Hyperledger Fabric uses the industry standard PKCS#11 interface to generate the identity keys and has more secure alternatives to software based keys. SafeNet 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 within a FIPS 140-2 certified hardware security module.

SafeNet 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 using SafeNet HSMs to generate the ECDSA signing keys for Blockchain Identities include the following:

- Secure generation, storage and protection of the private keys on FIPS 140-2 level 3 validated hardware.*
- Full life cycle management of the keys.
- HSM audit trail**.
- Take advantage of cloud services with confidence.
- Significant performance improvements by off-loading cryptographic operations from servers.

*validation for HSMoD services in progress.
**HSMoD services do not have access to the secure audit trail.

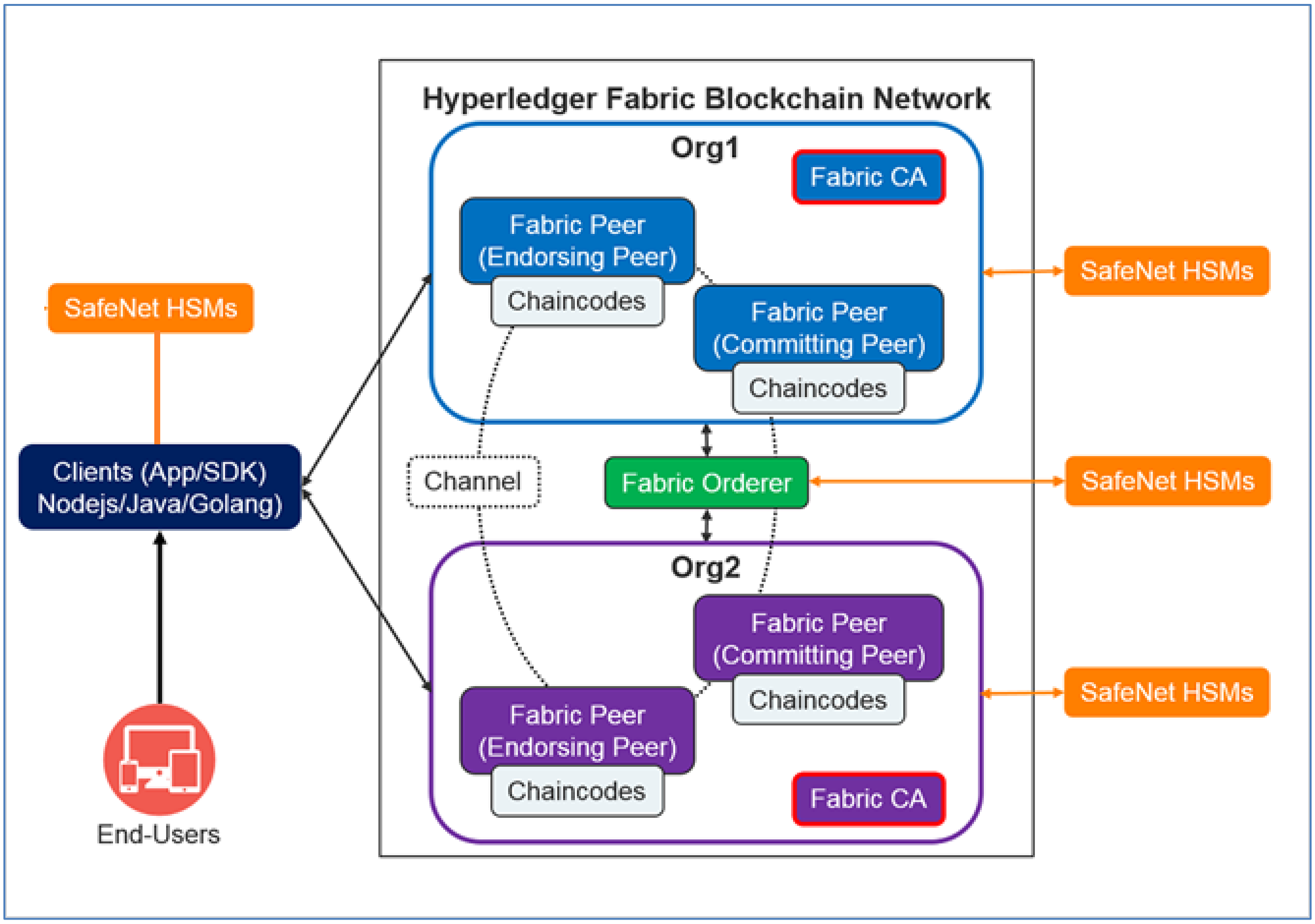
Understanding Hyperledger Fabric Blockchain Network

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 than 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 Fabric uses X.509 certificates as identities, 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 really matter because they determine the exact permissions over 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.

SafeNet HSMs are used to generate the key pairs for these identities (Peers, Orderers and Users). Below diagram shows the Hyperledger Fabric Blockchain Network using SafeNet HSMs.


Hyperledger Fabric Blockchain Network
Org1
Fabric CA
Fabric Peer (Endorsing Peer)
Chaincodes
Fabric Peer (Committing Peer)
Chaincodes
SafeNet HSMs
Clients (App/SDK) Nodejs/Java/Golang)
Channel
Fabric Orderer
SafeNet HSMs
Org2
Fabric Peer (Committing Peer)
Chaincodes
Fabric Peer (Endorsing Peer)
Chaincodes
Fabric CA
SafeNet HSMs
SafeNet HSMs
End-Users

Third Party Application Details

This integration uses the following third party applications:

- Hyperledger Fabric
- Hyperledger Fabric CA
- Hyperledger Fabric Client SDK (Node.js and Java)

Supported Platforms

List of the platforms which are tested with the following HSMs:

SafeNet Luna HSM: SafeNet 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. SafeNet Luna HSMs physically and logically secure cryptographic keys and accelerate cryptographic processing.

The SafeNet Luna HSM on premise offerings include the SafeNet Luna Network HSM, SafeNet PCIe HSM, and SafeNet Luna USB HSMs. SafeNet Luna HSMs are also available for access as an offering from cloud service providers such as IBM cloud HSM and AWS cloud HSM classic

The following platforms are supported for Hyperledger Fabric:

Platforms	Golang	Docker	Docker-Compose	Hyperledger
RHEL CentOS	1.10.1 1.11.2	17.06.2-ee	1.20.1 1.23.1	Hyperledger Fabric
Ubuntu	1.10	17.12.1-ce	1.19.0	Hyperledger Fabric
CentOS	1.12	18.06.1-ce	1.18.0	Hyperledger Fabric Client SDK Node
CentOS	1.12	18.06.1-ce	1.18.0	Hyperledger Fabric Client SDK Java

SafeNet DPoD: SafeNet Data Protection on Demand (DPoD) is a cloud-based platform that provides on-demand HSM and Key Management services through a simple graphical user interface. With DPoD, 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.

The following platforms are supported Hyperledger Fabric:

Platforms	Golang	Docker	Docker-Compose	Hyperledger
RHEL 64-bit	1.10.1 1.11.2	17.06.2-ee	1.20.1 1.23.1	Hyperledger Fabric
CentOS	1.12	18.06.1-ce	1.18.0	Hyperledger Fabric Client SDK Node
CentOS	1.12	18.06.1-ce	1.18.0	Hyperledger Fabric Client SDK Java

Prerequisites

Before you proceed with the integration, complete the following:

Configuring the SafeNet HSM

The SafeNet Luna HSM or HSMoD service configuration procedural material is included in the relevant chapter of this integration guide.

Installing Hyperledger Fabric and Fabric CA

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

To install the Hyperledger Fabric and Fabric CA prerequisite libraries

1. Install the following components on the system where you are generating the Peer, Orderer and User keys. Use the below command to install the components:

   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
   ```

To install and setup Golang

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

   - Installation steps: https://golang.org/doc/install
   - Download binaries: https://golang.org/dl/

To install and setup Docker and Docker Compose

1. Install Docker and Docker Compose on the host system. Follow the instructions available in the Docker documentation to install Docker and Docker Compose

   - Installation steps: https://docs.docker.com/engine/installation/linux/docker-ce/ubuntu/
   - Installation steps: https://docs.docker.com/install/linux/docker-ee/rhel/

2. Install the **docker-compose** run the following command.

   ```
   # sudo pip install docker-compose==<version>
   ```

   NOTE: It is always recommended to use the latest version available.

3. Configure the docker so that sudo is not required to run further commands:

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

4. Ensure that the **go** executable is in the **PATH**.

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

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://gerrit.hyperledger.org/r/fabric
   # cd fabric
   # git checkout -b v1.1.0 v1.1.0      (Hypeledger Fabric)
   # git checkout -b v1.3.0 v1.3.0      (Hypeledger Fabric)
   # git checkout -b v1.4.0 v1.4.0      (Hypeledger Fabric Client SDK)
   ```

   > **NOTE:** The instructions were developed against the tag v1.1.0 and v1.3.0 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. If using **v1.1.0** proceed to Step 4. For **v1.3.0** or above modify the **Makefile** as below:

   Modify:

   ```
   GO_TAGS ?=
   ```

   To:

   ```
   GO_TAGS ?= pkcs11
   ```

4. Remove the -static linking option in “**docker-env.mk**” file**.**

   Change:

   ```
   DOCKER_GO_LDFLAGS += -linkmode external -extldflags '-static -lpthread'
   ```

   To:

   ```
   DOCKER_GO_LDFLAGS += -linkmode external -extldflags '-lpthread'
   ```

   However static linking can be removed by exporting “**DOCKER_DYNAMIC_LINK**” environment variable.

   ```
   # export DOCKER_DYNAMIC_LINK=true
   ```

5. Add command to **images/orderer/Dockerfile.in** and **images/peer/Dockerfile.in** to install **libtool.**

   Right after:

   ```
   RUN mkdir -p /var/hyperledger/production $FABRIC_CFG_PATH
   ```

   Add:

   ```
   RUN apt-get update && apt-get install -y libtool
   ```

6. Build the docker images and executables.

```
# make docker
# make release
```

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

```
# cd $GOPATH/src/github.com/hyperledger
# git clone https://gerrit.hyperledger.org/r/fabric-ca
# cd fabric-ca
# git checkout -b v1.1.0 v1.1.0      (Hypeledger Fabric)
# git checkout -b v1.3.0 v1.3.0      (Hypeledger Fabric)
# git checkout -b v1.4.0 v1.4.0      (Hypeledger Fabric Client SDK)
```

> **NOTE:** The instructions were developed against the tag v1.1.0 and v1.3.0 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.

```
# make fabric-ca-client
```

CHAPTER 2: Integrating Hyperledger Fabric with an HSM on Demand service

SafeNet Data Protection on Demand (DPoD) provides strong cloud protection of secure assets, including keys, and should be considered a best practice when building systems based on Hyperledger Fabric.

This section provides procedural material on using a SafeNet Data Protection on Demand HSM on Demand (HSMoD) service to secure the keys for the Peer, Orderer and User nodes for a Hyperledger Fabric configuration. It guides administrator users through configuring the Blockchain Crypto Service Provider BCCSP to use the HSMoD service's PKCS#11 API to secure the application security keys.

This integration contains the following topics:

> Provisioning your HSMoD service

> Integrating Hyperledger Fabric with a HSMoD service

> Example: Running e2e_cli end-2-end execution using an HSMoD service

Provisioning your HSMoD service

This service provides your client machine with access to an HSM Application Partition for storing cryptographic objects used by your applications. Application partitions can be assigned to a single client, or multiple clients can be assigned to, and share, a single application partition.

> **NOTE:** Refer to the *SafeNet Data Protection on Demand Application Owner Quick Start Guide* for more information about provisioning the HSMoD service and initializing the service client.
>
> The HSMoD service client package is a zip file that contains system information you require to connect your client machine to an existing HSMoD service.

To create the HSMoD services for Hyperledger Fabric in Data Protection on Demand

1. Create the following HSMoD services in Data Protection on Demand
   - org1.example.com
   - org2.example.com
   - orderer.example.com
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. Follow the instructions in the DPOD *Application Owner Quick Start Guid*e and complete the following:
   - 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 e2e_cli.
>
> You need to set the partition password as per your organization security policy for a production environment.

Integrating Hyperledger Fabric with a HSMoD service

This section provides instructions on configuring the PKCS#11 BCCSP to use the HSMoD service and generating the keys for Peers, Orderers, and Users on an HSMoD service using the PKCS#11 BCCSP.

This section contains the following topics:

> Generating a CSR Using fabric-ca-client and PKCS11 BCCSP for an MSP Directory

> Configuring the Peer Nodes

> Configuring the Orderer Nodes

Generating a CSR Using fabric-ca-client and PKCS11 BCCSP for an 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 the BCCSP then it can be used to generate keys on the HSMoD service.

The fabric-ca-client BCCSP can be configured in the **~/.fabric-ca-client/fabric-ca-client-config.yaml** file.

To configure BCCSP to use the HSMoD services PKCS#11 API

The following is an example of a BCCSP configured to use the HSMoD services PKCS#11 API.

```
bccsp:
default: PKCS11
sw:
hash: SHA2
security: 256
filekeystore:
# The directory used for the software file-based keystore
keystore: msp/keystore
pkcs11:
```

```
library: /etc/hyperledger/fabric/dpod/org1.example.com/libs/64/libCryptoki2.so
label: org1.example.com
pin: userpin
hash: SHA2
security: 384
filekeystore:
# The directory used for the software file-based keystore
keystore: msp/keystore
```

You can add a **keyrequest** setting to the **csr** section of ~**/.fabric-ca-client/fabric-ca-client-config.yaml** to specify the key size as follows:

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

> **NOTE:** Use spaces not tabs and pay attention to the indentation. 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**

To generate the cryptographic keys using the gencsr command

The fabric-ca-client **gencsr** command generates ECDSA private keys on the HSM using the PKCS#11 BCCSP and creates a CSR in the **msp/signcerts** directory for the private key.

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>
```

The command to generate CSRs 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)
```

When generating a CSR request ensure that you have exported the correct partition label and 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.

To generate the key for **orderer.example.com** the command should look like the following:

```
# ./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.

As an example for generating the CSR for **peer0** of **org1.example.com**, execute the following:

```
# 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
```

The following command will generate 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.co
m/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"
```

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

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"
```

Configuring the Peer Nodes

To configure Peer nodes to access the keys from the HSMoD service you must set the environment variables for the PKCS#11 BCCSP library, partition, pin, and configuration files. In addition, you must mount the **core.yaml** file in the volumes section. The **core.yaml** file provides basic configuration options for various Peer modules.

To configure BCCSP to use the HSMoD Service PKCS#11 API

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

```
BCCSP:
Default: PKCS11
PKCS11:
# TODO: The default Hash and Security level needs refactoring to be
# fully configurable. Changing these defaults requires coordination
# SHA2 is hardcoded in several places, not only BCCSP
Hash: SHA2
Security: 384
Library: /etc/hyperledger/fabric/dpod/org1.example.com/libs/64/libCryptoki2.so
Label: org1.example.com
Pin: userpin
SoftwareVerify: true
SensitiveKeys: true
#FileKeyStore:
#    KeyStore:
```

> **NOTE:** Remove the “SensitiveKeys: true” from above snippet when using Hyperledger Fabric v1.3.0.
>
> Use spaces not tabs and pay attention to the indentation.

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>
```

Ensure that the DPOD client directory is available to the peer and to use the correct partition for that particular peer. The **ChrystokiConfigurationPath** must point to the DPOD client directory where **Chrystoki.conf** is available.

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

Configuring the Orderer Nodes

To configure the Orderer nodes to access the keys from the HSMoD service you must set the environment variables for the PKCS#11 BCCSP library, partition, pin and configuration file. Additionally you must mount the **orderer.yaml** file in the volume section. The **orderer.yaml** file provides basic configuration options for various Orderer modules.

To configure BCCSP to use the HSMoD services PKCS#11 API

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

```
BCCSP:
Default: PKCS11
PKCS11:
# TODO: The default Hash and Security level needs refactoring to be
# fully configurable. Changing these defaults requires coordination
# SHA2 is hardcoded in several places, not only BCCSP
Hash: SHA2
Security: 384
Library: /etc/hyperledger/fabric/dpod/org1.example.com/libs/64/libCryptoki2.so
Label: org1.example.com
Pin: userpin
SoftwareVerify: true
SensitiveKeys: true
#FileKeyStore:
#    KeyStore:
```

> **NOTE:** Remove the “SensitiveKeys: true” from above snippet when using Hyperledger Fabric v1.3.0.
>
> Use spaces not tabs and pay attention to the indentation.

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>
```

Ensure that the DPOD client directory is available to the Orderer and to use the correct partition for that Orderer. The **ChrystokiConfigurationPath** must point to the DPOD client directory of the Orderer where **Chrystoki.conf** is available.

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

You have completed the integration as 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 HSMoD service using the PKCS11 BCCSP. You have configured the **core.yaml** and the **orderer.yaml** to use the PKCS11 BCCSP and mounted them as a volume section of the Peer and Orderer configuration files.

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 DPOD 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 HSMoD service.

Example: Running e2e_cli end-2-end execution using an HSMoD service

This section of the guide demonstrates creating the Blockchain network, invoking transactions, and querying the state of the Blockchain using an e2e_cli example.

Before running end-2-end execution it is assumed that you have provisioned your HSMoD service and completed the **Prerequisites** section of this guide.

To run the e2e_cli end-2-end execution using an HSM on Demand service

1. Copy the compiled **fabric-ca-client** to **examples/e2e_cli** directory.

   ```
   # cd $GOPATH/src/github.com/hyperledger/fabric-ca/
   # cp bin/fabric-ca-client ../fabric/examples/e2e_cli
   # cd ../fabric/examples/e2e_cli
   ```

2. Modify **~/.fabric-ca-client/fabric-ca-client-config.yaml** file and do the following changes.

   ```
   bccsp:
   default: sw
   sw:
   hash: SHA2
   security: 256
   filekeystore:
   # The directory used for the software file-based keystore
   keystore: msp/keystore
   ```

```
 pkcs11:
 library:
 pin:
 label:
 hash: SHA2
 security: 384
 filekeystore:
 # The directory used for the software file-based keystore
 keystore: msp/keystore
```

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

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

> **NOTE:** Use spaces not tabs and pay attention to the indentation.

3. Copy the below script and save it as **gencerts.sh** to generate crypto material on the HSMoD service. 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
#Copyright (C) 2018 SafeNet. All rights reserved.
###########################################################################
# This script generates certificates and keys to work with the cryptogen util
# to be used with the e2e_cli Hyperledger fabric 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=./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
```

```
-------------------------------------------------------------------------------
```

4. Copy the **core.yaml** and **orderer.yaml** file to the **examples/e2e_cli** directory.

```
# cp ../../sampleconfig/core.yaml ../../sampleconfig/orderer.yaml .
```

> **NOTE:** You must copy the **core.yaml** and **orderer.yaml** files for all nodes.

5. Change the **BCCSP** section in the copied **core.yaml** and **orderer.yaml** files:

```
BCCSP:

Default: PKCS11

PKCS11:

# TODO: The default Hash and Security level needs refactoring to be

# fully configurable. Changing these defaults requires coordination

# SHA2 is hardcoded in several places, not only BCCSP

Hash: SHA2

Security: 384

Library:

Label:
```

```
Pin:
SoftwareVerify: true
SensitiveKeys: true
#FileKeyStore:
#    KeyStore:
```

> **NOTE:** Remove the “SensitiveKeys: true” from above snippet when using Hyperledger Fabric v1.3.0.
>
> Use spaces not tabs and pay attention to the indentation.

6. Make changes to **base/peer-base.yaml** (peer configuration) file.

   Add:

   ```
   - CORE_PEER_BCCSP_PKCS11_PIN=userpin
   ```

   To the **environment** section.

   Add a **volumes** section at the bottom of the file:

   ```
   services:
   peer-base:
   ...
   volumes:
   - ../core.yaml:/etc/hyperledger/fabric/core.yaml
   ```

7. Make changes to the **base/docker-compose-base.yaml** (orderer configuration) file.

   Add:

   ```
   - ORDERER_GENERAL_BCCSP_PKCS11_PIN=userpin
   ```

   To the **environment** section of **orderer.example.com**.

   Add:

   ```
   - ../orderer.yaml:/etc/hyperledger/fabric/orderer.yaml
   ```

   To the **volumes** section of **orderer.example.com**.

8. Make changes to the **docker-compose-cli.yaml** (client configuration) file:

   Add:

   ```
   environment:
   - ORDERER_GENERAL_BCCSP_PKCS11_LABEL=orderer.example.com
   -
   ORDERER_GENERAL_BCCSP_PKCS11_LIBRARY=/etc/hyperledger/fabric/dpod/orderer.examp
   le.com/libs/64/libCryptoki2.so
   - ChrystokiConfigurationPath=/etc/hyperledger/fabric/dpod/orderer.example.com
   ```

```
volumes:
-
/etc/hyperledger/fabric/dpod/orderer.example.com:/etc/hyperledger/fabric/dpod/orderer.example.com
```

To the bottom of the **orderer.example.com** section.

Add:

```
environment:
- 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
Volume
-  /etc/hyperledger/fabric/dpod/org1.example.com:/etc/hyperledger/fabric/dpod/org1.example.com
```

To the bottom of the **peer0.org1.example.com** and **peer1.org1.example.com** sections.

Add:

```
environment:
- 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
Volumes
-  /etc/hyperledger/fabric/dpod/org2.example.com:/etc/hyperledger/fabric/dpod/org2.example.com
```

To the bottom of the **peer0.org2.example.com** and **peer1.org2.example.com** section.

Add:

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

To the **cli** volumes section.

9. Modify the **scripts/scripts.sh** file:

   For Fabric v1.1.0

   In the **setGlobals** function after:

   ```
   if [ $1 -eq 0 -o $1 -eq 1 ] ; then
   ```

Add:

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

export CORE_PEER_BCCSP_PKCS11_PIN=userpin

export CORE_PEER_BCCSP_PKCS11_LABEL=org1.example.com

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

In the **setGlobals** function after:

```
else
```

Add:

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

export CORE_PEER_BCCSP_PKCS11_PIN=userpin

export CORE_PEER_BCCSP_PKCS11_LABEL=org2.example.com

export ChrystokiConfigurationPath=/etc/hyperledger/fabric/dpod/org2.example.com
```

And in the beginning of the **checkOSNAvailability** function add:

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

export CORE_PEER_BCCSP_PKCS11_PIN=userpin

export CORE_PEER_BCCSP_PKCS11_LABEL=orderer.example.com

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

For Fabric v1.3.0

In the **setGlobals** function after:

```
if [ $ORG -eq 1 ] ; then
```

Add:

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

export CORE_PEER_BCCSP_PKCS11_PIN=userpin

export CORE_PEER_BCCSP_PKCS11_LABEL=org1.example.com

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

After:

```
elif [ $ORG -eq 3 ] ; then
```

Add:

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

export CORE_PEER_BCCSP_PKCS11_PIN=userpin

export CORE_PEER_BCCSP_PKCS11_LABEL=org1.example.com

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

After:

```
else
```

Add:

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

export CORE_PEER_BCCSP_PKCS11_PIN=userpin

export CORE_PEER_BCCSP_PKCS11_LABEL=org2.example.com

export ChrystokiConfigurationPath=/etc/hyperledger/fabric/dpod/org2.example.com
```

And in the **checkOSNAvailability** function:

After:

```
CORE_PEER_LOCALMSPID="OrdererMSP"

CORE_PEER_TLS_ROOTCERT_FILE=$ORDERER_CA

CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp
```

Add:

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

export CORE_PEER_BCCSP_PKCS11_PIN=userpin

export CORE_PEER_BCCSP_PKCS11_LABEL=orderer.example.com

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

10. Modify the **generateArtifacts.sh** to use **gencerts.sh** to create key material by modifying the bottom of **generateArtifacts.sh** to as follows.

For Fabric v1.1.0

```
generateCerts

replacePrivateKey

./gencerts.sh

generateChannelArtifacts
```

For Fabric v1.3.0

```
generateCerts
generateIdemixMaterial
replacePrivateKey
./gencerts.sh
generateChannelArtifacts
```

11. In **network_setup.sh** comment the following line in **networkDown** function so that artifacts are not deleted.

    Change:

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

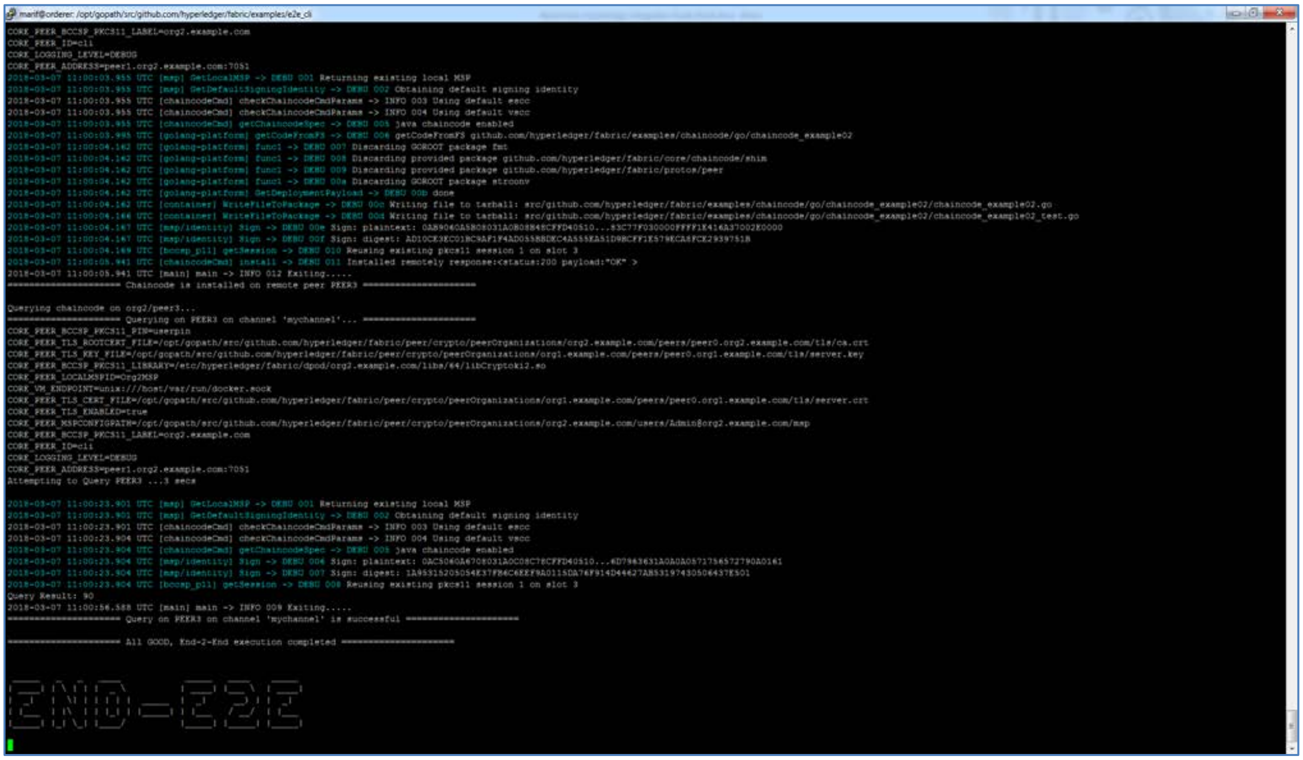
    To:

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

12. Run end-end execution of **e2e_cli**.

    ```
    # rm -rf crypto-config
    # ./generateArtifacts.sh
    # ./network_setup.sh up
    ```

    If all the above steps are followed then the "All GOOD, End-2-End execution completed" message displays. Refer the below screenshot for details.



manif@orderer: /opt/gopath/src/github.com/hyperledger/fabric/examples/e2e_cli

```
CORE_PEER_BCCSP_PKCS11_LABEL=org2.example.com
CORE_PEER_ID=cli
CORE_LOGGING_LEVEL=DEBUG
CORE_PEER_ADDRESS=peer1.org2.example.com:7051
2018-03-07 11:00:03.955 UTC [msp] GetLocalMSP -> DEBU 001 Returning existing local MSP
2018-03-07 11:00:03.955 UTC [msp] GetDefaultSigningIdentity -> DEBU 002 Obtaining default signing identity
2018-03-07 11:00:03.955 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 003 Using default escc
2018-03-07 11:00:03.955 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 004 Using default vscc
2018-03-07 11:00:03.955 UTC [chaincodeCmd] getChaincodeSpec -> DEBU 005 java chaincode enabled
2018-03-07 11:00:03.995 UTC [golang-platform] getCodeFromFS -> DEBU 006 getCodeFromFS github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02
2018-03-07 11:00:04.162 UTC [golang-platform] func1 -> DEBU 007 Discarding GOROOT package fmt
2018-03-07 11:00:04.162 UTC [golang-platform] func1 -> DEBU 008 Discarding provided package github.com/hyperledger/fabric/core/chaincode/shim
2018-03-07 11:00:04.162 UTC [golang-platform] func1 -> DEBU 009 Discarding provided package github.com/hyperledger/fabric/protos/peer
2018-03-07 11:00:04.162 UTC [golang-platform] func1 -> DEBU 00a Discarding GOROOT package strconv
2018-03-07 11:00:04.162 UTC [golang-platform] GetDeploymentPayload -> DEBU 00b done
2018-03-07 11:00:04.162 UTC [container] WriteFileToPackage -> DEBU 00c Writing file to tarball: src/github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02/chaincode_example02.go
2018-03-07 11:00:04.166 UTC [container] WriteFileToPackage -> DEBU 00d Writing file to tarball: src/github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02/chaincode_example02_test.go
2018-03-07 11:00:04.167 UTC [msp/identity] Sign -> DEBU 00e Sign: plaintext: 0AB9060A5B08031A0B08B48CFFD40510...83C77F030000FFFF1E416A37002E0000
2018-03-07 11:00:04.167 UTC [msp/identity] Sign -> DEBU 00f Sign: digest: AD10CE3EC01BC9AF1F4AD055BBDEC4A555EA51D9BCFF1E579ECA8FCE2939751B
2018-03-07 11:00:04.169 UTC [bccsp_p11] getSession -> DEBU 010 Reusing existing pkcs11 session 1 on slot 3
2018-03-07 11:00:05.941 UTC [chaincodeCmd] install -> DEBU 011 Installed remotely response:<status:200 payload:"OK" >
2018-03-07 11:00:05.941 UTC [main] main -> INFO 012 Exiting.....
===================== Chaincode is installed on remote peer PEER3 =====================

Querying chaincode on org2/peer3...
===================== Querying on PEER3 on channel 'mychannel'... =====================
CORE_PEER_BCCSP_PKCS11_PIN=userpin
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.key
CORE_PEER_BCCSP_PKCS11_LIBRARY=/etc/hyperledger/fabric/dpod/org2.example.com/libs/64/libCryptoki2.so
CORE_PEER_LOCALMSPID=Org2MSP
CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.crt
CORE_PEER_TLS_ENABLED=true
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
CORE_PEER_BCCSP_PKCS11_LABEL=org2.example.com
CORE_PEER_ID=cli
CORE_LOGGING_LEVEL=DEBUG
CORE_PEER_ADDRESS=peer1.org2.example.com:7051
Attempting to Query PEER3 ...3 secs

2018-03-07 11:00:23.901 UTC [msp] GetLocalMSP -> DEBU 001 Returning existing local MSP
2018-03-07 11:00:23.901 UTC [msp] GetDefaultSigningIdentity -> DEBU 002 Obtaining default signing identity
2018-03-07 11:00:23.901 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 003 Using default escc
2018-03-07 11:00:23.904 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 004 Using default vscc
2018-03-07 11:00:23.904 UTC [chaincodeCmd] getChaincodeSpec -> DEBU 005 java chaincode enabled
2018-03-07 11:00:23.904 UTC [msp/identity] Sign -> DEBU 006 Sign: plaintext: 0AC5060A6708031A0C08C78CFFD40510...6D7963631A0A0A0571756572790A0161
2018-03-07 11:00:23.904 UTC [msp/identity] Sign -> DEBU 007 Sign: digest: 1A95315205054E37FB6C6EEF9A0115DA76F914D44627AB531974305064 37E501
2018-03-07 11:00:23.904 UTC [bccsp_p11] getSession -> DEBU 008 Reusing existing pkcs11 session 1 on slot 3
Query Result: 90
2018-03-07 11:00:56.588 UTC [main] main -> INFO 009 Exiting.....
===================== Query on PEER3 on channel 'mychannel' is successful =====================

===================== All GOOD, End-2-End execution completed =====================
```

CHAPTER 3: Integrating Hyperledger Fabric with a SafeNet Luna Network HSM

SafeNet Luna HSMs provides strong physical protection of secure assets, including keys, and should be considered a best practice when building systems based on Hyperledger Fabric.

This section provides procedural material on using a SafeNet Luna HSM to secure the keys for the Peer, Orderer and User nodes for a Hyperledger Fabric configuration. It guides administrator users through configuring the Blockchain Crypto Service Provider BCCSP to use the SafeNet Luna HSM's PKCS#11 API to secure the application security keys.

This integration contains the following topics:

> Configuring the SafeNet Luna HSM

> Integrating Hyperledger Fabric with a SafeNet Luna HSM

> Example: Running e2e_cli end-2-end execution using SafeNet Luna HSM

Configuring the SafeNet Luna HSM

Configure the SafeNet Luna HSM partitions on your system for the Hyperledger Fabric Blockchain integration.

To configure the SafeNet Luna HSM

1. Ensure the HSM is setup, initialized, provisioned and ready for deployment. Refer to the HSM product documentation for help.
2. Create a partition on the HSM that will be later used by Hyperledger Fabric.
3. Depending on the variant of the HSM, create a separate partition for each Peer and Orderer organization on the Luna HSM and label them as follows.
   - org1.example.com
   - org2.example.com
   - orderer.example.com
4. Register a client for the system and assign the client to each partition to create an NTLS connection for the three partitions. Initialize Crypto Officer and Crypto User roles for each registered partition.
5. Ensure that each partition is successfully registered and configured. The command to see the registered partitions is:

```
# /usr/safenet/lunaclient/bin/lunacm
LunaCM v7.1.0-379. Copyright (c) 2006-2017 SafeNet.


Available HSMs:


Slot Id ->                0
Label ->                  org1.example.com
```

```
Serial Number ->        1238712343066
Model ->                LunaSA 7.1.0
Firmware Version ->     7.1.0
Configuration ->        Luna User Partition With SO (PED) Key Export With
Cloning Mode
Slot Description ->     Net Token Slot

Slot Id ->              1
Label ->                org2.example.com
Serial Number ->        1238712343062
Model ->                LunaSA 7.1.0
Firmware Version ->     7.1.0
Configuration ->        Luna User Partition With SO (PED) Key Export With
Cloning Mode
Slot Description ->     Net Token Slot

Slot Id ->              2
Label ->                orderer.example.com
Serial Number ->        1238712343063
Model ->                LunaSA 7.1.0
Firmware Version ->     7.1.0
Configuration ->        Luna User Partition With SO (PED) Key Export With
Cloning Mode
Slot Description ->     Net Token Slot
```

NOTE: Follow the SafeNet Network Luna HSM documentation for detailed steps for creating NTLS connection, initializing the partitions and various user roles.

NOTE: For demonstration purposes we have registered a single client for all of the separate partitions 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.

Integrating Hyperledger Fabric with a SafeNet Luna HSM

This section provides instructions on how to generate the keys for Peers, Orderers and Users on SafeNet Luna HSM using PKCS11 BCCSP and configuring the PKCS11 BCCSP to use the SafeNet Luna HSM.

This section contains the following topics:

> Generating a CSR Using fabric-ca-client and PKCS11 BCCSP for an MSP Directory

> Configuring Peer Nodes

> Configuring Orderer Nodes

Generating 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 SafeNet Luna HSM.

The fabric-ca-client BCCSP can be configured in the **~/.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**

To configure BCCSP to use the HSMoD services PKCS#11 API

Here is an example of a BCCSP configured to use the PKCS11 implementation and SafeNet Luna HSM:

```
bccsp:

default: PKCS11

sw:

hash: SHA2

security: 256

filekeystore:
# The directory used for the software file-based keystore

keystore: msp/keystore

pkcs11:

library: /usr/safenet/lunaclient/lib/libCryptoki2_64.so

pin: userpin

label: org1.example.com

hash: SHA2

security: 384

filekeystore:

# The directory used for the software file-based keystore

keystore: msp/keystore
```

You can add a keyrequest setting to the **csr** section of ~**/.fabric-ca-client/fabric-ca-client-config.yaml** to specify the key size as follows:

```
csr:

.

.

.

keyrequest:

algo: ecdsa
```

```
size: 384
```

NOTE: Use spaces not tabs and pay attention to the indentation.

To generate the cryptographic keys using the gencsr command

The **fabric-ca-client gencsr** command generates ECDSA private keys on the HSM using the PKCS11 BCCSP and creates a CSR stored in the **msp/signcerts** directory for the private key.

The PKCS11 values in the **fabric-ca-client-config.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 fabric-ca-client BCCSP.

```
# 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>
```

The command to generate CSRs 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)
```

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. To generate the key for **orderer.example.com** the command should look like the following:

```
# ./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.

As an example if you need to generate the CSR for the **peer0** of **org1.example.com**, execute the following:

```
# 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.

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"
```

Configuring Peer Nodes

To configure Peer nodes to access the keys from a SafeNet Luna HSM it is required to configure the BCCSP for PKCS11 in the **core.yaml** file and mount the **core.yaml** file in the volume section of the peer configuration file. The **core.yaml** file provides basic configuration options for various Peer modules.

To configure BCCSP to use the HSMoD services PKCS#11 API

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

```
BCCSP:

Default: PKCS11

PKCS11:

# TODO: The default Hash and Security level needs refactoring to be

# fully configurable. Changing these defaults requires coordination

# SHA2 is hardcoded in several places, not only BCCSP

Hash: SHA2

Security: 384

Library: /usr/safenet/lunaclient/lib/libCryptoki2_64.so

Label: org1.example.com

Pin: userpin

SoftwareVerify: true

SensitiveKeys: true

#FileKeyStore:

KeyStore:
```

NOTE: Remove the “SensitiveKeys: true” from above snippet when using Hyperledger Fabric v1.3.0.

Use spaces not tabs and pay attention to the indentation.

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>
```

Configuring Orderer Nodes

To configure Orderer nodes to access the keys from a SafeNet Luna HSM it is required to configure the BCCSP for PKCS11 in the **orderer.yaml** and mount the **orderer.yaml** file in the volume section of the Orderer configuration file. The **orderer.yaml** file provides basic configuration options for various Orderer modules.

To configure BCCSP to use the HSMoD services PKCS#11 API

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

```
BCCSP:

Default: PKCS11

PKCS11:

# TODO: The default Hash and Security level needs refactoring to be

# fully configurable. Changing these defaults requires coordination

# SHA2 is hardcoded in several places, not only BCCSP

Hash: SHA2

Security: 384

Library: /usr/safenet/lunaclient/lib/libCryptoki2_64.so

Label: orderer.example.com

Pin: userpin

SoftwareVerify: true

SensitiveKeys: true

#FileKeyStore:

#    KeyStore:
```

> **NOTE:** Remove the “SensitiveKeys: true” from above snippet when using Hyperledger Fabric v1.3.0.
>
> Use spaces not tabs and pay attention to the indentation.

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 as 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 SafeNet Luna HSM using the PKCS11 BCCSP. You have configured the **core.yaml** and the **orderer.yaml** 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 SafeNet 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.

Example: Running e2e_cli end-2-end execution using SafeNet Luna HSM

The next section of the guide describes the procedure to create the Blockchain network, to invoke transactions and to query the state using the e2e_cli example showing how to create the key materials on a SafeNet Luna HSM and run the end to end execution.

Before running **e2e_cli** end-2-end execution it is assumed that you have completed the **Prerequisites** section of this guide.

To run the e2e_cli end-2-end execution using an HSM on Demand service

1. Copy the compiled **fabric-ca-client** to **examples/e2e_cli** directory.

   ```
   # cd $GOPATH/src/github.com/hyperledger/fabric-ca/

   # cp bin/fabric-ca-client ../fabric/examples/e2e_cli

   # cd ../fabric/examples/e2e_cli
   ```

2. Modify **~/.fabric-ca-client/fabric-ca-client-config.yaml** file and do the following changes.

   Change the **bccsp** section to:

   ```
   bccsp:

   default: SW

   sw:

   hash: SHA2

   security: 256
   ```

```
filekeystore:

# The directory used for the software file-based keystore

keystore: msp/keystore

pkcs11:

library: /usr/safenet/lunaclient/lib/libCryptoki2_64.so

pin: userpin

label:

hash: SHA2

security: 384

filekeystore:

# The directory used for the software file-based keystore

keystore: msp/keystore
```

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

```
csr:

.

.

.

keyrequest:

algo: ecdsa

size: 384
```

> **NOTE:** Use spaces not tabs and pay attention to the indentation.

4. Copy the below script and save it as **gencerts.sh** to generate crypto material on SafeNet 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

#Copyright (C) 2018 SafeNet. All rights reserved.

###########################################################################

# This script generates certificates and keys to work with the cryptogen

# util

# to be used with the e2e_cli hyperledger fabric 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=./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
---------------------------------------------------------------------------
```

5. Copy the core.yaml and orderer.yaml in examples/e2e_cli directory:

```
# cp ../../sampleconfig/core.yaml ../../sampleconfig/orderer.yaml .
```

> **NOTE:** You must copy the core.yaml and orderer.yaml files for all nodes.

6. Change the BCCSP section in the copied **core.yaml** and **orderer.yaml** files:

```
BCCSP:
Default: PKCS11
PKCS11:
# TODO: The default Hash and Security level needs refactoring to be
# fully configurable. Changing these defaults requires coordination
# SHA2 is hardcoded in several places, not only BCCSP
Hash: SHA2
Security: 384
Library:
Label:
Pin:
```

```
SoftwareVerify: true
SensitiveKeys: true
#FileKeyStore:
#    KeyStore:
```

> **NOTE:** Remove the “SensitiveKeys: true” from above snippet when using Hyperledger Fabric v1.3.0.
>
> Use spaces not tabs and pay attention to the indentation.

Make changes to **base/peer-base.yaml** (peer configuration) file.

Add:

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

To the **environment** section.

Add a **volumes** section at the bottom of the file:

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

7. Make changes to the **base/docker-compose-base.yaml** (orderer configuration) file.

Add:

```
- ORDERER_GENERAL_BCCSP_PKCS11_LIBRARY=/usr/safenet/lunaclient/lib/libCryptoki2_64.so
- ORDERER_GENERAL_BCCSP_PKCS11_PIN=userpin
```

To the **environment** section of **orderer.example.com**.

Add:

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

To the **volumes** section of **orderer.example.com**.

8. Make changes to the **docker-compose-cli.yaml** (client configuration) file:

Add:

```
environment:
- ORDERER_GENERAL_BCCSP_PKCS11_LABEL=orderer.example.com
```

To the bottom of the **orderer.example.com** section.

Add:

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

To the bottom of the **peer0.org1.example.com** and **peer1.org1.example.com** sections.

Add:

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

To the bottom of the **peer0.org2.example.com** and **peer1.org2.example.com** section.

Add:

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

To the **cli** volumes section.

9. Modify the **scripts/scripts.sh** file:

For Fabric v1.1.0

Add:

```
export
CORE_PEER_BCCSP_PKCS11_LIBRARY=/usr/safenet/lunaclient/lib/libCryptoki2_64.so
export CORE_PEER_BCCSP_PKCS11_PIN=userpin
```

As the first two lines of the **setGlobals** function.

In the **setGlobals** function after:

```
if [ $1 -eq 0 -o $1 -eq 1 ] ; then
```

Add:

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

In the **setGlobals** function after:

```
else
```

Add:

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

And in the beginning of the **checkOSNAvailability** function add:

```
export
CORE_PEER_BCCSP_PKCS11_LIBRARY=/usr/safenet/lunaclient/lib/libCryptoki2_64.so
export CORE_PEER_BCCSP_PKCS11_PIN=userpin
export CORE_PEER_BCCSP_PKCS11_LABEL=orderer.example.com
```

For Fabric v1.3.0

In the **setGlobals** function

After:

```
PEER=$1
```

```
ORG=$2
```

Add:

```
export
CORE_PEER_BCCSP_PKCS11_LIBRARY=/usr/safenet/lunaclient/lib/libCryptoki2_64.so
export CORE_PEER_BCCSP_PKCS11_PIN=userpin
```

After:

```
if [ $ORG -eq 1 ] ; then
```

Add:

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

After:

```
elif [ $ORG -eq 3 ] ; then
```

Add:

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

After:

```
else
```

Add:

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

And in the **checkOSNAvailability** function:

After:

```
CORE_PEER_LOCALMSPID="OrdererMSP"
CORE_PEER_TLS_ROOTCERT_FILE=$ORDERER_CA
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/cryp
to/ordererOrganizations/example.com/orderers/orderer.example.com/msp
```

Add:

```
export
CORE_PEER_BCCSP_PKCS11_LIBRARY=/usr/safenet/lunaclient/lib/libCryptoki2_64.so
export CORE_PEER_BCCSP_PKCS11_PIN=userpin
export CORE_PEER_BCCSP_PKCS11_LABEL=orderer.example.com
```

10. Modify the **generateArtifacts.sh** to use **gencerts.sh** to create key material by modifying the bottom of **generateArtifacts.sh** to as follows.

For Fabric v1.1.0

```
generateCerts
replacePrivateKey
./gencerts.sh
generateChannelArtifacts
```

For Fabric v1.3.0

```
generateCerts
```

```
generateIdemixMaterial
replacePrivateKey
./gencerts.sh
generateChannelArtifacts
```

11. In **network_setup.sh** comment the following line in **networkDown** function so that artifacts are not deleted.

    Change:

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

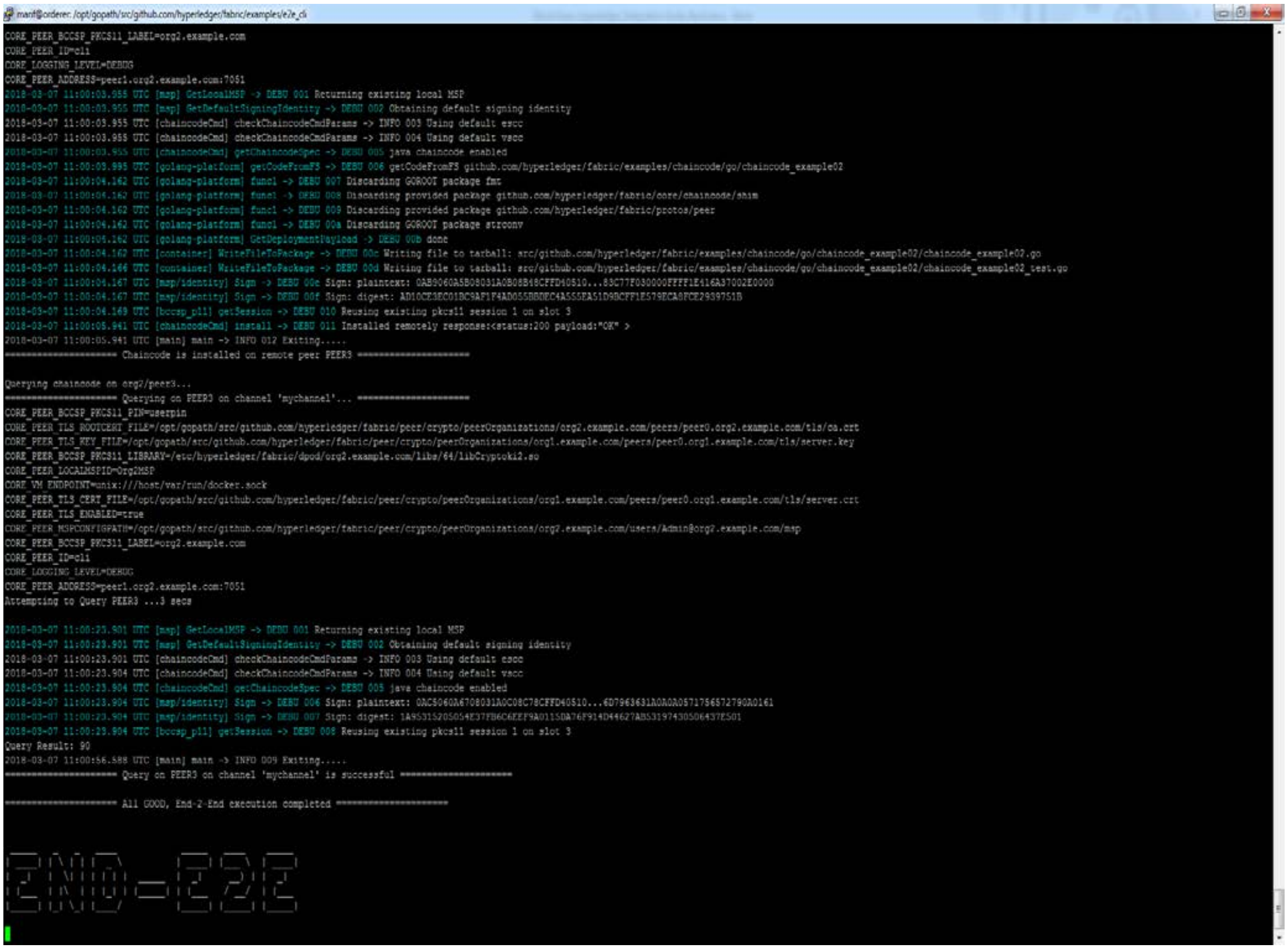
    To:

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

12. Run end-end execution of **e2e_cli**.

    ```
    # rm -rf crypto-config
    # ./generateArtifacts.sh
    # ./network_setup.sh up
    ```

    If the above steps are followed then the "All GOOD, End-2-End execution completed" message displays. Refer to the below screenshot for details.


manf@orderer: /opt/gopath/src/github.com/hyperledger/fabric/examples/e2e_cli

```
CORE_PEER_BCCSP_PKCS11_LABEL=org2.example.com
CORE_PEER_ID=cli
CORE_LOGGING_LEVEL=DEBUG
CORE_PEER_ADDRESS=peer1.org2.example.com:7051
2018-03-07 11:00:03.955 UTC [msp] GetLocalMSP -> DEBU 001 Returning existing local MSP
2018-03-07 11:00:03.955 UTC [msp] GetDefaultSigningIdentity -> DEBU 002 Obtaining default signing identity
2018-03-07 11:00:03.955 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 003 Using default escc
2018-03-07 11:00:03.955 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 004 Using default vscc
2018-03-07 11:00:03.955 UTC [chaincodeCmd] getChaincodeSpec -> DEBU 005 java chaincode enabled
2018-03-07 11:00:03.995 UTC [golang-platform] getCodeFromFS -> DEBU 006 getCodeFromFS github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02
2018-03-07 11:00:04.162 UTC [golang-platform] func1 -> DEBU 007 Discarding GOROOT package fmt
2018-03-07 11:00:04.162 UTC [golang-platform] func1 -> DEBU 008 Discarding provided package github.com/hyperledger/fabric/core/chaincode/shim
2018-03-07 11:00:04.162 UTC [golang-platform] func1 -> DEBU 009 Discarding provided package github.com/hyperledger/fabric/protos/peer
2018-03-07 11:00:04.162 UTC [golang-platform] func1 -> DEBU 00a Discarding GOROOT package strconv
2018-03-07 11:00:04.162 UTC [golang-platform] GetDeploymentPayload -> DEBU 00b done
2018-03-07 11:00:04.162 UTC [container] WriteFileToPackage -> DEBU 00c Writing file to tarball: src/github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02/chaincode_example02.go
2018-03-07 11:00:04.166 UTC [container] WriteFileToPackage -> DEBU 00d Writing file to tarball: src/github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02/chaincode_example02_test.go
2018-03-07 11:00:04.167 UTC [msp/identity] Sign -> DEBU 00e Sign: plaintext: 0AB9060A5B08031A0B08B48CFFD40510...83C77F030000FFFF1E416A37002E0000
2018-03-07 11:00:04.167 UTC [msp/identity] Sign -> DEBU 00f Sign: digest: AD10CE3EC01BC9AF1F4AD055BBDEC4A555EA51D9BCFF1E579ECA8FCE2939751B
2018-03-07 11:00:04.169 UTC [bccsp_p11] getSession -> DEBU 010 Reusing existing pkcs11 session 1 on slot 3
2018-03-07 11:00:05.941 UTC [chaincodeCmd] install -> DEBU 011 Installed remotely response:<status:200 payload:"OK" >
2018-03-07 11:00:05.941 UTC [main] main -> INFO 012 Exiting.....
===================== Chaincode is installed on remote peer PEER3 =====================

Querying chaincode on org2/peer3...
===================== Querying on PEER3 on channel 'mychannel'... =====================
CORE_PEER_BCCSP_PKCS11_PIN=userpin
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.key
CORE_PEER_BCCSP_PKCS11_LIBRARY=/etc/hyperledger/fabric/dpod/org2.example.com/libs/64/libCryptoki2.so
CORE_PEER_LOCALMSPID=Org2MSP
CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.crt
CORE_PEER_TLS_ENABLED=true
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
CORE_PEER_BCCSP_PKCS11_LABEL=org2.example.com
CORE_PEER_ID=cli
CORE_LOGGING_LEVEL=DEBUG
CORE_PEER_ADDRESS=peer1.org2.example.com:7051
Attempting to Query PEER3 ...3 secs

2018-03-07 11:00:23.901 UTC [msp] GetLocalMSP -> DEBU 001 Returning existing local MSP
2018-03-07 11:00:23.901 UTC [msp] GetDefaultSigningIdentity -> DEBU 002 Obtaining default signing identity
2018-03-07 11:00:23.901 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 003 Using default escc
2018-03-07 11:00:23.904 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 004 Using default vscc
2018-03-07 11:00:23.904 UTC [chaincodeCmd] getChaincodeSpec -> DEBU 005 java chaincode enabled
2018-03-07 11:00:23.904 UTC [msp/identity] Sign -> DEBU 006 Sign: plaintext: 0AC5060A6708031A0C08C78CFFD40510...6D7963631A0A0A0571756572790A0161
2018-03-07 11:00:23.904 UTC [msp/identity] Sign -> DEBU 007 Sign: digest: 1A95315205054E37FB6C6EEF9A0115DA76F914D44627AB53197430506437E501
2018-03-07 11:00:23.904 UTC [bccsp_p11] getSession -> DEBU 008 Reusing existing pkcs11 session 1 on slot 3
Query Result: 90
2018-03-07 11:00:56.588 UTC [main] main -> INFO 009 Exiting.....
===================== Query on PEER3 on channel 'mychannel' is successful =====================

===================== All GOOD, End-2-End execution completed =====================
```

CHAPTER 4: Integrating Hyperledger Fabric Client with a SafeNet Luna Network HSM or HSM on Demand Service

This section provides procedural material on integrating a SafeNet Luna HSM or SafeNet Data Protection on Demand (DPoD) HSM on Demand (HSMoD) service with a Hyperledger Fabric Client configuration to secure the Peer, Orderer and Admin User.

This chapter guides administrator users through configuring the Blockchain Crypto Service Provider (BCCSP) to use the SafeNet Luna HSM's or HSMoD service PKCS#11 API to secure the client security keys.

Configure or provision the appropriate HSM for your integration. This integration contains the following topics:

> Configuring the SafeNet Luna HSM

> Provisioning the HSM on Demand service

> Integrating the Hyperledger Fabric Client SDK for Node.js with a SafeNet HSM

> Integrating Hyperledger Fabric Client SDK for Java with a SafeNet HSM

Configuring the SafeNet Luna HSM

Configure the SafeNet Luna HSM on the system to carry on the integration.

Follow the *SafeNet Luna Network HSM Product Documentation* for detailed steps for creating the NTLS connection, initializing the partitions, and initializing the Security Officer, Crypto Officer, and Crypto User roles.

1. Ensure the HSM is setup, initialized, provisioned and ready for deployment. Refer to the HSM product documentation for help.
2. Create a partition on the HSM that will be later used by Hyperledger Fabric Client.
3. If using a SafeNet 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.
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) 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 security policy if implementing in a production environment.

Provisioning the HSM on Demand service

This service provides your client machine with access to an HSM Application Partition for storing cryptographic objects used by your applications. Application partitions can be assigned to a single client, or multiple clients can be assigned to, and share, a single application partition.

> **NOTE:** Refer to the *SafeNet Data Protection on Demand Application Owner Quick Start Guide* for more information about provisioning the HSMoD service and initializing the service client.
>
> The HSMoD service client package is a zip file that contains system information you require to connect your client machine to an existing HSMoD service.

To create the HSMoD service for Hyperledger Fabric in Data Protection on Demand

1. Create the **fabric-sdk** HSMoD service in Data Protection on Demand.
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:
   - 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.

Integrating the Hyperledger Fabric Client SDK for Node.js with a SafeNet 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 SafeNet Luna HSM or HSMoD service and run end-2-end execution, complete the following.

> **NOTE:** Ensure that all Prerequisites are completed and the SafeNet Luna HSM or HSMoD service client is configured on the system to carry on the integration.

To integrate the Hyperledger Fabric Client SDK Node

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. Checkout 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. Checkout 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:
# The directory used for the software file-based keystore
keystore: msp/keystore
```

> **NOTE:** Use spaces not tabs and pay attention to the indentation. Ensure that the library path points to the correct SafeNet 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
```

> **NOTE:** 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 SafeNet Cryptoki library when using Safenet Luna HSM or HSMoD Service.

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

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

Following snippet will be shown when test 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 cleanup 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
    ```

    Now step 12 can be performed to execute end-2-end HSM integration test again.

Integrating Hyperledger Fabric Client SDK for Java with a SafeNet 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 helps facilitate 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 monitor events on the channel.

To generate the keys on SafeNet Luna HSM or HSMoD service and run end-2end execution, complete the following.

> **NOTE:** Ensure that all Prerequisites are completed and the SafeNet Luna HSM or HSMoD service client is configured on the system to carry on the integration.

To 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 HSMoD are available at <DPoD Installation Directory>/jsp/LunaProvider.jar and <DPoD 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:

# The directory used for the software file-based keystore

keystore: msp/keystore
```

> **NOTE:** Use spaces not tabs and pay attention to the indentation. Ensure that the library path points to the correct SafeNet Cryptoki library when using Safenet Luna HSM or HSMoD 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
    ```

    Following snippet will be shown when test 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 cleanup 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
```

Now step 12 can be performed to execute end-2-end HSM integration test again.