THALES

OpenSSL: Integration Guide

THALES LUNA HSM AND DPOD LUNA CLOUD HSM

Document Information

Document Part Number	007-012068-001
Revision	T
Release Date	5 October 2021

Trademarks, Copyrights, and Third-Party Software

CONTENTS

Overview

This document contains steps to install, configure, and integrate OpenSSL with a Luna HSM or Luna Cloud HSM service. OpenSSL is an open source project that consists of a cryptographic library and an SSL/TLS toolkit. OpenSSL provides command-line tools for cryptographic operations including symmetric encryption, public-key encryption, and digital signing hash.

Luna HSMs can be used to securely store the OpenSSL cryptographic keys. OpenSSL integrates with GemEngine to consume HSM resources. The benefits of using Luna HSMs to generate the cryptographic keys for OpenSSL are:

- Secure generation, storage and protection of the Identity signing private key on FIPS 140-2 level 3 validated hardware
- Full life cycle management of the keys
- Significant performance improvements by off-loading cryptographic operations from application servers
- HSM audit trail*

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

Certified Platforms

This integration is certified on the following platforms:

Certified platforms on Luna HSM

Certified platforms on Luna Cloud HSM

Certified platforms on Luna HSM

HSM Type	OpenSSL Toolkit	Platforms
Luna HSM	GemEngine 1.5	Windows Server 2019 RHEL 8
Luna HSM	GemEngine 1.2 GemEngine 1.3	Windows Server 2019 Windows Server 2016 Windows Server 2012 R2 RHEL 7

NOTE: This integration is tested with Luna Clients in HA and FIPS Mode.

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

Certified platforms on Luna Cloud HSM

HSM Type	OpenSSL Toolkit	Platforms
Luna Cloud HSM	GemEngine 1.5	Windows Server 2019 RHEL 8
Luna Cloud HSM	GemEngine 1.3	Windows Server 2019 Windows Server 2016 RHEL 7

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

Prerequisites

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

Configure Luna HSM

Configure Luna HSM service

Set up OpenSSL toolkit

Configure Luna HSM

If you are using Luna HSM:

1. Verify that the HSM is set up, initialized, provisioned, and ready for deployment. Refer to the Luna HSM documentation for more information.
2. Create a partition on the HSM that will be later used by OpenSSL.
3. If you are using a Luna Network HSM, register a client for the system and assign the client to the partition to create an NTLS connection. Initialize the Crypto Officer and Crypto User roles for the registered partition.
4. Ensure that each partition is successfully registered and configured. The command to see the registered partitions is:

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

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

Available HSMs:

Slot Id ->              0

Label ->                OPENSSL

Serial Number ->        1238686731832

Model ->                LunaSA 7.7.1

Firmware Version ->     7.7.1

Bootloader Version ->   1.1.2

Configuration ->        Luna User Partition With SO (PW) Key Export With
```

```
Cloning Mode
Slot Description ->      Net Token Slot
FM HW Status ->          Non-FM
```

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

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

Set up Luna HSM High-Availability

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

Using Luna HSM in FIPS mode

Under FIPS 186-3/4, the RSA methods permitted for generating keys are 186-3 with primes and 186-3 with aux primes. This means that RSA PKCS and X9.31 key generation is no longer approved for operation in a FIPS-compliant HSM. If you are using the Luna HSM in FIPS mode, you have to make the following change in configuration file:

For Linux

```
Misc = {
RSAKeyGenMechRemap = 1;
}
```

For Windows

```
[Misc]
RSAKeyGenMechRemap=1
```

The above setting redirects the older calling mechanism to a new approved mechanism when Luna HSM is in FIPS mode.

> **NOTE:** The above setting is not required for Universal Client. This setting is applicable only for Luna Client 7.x.

Configure Luna Cloud HSM service

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

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

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

Standalone Cloud HSM service using minimum client package

To configure Luna Cloud HSM service using minimum client package:

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

   ```
   [Windows]
   cvclient-min.zip
   [Linux]
   cvclient-min.tar
   # tar -xvf cvclient-min.tar
   ```

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

   ```
   [Windows]
   Right-click setenv.cmd and select Run as Administrator.
   [Linux]
   Source the setenv script.
   # source ./setenv
   ```

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

Standalone Cloud HSM service using full Luna client package

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

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

   ```
   [Windows]
   cvclient-min.zip
   [Linux]
   cvclient-min.tar
   # tar -xvf cvclient-min.tar
   ```

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

   ```
   [Windows]
   Right-click setenv.cmd and select Run as Administrator.
   [Linux]
   Source the setenv script.
   # source ./setenv
   ```

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

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

   Cloud HSM Certificates:

   ```
   server-certificate.pem
   partition-ca-certificate.pem
   partition-certificate.pem
   ```

   LunaClient Certificate Directory:

   ```
   [Windows default location for Luna Client]
   C:\Program Files\Safenet\Lunaclient\cert\
   [Linux default location for Luna Client]
   /usr/safenet/lunaclient/cert/
   ```

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

   ```
   [Windows]
   crystoki.ini
   [Linux]
   Chrystoki.conf
   ```

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

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

   [XTC]

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

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

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

   ```
   Misc]
   PluginModuleDir=<LunaClient_plugins_directory>

   [Windows Default]
   C:\Program Files\Safenet\Lunaclient\plugins\
   ```

```
[Linux Default]
/usr/safenet/lunaclient/plugins/
```

Save the configuration file. If you wish, you can now safely delete the extracted Cloud HSM service client directory.

10. Reset the ChrystokiConfigurationPath environment variable and point back to the location of the Luna Client configuration file.

    Windows

    In the Control Panel, search for "environment" and select Edit the system environment variables. Click Environment Variables. In both list boxes for the current user and system variables, edit ChrystokiConfigurationPath and point to the crystoki.ini file in the Luna client install directory.

    Linux

    Either open a new shell session, or export the environment variable for the current session pointing to the location of the Chrystoki.conf file:

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

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

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

Luna HSM and Luna Cloud HSM service in hybrid mode

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

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

To use Luna Cloud HSM Service in FIPS mode

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

Set up OpenSSL toolkit

Acquire the OpenSSL toolkit with GemEngine support from Thales Customer Support.

> **NOTE:** The Doc ID for downloading the GemEngine v1.2 from support portal is KB0016309. The Doc ID for downloading the GemEngine v1.3 from support portal is KB0017806.
>
> The Doc ID for downloading the GemEngine v1.5 from support portal is KB0024584.

We recommend you familiarize yourself with OpenSSL. Refer to OpenSSL Documentation for more information about OpenSSL.

Integrating Luna HSM with OpenSSL using GemEngine

To integrate Luna HSM with OpenSSL using GemEngine, follow the steps mentioned below in accordance with your environment configuration:

> Integrate OpenSSL with Luna HSM on UNIX

> Integrate OpenSSL with Luna HSM on Windows

Integrate OpenSSL with Luna HSM on UNIX

To integrate OpenSSL with Luna HSM on Unix, follow these steps:

> Install and configure OpenSSL toolkit

> Configure GemEngine to use Luna HSM on Unix

> Verify the OpenSSL and GemEngine Integration on Unix

Install and configure OpenSSL toolkit

To install and configure OpenSSL toolkit on Unix, follow the steps mentioned in one of the scenarios below, as per your requirements:

> Scenario A: Integrate pre-built dynamic engine with an existing installation of OpenSSL

> Scenario B: Compile the dynamic engine and integrating with an existing installation of OpenSSL

> Scenario C: Compile and install OpenSSL from source and compiling and installing GemEngine

> Scenario D: Configure OpenSSL to enable GemEngine by default

Scenario A: Integrate pre-built dynamic engine with an existing installation of OpenSSL

1. Extract the GemEngine toolkit and navigate to the GemEngine directory.
2. Locate the libgem.so and sautil files in:

   ```
   builds/linux/<distributor>/<bit_version>/<OpenSSL_version>
   ```

   Example:

   ```
   builds/linux/rhel/64/1.0
   ```

   > **NOTE:** Locate the correct dynamic engine library **libgem.so** in the correct **<OpenSSL_version>** directory. For OpenSSL version 1.1.1x, the dynamic engine library is gem.so.

3. Use gembuild to locate the OpenSSL Engines directory.

   ```
   # ./gembuild locate-engines
   "/usr/local/lib64/engines"
   ```

4. Copy the libgem.so or gem.so to the engines directory and test engine.

   Example:

   ```
   # cp builds/linux/rhel/64/1.0.1/libgem.so /usr/local/lib64/engines/
   ```

5. Verify the GemEngine is present.

```
# openssl engine gem –v
(gem) Gem engine support
enginearg, openSession, closeSession, login, logout, engineinit,
CONF_PATH, ENGINE_INIT, ENGINE2_INIT, engine2init, DisableCheckFinalize
SO_PATH, GET_HA_STATE, SET_FINALIZE_PENDING, SKIP_C_INITIALIZE,
IntermediateProcesses
```

   If the output looks as above, then the GemEngine is successfully installed.

6. Copy the sautil utility to /usr/local/bin

   Example:

```
# cp builds/linux/rhel/64/1.0.1/sautil /usr/local/bin
# /usr/local/bin/sautil
```

Scenario B: Compile the dynamic engine and integrating with an existing installation of OpenSSL

> **NOTE:** Ensure the system has a C compiler and access to the make utility.

1. Download and extract the OpenSSL source tarball. We recommend the version that is closest to your existing OpenSSL installation (run openssl version).

   Example:

   Download openssl-x.x.xx.tar.gz from https://www.openssl.org/source/

```
# tar xvfz openssl-x.x.xx.tar.gz
```

> **NOTE:** Ensure you download a supported OpenSSL version. See **Error! Reference source not found.** and Certified Platforms for more information about supported OpenSSL and GemEngine toolkit versions.

2. Locate the OpenSSL engine directory.

```
# ./gembuild locate-engines
```

3. Run `gembuild config`.

   Example:

```
# ./gembuild config --openssl-source=/home/openssl-x.x.xx --openssl-
engines=/usr/lib64/openssl/engines --config-bits=64
```

> **NOTE:** If the OpenSSL development package is not available, you need to install it on the system. This example command assumed that the OpenSSL headers directory is located in /usr/include. If the header files are located in a different location, the --openssl-includes option can be used. The --openssl-libs options can be used to specify the location of the lib directory with libcrypto.so. All paths need to be absolute.

4. Install the required EC header files.

```
/gembuild openssl-ec-headers
```

5. Compile the engine.

```
# ./gembuild engine-build
```

> **NOTE:** If you encounter error in this step, refer to Problem in the Troubleshooting section.

6. Install and test the engine.

```
# ./gembuild engine-install
# openssl engine gem –v
gem) Gem engine support
enginearg, openSession, closeSession, login, logout, engineinit,
CONF_PATH, ENGINE_INIT, ENGINE2_INIT, engine2init, DisableCheckFinalize,
SO_PATH, GET_HA_STATE, SET_FINALIZE_PENDING, SKIP_C_INITIALIZE
```

7. Compile and install sautil. By default this will install the sautil command to /usr/local/bin/sautil. If a different location is desired, use the --sautil-prefix option to specify the desired directory either by redoing STEP 3 with the option or by specifying the option as part of the ./gembuild sautil-install command.

```
# ./gembuild sautil-build
# ./gembuild sautil-install
```

Scenario C: Compile and install OpenSSL from source and compiling and installing GemEngine

1. Download and extract the OpenSSL source tarball.

   Example:

   Download openssl-x.x.xx.tar.gz from https://www.openssl.org/source/

```
# tar xvfz openssl-x.x.xx.tar.gz
```

2. If using FIPS download and extract the OpenSSL FIPS module. If not using FIPS proceed to step 3.

   Example:

   Download openssl-fips-2.0.x.tar.gz from https://www.openssl.org/source/

```
# tar xvfz openssl-fips-2.0.x.tar.gz
```

3. Run gembuild config using the --prefix option. If using the FIPS module, add the --openssl-fips-source=/home/openssl-fips-2.0.x to the ./gembuild config command.

   Example:

```
# ./gembuild config --openssl-source=/home/openssl-x.x.xx --prefix=/usr/local --config-bits=64
```

> **NOTE:** If you are using GemEngine 1.3 or above and OpenSSL 1.0.2x, then use the option **--compat-102** in the above command.

4. If using FIPS, compile and install the FIPS module. If not using FIPS, proceed to step 5.

   FIPS:

```
# ./gembuild openssl-fips-build
# ./gembuild openssl-fips-install
```

5. Compile and install the OpenSSL module.

```
# ./gembuild openssl-build
```

```
# ./gembuild openssl-install
```

6. Compile and install gem dynamic engine and verify the engine.

```
# ./gembuild engine-build
```

> **NOTE:** If you encounter error in this step, refer to Problem 2 and 3 in the Troubleshooting section.

```
# ./gembuild engine-install
# /usr/local/ssl/bin/openssl engine gem -v
(gem) Gem engine support
enginearg, openSession, closeSession, login, logout, engineinit,
CONF_PATH, ENGINE_INIT, ENGINE2_INIT, engine2init, DisableCheckFinalize,
SO_PATH, GET_HA_STATE, SET_FINALIZE_PENDING, SKIP_C_INITIALIZE
```

7. Compile and install sautil.

```
# ./gembuild sautil-build
# ./gembuild sautil-install
```

> **NOTE:** This installs the sautil utility to <prefix>/sautil/bin/sautil where <prefix> is the directory specified with the –prefix option in step 3.
>
> If you require a different location include the –sautil-prefix option to specify the desired directory as part of the /gembuild sautil –install command.

8. Add openssl and sautil to PATH

   Example:

```
# export PATH=/usr/local/ssl/bin:/usr/local/sautil/bin:$PATH
```

9. Add openssl library to LD_LIBRARY_PATH

   Example:

```
# export LD_LIBRARY_PATH=/usr/local/ssl/lib:$LD_LIBRARY_PATH
```

Scenario D: Configure OpenSSL to enable GemEngine by default

This integration assumes that OpenSSL with GemEngine is installed at /usr/local/ssl/bin/openssl

1. Locate the openssl.cnf and engines directory.

```
# openssl version -d
OPENSSLDIR: "/usr/local/ssl"
```

   This provides the location of the OpenSSL directory.

```
# ./gembuild locate-engines
```

   This gives the location of the directory where the libgem.so or gem.so should be located.

> **NOTE:** This is the OpenSSL that is set to the PATH. Run **which openssl** to verify you are accessing the correct configuration file.

2. Edit the openssl.cnf file.

   Example:

```
# Insert near top of file openssl.cnf:
openssl_conf = openssl_init
# Insert at bottom of file openssl.cnf:
[ openssl_init ]
engines = engine_section
[ engine_section ]
gem = gem_section
[ gem_section ]
dynamic_path = /usr/local/ssl/lib/engines/libgem.so
default_algorithms = ALL
```

   NOTE: Make sure to give correct dynamic engine library in dynamic_path.

3. Verify the engine is loaded without specifying it.

```
# openssl engine -v
(dynamic) Dynamic engine loading support
SO_PATH, NO_VCHECK, ID, LIST_ADD, DIR_LOAD, DIR_ADD, LOAD
(gem) Gem engine support
enginearg, openSession, closeSession, login, logout, engineinit,
CONF_PATH, ENGINE_INIT, ENGINE2_INIT, engine2init, DisableCheckFinalize,
SO_PATH, GET_HA_STATE, SET_FINALIZE_PENDING, SKIP_C_INITIALIZE,
```

4. Test the application by generating a certificate request using RSA key.

```
# openssl req -out CSR.csr -new -newkey rsa:2048 -nodes -keyout privateKey.key
```

Configure GemEngine to use Luna HSM on Unix

To configure GemEngine to use Luna HSM on Unix, complete one of the following tasks, depending on your requirements:

Configure GemEngine for Luna HSM

Configure GemEngine for Luna HA slot or Luna Cloud HSM service

Configure GemEngine for Luna HSM (single partition)

To configure GemEngine for Luna HSM (single partition):

1. Open the /etc/Chrystoki.conf file and add the following GemEngine section:

```
GemEngine = {
LibPath = /usr/safenet/lunaclient/lib/libCryptoki2.so;
LibPath64 = /usr/safenet/lunaclient/lib/libCryptoki2_64.so;
EnableDsaGenKeyPair = 1;
```

```
EnableRsaGenKeyPair = 1;
DisablePublicCrypto = 1;
EnableRsaSignVerify = 1;
EnableLoadPubKey = 1;
EnableLoadPrivKey = 1;
DisableCheckFinalize = 1;
DisableEcdsa = 1;
DisableDsa = 0;
DisableRand = 0;
EngineInit = <slot_id>:10:11;
}
```

2. Run the sautil utility to open the persistent sautil session on Luna HSM slot.

```
# /usr/local/sautil/bin/sautil -v -s <slot_id> -i 10:11 -o -q
```

> **NOTE:** For more login methods for session refer to the *README-GEM-CONFIG* file present in the **<GemEngine_Directory>/docs** folder.

Configure GemEngine for Luna HA slot or Luna Cloud HSM service

1. Create a text file and store the partition password in it.

```
# echo <partition_password> > <path_to_my_passfile>/passfile
```

2. Open the /<ChrystokiConfigurationFile_Directory>/Chrystoki.conf file and add the following code to the GemEngine section:

```
GemEngine = {
LibPath = <path to LibCryptoki2.so>;
LibPath64 = <path to LibCryptoki2.so>;
EnableDsaGenKeyPair = 1;
EnableRsaGenKeyPair = 1;
DisablePublicCrypto = 1;
EnableRsaSignVerify = 1;
EnableLoadPubKey = 1;
EnableLoadPrivKey = 1;
DisableCheckFinalize = 1;
DisableEcdsa = 1;
DisableDsa = 0;
DisableRand = 0;
EngineInit = "<Partition Label>":0:0:passfile=<path_to_my_passfile>/passfile;
EnableLoginInit = 1;
```

3. If you are using Luna Cloud HSM, then open /<ChrystokiConfigurationFile_Directory>/Chrystoki.conf file and add the following code inside the Misc section:

```
Misc = {
   FinalizeOnClose = 1;
}
```

Verify the OpenSSL and GemEngine Integration on Unix

Complete the following to verify the OpenSSL and GemEngine integration on Unix:

> Generate cryptographic objects for OpenSSL CMS

> Use OpenSSL CMS cryptographic objects to sign, verify, encrypt, and decrypt a message

Generate cryptographic objects for OpenSSL CMS

Generate the CA private key and CA certificate for OpenSSL CMS and then use the CA private key and certificate to generate the receiver and sender certificates. To generate cryptographic objects for OpenSSL CMS:

1. Open the /< OPENSSLDIR>/openssl.cnf file in a text editor and edit the [CA_default] section. Make the following changes:

```
dir                = /usr/local/ssl
new_certs_dir      = $dir/certs
```

> **NOTE:** You can change dir to the directory of your choice, but make sure to use correct path in the subsequent steps.

2. Create the text files /usr/local/ssl/index.txt and /usr/local/ssl/serial.
3. Open the /usr/local/ssl/serial file and write 01 at the top and press enter. Save the file.
4. Create a 2048-bit private key using GemEngine

```
# openssl genrsa -engine gem 2048
```

   This generates a RSA 2048 CA private key on the Luna HSM or Luna Cloud HSM service.

5. List the generated key pair using cmu utility.

```
# <LunaClient_Installation_Directory>/bin/cmu list
```

   For example:

```
# /usr/safenet/lunaclient/bin/cmu list
```

   Provide partition password when prompted.

6. Create a CA certificate based on the generated key that is used for signing other certificates:

```
# openssl req -engine gem -new -x509 -days 365 -key rsa-private-
a55e015d94ee6c4a559dfab7c39a2069d4064bcd -keyform engine -out
/usr/local/ssl/certs/ca.cer
```

   Where rsa-private-a55e015d94ee6c4a559dfab7c39a2069d4064bcd is the object label for the CA private key on the Luna HSM or Luna Cloud HSM service created in step 4.

7. Create a directory to generate the certificate request for sender and receiver.

```
# mkdir /usr/local/ssl/certs/sender
```

```
# mkdir /usr/local/ssl/certs/receiver
```

8. Generate a certificate request for sender.

```
# openssl req -engine gem -newkey rsa:2048 -out
/usr/local/ssl/certs/sender/sender.txt
```

Sender request is used to generate the sender's certificate signed by the Certificate Authority.

9. List the generated key pair using cmu utility.

```
# <LunaClient_Installation_Directory>/bin/cmu list
```

For example:

```
# /usr/safenet/lunaclient/bin/cmu list
```

Provide partition password when prompted.

10. Generate a certificate request for receiver.

```
# openssl req -engine gem -newkey rsa:2048 -out
/usr/local/ssl/certs/receiver/receiver.txt
```

Receiver request is used to generate the receiver's certificate signed by the Certificate Authority.

11. List the generated key pair using cmu utility.

```
# <LunaClient_Installation_Directory>/bin/cmu list
```

For example:

```
# /usr/safenet/lunaclient/bin/cmu list
```

Provide partition password when prompted.

12. Sign the certificate request for Sender by CA (Certificate Authority).

```
# openssl ca -engine gem -policy policy_anything -cert
/usr/local/ssl/certs/ca.cer -in /usr/local/ssl/certs/sender/sender.txt -keyfile
rsa-private-a55e015d94ee6c4a559dfab7c39a2069d4064bcd -keyform engine -out
/usr/local/ssl/certs/sender/sender.cer
```

Where rsa-private-a55e015d94ee6c4a559dfab7c39a2069d4064bcd is the object label for the ca private key on the Luna HSM or Luna Cloud HSM service created in step 4.

13. Sign the certificate request for Receiver by CA (Certificate Authority).

```
# openssl ca -engine gem -policy policy_anything -cert
/usr/local/ssl/certs/ca.cer -in /usr/local/ssl/certs/receiver/receiver.txt -
keyfile rsa-private-a55e015d94ee6c4a559dfab7c39a2069d4064bcd -keyform engine -
out /usr/local/ssl/certs/receiver/receiver.cer
```

Where rsa-private-a55e015d94ee6c4a559dfab7c39a2069d4064bcd is the object label for the ca private key on the Luna HSM or Luna Cloud HSM service created in step 4.

Use OpenSSL CMS cryptographic objects to sign, verify, encrypt, and decrypt a message

The sender will send a message to the receiver by signing the message with its own private key and encrypting the message with the receiver's public key. The receiver then decrypts the message using its own private key and verifies the message using the sender's public key. To do so, follow the below steps:

1. Create a text file message.txt.

> **NOTE:** Sender and Receiver keys are stored on the Luna HSM or Luna Cloud HSM service. Use the object label of those keys to encrypt, decrypt and sign using those keys

2. Sign the message.txt file using the sender's private key.

```
# openssl cms -engine gem -sign -in message.txt -signer
/usr/local/ssl/certs/sender/sender.cer -inkey rsa-private-
7dfdb3caaf25a63b3d8a81d2a3b51668decafe0f -keyform engine -out sendmail.msg
```

Where rsa-private-7dfdb3caaf25a63b3d8a81d2a3b51668decafe0f is the object label for the sender private key on the Luna HSM or Luna Cloud HSM service.

3. Encrypt the sendmail.msg using the receiver's public key, supplied with the receiver's certificate.

```
# openssl cms -engine gem -encrypt -in sendmail.msg -out sendmail_enc.msg
/usr/local/ssl/certs/receiver/receiver.cer
```

4. Decrypt the sendmail_enc.msg using the receiver's private key.

```
# openssl cms -engine gem -decrypt -in sendmail_enc.msg -inkey rsa-private-
ec0fcb1ce9114662556caab35fc2baf0565752ec -keyform engine -out sendmail_dec.msg
```

Where rsa-private-ec0fcb1ce9114662556caab35fc2baf0565752ec is the object label for the receiver private key on the Luna HSM or Luna Cloud HSM service.

5. Verify the signature of sendmail_dec.msg using the sender's certificate.

```
# openssl cms -engine gem -verify -in sendmail_dec.msg -CAfile
/usr/local/ssl/certs/ca.cer -out out.txt /usr/local/ssl/certs/sender/sender.cer
```

6. Open the out.txt and verify the message which you have typed in the message.txt
7. Close the sautil session if you are using Luna HSM slot.

```
# /usr/local/sautil/bin/sautil -c -s <slot_id> -i 10:11 -q
```

This completes the integration of OpenSSL with Luna Network HSM or a Luna Cloud HSM service using GemEngine on UNIX.

Integrate OpenSSL with Luna HSM on Windows

To integrate OpenSSL with Luna HSM using GemEngine, follow these steps:

> Install and configure OpenSSL toolkit

> Configure GemEngine to use Luna HSM

> Verify the OpenSSL and GemEngine integration on Windows

Install and configure OpenSSL toolkit

To install and configure the OpenSSL toolkit on Windows:

1. Navigate to the OpenSSL toolkit gemengine-1.x\builds\win and extract the file sautil-win64-openssl-x.x.xx.tar.gz and ssl-win64-openssl-x.x.xx.tar.gz in a folder C:\
2. Add C:\cygwin\usr\local\sautil\bin and C:\cygwin\usr\local\ssl\bin to your system path（Control Panel -> System -> Change Settings -> Advanced -> Environment Variables -> System Variables）.

   NOTE: We recommend adding these files to the Path. This step is not mandatory.

3. Create the directory C:\ssl. C:\ssl is the common working directory for the Windows installation.
4. Create an openssl.cnf file under the working directory. Copy and paste the following in the openssl.cnf file and save it as C:\ssl\openssl.cnf:

```
# SSLeay example configuration file.
# This is mostly being used for generation of certificate requests.
#
RANDFILE      = .rnd
#####################################################################
[ ca ]
default_ca    = CA_default           # The default ca section

#####################################################################
[ CA_default ]

   certs      = certs                # Where the issued certs are kept
   crl_dir          = crl            # Where the issued crl are kept
   database   = index.txt      # database index file.
   new_certs_dir    = certs               # default place for new certs.
   certificate= cacert.pem           # The CA certificate
   serial           = serial.txt           # The current serial number
   crl        = crl.pem        # The current CRL
   private_key= private\cakey.pem    # The private key
   RANDFILE   = private\private.rnd  # private random number file
```

```
x509_extensions  = x509v3_extensions    # The extentions to add to the cert
default_days     = 365            # how long to certify for
default_crl_days= 30              # how long before next CRL
default_md  = md5            # which md to use.
preserve    = no             # keep passed DN ordering

# A few different ways of specifying how similar the request should look
# For type CA, the listed attributes must be the same, and the optional
# and supplied fields are just that :-)
policy      = policy_match

# For the CA policy
[ policy_match ]
countryName       = match
stateOrProvinceName     = match
organizationName  = match
organizationalUnitName  = match
commonName        = supplied
emailAddress      = optional

# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types.
[ policy_anything ]
countryName       = optional
stateOrProvinceName     = optional
localityName      = optional
organizationName  = optional
organizationalUnitName  = optional
commonName        = supplied
emailAddress      = optional

####################################################################
[ req ]
default_bits      = 1024
default_keyfile   = privkey.pem
```

```
distinguished_name      = req_distinguished_name
attributes          = req_attributes

[ req_distinguished_name ]
countryName             = Country Name (2 letter code)
countryName_min                 = 2
countryName_max                 = 2
stateOrProvinceName             = State or Province Name (full name)
localityName            = Locality Name (eg, city)
0.organizationName              = Organization Name (eg, company)
organizationalUnitName          = Organizational Unit Name (eg, section)
commonName              = Common Name (eg, your website's domain name)
commonName_max                  = 64
emailAddress            = Email Address
emailAddress_max        = 40

[ req_attributes ]
challengePassword       = A challenge password
challengePassword_min           = 4
challengePassword_max           = 20

[ x509v3_extensions ]
# under ASN.1, the 0 bit would be encoded as 80
# nsCertType            = 0x40
#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
#nsCaPolicyUrl
#nsSslServerName
#nsCertSequence
#nsCertExt
#nsDataType
```

NOTE: You can configure the toolkit using the **openssl.cnf** file.

5. Set the OpenSSL configuration file path. Execute the following in the command prompt:

```
set OPENSSL_CONF=C:\ssl\openssl.cnf
```

Configure GemEngine to use Luna HSM on Windows

To configure GemEngine to use Luna HSM on Windows, complete one of the following tasks, depending on your requirements:

Configure GemEngine for Luna HSM

Configure GemEngine for Luna HA slot or Luna Cloud HSM service

Configure GemEngine for Luna HSM

To configure GemEngine for Luna HSM (single partition):

1. Open the crystoki.ini file and add the following GemEngine section:

```
[GemEngine]
LibPath = <Luna Client installation Directory>\cryptoki.dll
LibPath64 = <Luna Client installation Directory>\cryptoki.dll
EnableDsaGenKeyPair = 1
EnableRsaGenKeyPair = 1
DisablePublicCrypto = 1
EnableRsaSignVerify = 1
EnableLoadPubKey = 1
EnableLoadPrivKey = 1
DisableCheckFinalize = 1
DisableEcdsa = 1
DisableDsa = 0
DisableRand = 0
EngineInit = <slot_id>:10:11
```

2. Run the sautil utility to open the persistent sautil session on Luna HSM slot.

```
C:\OpenSSL\sautil\bin>sautil -v -s <slot_id> -i 10:11 -o -q
```

> **NOTE:** For more login methods for session, refer to the *README-GEM-CONFIG* file present in **<GemEngine_Directory>\docs** folder.

Configure GemEngine for Luna HA slot or Luna Cloud HSM service

1. Create a text file passfile in <path_to_my_passfile> and store the partition password in it.
2. Open the crystoki.ini file and add the following GemEngine section:

```
[GemEngine]
LibPath = <Path to cryptoki.dll>
LibPath64 = <Path to cryptoki.dll>
EnableDsaGenKeyPair = 1
EnableRsaGenKeyPair = 1
```

```
DisablePublicCrypto = 1
EnableRsaSignVerify = 1
EnableLoadPubKey = 1
EnableLoadPrivKey = 1
DisableCheckFinalize = 1
DisableEcdsa = 1
DisableDsa = 0
DisableRand = 0
EngineInit = "myTokenLabel":0:0:passfile=<path_to_my_passfile>\passfile
EnableLoginInit = 1
```

Verify the OpenSSL and GemEngine integration on Windows

Complete the following to verify the OpenSSL and GemEngine integration:

> Generate cryptographic objects for OpenSSL CMS

> Use OpenSSL CMS cryptographic objects to sign, verify, encrypt, and decrypt a message

Generate cryptographic objects for OpenSSL CMS

Generate the CA private key and CA certificate for OpenSSL CMS and then use the CA private key and certificate to generate the receiver and sender certificates. To generate cryptographic objects for OpenSSL CMS:

1. Set the crystoki.ini path to use with OpenSSL:

   ```
   set CONF_PATH=<Luna Client installation Directory>\crystoki.ini
   ```

2. Create the following directories for OpenSSL operations:

   ```
   C:\ssl>mkdir keys
   C:\ssl>mkdir requests
   C:\ssl>mkdir certs
   ```

3. Create the file index.txt – an empty (zero-byte) text file under C:\ssl\index.txt
4. Create the serial number file serial.txt. This is a plain ASCII file containing the string 01 on the first line, followed by a new line under C:\ssl\serial.txt.
5. Copy the libeay32.dll and the ssleay32.dll library to the directory containing sautil.exe.

   NOTE: Skip this step if you are using GemEngine 1.3 or above.

6. Create a 2048-bit private key on Luna HSM using GemEngine. This key pair will later be used to create the CA.

   ```
   C:\ssl>openssl genrsa -engine gem 2048
   ```

   The RSA 2048 bit key for CA is generated on the Luna HSM partition or Luna Cloud HSM service.

7. List the generated key pair using cmu utility.

   ```
   # <LunaClient_Installation_Directory>\Cmu.exe list
   ```

   For example:

   ```
   # C:\Program Files\SafeNet\LunaClient\Cmu.exe list
   ```

   Provide partition password when prompted.

8. Create a CA certificate based on the generated key:

   ```
   C:\ssl>openssl req -engine gem -new -x509 -days 365 -key rsa-private-
   08bde9331fa3be515d2e7db9dd1e28b36b50632e -keyform engine -out certs/ca.cer
   ```

   Where rsa-private-08bde9331fa3be515d2e7db9dd1e28b36b50632e is the object label for the CA private key on the Luna HSM or Luna Cloud HSM service created in step 6.

9. Create a certificate request for sender.

   ```
   C:\ssl>openssl req -engine gem -newkey rsa:2048 -out requests/sender.txt
   ```

   Sender request is used to generate the sender's certificate signed by the Certificate Authority.

10. List the generated key pair using cmu utility.

    ```
    # <LunaClient_Installation_Directory>\Cmu.exe list
    ```

    For example:

    ```
    # C:\Program Files\SafeNet\LunaClient\Cmu.exe list
    ```

    Provide partition password when prompted.

11. Create a certificate request for receiver.

    ```
    C:\ssl>openssl req -engine gem -newkey rsa:2048 -out requests/receiver.txt
    ```

    Receiver request is used to generate the receiver's certificate signed by the Certificate Authority.

12. List the generated key pair using cmu utility.

    ```
    # <LunaClient_Installation_Directory>\Cmu.exe list
    ```

    For example:

    ```
    # C:\Program Files\SafeNet\LunaClient\Cmu.exe list
    ```

    Provide the partition password when prompted.

13. Use the Certificate Authority (CA) to sign the certificate request for Sender.

    ```
    C:\ssl>openssl ca -engine gem -policy policy_anything -cert certs/ca.cer -in
    requests/sender.txt -keyfile rsa-private-
    08bde9331fa3be515d2e7db9dd1e28b36b50632e -keyform engine -out certs/sender.cer
    ```

    Where rsa-private-08bde9331fa3be515d2e7db9dd1e28b36b50632e is the object label for the CA private key on the Luna HSM or HSM on Luna Cloud HSM service created in step 6.

14. Use the Certificate Authority (CA) to sign the certificate request for Receiver.

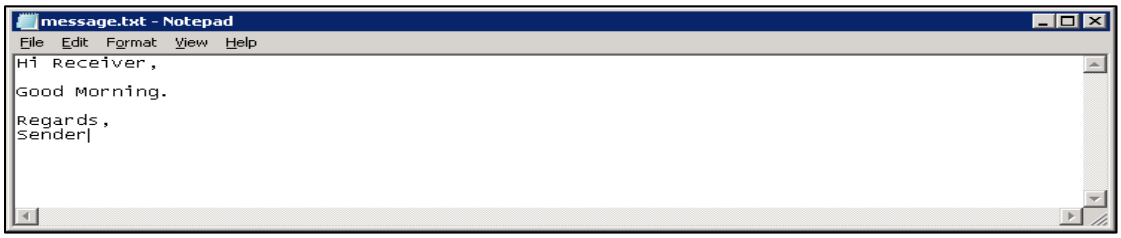
    ```
    C:\ssl>openssl ca -engine gem -policy policy_anything -cert certs/ca.cer -in
    requests/receiver.txt -keyfile rsa-private-
    08bde9331fa3be515d2e7db9dd1e28b36b50632e -keyform engine -out
    certs/receiver.cer
    ```

    Where rsa-private-08bde9331fa3be515d2e7db9dd1e28b36b50632e is the object label for the CA private key on the Luna HSM or HSM on Luna Cloud HSM service created in step 6.

Use OpenSSL CMS cryptographic objects to sign, verify, encrypt, and decrypt a message

The sender will send a message to the receiver by signing the message with its own private key and encrypting the message with the receiver's public key. The receiver then decrypts the message using its own private key and verifies the message using the sender's public key. To do so, follow the below steps:

1. Create the text file message.txt in the C:\ssl directory.



NOTE: Sender and Receiver keys are stored on the Luna HSM or Luna Cloud HSM service. Use the object label of those keys to encrypt, decrypt, and sign using those keys.

2. Sign the message.txt using the sender's private key.

```
C:\ssl>openssl cms -engine gem -sign -in message.txt -signer certs\sender.cer -
inkey rsa-private-605ddfab1e95bfac36eec44f291647e8a3ff5f64 -keyform engine -out
sendmail.msg
```

Where rsa-private-605ddfab1e95bfac36eec44f291647e8a3ff5f64 is the object label for the sender's private key on the Luna HSM or Luna Cloud HSM service.

3. Encrypt the sendmail.msg using the receiver's public key, supplied with the receiver's certificate.

```
C:\ssl>openssl cms -engine gem -encrypt -in sendmail.msg -out sendmail_enc.msg
certs\receiver.cer
```

4. Decrypt the sendmail_enc.msg using the receiver's private key.

```
C:\ssl>openssl cms -engine gem -decrypt -in sendmail_enc.msg -inkey rsa-
private-a216d3b9276268459598f163ff7163aa30cbd3d0 -keyform engine -out
sendmail_dec.msg
```

Where rsa-private-a216d3b9276268459598f163ff7163aa30cbd3d0 is the object label for the receiver's private key on the Luna HSM or Luna Cloud HSM service.

5. Now verify the signature of sendmail_dec.msg using the sender's public key supplied with sender's certificate.

```
C:\ssl>openssl cms -engine gem -verify -in sendmail_dec.msg -CAfile
certs\ca.cer -out out.txt certs\sender.cer
```

6. Open the out.txt and verify the message which you have typed in the message.txt.
7. If you are using Luna HSM partition, close the session with sautil.

```
C:\OpenSSL\sautil\bin>sautil -c -s <slot_id> -i 10:11 -q
```

This completes the integration of OpenSSL with Luna Network HSM or Luna Cloud HSM service using GemEngine on Windows.

Troubleshooting

Problem 1

SAUTIL not found on your system. If OpenSSL was previously installed, or you have patched the Luna Engine in the OpenSSL source code, then the SAUTIL utility will not be available on the system.

Solution

SAUTIL is installed by default with OpenSSL when you install the OpenSSL from Apache Toolkit. If SAUTIL is not available, you can install it by completing the following:

1. Traverse to the toolkit and untar the luna-samples file.
2. Under the luna-samples, you will find sautil. Under sautil, you will find the sautil.c file.
3. Open the sautil.c, search for “#define LUNA_OSSL_ECDSA (1)”, and disable it.
4. Save and close the file and run the following commands in sautil directory under luna-samples.

   ```
   # ./configure.sh

   # make
   ```

5. Verify /usr/local/sautil/bin/sautil is installed on your system.

Problem 2

OpenSSL source installation fails with the following error using gembuild script provided with OpenSSL toolkit:

```
make[2]: *** No rule to make target `../../include/openssl/idea.h', needed by
`e_idea.o'.  Stop.

make[2]: Leaving directory `/home/openssl-1.0.1s/crypto/evp'

make[1]: *** [subdirs] Error 1

make[1]: Leaving directory `/home/openssl-1.0.1s/crypto'

make: *** [build_crypto] Error 1

ERROR: There was an issue compiling OpenSSL. See /home/gemengine-1.1/logs/openssl-
build.log for details.
```

Solution

Add make depend to the gembuild script on line 453 right after the make clean. The error is the result of a recent change in OpenSSL that requires make depend to be run before make install.

Problem 3

OpenSSL sign operation is showing Segmentation fault (core dumped) error, when using OpenSSL v1.1.0 with GemEngine.

```
# openssl req -engine gem -new -x509 -days 365 -key rsa-private-
d6b5261ac7f89d6b3b80d4c0f8aea11f185b95a1 -keyform engine -out
/usr/local/ssl/certs/ca.cer

engine "gem" set.

You are about to be asked to enter information that will be incorporated

into your certificate request.
```

```
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) []:IN
State or Province Name (full name) []:Uttar Pradesh
Locality Name (eg, city) []:Noida
Organization Name (eg, company) []: Thales
Organizational Unit Name (eg, section) []: HSM
Common Name (eg, your websites domain name) []:ca.example.com
Email Address []:
Segmentation fault (core dumped)
```

Solution

This error occurs when the GemEngine calls the function luna_fini_p11 while in the execution path of exit(). Complete the following procedure to overcome the error.

1. Open the <gemengine directory>/engine/e_gem.c file in vi editor.

   ```
   # vi /home/gemengine-1.3/engine/e_gem.c
   ```

2. At line 934 comment the function as follows:

   ```
   //luna_fini_p11();
   ```

3. Save and close the file.
4. Remove the gem.so from OpenSSL engine directory.

   ```
   # rm -rf /usr/local/ssl/lib/engines-1.1/gem.so
   ```

5. Recompile the gem.so.

   ```
   # ./gembuild engine-build
   # ./gembuild engine-install
   ```

8. Rerun the OpenSSL sign operation command that was showing the Segmentation Fault earlier.

   ```
   # openssl req -engine gem -new -x509 -days 365 -key rsa-private-
   d6b5261ac7f89d6b3b80d4c0f8aea11f185b95a1 -keyform engine -out
   /usr/local/ssl/certs/ca.cer
   engine "gem" set.
   You are about to be asked to enter information that will be incorporated
   into your certificate request.
   What you are about to enter is what is called a Distinguished Name or a DN.
   There are quite a few fields but you can leave some blank
   For some fields there will be a default value,
   If you enter '.', the field will be left blank.
   ```

```
-----
Country Name (2 letter code) []:IN
State or Province Name (full name) []:Uttar Pradesh
Locality Name (eg, city) []:Noida
Organization Name (eg, company) []:Thales
Organizational Unit Name (eg, section) []:HSM
Common Name (eg, your websites domain name) []:ca.example.com
Email Address []:
```

Certificate signing gets completed without showing Segmentation Fault.

Contacting Customer Support

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

Customer Support Portal

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

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

Telephone Support

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

Email Support

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