THALES

OpenSSL: Integration Guide

THALES LUNA HSM AND DPOD LUNA CLOUD HSM

Document Information

Document Part Number	007-012068-001
Revision	V
Release Date	1 December 2023

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 HSM is used to securely store the OpenSSL cryptographic keys. When OpenSSL is integrated with Luna HSM, it leverages the Gem Engine to consume HSM resources. The benefits of using Luna HSM to generate the cryptographic keys for OpenSSL are:

> Secure generation, storage, and protection of the cryptographic keys 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*

> Using cloud services with confidence.

* 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 1.6	Windows Server 2022 RHEL 9 RHEL 8
Luna HSM	GemEngine 1.5	Windows Server 2019 RHEL 9 RHEL 8
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 HSM secures the cryptographic keys physically and logically and accelerate cryptographic processing. Luna HSM on premise offerings include the Luna Network HSM, Luna PCIe HSM, and Luna USB HSMs. Luna HSM is 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 1.6	Windows Server 2022 RHEL 8
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 (64-bit) v10.5.1-174. Copyright (c) 2022 Thales Group. All rights
reserved.

Available HSMs:
Slot Id ->                0
Label ->                  TPA01
Serial Number ->          1312109862206
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

Follow these steps to set up your Luna Cloud HSM:

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 using the following command:

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

   > **NOTE:** Do not extract to a new subdirectory. Place the files in the client install directory.

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

   ```
   source ./setenv
   ```

   > **NOTE:** To add the configuration to an already installed UC client, use the -addcloudhsm option when running the setenv script.

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

   > **NOTE:** If your organization requires non-FIPS algorithms for your operations, ensure that the Allow non-FIPS approved algorithms check box is checked. For more information, refer to Supported Mechanisms.

Set up OpenSSL toolkit

Download the OpenSSL toolkit with Gem Engine support from Thales Customer Support portal.

> **NOTE:**
>
> The Doc ID for downloading the GemEngine v1.6 from support portal is KB0026742.
>
> The Doc ID for downloading the GemEngine v1.5 from support portal is KB0024584.
>
> The Doc ID for downloading the GemEngine v1.3 from support portal is KB0017806.
>
> The Doc ID for downloading the GemEngine v1.2 from support portal is KB0016309.

It is recommended that you should familiarize yourself with OpenSSL. Refer to OpenSSL Documentation for more information about OpenSSL.

Integrating Luna HSM with OpenSSL using Gem Engine

To integrate Luna HSM with OpenSSL using Gem Engine, 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 Gem Engine to use Luna HSM on Unix

> Verify the OpenSSL and Gem Engine Integration on Unix

Install and configure OpenSSL toolkit

To install and configure OpenSSL toolkit on UNIX, follow 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 Gem Engine

> Scenario D: Configure OpenSSL to enable Gem Engine by default

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

1. Extract the Gem Engine toolkit and navigate to the Gem Engine directory.
2. Locate the `libgem.so` or `gem.so` and `sautil` binaries available at:

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

   Example:

   ```
   builds/linux/rhel/64/3.0/
   ```

   NOTE: Locate the dynamic engine library `libgem.so` in the exact `<OpenSSL_version>` directory. For OpenSSL version 1.1.x onwards, the dynamic engine library is `gem.so`.

3. Use `gembuild` to locate the OpenSSL Engines directory.

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

4. Copy the `libgem.so` or `gem.so` to the OpenSSL (depending on the version installed) engines directory to install the Gem Engine.

   Example:

   ```
   # cp <gem-engine directory>/builds/linux/rhel/64/3.0/gem.so /usr/lib64/engines-
   3
   ```

5. Copy the `sautil` utility to `/usr/local/bin` according to your OpenSSL version.

   Example:

   ```
   # cp <gem-engine directory>/builds/linux/rhel/64/3.0/sautil /usr/local/bin
   ```

6. Run the `sautil` utility to see if you get all the options listed.

   ```
   # /usr/local/bin/sautil
   ```

7. Add openssl and sautil location to `PATH` environment variable, if they are not in system defaults.

   Example:

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

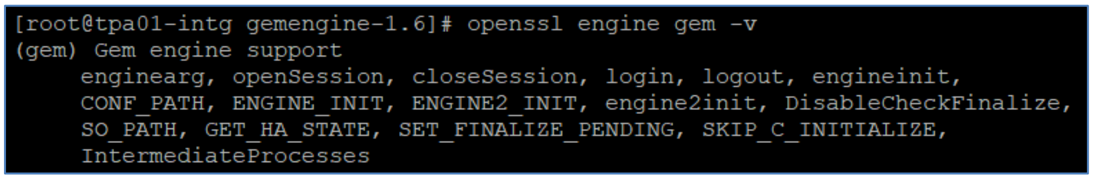
8. Add openssl library location to `LD_LIBRARY_PATH` environment variable, if not present at system defaults.

   Example:

   ```
   # export LD_LIBRARY_PATH=/usr/local/lib64:$LD_LIBRARY_PATH
   ```

9. Verify that the Gem Engine support is available.

   ```
   # openssl engine gem -v
   ```



```
[root@tpa01-intg gemengine-1.6]# 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 like the example above, it indicates that the Gem Engine is successfully installed. Now follow the steps to configure Gem Engine to use Luna HSM.

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 tar ball from https://www.openssl.org/source/. It is required to download the version which is closest to your existing OpenSSL installation. For example if you have OpenSSL v1.1.1 is installed you need to download any OpenSSL v1.1.x where x can be any number.

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

2. Navigate to the Gem Engine directory to locate the engine location for the existing OpenSSL.

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

   Note the OpenSSL Engine directory that will be used as an input for next command.

3. Run `gembuild config` and provide the inputs required to compile the engine.

   Example:

   ```
   # ./gembuild config --openssl-source=<Path to extracted OpenSSL source
   directory> --openssl-engines=<Path to Engine directory from step 2> --config-
   bits=64
   ```

   NOTE: If the OpenSSL development package is not available, you need to install it on the system. In this example, it is 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 must be used. The `--openssl-libs` option must be used to specify the location of the lib directory if `libcrypto.so` is available at a location other than the default libs location. All paths need to be absolute.

   The `gembuild` script in GemEngine 1.6 has a new option `"--openssl-api=<major>"` where <major> is the major version of the openssl build.

   Examples:

   `--openssl-api=3.0`

   or

   `--openssl-api=1.1.1`

4. Install the required EC header files.

   ```
   # ./gembuild openssl-ec-headers
   ```

5. Compile the engine.

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

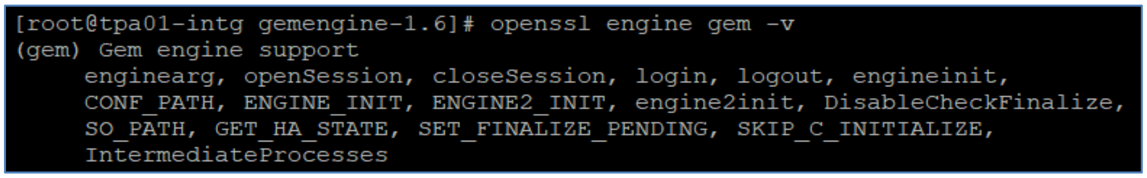
   NOTE: If you encounter a known issue in this step, refer the Troubleshooting section.

6. Install the Gem Engine.

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

7. Verify that the Gem Engine support is available.

```
# openssl engine gem -v
```



```
[root@tpa01-intg gemengine-1.6]# 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 resembles the example above, it indicates that the Gem Engine is successfully installed.

8. 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 in Step 3, or use the `--sautil-prefix` option with the "`./gembuild sautil-install`" command.

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

Now follow the steps to configure Gem Engine to use Luna HSM.

Scenario C: Compile and install OpenSSL from source and compiling and installing Gem Engine

1. Download and extract the OpenSSL source tarball. Download openssl-x.x.xx.tar.gz from https://www.openssl.org/source/

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

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

> **NOTE:** This step is applicable till OpenSSL v1.0.2. Skip this step for latest OpenSSL.

Example:

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

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

3. Extract the Gem Engine toolkit and navigate to the Gem Engine directory. Run the `gembuild config` command using the `--prefix` option.

> **NOTE:** If using the OpenSSL v1.0.2 with FIPS module, add the `--openssl-fips-source=<Path to openssl-fips-2.0.x>` to the "`gembuild config`" command.

Example:

```
# ./gembuild config --openssl-source=<Path to extracted OpenSSL source
directory> --prefix=/usr/local --config-bits=64
```

> **NOTE:** If using GemEngine v1.3 or above to build and install OpenSSL v1.0.2, must include the option `--compat-102` in the above command.
>
> OpenSSL v1.0.2 support is removed from GemEngine v1.6.

The `gembuild` script in GemEngine 1.6 has a new option `"--openssl-api=<major>"` where <major> is the major version of the openssl build.

Examples:

`--openssl-api=3.0`

or

`--openssl-api=1.1.1`

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

   NOTE: This step is applicable till OpenSSL v1.0.2. Skip this step for latest OpenSSL.

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

5. Compile and install the OpenSSL using toolkit.

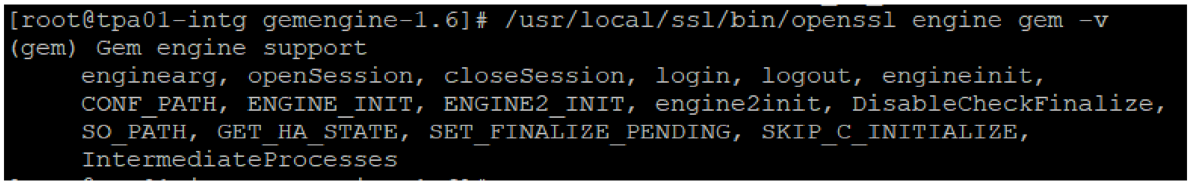
```
# ./gembuild openssl-build
# ./gembuild openssl-install
```

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

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

   NOTE: If you encounter any known issue here, refer the Troubleshooting section.

```
# ./gembuild engine-install
# /usr/local/ssl/bin/openssl engine gem -v
```



```
[root@tpa01-intg gemengine-1.6]# /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,
     IntermediateProcesses
```

If the output looks like above, then the Gem Engine is successfully installed.

7. Compile and install sautil.

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

   NOTE: sautil will install 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 location to the `PATH` environment variable.

   Example:

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

9. Add openssl library location to `LD_LIBRARY_PATH` environment variable.

   Example:

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

Now follow the steps to Configure Gem Engine to use Luna HSM.

Scenario D: Configure OpenSSL to enable Gem Engine by default

1. Locate the OpenSSL configuration file `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` file is available.

   > **NOTE:** Above command will show OpenSSL which is set via the `PATH` environment variable. Run `which openssl` to verify you are accessing the correct configuration file.

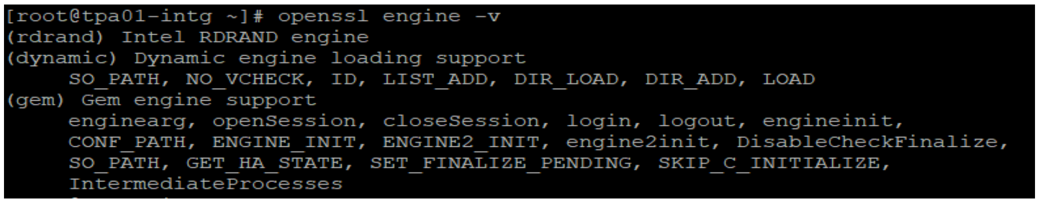
2. Edit the openssl.cnf file, as provided below.

   ```
   # 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 that the engine is loaded without specifying it.

   ```
   # openssl engine -v
   ```

   ```
   [root@tpa01-intg ~]# openssl engine -v
   (rdrand) Intel RDRAND engine
   (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,
        IntermediateProcesses
   ```

   If the output resembles the example above, then the Gem Engine is configured as default engine.

4. Test the application by generating a certificate request without using the `engine` parameter.

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

Configure Gem Engine to use Luna HSM on UNIX

To configure Gem Engine to use Luna HSM on UNIX, complete one of the following tasks, depending on the Luna HSM that you are using:

Configure Gem Engine for Luna HSM

Configure Gem Engine for Luna Cloud HSM service

Configure Gem Engine for Luna HSM

To configure Gem Engine for Luna HSM:

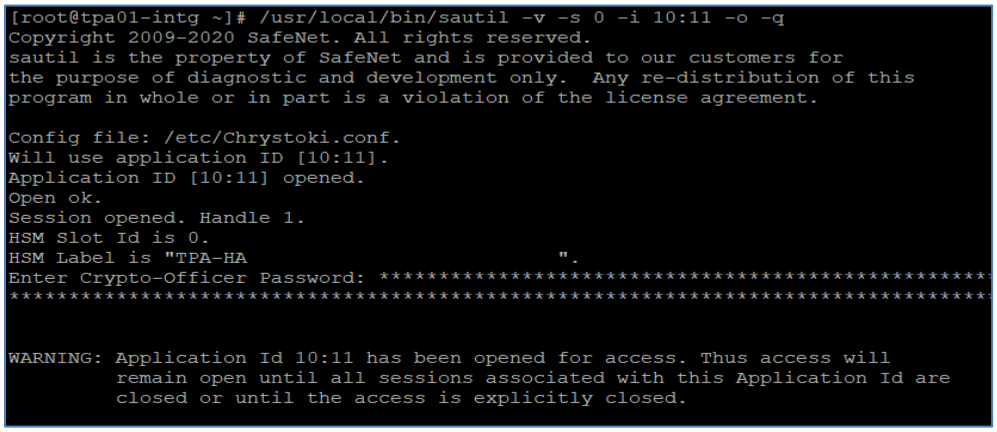
1. Open the `/etc/Chrystoki.conf` file and add the following `GemEngine` section:

```
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;
}
```

NOTE: `<slot_id>` must be replaced with the actual value of physical or virtual slot.

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

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



```
[root@tpa01-intg ~]# /usr/local/bin/sautil -v -s 0 -i 10:11 -o -q
Copyright 2009-2020 SafeNet. All rights reserved.
sautil is the property of SafeNet and is provided to our customers for
the purpose of diagnostic and development only.  Any re-distribution of this
program in whole or in part is a violation of the license agreement.

Config file: /etc/Chrystoki.conf.
Will use application ID [10:11].
Application ID [10:11] opened.
Open ok.
Session opened. Handle 1.
HSM Slot Id is 0.
HSM Label is "TPA-HA                          ".
Enter Crypto-Officer Password: ***********************************************************
*****************************************************************************************


WARNING: Application Id 10:11 has been opened for access. Thus access will
         remain open until all sessions associated with this Application Id are
         closed or until the access is explicitly closed.
```

> **NOTE:** If the persistent session is not the requirement, you can refer to the *README-GEM-CONFIG* file present in the `<GemEngine_Directory>/docs` folder for other methods to login.

Configure Gem Engine for Luna Cloud HSM service

1. Create a text file to store the partition crypto officer password in that file.

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

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

   ```
   GemEngine = {
   LibPath = <path to LibCryptoki2.so>;
   LibPath64 = <path to LibCryptoki2_64.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;
   ```

   > **NOTE:** `<Partition Label>` must be replaced with the actual label of your partition and `passfile` must point to the actual path of the file containing CO password.

3. In the `Misc` section of `Chrystoki.conf` file, add the following flag and save the changes. Do not delete the default values already present in the `Misc` section.

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

Verify the OpenSSL and Gem Engine Integration on UNIX

Use the following steps to verify the OpenSSL and Gem Engine integration on UNIX:

> Generate cryptographic objects for OpenSSL

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

Generate cryptographic objects for OpenSSL

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

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 a directory for saving the generated certificates if the directory doesn't exist already.

   ```
   # mkdir -p /usr/local/ssl/certs
   ```

3. Create the text files `/usr/local/ssl/index.txt` and `/usr/local/ssl/serial`.
4. Open the `/usr/local/ssl/serial` file and write `01` at the top and press enter. Save the file.
5. Create a 2048-bit private key for CA using Gem Engine. The key will be generated on the Luna HSM's partition.

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

6. List the generated key pair using `cmu` utility.

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

   Example:

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

   Provide partition password when prompted and note down the private key label of CA keys.

7. 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 5.

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

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

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

9. 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 CA.

10. List the generated key pair using `cmu` utility.

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

    Example:

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

Provide partition password when prompted and note down the private key label of sender keys.

11. 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 CA.

12. List the generated key pair using the `cmu` utility.

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

Example:

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

Provide partition password when prompted and note down the private key label of receiver keys.

13. Sign the certificate request of sender by CA .

```
# 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 5.

14. Sign the certificate request for receiver by CA.

```
# 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 that was created in step 5.

Use OpenSSL to sign, verify, encrypt, and decrypt a message

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

> **NOTE:** Luna Cloud HSM and Luna HSM f/w 7.7.2 onwards do not support RSA_PKCS for encryption/decryption in FIPS mode whereas OpenSSL CMS uses RSA_PKCS only. If your HSM is operating in FIPS as mentioned please follow APPENDIX-A to performing the same crypto operations.

1. Create a text file named `message.txt`.

```
Hi Receiver,

Good Morning.

Regards,
Sender
```

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

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 that is 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 public key with 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` file and compare the contents from the `message.txt`. Both the files must have the same message that proves all crypto operations are performed successfully.

7. Close the `sautil` session if you are using Luna HSM slot.

   ```
   # /usr/local/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 Gem Engine on UNIX.

Integrate OpenSSL with Luna HSM on Windows

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

> Install and configure OpenSSL toolkit

> Configure Gem Engine to use Luna HSM

> Verify the OpenSSL and Gem Engine integration on Windows

Install and configure OpenSSL toolkit

To install and configure the OpenSSL toolkit on Windows:

1. Navigate to the OpenSSL toolkit `<engine-directory>\builds\win` and extract the file `sautil-win64-openssl-x.x.xx.tar.gz` and `ssl-win64-openssl-x.x.xx.tar.gz` in `C:\` directory.

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` as the common working directory for the Windows installation.
4. Create an `openssl.cnf` file under the working directory. Copy and paste the following text 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 command in the command prompt:

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

Configure Gem Engine to use Luna HSM on Windows

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

Configure Gem Engine for Luna HSM

Configure Gem Engine for Luna HA slot or Luna Cloud HSM service

Configure Gem Engine for Luna HSM

To configure Gem Engine for Luna HSM:

1. Open the `crystoki.ini` file and add the following text to the `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 Gem Engine for 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 text to the `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
   ```

> **NOTE:** `<Partition Label>` must be replaced with the actual label of your partition and `passfile` must point to the actual path of the file containing CO password.

Verify the OpenSSL and Gem Engine integration on Windows

Complete the following steps to verify the OpenSSL and Gem Engine integration:

> Generate cryptographic objects for OpenSSL

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

Generate cryptographic objects for OpenSSL

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

1. Set the `OPESSL_CONF` path to use with OpenSSL:

   ```
   set OPESSL_CONF =<Path to openssl.conf>
   ```

2. Create the following directories for the OpenSSL operations:

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

3. Create the index.txt file, which will be 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 the `sautil.exe` file.

   > **NOTE:** Skip this step if you are using Gem Engine 1.3 or above.

6. Create a 2048-bit private key on Luna HSM using Gem Engine. This key pair will be used later 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 by using the `cmu` utility.

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

   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 the 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 CA.

10. List the generated key pair using `cmu` utility.

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

    Example:

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

    Provide partition password when prompted.

11. Create a certificate request for the 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 CA.

12. List the generated key pair by using the `cmu` utility.

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

    Example:

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

    Provide the partition password when prompted.

13. Use the 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 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
    ```

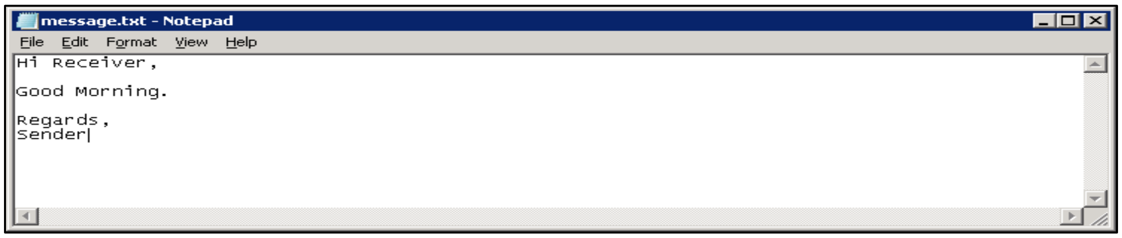
    Where `rsa-private-08bde9331fa3be515d2e7db9dd1e28b36b50632e` is the object label for the CA private key on the Luna HSM or HSM on Luna Cloud HSM service created in step 6.

Use OpenSSL to sign, verify, encrypt, and decrypt a message

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

> **NOTE:** Luna Cloud HSM and Luna HSM f/w 7.7.2 onwards do not support RSA_PKCS for encryption/decryption in FIPS mode whereas OpenSSL CMS uses RSA_PKCS only. If your HSM is operating in FIPS as mentioned please follow APPENDIX-A to performing the same crypto operations.

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



```
Hi Receiver,

Good Morning.

Regards,
Sender
```

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. Verify the signature of `sendmail_dec.msg` using the sender's public key supplied with the 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` file and compare the contents with the `message.txt` file. Both the files must have the same message that proves all crypto operations are performed successfully.

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 Gem Engine on Windows.

Appendix-A

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

Use OpenSSL to sign, verify, encrypt, and decrypt a message

NOTE: Luna Cloud HSM and Luna HSM f/w 7.7.2 onwards do not support RSA_PKCS for encryption/decryption in FIPS mode whereas OpenSSL CMS uses RSA_PKCS only. Performing the crypto operations required the below steps when your HSM in FIPS mode as mentioned here.

1. Create a text file named `message.txt`.

```
Hi Receiver,

Good Morning.

Regards,
Sender
```

NOTE: Sender and Receiver keys are stored on the Luna HSM or Luna Cloud HSM service. Use the object label of keys on Luna HSM partition to decrypt and sign the messages.

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-
613247c58f0ca0d7f2bb5a14efb2c7d53ed45322 -keyform engine -out sendmail.msg
```

   Where `rsa-private-613247c58f0ca0d7f2bb5a14efb2c7d53ed45322` is the object label for the sender’s private key on the Luna HSM.

3. Encrypt the signed file `sendmail.msg` using the receiver’s public key that is supplied with the receiver’s certificate. To encrypt the file, first generate an AES key to encrypt the signed file and then encrypt the AES key using receiver’s public key.

   - To generate AES key:

```
# openssl rand -engine gem -hex 32 > encryption_aes.key
```

   - Encrypt the signed message by generated AES key:

```
# openssl enc -aes-256-cbc -a -pbkdf2 -k encryption_aes.key -in sendmail.msg -
out sendmail.enc
```

   - Encrypt the AES key by Receiver’s Public key:

```
# openssl rsautl -encrypt -oaep -engine gem -in encryption_aes.key -out
encryption_aes.enc -certin -inkey /usr/local/ssl/certs/receiver/receiver.cer
```

4. Send the encrypted message and encryption key to receiver.
5. Receiver will decrypt the received AES key `encryption_aes.enc` using own Private key and then decrypt the encrypted message `sendmail.enc` using the AES key.

- To decrypt AES key using Receiver's Private Key:

```
# openssl rsautl -decrypt -oaep -engine gem -in encryption_aes.enc -out
encryption_aes.key -inkey rsa-private-489393ff23d93012ab85d859cf3386ef42fcc186
-keyform engine
```

Where `rsa-private-489393ff23d93012ab85d859cf3386ef42fcc186` is the object label for the receiver's private key on the Luna HSM.

- Decrypt the signed message by decrypted AES key:

```
# openssl enc -aes-256-cbc -d -a -pbkdf2 -k encryption_aes.key -in sendmail.enc
-out sendmail.dec
```

6. Verify the signature of `sendmail.dec` using the public key with the sender's certificate.

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

7. Open the `sendmail.txt` file and compare the contents from the `message.txt`. Both the files must have the same message that proves all crypto operations are performed successfully.

Troubleshooting

Problem 1

SAUTIL is not found on your system. If the OpenSSL was previously installed, or if 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 steps:

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

```
# ./configure.sh
```

```
# make
```

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

Problem 2

OpenSSL source installation fails with the following error using the `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 displaying the Segmentation Fault (core dumped) error, when using OpenSSL v1.1.0 with Gem Engine.

```
# 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 Gem Engine calls the function `luna_fini_p11` while in the execution path of `exit()`. Complete the following procedure to overcome the error.

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

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

- At line 934, comment the function as follows:

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

- Save and close the file.
- Remove the `gem.so` from the OpenSSL engine directory.

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

- Recompile the `gem.so`.

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

- Rerun the OpenSSL sign operation command that was displaying the Segmentation Fault.

  ```
  # 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 []:
```

The certificate signing will get completed without displaying the 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.