OpenSSL

v1.1.1

Integration Guide

u.trust GP HSM Se-Series

CryptoServer CSe-Series/Se-Series LAN with firmware SecurityServer 4.45.5.0 or higher

utimaco®

Imprint

Copyright 2025	Utimaco IS GmbH Krefelder Straße 220 52070 Aachen Germany
Phone	AMERICAS +1-844-UTIMACO (+1 844-884-6226) EMEA +49 800-627-3081 APAC +81 800-919-1301
Internet	https://support.hsm.utimaco.com/
e-mail	support@utimaco.com
Document Version	1.0.0
Date	2025-10-17
Status	PUBLISHED
Document No.	IG-2025-0044

Table of Contents

1 Introduction

This guide is part of the information and support provided by Utimaco. Additional documents produced to support your Utimaco u.trust GP HSM Se-Series product can be found in the document directory of the Utimaco SecurityServer product bundle. All Utimaco u.trust GP HSM Se-Series product documentation are available on Utimaco's website at https://utimaco.com/

1.1 About This Guide

This guide provides an integration explaining how to integrate an Utimaco CryptoServer Hardware Security Module (HSM) with OpenSSL.

1.1.1 Target Audience for This Guide

This guide is intended for administrators of OpenSSL and of Utimaco HSMs.

1.1.2 Contents of This Guide

After the introduction, this guide is divided up as follows:

Chapter 2 Overview

Chapter 3 Integration Requirements and Prerequisites

Chapter 4 Integrating OpenSSL on Linux

Chapter 5 Integrating OpenSSL on Windows

Chapter 6 Troubleshooting

Chapter 7 Further Information

1.1.3 Document Conventions

The following conventions are used in this guide:

Convention	*Use*	*Example*

Bold	Items of the Graphical User Interface (GUI), e.g., menu options	Press the OK button.
Monospaced	File names, folder and directory names, commands, file outputs, programming code samples	You will find the file example.conf in the /exmp/demo/ directory.
Italic	References and important terms	See Chapter 3, "Sample Chapter", in the *CryptoServer - csadm Manual* or [CSADMIN].

Table 1: Document conventions

Special icons are used to highlight the most important notes and information.

Here you find important safety information that should be followed.

Here you find additional notes or supplementary information.

missing thing

1.1.4 Abbreviations

The following abbreviations are used in this guide:

Abbreviation	*Meaning*
CA	Certificate Authority
CAT	CryptoServer Administration Tool

CD	Compact Disc
CSR	Certificate Signing Request
GUI	Graphical User Interface
HSM	Hardware Security Module
IP	Internet Protocol
LAN	Local Area Network
PCIe	PCI Express Interface
PKCS#11	Public-Key Cryptography Standard #11
RSA	Rivest-Shamir-Adleman
SO	Security Officer
SSL	Secure Socket Layer
TLS	Transport Layer Security
URL	Uniform Resource Locator

Table 2: List of Abbreviations

2 Overview

2.1 OpenSSL

OpenSSL is an open-source tool for using the Secure Socket Layer (SSL) and Transport Layer Security (TLS) protocols for Web authentication. It offers cryptographic functions to support SSL/TLS protocols.

It allows users to perform various SSL related tasks, including CSR (Certificate Signing Request) and private keys generation and SSL certificate installation. Most of the Linux distributions come with pre-compiled OpenSSL, but if you are on a Windows system, you can get it through manual installation.

OpenSSL has an abstraction layer called "engine" which can delegate cryptographic operations to different pieces of software or hardware.

Libp11 provides an engine_pkcs11 that tries to fit the PKCS#11 API within the engine API of OpenSSL. That is, it provides a gateway between PKCS#11 modules and the OpenSSL engine API.

2.2 Utimaco CryptoServer HSM

CryptoServer is a hardware security module developed by Utimaco IS GmbH. CryptoServer is a physically protected, specialized computer unit designed to perform sensitive cryptographic tasks and to securely manage as well as store cryptographic keys and data. It can be used as a universal, independent security component for heterogeneous computer systems.

3 Integration Requirements and Prerequisites

Ensure the system environment you will be using meets the following hardware and software requirements.

This guide assumes that the user has already installed and configured required Software.

3.1 Tested Versions

The integrations that have been successfully tested with the Utimaco HSM with OpenSSL.

Operating System	*OpenSSL*	*Utimaco* u.trust GP HSM Se-Series *Version*	*Utimaco HSM*
RHEL7/ CentOS7 RHEL8/ CentOS8 Windows 2019 Ubuntu 20	OpenSSL 1.1.1q	SecurityServer 4.45.5 p11tool2 from product package Utimaco SecurityServer	CryptoServer CSe- Series/Se-Series

Table 3: List of Tested versions

3.2 Software Requirements

Software	*Software Requirements*
OpenSSL	OpenSSL 1.1.1q
Libp11 (Linux)	0.4.12
Libp11 (Windows)	0.4.12
HSM Interface	SecurityServer PKCS#11 Provider

Table 4: List of Software Requirements

3.3 Hardware Requirements

Hardware	*Hardware Requirements*
Utimaco LAN HSM	CryptoServer CSe-Series/Se-Series LAN with firmware SecurityServer 4.45.5.0 or higher
Utimaco PCI-e HSM	CryptoServer CSe-Series/Se-Series PCI-e with firmware SecurityServer 4.45.5.0 or higher

Table 5: List of Hardware Requirements

Setup an account on the Utimaco support portal and request download access at the following URL:

https://support.hsm.utimaco.com/

3.4 Prerequisites

Before you begin, please ensure that you have installed/setup:

- Operating system listed in Tested Versions
- SecurityServer listed in Tested Version
- CryptoServer Default Admin should be replaced with a new admin user
- Public and private key pair must be created and stored onto each HSM. Refer to the CryptoServer documentations to setup the keys
- CryptoServer is setup and configured. Refer to the CryptoServer documentations to setup the HSM
- PKCS#11 library is setup and configured as per the environment. Refer to the CryptoServer documentations to setup and configure the PKCS#11 library for CryptoServer
- Familiarize yourself with the OpenSSL documents and setup process

4 Integrating OpenSSL on Linux

4.1 Installing OpenSSL

1. (Optional) It is recommended to update the system with the latest security patch

›_ Console

```
For RHEL
# yum update For Ubuntu:
# apt-get update
```

If you are using existing or pre-installed openssl then skip steps 2,3, 4, and 5.

2. Install dependent packages for openssl

›_ Console

```
For RHEL
# yum install make gcc perl pcre-devel zlib-devel For Ubuntu
# apt install build-essential checkinstall zlib1g-dev libtemplate-perl
```

3. Download the latest version of openssl on Linux machine from https://www.openssl.org

›_ Console

```
# wget https://www.openssl.org/source/openssl-1.1.1q.tar.gz
```

4. Extract the downloaded file

›_ Console

```
# tar xvf openssl-1.1.1q.tar.gz
```

5. Go to openssl directory and run the following commands to build and install openssl

›_ Console

```
# cd openssl-1.1.1q
# ./config --prefix=/usr/local/openssl # make
# make test
# make install
# export LD_LIBRARY_PATH=/usr/local/openssl/lib:$LD_LIBRARY_PATH # export PATH=/
usr/local/openssl/bin:$PATH
# openssl version -a
```

```
[root@openssl-ubuntu ~]# openssl version -a
OpenSSL 1.1.1q  5 Jul 2022
built on: Sun Jul 24 17:50:43 2022 UTC
platform: linux-x86_64
options:  bn(64,64) rc4(16x,int) des(int) idea(int) blowfish(ptr)
compiler: gcc -fPIC -pthread -m64 -Wa,--noexecstack -Wall -O3 -DOPENSSL_USE_NODELETE -DL_ENDIAN -DOPENSSL_PIC -DOPENSSL_CPUID_OBJ -DOPENSSL_IA32_SSE2 -D
OPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA256_ASM -DSHA512_ASM -DKECCAK1600_ASM -DRC4_ASM -DMD5_ASM -DAESNI_ASM -
DVPAES_ASM -DGHASH_ASM -DECP_NISTZ256_ASM -DX25519_ASM -DPOLY1305_ASM -DNDEBUG
OPENSSLDIR: "/usr/local/openssl/ssl"
ENGINESDIR: "/usr/local/openssl/lib/engines-1.1"
Seeding source: os-specific
[root@openssl-ubuntu ~]#
```

Figure 1: OpenSSL version output

After rebooting the server, export the library path every time.

 # export LD_LIBRARY_PATH=/usr/local/openssl/lib:$LD_LIBRARY_PATH # export PATH=/usr/local/openssl/bin:$PATH

4.2 Installing Libp11

1. Download the latest libp11 package from Releases · OpenSC/libp11 · GitHub.

›_ Console

```
# wget https://github.com/OpenSC/libp11/releases/download/libp11- 0.4.12/
libp11-0.4.12.tar.gz
```

2. Extract the file

›_ Console

```
# tar -xvf libp11-0.4.12.tar.gz
```

3. Go to libp11 directory, build and install libp11 using the following commands

›_ Console

```
# cd libp11-0.4.12
# ./configure OPENSSL_CFLAGS="-I/usr/local/openssl/include/openssl"
 OPENSSL_LIBS="-L/usr/local/openssl/lib -lcrypto" prefix="/usr/local/libp11/"
# make
# make install
# export LD_LIBRARY_PATH=/usr/local/openssl/lib:
/usr/local/libp11/lib/:$LD_LIBRARY_PATH
```

```
root@Openssl-ubuntu:~/libp11-0.4.12# ./configure OPENSSL_CFLAGS="-I/usr/local/openssl/include/openssl" OPENSSL_LIBS="-L/usr/lo
cal/openssl/lib -lcrypto" prefix="/usr/local/libp11/"
checking build system type... x86_64-pc-linux-gnu
checking host system type... x86_64-pc-linux-gnu
checking target system type... x86_64-pc-linux-gnu
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /usr/bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking whether make supports nested variables... yes
checking whether make supports nested variables... (cached) yes
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking whether gcc understands -c and -o together... yes
checking whether make supports the include directive... yes (GNU style)
checking dependency style of gcc... gcc3
checking for pkg-config... /usr/bin/pkg-config
checking pkg-config is at least version 0.9.0... yes
checking how to run the C preprocessor... gcc -E
checking for grep that handles long lines and -e... /usr/bin/grep
checking for egrep... /usr/bin/grep -E
checking for ANSI C header files... yes
checking for sys/types.h... yes
```

Figure 2: Output of configure command for libp11

```
config.status: creating src/libp11.pc
config.status: creating src/libp11.rc
config.status: creating src/pkcs11.rc
config.status: creating doc/Makefile
config.status: creating doc/doxygen.conf
config.status: creating examples/Makefile
config.status: creating tests/Makefile
config.status: creating src/config.h
config.status: executing depfiles commands
config.status: executing libtool commands
configure: creating src/libp11.map

libp11 has been configured with the following options:

Version:                0.4.12
libp11 directory:       /usr/local/libp11/lib
Engine directory:       $(libdir)
Default PKCS11 module:
API doc support:        no

Host:                   x86_64-pc-linux-gnu
Compiler:               gcc
Preprocessor flags:
Compiler flags:         -g -O2 -pthread
Linker flags:
Libraries:              -lpthread -ldl

OPENSSL_CFLAGS:         -I/usr/local/openssl/include/openssl
OPENSSL_LIBS:           -L/usr/local/openssl/lib -lcrypto
```

Figure 3: Output of configure command continued for libp11

If you are using existing or pre-installed openssl then change the value of

OPENSSL_CFLAGS and OPENSSL_LIBS to their correct path.

Make a note of "Engine Directory" path while running the configure command as the pkcs11.so file is generated inside this directory after running "make install" command.

After rebooting the server, export the library path every time.

 # export LD_LIBRARY_PATH=/usr/local/openssl/lib:

/usr/local/libp11/lib/:$LD_LIBRARY_PATH and # export PATH=/usr/local/openssl/bin:$PATH

4.3 PKCS#11 Configuration for CryptoServer

1. Create the directory /etc/utimaco. Locate the Utimaco PKCS#11 configuration file in your SecurityServer directory, Linux/x86-64/Crypto_APIS/PKCS11_R3/sample. Copy the Utimaco PKCS#11 configuration file cs_pkcs11_R3.cfg into /etc/utimaco directory

›_ Console

```
# mkdir /etc/utimaco
# cd <install directory>/Software/Linux/x86- 64/Crypto_APIs/PKCS11_R3/sample
# cp cs_pkcs11_R3.cfg /etc/utimaco # cd /etc/utimaco
```

2. Edit the cs_pkcs11_R3.cfg file and make the appropriate changes to the file

cs_pkcs11_R3.cfg

```
[Global]
# For unix:
Logpath = /tmp
# Loglevel (0 = NONE; 1 = ERROR; 2 = WARNING; 3 = INFO; 4 = TRACE)
Logging = 1

# Set the Device to connect with [CryptoServer]
# Device specifier
Device = <HSM_IP>
```

For more information regarding the commands and command parameters please check the Utimaco CryptoServer documentation. The device may be a CryptoServer (PCIe or LAN) device. The device line will follow one of these patterns, based on the HSM form-factor:
Device = 288@<HSM IP address> Hardware (LAN) HSM

OR

Device = /dev/cs2.0 Hardware (PCIe) HSM

To make your testing easier, it would be good to enable the PKCS#11 log file. That can be enabled by editing the Logging Loglevel. Set the LogPath and Logging Loglevel to 1. For testing you may want to increase it to 4.

The added LogPath points to a writable directory, not to a file.

If you encounter problems, check the log file named cs_pkcs11_R3.log in the LogPath defined directory. When you are done testing, you should change Logging to 1 or 2. This will limit the logging to only critical and important messages.

4.3.1 Create SO User and Initialize a Slot

You should initialize a slot with a custom label using p11tool2.

First using p11tool2 create, the SO or Security Officer and then using p11tool2 command initialize the Slot that you want to use, and the slot user as shown below.

›_ Console

```
# ./p11tool2 slot=<slot_no> Label=<token_label> Login=ADMIN,ADMIN.key
InitToken=<SO_PIN>
# ./p11tool2 slot=0 LoginSO=<SO_PIN> InitPin=<CryptoUser_PIN>
```

4.4 Configuring OpenSSL to Use Utimaco HSM

4.4.1 Setting up Utimaco CryptoServer library in OpenSSL Configuration File

1. Open the file /usr/local/openssl/ssl/openssl.cnf and enter the following line in the first line of the file

›_ Console

```
openssl_conf = openssl_init
```

2. Enter the following lines under last section of openssl.cnf file

›_ Console

```
[openssl_init] engines=engine_section
[engine_section]
pkcs11 = pkcs11_section
[pkcs11_section] engine_id = pkcs11
dynamic_path = /usr/local/libp11/lib/pkcs11.so
MODULE_PATH = /opt/utimaco/lib/libcs_pkcs11_R3.so
init = 0
```

Dynamic path and Module path will get changed according to the user environment.

4.4.2 Verify PKCS#11 Engine

Run the command below to verify the OpenSSL Engine is available or not.

›_ Console

```
# openssl engine pkcs11 -t
```

```
root@Openssl-ubuntu:~/libp11-0.4.12# openssl engine pkcs11 -t
(pkcs11) pkcs11 engine
     [ available ]
root@Openssl-ubuntu:~/libp11-0.4.12#
```

Figure 4: Verification of pkcs11 engine

4.4.3 Test OpenSSL Functionalities with Utimaco HSM

4.4.3.1 Testing with RSA Key

1. Generate the RSA key using p11tool2

›_ Console

```
# p11tool2 slot=2 LoginUser=123456 PubKeyAttr=CKA_LABEL="TestRSAKey"
 PrvKeyAttr=CKA_LABEL="TestRSAKey" GenerateKeyPair=RSA
```

2. Verify that the keys are generated onto the HSM using the following command

›_ Console

```
# p11tool2 LoginUser=<cryptouser_password> ListObjects
```

Example

›_ Console

```
# p11tool2 slot=2 LoginUser=123456 ListObjects

CKO_PUBLIC_KEY:

+ 1.1
CKA_KEY_TYPE    = CKK_RSA
CKA_LABEL   = TestRSAKey
CKA_ID  =

CKO_PRIVATE_KEY:

+ 2.1
CKA_KEY_TYPE = CKK_RSA
CKA_SENSITIVE = CK_TRUE
CKA_EXTRACTABLE = CK_FALSE
CKA_LABEL = TestRSAKey
CKA_ID
```

3. Generate a self-signed certificate

›_ Console

```
# openssl req -engine pkcs11 -new -key "pkcs11:token=OpensslSlot;object=
TestRSAKey" -keyform engine -out TestRSACSR.csr
```

Here, OpensslSlot is the token label and TestRSAKey is the key on the HSM. Provide Cryptouser PIN when prompted.

```
root@Openssl-ubuntu:~/libp11-0.4.12# openssl req -engine pkcs11 -new -key "pkcs11:token=OpensslSlot;object=TestRSAKey" -keyfor
m engine -out TestRSACSR.csr
engine "pkcs11" set.
Enter PKCS#11 token PIN for OpensslSlot:
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) [AU]:US
State or Province Name (full name) [Some-State]:CA
Locality Name (eg, city) []:campbell
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Utimaco
Organizational Unit Name (eg, section) []:Utimaco
Common Name (e.g. server FQDN or YOUR name) []:openssl
Email Address []:support@utimaco.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
root@Openssl-ubuntu:~/libp11-0.4.12#
```

Figure 5:Certificate request output

```
root@Openssl-ubuntu:~/libp11-0.4.12# cat TestRSACSR.csr
-----BEGIN CERTIFICATE REQUEST-----
MIICzTCCAbUCAQAwgYcxCzAJBgNVBAYTAlVTMQswCQYDVQQIDAJDQTERMA8GA1UE
BwwIY2FtcGJlbGwxEDAOBgNVBAoMB1V0aW1hY28xEDAOBgNVBAsMB1V0aW1hY28x
EDAOBgNVBAMMB29wZW5zc2wxIjAgBgkqhkiG9w0BCQEWE3N1cHBvcnRAdXRpbWFj
by5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCJrlR7CKpB5c9l
qy1PiQqljcBQCR3PU+vn6ZRSKNMqKGwkVFcqriB8ZyO8Agad5WP4uVvSgZ650eci
NmwRaQLcacgtYV24Rqg49+l3edQhyTMJUQOCZw+kyciAqfqpu5eLBUqnKPUJ6wFu
CXO9lGopCLfsU06kq6/iO2v/KyOWQPgo86lTHsRmCu0anSf8gjIeBnMWsTmXffZZ
dFvFDfDT0ELLVqnqOaQzsCA8A3SlbKAH+bKV8OkIAmwJrKjgDsGqnA+ryPdmrShb
4wzqv3fnrLRBq8TGpIBuHO5y5BUvOxu1PoYE6F4dPnlW0xkhupZ/88vuJ97CRUDU
CJZiRq/tAgMBAAGgADANBgkqhkiG9w0BAQsFAAOCAQEAHTSGc9r7j6G+KY3QYia1
cC1l0gNmdXgnZNbB9zl8LhzDz6odr98e2VyJCG6u+VS3VoOawtpGkJmKblqlQndt
MXMWbDF6J6IRoqEIJGS1YyJ2QJzk5VpuCLxs+hb74OjrZFh9xbGo2bPskRdtTSrV
4gjQqPAJy9v4cKS8nbtOWl3MXNk0M3OwhMStEhJfCcRLTkhA7XV8jiKjRHfrXgio
AfDDnue2dEGkss29iDt6v1dNqf3GJC5LV8sOTqywuTkKgtq7Y+b1RAu0I0FccmpN
MIcpYPRRU7tHOKFO8aixi3DxLBiEJTXwugsenaXfW/fkfA/6LOs9qrH+dwiqc5rJ
fg==
-----END CERTIFICATE REQUEST-----
root@Openssl-ubuntu:~/libp11-0.4.12#
```

Figure 6:Content of certificate request file

4. Create the self-signed certificate based on the generated key

›_ Console

```
# openssl req -engine pkcs11 -new -x509 -days 365 -key
"pkcs11:token=OpensslSlot;object=TestRSAKey" -keyform engine -out TestRSA.cert
```

Here, OpensslSlot is the token label and TestRSAKey is the key on the HSM. Provide Cryptouser PIN when prompted.

```
root@Openssl-ubuntu:~/libp11-0.4.12# openssl req -engine pkcs11 -new -x509 -days 365 -key "pkcs11:token=OpensslSlot;object=Tes
tRSAKey" -keyform engine -out TestRSA.cert
engine "pkcs11" set.
Enter PKCS#11 token PIN for OpensslSlot:
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) [AU]:US
State or Province Name (full name) [Some-State]:CA
Locality Name (eg, city) []:Campbell
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Utimaco
Organizational Unit Name (eg, section) []:Utimaco
Common Name (e.g. server FQDN or YOUR name) []:Openssl
Email Address []:support@utimaco.com
root@Openssl-ubuntu:~/libp11-0.4.12#
```

Figure 7: Self signed certificate generation output

```
root@Openssl-ubuntu:~/libp11-0.4.12# cat TestRSA.cert
-----BEGIN CERTIFICATE-----
MIID8TCCAtmgAwIBAgIUfQzBr+vul0FCcN7jhvMVyu2sAUswDQYJKoZIhvcNAQEL
BQAwgYcxCzAJBgNVBAYTAlVTMQswCQYDVQQIDAJDQTERMA8GA1UEBwwIQ2FtcGJl
bGwxEDAOBgNVBAoMB1V0aW1hY28xEDAOBgNVBAsMB1V0aW1hY28xEDAOBgNVBAMM
B09wZW5zc2wxIjAgBgkqhkiG9w0BCQEWE3N1cHBvcnRAdXRpbWFjby5jb20wHhcN
MjIwODIyMTc1NDI4WhcNMjMwODIyMTc1NDI4WjCBhzELMAkGA1UEBhMCVVMxCzAJ
BgNVBAgMAkNBMREwDwYDVQQHDAhDYW1wYmVsbDEQMA4GA1UECgwHVXRpbWFjbzEQ
MA4GA1UECwwHVXRpbWFjbzEQMA4GA1UEAwwHT3BlbnNzbDEiMCAGCSqGSIb3DQEJ
ARYTc3VwcG9ydEB1dGltYWNvLmNvbTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCC
AQoCggEBAImuVHsIqkHlz2WrLU+JCqWNwFAJHc9T6+fplFIo0yoobCRUVyquIHxn
I7wCBp3lY/i5W9KBnrnR5yI2bBFpAtxpyC1hXbhGqDj36Xd51CHJMwlRA4JnD6TJ
yICp+qm7l4sFSqco9QnrAW4Jc72UaikIt+xTTqSrr+I7a/8rI5ZA+CjzqVMexGYK
7RqdJ/yCMh4GcxaxOZd99ll0W8UN8NPQQstWqeo5pDOwIDwDdKVsoAf5spXw6QgC
bAmsqOAOwaqcD6vI92atKFvjDOq/d+estEGrxMakgG4c7nLkFS87G7U+hgToXh0+
eVbTGSG6ln/zy+4n3sJFQNQIlmJGr+0CAwEAAaNTMFEwHQYDVR0OBBYEFP9ndHRK
voa07bLS4Pj42a645/aKMB8GA1UdIwQYMBaAFP9ndHRKvoa07bLS4Pj42a645/aK
MA8GA1UdEwEB/wQFMAMBAf8wDQYJKoZIhvcNAQELBQADggEBAHTWEC3h+FNkix18
poOBi40B6GdYYg1M91x3uEC6rq4Cwvs3BS6cq+2vJaqsIh9mYHtcIkSurA/tASV6
vwh+75ajNnYVXzSIivg6w2Uy+9eijatvD04KLQmhHxxpucZrE/OyvrI+OIG6oudt
ScV7olm8r3veLW42Pha5bhFnL2gMRj/09DPtUxBNVs6lle2MbXKu73OhfvUvECgG
27cGE0L5hcfAh7XopadR5u21w9DaGTHrHi3lfBwIlnW/SWjsF6Y4pmsnG46AXs9K
yPUSAmMMrQG20hqy+Ph9+eriHCPHMopR3aeUg3kXuYzWrjBNxSszIdbkGSaZqKz7
p/eXWrE=
-----END CERTIFICATE-----
root@Openssl-ubuntu:~/libp11-0.4.12#
```

Figure 8: Content of self-signed certificate file

5. Create a sample text file with any content inside it

›_ Console

```
# touch message.txt
```

```
[root@openssl-ubuntu ~]# cat message.txt
WELCOME to Utimaco
[root@openssl-ubuntu ~]#
```

Figure 9: Content of message.txt

6. Sign the message file

›_ Console

```
# openssl cms -engine pkcs11 -sign -in message.txt -signer TestRSA.cert - inkey
"pkcs11:token=OpensslSlot;object=TestRSAKey" -keyform engine -out
signedRSAmessage.txt
```

Here, OpensslSlot is the token label and TestRSAKey is the key on the HSM. Provide Cryptouser PIN when prompted.

```
root@Openssl-ubuntu:~/Test# cat signedRSAmessage.txt
MIME-Version: 1.0
Content-Type: multipart/signed; protocol="application/pkcs7-signature"; micalg="sha-256"; boundary="----11E44D2386E7765BD0B12F
FAA5A13161"

This is an S/MIME signed message

------11E44D2386E7765BD0B12FFAA5A13161
WELCOME to Utimaco

------11E44D2386E7765BD0B12FFAA5A13161
Content-Type: application/pkcs7-signature; name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"

MIIG5gYJKoZIhvcNAQcCoIIG1zCCBtMCAQExDTALBglghkgBZQMEAgEwCwYJKoZI
hvcNAQcBoIID+TCCA/UwggLdoAMCAQICFHMfgmGnxHsAFuETpGwOHVZMuH6SMA0G
CSqGSIb3DQEBCwUAMIGJMQswCQYDVQQGEwJVUzELMAkGA1UECAwCQ0ExETAPBgNV
BAcMCGNhbXBiZWxsMRIwEAYDVQQKDAlvcGVuc3NsQ0ExEDAOBgNVBAsMB1V0aW1h
Y28xEDAOBgNVBAMMB1V0aW1hY28xIjAgBgkqhkiG9w0BCQEWE3N1cHBvcnRAdXRp
bWFjby5jb20wHhcNMjIwODE2MTYyNjQ2WhcNMjMwODE2MTYyNjQ2WjCBiTELMAkG
A1UEBhMCVVMxCzAJBgNVBAgMAkNBMREwDwYDVQQHDAhjYW1wYmVsbDESMBAGA1UE
CgwJb3BlbnNzbENBMRAwDgYDVQQLDAdVdGltYWNvMRAwDgYDVQQDDAdVdGltYWNv
MSIwIAYJKoZIhvcNAQkBFhNzdXBwb3J0QHV0aW1hY28uY29tMIIBIjANBgkqhkiG
9w0BAQEFAAOCAQ8AMIIBCgKCAQEAia5UewiqQeXPZastT4kKpY3AUAkdz1Pr5+mU
UijTKihsJFRXKq4gfGcjvAIGneVj+Llb0oGeudHnIjZsEWkC3GnILWFduEaoOPfp
d3nUIckzCVEDgmcPpMnIgKn6qbuXiwVKpyj1CesBbglzvZRqKQi37FNOpKuv4jtr
/ysjlkD4KPOpUx7EZgrtGp0n/IIyHgZzFrE5l332WXRbxQ3w09BCy1ap6jmkM7Ag
PAN0pWygB/mylfDpCAJsCayo4A7BqpwPq8j3Zq0oW+MM6r9356y0QavExqSAbhzu
```

```
qqLunwb1xkzP47jvKOz7h0DEz27ngrHUN/0Dm7z9fPfnbdV9zxPdGCUe6Bx1GDmy
Z/Cw/GchB89fcmgMh22FM6aayOkufnKyFVuYh8gMJARTvhIS/pxXJzDAlK+pQ7R6
mTyI3nsQgems6I/miNq8003+Pxl0oej5btH1nCnakhbe83Ty6MW4NOavE6IXqrrD
C3UeLqvABvHdQp3m2BfwbGjxVK2PXpbjVxaL1yQyeDrtTamyIE6Y2YApeWH28XTG
DQjiYhKty9h0dAahmDLRIm1BITGCArMwggKvAgEBMIGiMIGJMQswCQYDVQQGEwJV
UzELMAkGA1UECAwCQ0ExETAPBgNVBAcMCGNhbXBiZWxsMRIwEAYDVQQKDAlvcGVu
c3NsQ0ExEDAOBgNVBAsMB1V0aW1hY28xEDAOBgNVBAMMB1V0aW1hY28xIjAgBgkq
hkiG9w0BCQEWE3N1cHBvcnRAdXRpbWFjby5jb20CFHMfgmGnxHsAFuETpGwOHVZM
uH6SMAsGCWCGSAFlAwQCAaCB5DAYBgkqhkiG9w0BCQMxCwYJKoZIhvcNAQcBMBwG
CSqGSIb3DQEJBTEPFw0yMjA4MTYxNjMwNDVaMC8GCSqGSIb3DQEJBDEiBCA+e308
Ewlpnw6OKn0q5ThFBVYxAYxOsfJdfTmSPIPySjB5BgkqhkiG9w0BCQ8xbDBqMAsG
CWCGSAFlAwQBKjALBglghkgBZQMEARYwCwYJYIZIAWUDBAECMAoGCCqGSIb3DQMH
MA4GCCqGSIb3DQMCAgIAgDANBggqhkiG9w0DAgIBQDAHBgUrDgMCBzANBggqhkiG
9w0DAgIBKDANBgkqhkiG9w0BAQEFAASCAQAVieqOR9qv4CXr9a9nUwueBv/txjvn
jkzm+sPErMeQ2MKRIIWlQxWD2N3+hG4B4zMd8o4WVT7SA2rhN1MzLZoXQ5zudKqq
wuLKZC1nBuER8n4aoQmAwnFbPqtxICSBMPUdWT1+wEeXReh0W4qnt6228p8jfR2+
fMzUE0iqheORpe0Z38EUD25s0RgO+dYiCjjZod9S20UR/FBuXvbSU/QstrnxCXNY
GVSX6T1cHoDUXR/3sq/op+4fHGDd4SQJUj+HGfZyKmCwWjewbcAuHvx2uNQBLQEC
exLW5PkbF1GjjTQzSpcsCDdQprUCTXcywR/aH4CrbS1NoCiDlMOGO2yg

------11E44D2386E7765BD0B12FFAA5A13161--

root@Openssl-ubuntu:~/Test#
```

Figure 10: Content of signed message file

7. Encrypt the signed message file

›_ Console

```
# openssl cms -engine pkcs11 -encrypt -in signedRSAmessage.txt-out
encryptedRSAsignedmessage.txt TestRSA.cert
```

```
root@Openssl-ubuntu:~/Test# cat encryptedRSAsignedmessage.txt
MIME-Version: 1.0
Content-Disposition: attachment; filename="smime.p7m"
Content-Type: application/pkcs7-mime; smime-type=enveloped-data; name="smime.p7m"
Content-Transfer-Encoding: base64

MIIJBAYJKoZIhvcNAQcDoIII9TCCCPECAQIxggEhoYIBHQIBA6BRoU8wCQYHKoZI
zj0CAQNCAARkJarYHMQ+u3tHDjSKslB5/fKcj/I+M0zsKanaRoBfb/GspqkP7mrd
HA3oSDcDEk5QOwieb1LSrLUfM2M3+0L/MBwGCSuBBRCGSD8AAjAPBgsqhkiG9w0B
CRADBgUAMIGmMIGjMHcwXzELMAkGA1UEBhMCSU4xCzAJBgNVBAgMAk1IMQ0wCwYD
VQQHDARQdW5lMRAwDgYDVQQKDAd1dGltYWNvMRAwDgYDVQQLDAd1dGltYWNvMRAw
DgYDVQQDDAdvcGVuc3NsAhRWpoSUs3nnfx8pbhoQB46XEQCKUQQoJxOtmYYQqWVu
qaleraMhZbRflhC3ml2ywr2aTJTAfL2+NcCqD+4XqjCCB8UGCSqGSIb3DQEHATAU
BggqhkiG9w0DBwQI9QGHas61VXuAggeg3w1+bGJjYzH033Y8bqAXpOmxOTD6BnQ+
34inId6mVmFPbGB9Sii5L4KEwv0J3V1pXrr+/FRLa1WExCXPhFzFtsZVGv9DQnmK
gGPIYGULF6pU9+V93TboVl9lhnimPCiczb9ehCEybMq+gMwKJiQi43BlSSJNV0qQ
/M9tMoefAdnx5wVwg3XUvzrP9YIsujmdc733/8i7HtvHz3QIiTf/sUX1n0wtF/gt
xZKx5HdKHcMyxv8ToSfa8No6ioKFcewL/yzdc6mTz/+vnc2wSfhQIVucrVOCyrmW
Lzk4Q8kAbo+n2xdcSd3vL16ytioD1TXLWOw4G5Ec7C4bhp2d9Aq/lySbfDt3Tvuk
6Ouegul/dqjvB4da+EkI88e1RrXkgkWilFHOOOuQ+nb1Wg8M7/skcgVLQUobTzYx
zHOJe5hcUbHTE/o/C5D6FJavlhgeZ2TrFEsctobMdImR02kf54B8G2WXxTUb3efg
9Tfx3UXPSMtHeZEKW0t1cmLJHvOGO7XrK27J11asFTJBBfj5A7roauyoqWjYojtS
ld6syYPYMn6g/KMIxgpxZn6orzsV2fI06NXUqfaZL9goxRaurQk3vDhJb9pBp2Mo
EpRuXHM0DHJuvZEX2VvDkAf6aoY38uE5L+Ti+ETHPRUckvWbme1fToV8tuW/PENy
```

Figure 11: Content of encrypted signed message file

8. Decrypt the encrypted signed message file

›_ Console

```
# openssl cms -engine pkcs11 -decrypt -in encryptedRSAsignedmessage.txt - inkey
"pkcs11:token=OpensslSlot;object=TestRSAKey" -keyform engine -out
decryptedRSAsignedmessage.txt
```

Here, OpensslSlot is the token label and TestRSAKey is the key on the HSM. Provide Cryptouser PIN when prompted.

```
root@Openssl-ubuntu:~/Test# cat decryptedRSAsignedmessage.txt
MIME-Version: 1.0
Content-Type: multipart/signed; protocol="application/pkcs7-signature"; micalg="sha-256"; boundary="----11E44D2386E7765BD0B12F
FAA5A13161"

This is an S/MIME signed message

------11E44D2386E7765BD0B12FFAA5A13161
WELCOME to Utimaco

------11E44D2386E7765BD0B12FFAA5A13161
Content-Type: application/pkcs7-signature; name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"

MIIG5gYJKoZIhvcNAQcCoIIG1zCCBtMCAQExDTALBglghkgBZQMEAgEwCwYJKoZI
hvcNAQcBoIID+TCCA/UwggLdoAMCAQICFHMfgmGnxHsAFuETpGwOHVZMuH6SMA0G
CSqGSIb3DQEBCwUAMIGJMQswCQYDVQQGEwJVUzELMAkGA1UECAwCQ0ExETAPBgNV
BAcMCGNhbXBiZWxsMRIwEAYDVQQKDAlvcGVuc3NsQ0ExEDAOBgNVBAsMB1V0aW1h
Y28xEDAOBgNVBAMMB1V0aW1hY28xIjAgBgkqhkiG9w0BCQEWE3N1cHBvcnRAdXRp
bWFjby5jb20wHhcNMjIwODE2MTYyNjQ2WhcNMjMwODE2MTYyNjQ2WjCBiTELMAkG
A1UEBhMCVVMxCzAJBgNVBAgMAkNBMREwDwYDVQQHDAhjYW1wYmVsbDESMBAGA1UE
CgwJb3BlbnNzbENBMRAwDgYDVQQLDAdVdGltYWNvMRAwDgYDVQQDDAdVdGltYWNv
MSIwIAYJKoZIhvcNAQkBFhNzdXBwb3J0QHV0aW1hY28uY29tMIIBIjANBgkqhkiG
9w0BAQEFAAOCAQ8AMIIBCgKCAQEAia5UewiqQeXPZastT4kKpY3AUAkdz1Pr5+mU
UijTKihsJFRXKq4gfGcjvAIGneVj+Llb0oGeudHnIjZsEWkC3GnILWFduEaoOPfp
```

Figure 12: Content of decrypted signed message file

9. Verify the decrypted signed message file

›_ Console

```
# openssl cms -engine pkcs11 -verify -in decryptedRSAsignedmessage.txt - CAfile
TestRSA.cert -out originalmessage.txt TestRSA.cert
```

```
root@Openssl-ubuntu:~/Test# openssl cms -engine pkcs11 -verify -in decryptedRSAsignedmessage.txt -CAfile TestRSA.cert -out ori
ginalmessage.txt TestRSA.cert
engine "pkcs11" set.
Verification successful
root@Openssl-ubuntu:~/Test#
```

Figure 13: Output of openssl verification command

```
root@Openssl-ubuntu:~/Test# cat originalmessage.txt
WELCOME to Utimaco
root@Openssl-ubuntu:~/Test#
```

Figure 14: Content of original message file

4.4.3.2 Testing with ECDSA Key

1. Generate the ECDSA key using p11tool2

›_ Console

```
# p11tool2 slot=2 LoginUser=123456 PubKeyAttr=CKA_LABEL="TestECDSAKey"
 PrvKeyAttr=CKA_LABEL="TestECDSAKey",CKA_DERIVE=CK_TRUE GenerateKeyPair=ECC
```

Once key generation is complete then add CKA_ID for both public and private ECDSA keys using PKCS11# CryptoServer Administration tool. Also, make sure to set CKA_DERIVE=CK_TRUE in above command.

2. Verify that the keys are generated onto the HSM using the following command

›_ Console

```
# p11tool2 slot=<Slot_No.> LoginUser=<CryptoUser_PIN> ListObjects
```

Example

›_ Console

```
# p11tool2 slot=2 LoginUser=123456 ListObjects

CKO_PUBLIC_KEY:

+ 1.1
CKA_KEY_TYPE    = CKK_ECDSA
CKA_LABEL   = TestECDSAKey
CKA_ID  = 0x56 (V)

CKO_PRIVATE_KEY:

+ 2.1
CKA_KEY_TYPE    = CKK_ECDSA
CKA_SENSITIVE   = CK_TRUE
CKA_EXTRACTABLE = CK_FALSE
CKA_LABEL = TestECDSAKey
CKA_ID = 0x56 (V)
```

3. Generate a certificate request

›_ Console

```
# openssl req -engine pkcs11 -new -key
"pkcs11:token=OpensslSlot;object=TestECDSAKey" -keyform engine -out
TestECDSACSR.csr
```

```
root@Openssl-ubuntu:~/test# openssl req -engine pkcs11 -new -key "pkcs11:token=OpensslSlot;object=TestECDSAKey" -keyform engin
e -out TestECDSACSR.csr
engine "pkcs11" set.
Enter PKCS#11 token PIN for OpensslSlot:
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) [AU]:US
State or Province Name (full name) [Some-State]:CA
Locality Name (eg, city) []:cambpell
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Utimaco
Organizational Unit Name (eg, section) []:Utimaco
Common Name (e.g. server FQDN or YOUR name) []:openssl
Email Address []:support@utimaco.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
root@Openssl-ubuntu:~/test#
```

```
root@Openssl-ubuntu:~/test# cat TestECDSACSR.csr
-----BEGIN CERTIFICATE REQUEST-----
MIIBQjCB6gIBADCBhzELMAkGA1UEBhMCVVMxCzAJBgNVBAgMAkNBMREwDwYDVQQH
DAhjYW1icGVsbDEQMA4GA1UECgwHVXRpbWFjbzEQMA4GA1UECwwHVXRpbWFjbzEQ
MA4GA1UEAwwHb3BlbnNzbDEiMCAGCSqGSIb3DQEJARYTc3VwcG9ydEB1dGltYWNv
LmNvbTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABMWwkkzXkrpG66ZRCYVigX9I
tZtuRdiANIaMtfhjHTKFFKt96sLYcCIdFeVAbW6UF7sKV2MO9Ez+m1xwLYqsHzeg
ADAKBggqhkjOPQQDAgNHADBEAiA3guBiT4Rkx3poEF4lSMTXtEdhSdaYzGmFiWSB
WLs0PwIgJ62vXOIaS0oQC8pWWRj7hv2T4QK+je7TokjsGgo31Ms=
-----END CERTIFICATE REQUEST-----
root@Openssl-ubuntu:~/test#
```

Figure 15: Certificate request command output and the Content of certificate request file

Here, OpensslSlot is the token label and TestECDSAKey is the key on the HSM. Provide Cryptouser PIN when prompted.

4. Create a self-signed certificate based on the generated key

›_ Console

```
# openssl req -engine pkcs11 -new -x509 -days 365 -key
"pkcs11:token=OpensslSlot;object=TestECDSAKey" -keyform engine -out
TestECDSA.cert
```

Here, OpensslSlot is the token label and TestECDSAKey is the key on the HSM. Provide Cryptouser PIN when prompted.

```
root@Openssl-ubuntu:~/test# openssl req -engine pkcs11 -new -x509 -days 365 -key "pkcs11:token=OpensslSlot;object=TestECDSAKey
" -keyform engine -out TestECDSA.cert
engine "pkcs11" set.
Enter PKCS#11 token PIN for OpensslSlot:
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) [AU]:US
State or Province Name (full name) [Some-State]:CA
Locality Name (eg, city) []:Cambpell
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Utimaco
Organizational Unit Name (eg, section) []:Utimaco
Common Name (e.g. server FQDN or YOUR name) []:openssl123
Email Address []:support@utimaco.com
```

```
root@Openssl-ubuntu:~/test# cat TestECDSA.cert
-----BEGIN CERTIFICATE-----
MIICbDCCAhGgAwIBAgIUE2lgZOs4DOmK/LUidWky7cNgJ18wCgYIKoZIzj0EAwIw
gYoxCzAJBgNVBAYTAlVTMQswCQYDVQQIDAJDQTERMA8GA1UEBwwIQ2FtYnBlbGwx
EDAOBgNVBAoMB1V0aW1hY28xEDAOBgNVBAsMB1V0aW1hY28xEzARBgNVBAMMCm9w
ZW5zc2wxMjMxIjAgBgkqhkiG9w0BCQEWE3N1cHBvcnRAdXRpbmFjby5jb20wHhcN
MjIwODI0MTExOTQyWhcNMjMwODI0MTExOTQyWjCBijELMAkGA1UEBhMCVVMxCzAJ
BgNVBAgMAkNBMREwDwYDVQQHDAhDYW1icGVsbDEQMA4GA1UECgwHVXRpbWFjbzEQ
MA4GA1UECwwHVXRpbWFjbzETMBEGA1UEAwwKb3BlbnNzbDEyMzEiMCAGCSqGSIb3
DQEJARYTc3VwcG9ydEB1dGluYWNvLmNvbTBZMBMGByqGSM49AgEGCCqGSM49AwEH
A0IABMWwkkzXkrpG66ZRCYVigX9ItZtuRdiANIaMtfhjHTKFFKt96sLYcCIdFeVA
bW6UF7sKV2MO9Ez+m1xwLYqsHzejUzBRMB0GA1UdDgQWBBRbgefwzvm857//7Sya
ZoS4twsg6zAfBgNVHSMEGDAWgBRbgefwzvm857//7SyaZoS4twsg6zAPBgNVHRMB
Af8EBTADAQH/MAoGCCqGSM49BAMCA0kAMEYCIQCCFZIsRAsviUO3QWihv+A8/mzM
bHH1kRD97yU6BlKJBAIhAOGKl7WaqthbCsPI2xnPFHh4TBrlRPwNYJgBitfGi6zn
-----END CERTIFICATE-----
root@Openssl-ubuntu:~/test#
```

Figure 16: Self signed certificate generation output and content of certificate file

5. Create a sample text file and write any content inside it

›_ Console

```
# touch message.txt
```

```
root@Openssl-ubuntu:~/test# cat message.txt
WELCOME to Utimaco Security World
root@Openssl-ubuntu:~/test#
```

Figure 17: Content of message file

6. Sign the message file

›_ Console

```
# openssl cms -engine pkcs11 -sign -in message.txt -signer TestECDSA.cert-inkey
"pkcs11:token=OpensslSlot;object=TestECDSAKey" -keyform engine -out
signedECDSAmessage.txt
```

Here, OpensslSlot is the token label and TestECDSAKey is the key on the HSM. Provide Cryptouser PIN when prompted.

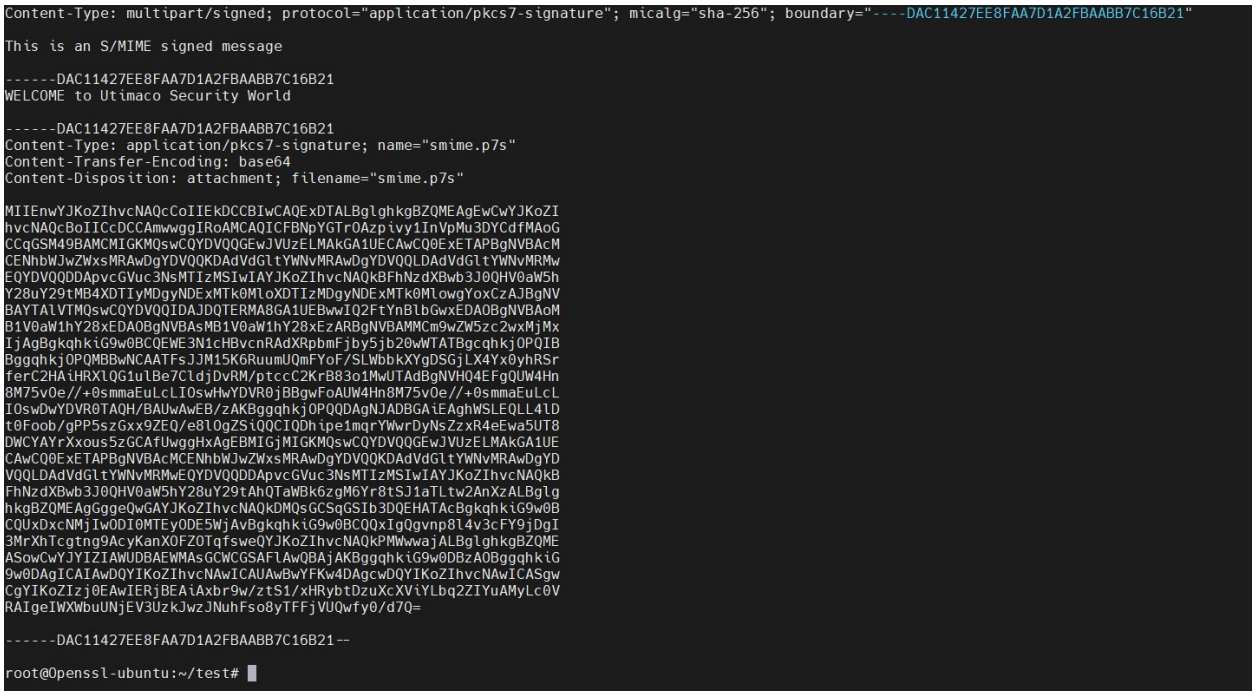

Figure 18: Content of signed message file

7. Encrypt the signed message file

›_ Console

```
# openssl cms -engine pkcs11 -encrypt -in signedECDSAmessage.txt -out
encryptedECDSAsignedmessage.txt TestECDSA.cert
```

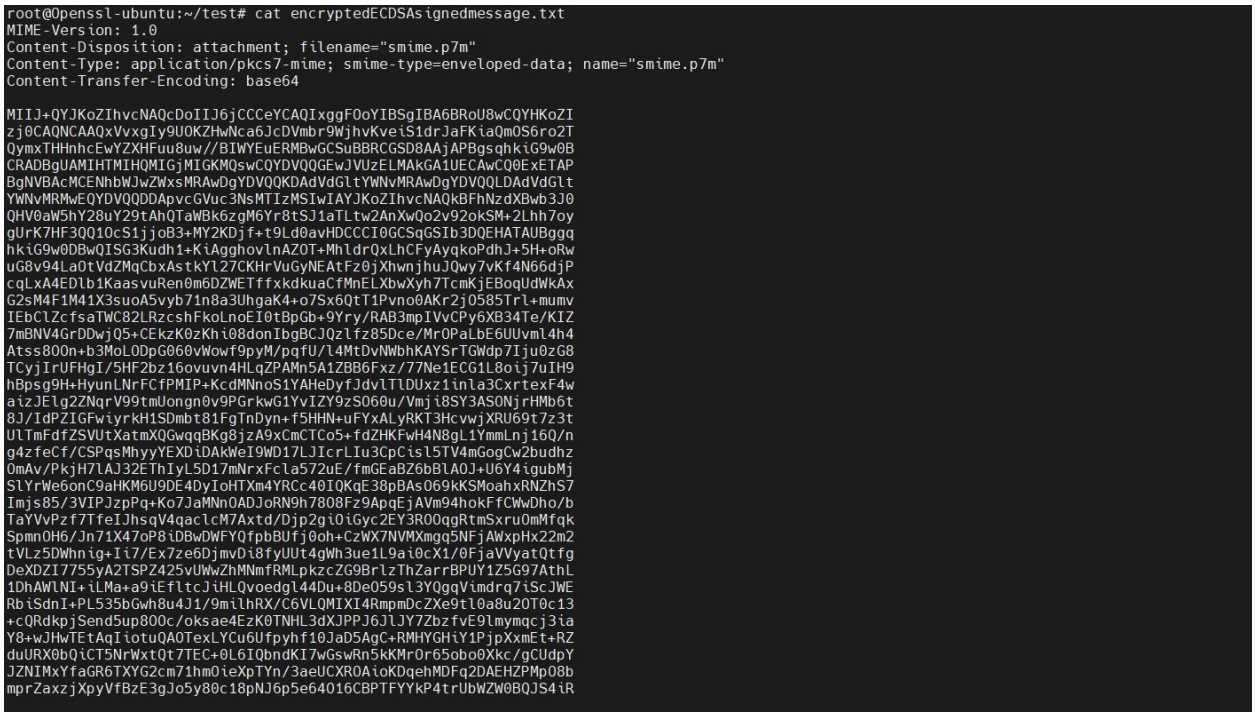

Figure 19: Content of encrypted signed message file

8. Decrypt the encrypted signed message file

›_ Console

```
# openssl cms -engine pkcs11 -decrypt -in encryptedECDSAsignedmessage.txt -inkey
"pkcs11:token=OpensslSlot;object=TestECDSAKey" -keyform engine -out
decryptedECDSAsignedmessage.txt
```

Here, OpensslSlot is the token label and TestECDSAKey is the key on the HSM. Provide Cryptouser PIN when prompted.

```
root@Openssl-ubuntu:~/test# cat decryptedECDSAsignedmessage.txt
MIME-Version: 1.0
Content-Type: multipart/signed; protocol="application/pkcs7-signature"; micalg="sha-256"; boundary="----F962038211F1EA542BCF0ECC3CEC6BC8"

This is an S/MIME signed message

------F962038211F1EA542BCF0ECC3CEC6BC8
WELCOME to Utimaco Security World

------F962038211F1EA542BCF0ECC3CEC6BC8
Content-Type: application/pkcs7-signature; name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"

MIIEVwYJKoZIhvcNAQcCoIIESDCCBEQCAQExDTALBglghkgBZQMEAgEwCwYJKoZI
hvcNAQcBoIICPzCCAjswggHhoAMCAQICFC515AfQ3skPsXQmAGNqqnQzzcRfMAoG
CCqGSM49BAMCMHMxCzAJBgNVBAYTAmluMQswCQYDVQQIDAJtaDENMAsGA1UEBwwE
cHVuZTERMA8GA1UECgwIdXRpZW1hY28xEDAOBgNVBAsMB3V0aW1hY28xEDAOBgNV
BAMMB29wZW5zc2wxETAPBgkqhkiG9w0BCQEWAmRqMB4XDTIyMDgyNDExNDIzMFoX
DTIzMDgyNDExNDIzMFowczELMAkGA1UEBhMCaW4xCzAJBgNVBAgMAm1oMQ0wCwYD
VQQHDARwdW5lMREwDwYDVQQKDAh1dGllbWFjbzEQMA4GA1UECwwHdXRpbWFjbzEQ
MA4GA1UEAwwHb3BlbnNzbDERMA8GCSqGSIb3DQEJARYCZGowWTATBgcqhkjOPQIB
BggqhkjOPQMBBwNCAARldSO1A7ZIOWR0VGOu2p3DWVWFyEwCIzy11rcui4FNUJqt
JeRme5v/thhd5o1aZt14EG3I+jV53nj5Gi43m9ioo1MwUTAdBgNVHQ4EFgQUYWnF
8W2s0+Zo2XlQWI7Z1UvJ0VQwHwYDVR0jBBgwFoAUYWnF8W2s0+Zo2XlQWI7Z1UvJ
0VQwDwYDVR0TAQH/BAUwAwEB/zAKBggqhkjOPQQDAgNIADBFAiA5EtKAMr4FbQ6w
XSVUauqWCdwKKkh8a19QWBCn0akRaQIhAPtY0v/zlLHEHRd8Eie+VIpI4FpncvdV
rjcgDVDJB4ubMYIB3jCCAdoCAQEwgYswczELMAkGA1UEBhMCaW4xCzAJBgNVBAgM
Am1oMQ0wCwYDVQQHDARwdW5lMREwDwYDVQQKDAh1dGllbWFjbzEQMA4GA1UECwwH
dXRpbWFjbzEQMA4GA1UEAwwHb3BlbnNzbDERMA8GCSqGSIb3DQEJARYCZGoCFC51
5AfQ3skPsXQmAGNqqnQzzcRfMAsGCWCGSAFlAwQCAaCB5DAYBgkqhkiG9w0BCQMx
CwYJKoZIhvcNAQcBMBwGCSqGSIb3DQEJBTEPFw0yMjA4MjQxMTQ0NDFaMC8GCSqG
SIb3DQEJBDEiBCC+enyXi/dwVj2MOAjcyteFNyC2eD0BzIpqdc4Vk5Op+zB5Bgkq
hkiG9w0BCQ8xbDBqMAsGCWCGSAFlAwQBKjALBglghkgBZQMEARYwCwYJYIZIAWUD
BAECMAoGCCqGSIb3DQMHMA4GCCqGSIb3DQMCAgIAgDANBggqhkiG9w0DAgIBQDAH
BgUrDgMCBzANBggqhkiG9w0DAgIBKDAKBggqhkjOPQQDAgRHMEUCIBUYdyRw1O92
YWDtDjQitSOk7rlJ5fFF+4+1CADl4HvBAiEAwsrZMNu8njC1wey1PQGZDJ9dhbGn
CVzWkO6Rwivu7aY=

------F962038211F1EA542BCF0ECC3CEC6BC8--
```

Figure 20: Content of decrypted signed message file

9. Verify the decrypted signed message file

›_ Console

```
# openssl cms -engine pkcs11 -verify -in decryptedECDSAsignedmessage.txt -
CAfile TestECDSA.cert -out originalmessage.txt TestECDSA.cert
```

```
root@Openssl-ubuntu:~/test# openssl cms -engine pkcs11 -verify -in decryptedECDSAsignedmessage.txt -CAfile TestECDSA.cert -out originalmessage.txt Test
ECDSA.cert
engine "pkcs11" set.
Verification successful
```

Figure 21: Output of openssl verify command

10. Open the content of originalmessage.txt and verify it is the same as original content.

```
root@Openssl-ubuntu:~/test# cat originalmessage.txt
WELCOME to Utimaco Security World
root@Openssl-ubuntu:~/test#
```

Figure 22: Content of original message file

4.4.4 Creating a local CA (Certificate Authority) and performing Cryptographic operation with OpenSSL

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

›_ Console

```
dir = /localCA 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. Here we have created directory /localCA under root directory and new_certs_dir= $dir/newcerts

2. Create the directory /localCA/newcerts

›_ Console

```
# mkdir /localCA/newcerts
```

3. Create the text files /localCA/index.txt and /localCA/serial

›_ Console

```
# touch /localCA/index.txt # touch /localCA/serial
```

4. Open the /localCA/serial file and write 01 in it and click enter. Save the file
5. Create a key pair by using pkcs11tool2 for root CA

For RSA

›_ Console

```
# p11tool2 slot=0 LoginUser=123456 PubKeyAttr=CKA_LABEL="CAKey"
 PrvKeyAttr=CKA_LABEL="CAKey" GenerateKeyPair=RSA
```

This generates RSA 2048 CA private and public keys on the HSM

For ECDSA

To generate ECDSA CA keys on the HSM

›_ Console

```
# p11tool2 slot=0 LoginUser=123456 PubKeyAttr=CKA_LABEL="CAKey"
 PrvKeyAttr=CKA_LABEL="CAKey" GenerateKeyPair=ECC
```

Once key generation is completed then add CKA_ID for both public and private ECDSA keys using PKCS11# CryptoServer Administration tool.

6. Verify that the keys are generated onto the HSM using following command:

›_ Console

```
# p11tool2 Slot=<Slot_No.> LoginUser=<Cryptouser_PIN> ListObjects
```

For RSA

```
[root@openssl-ubuntu ~]# p11tool2 Slot=0 LoginUser=123456 ListObjects


CKO_PUBLIC_KEY:

+ 1.1
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_LABEL                     = CAKey
  CKA_ID                        =



CKO_PRIVATE_KEY:

+ 2.1
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = CAKey
  CKA_ID                        =

[root@openssl-ubuntu ~]#
```

Figure 23: CA RSA Key list

For ECDSA:

```
[root@openssl-ubuntu ~]# p11tool2 Slot=0 LoginUser=123456 ListObjects


CKO_PUBLIC_KEY:

+ 1.1
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_LABEL                     = CAKey
  CKA_ID                        = 0x45 (E)


CKO_PRIVATE_KEY:

+ 2.1
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = CAKey
  CKA_ID                        = 0x45 (E)
[root@openssl-ubuntu ~]#
[root@openssl-ubuntu ~]#
```

Figure 24: CA ECDSA Key list

7. Create the CA certificate based on the generated key that is used for signing other certificates by running below command:

›_ Console

```
# openssl req -engine pkcs11 -new -x509 -days 365 -key
"pkcs11:token=OpensslSlot;object=CAKey" -keyform engine -out
/localCA/newcerts/ca.cer
```

```
[root@openssl-ubuntu ~]# openssl req -engine pkcs11 -new -x509 -days 365 -key "pkcs11:token=OpensslSlot;object=CAKey" -keyform engine -out /localCA/newc
erts/ca.cer
engine "pkcs11" set.
Enter PKCS#11 token PIN for OpensslSlot:
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) [AU]:US
State or Province Name (full name) [Some-State]:CA
Locality Name (eg, city) []:Citynew
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Utimaco
Organizational Unit Name (eg, section) []:HSM
Common Name (e.g. server FQDN or YOUR name) []:test.utimaco.com
Email Address []:support@utimaco.com
[root@openssl-ubuntu ~]#
```

Figure 25: CA certificate generation output

Here, CAKey is the Object label for the CA private key on the Utimaco HSM created in Step 5, and OpensslSlot is token label. Provide Cryptouser PIN when prompted.

4.4.5 Generate Certificate Request for Sender and Receiver

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

›_ Console

```
# mkdir /localCA/newcerts/sender
# mkdir /localCA/newcerts/receiver
```

2. Generate a sender key pair using p11tool2

For RSA

›_ Console

```
# p11tool2 slot=0 LoginUser=123456 PubKeyAttr=CKA_LABEL="SenderKey"
 PrvKeyAttr=CKA_LABEL="SenderKey" GenerateKeyPair=RSA
```

For ECDSA

›_ Console

```
# p11tool2 slot=0 LoginUser=123456 PubKeyAttr=CKA_LABEL="SenderKey"
 PrvKeyAttr=CKA_LABEL="SenderKey" GenerateKeyPair=ECC
```

Once key generation is completed then add **CKA_ID** for both public and private ECDSA keys using **PKCS11# CryptoServer Administration** tool.

3. Verify that the keys are generated onto the HSM using the following command:

For RSA›_ Console

```
# p11tool2 slot=<Slot_No.> LoginUser=<CryptoUser_PIN> ListObjects
```

```
[root@openssl-ubuntu ~]# p11tool2 Slot=0 LoginUser=123456 ListObjects


CKO_PUBLIC_KEY:

+ 1.1
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_LABEL                     = SenderKey
  CKA_ID                        =


+ 1.2
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_LABEL                     = CAKey
  CKA_ID                        = 0x45 (E)


CKO_PRIVATE_KEY:

+ 2.1
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = SenderKey
  CKA_ID                        =


+ 2.2
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = CAKey
  CKA_ID                        = 0x45 (E)
```

Figure 26: Sender RSA Key list

For ECDSA

```
[root@openssl-ubuntu ~]# p11tool2 Slot=0 LoginUser=123456 ListObjects


CKO_PUBLIC_KEY:

+ 1.1
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_LABEL                     = SenderKey
  CKA_ID                        = 0x46 (F)

+ 1.2
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_LABEL                     = CAKey
  CKA_ID                        = 0x45 (E)


CKO_PRIVATE_KEY:

+ 2.1
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = SenderKey
  CKA_ID                        = 0x46 (F)

+ 2.2
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = CAKey
  CKA_ID                        = 0x45 (E)
```

Figure 27: Sender ECDSA Key list

4. Generate a certificate request for sender.

›_ Console

```
# openssl req -engine pkcs11 -new -key
"pkcs11:token=OpensslSlot;object=SenderKey" -keyform engine -out /localCA/
newcerts/sender/sender.txt
```

```
[root@openssl-ubuntu ~]# openssl req -engine pkcs11 -new -key "pkcs11:token=OpensslSlot;object=SenderKey" -keyform engine -out /localCA/newcerts/sender/sender.txt
engine "pkcs11" set.
Enter PKCS#11 token PIN for OpensslSlot:
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) [AU]:IN
State or Province Name (full name) [Some-State]:MH
Locality Name (eg, city) []:Pune
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Utimaco
Organizational Unit Name (eg, section) []:HSM
Common Name (e.g. server FQDN or YOUR name) []:sender.utimaco.com
Email Address []:sender@utimaco.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
[root@openssl-ubuntu ~]#
```

Figure 28: Sender certificate request generation

Enter the prompted value for "A challenge password" as blank.

Here, OpensslSlot is the token label and SenderKey is the key on the HSM. Provide Cryptouser PIN when prompted.

5. Sign the certificate request for sender by CA

›_ Console

```
# openssl ca -engine pkcs11 -policy policy_anything -cert /localCA/newcerts/
ca.cer -in /localCA/newcerts/sender/sender.txt -keyfile
"pkcs11:token=OpensslSlot;object=CAKey" -keyform engine -out/localCA/newcerts/
sender/SenderSignedCertificate.cert
```

```
[root@openssl-ubuntu ~]# openssl ca -engine pkcs11 -policy policy_anything -cert /localCA/newcerts/ca.cer -in /localCA/newcerts/sender/sender.txt -keyfi
le "pkcs11:token=OpensslSlot;object=CAKey" -keyform engine -out /localCA/newcerts/sender/SenderSignedCertificate.cert
engine "pkcs11" set.
Using configuration from /usr/local/openssl/ssl/openssl.cnf
Enter PKCS#11 token PIN for OpensslSlot:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 1 (0x1)
        Validity
            Not Before: Aug 28 20:27:20 2022 GMT
            Not After : Aug 28 20:27:20 2023 GMT
        Subject:
            countryName               = IN
            stateOrProvinceName       = MH
            localityName              = Pune
            organizationName          = Utimaco
            organizationalUnitName    = HSM
            commonName                = sender.utimaco.com
            emailAddress              = sender@utimaco.com
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                92:FD:FA:01:20:BD:F2:20:4D:1D:3E:1E:64:93:AD:43:F0:0C:01:FE
            X509v3 Authority Key Identifier:
                keyid:5F:51:53:AB:98:1A:6B:1C:6E:3E:38:D8:D9:32:98:C8:E6:EC:29:D6

Certificate is to be certified until Aug 28 20:27:20 2023 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
[root@openssl-ubuntu ~]#
```

Figure 29: Sender certificate request signing by CA

Press y to sign and y again to commit.

Here, OpensslSlot is the token label and CAKey is the key on the HSM. Provide Cryptouser PIN when prompted.

6. Generate key pair for receiver using p11tool2

For RSA

›_ Console

```
./p11tool2 slot=0 LoginUser=123456 PubKeyAttr=CKA_LABEL="ReceiverKey"
 PrvKeyAttr=CKA_LABEL="ReceiverKey" GenerateKeyPair=RSA
```

For ECDSA

›_ Console

```
./p11tool2 slot=0 LoginUser=123456 PubKeyAttr=CKA_LABEL="ReceiverKey"
 PrvKeyAttr=CKA_LABEL="ReceiverKey",CKA_DERIVE=CK_TRUE GenerateKeyPair=ECC
```

Once key generation is completed then add CKA_ID for both public and private ECDSA keys using **PKCS11# CryptoServer Administration** tool.

Also, make sure to set **CKA_DERIVE=CK_TRUE** in the above command

7. Verify that key pair is generated onto the HSM using the following command:

›_ Console

```
# p11tool2 slot=<Slot_No.> LoginUser=<CryptoUser_PIN> ListObjects
```

For RSA

```
[root@openssl-ubuntu ~]# p11tool2 Slot=0 LoginUser=123456 ListObjects

CKO_PUBLIC_KEY:

+ 1.1
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_LABEL                     = ReceiverKey
  CKA_ID                        =


+ 1.2
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_LABEL                     = SenderKey
  CKA_ID                        = 0x46 (F)

+ 1.3
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_LABEL                     = CAKey
  CKA_ID                        = 0x45 (E)


CKO_PRIVATE_KEY:

+ 2.1
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = SenderKey
  CKA_ID                        = 0x46 (F)

+ 2.2
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = CAKey
  CKA_ID                        = 0x45 (E)

+ 2.3
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = ReceiverKey
  CKA_ID                        =
```

Figure 30: Receiver RSA Key list

For ECDSA

```
[root@openssl-ubuntu ~]# p11tool2 Slot=0 LoginUser=123456 ListObjects


CKO_PUBLIC_KEY:

+ 1.1
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_LABEL                     = ReceiverKey
  CKA_ID                        = 0x47 (G)

+ 1.2
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_LABEL                     = SenderKey
  CKA_ID                        = 0x46 (F)

+ 1.3
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_LABEL                     = CAKey
  CKA_ID                        = 0x45 (E)


CKO_PRIVATE_KEY:

+ 2.1
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = SenderKey
  CKA_ID                        = 0x46 (F)

+ 2.2
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = CAKey
  CKA_ID                        = 0x45 (E)

+ 2.3
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = ReceiverKey
  CKA_ID                        = 0x47 (G)
[root@openssl-ubuntu ~]#
```

Figure 31: Receiver ECDSA Key list

8. Generate a certificate request for receiver.

›_ Console

```
# openssl req -engine pkcs11 -new -key
"pkcs11:token=OpensslSlot;object=ReceiverKey" -keyform engine -out/localCA/
newcerts/receiver/Receiver.txt
```

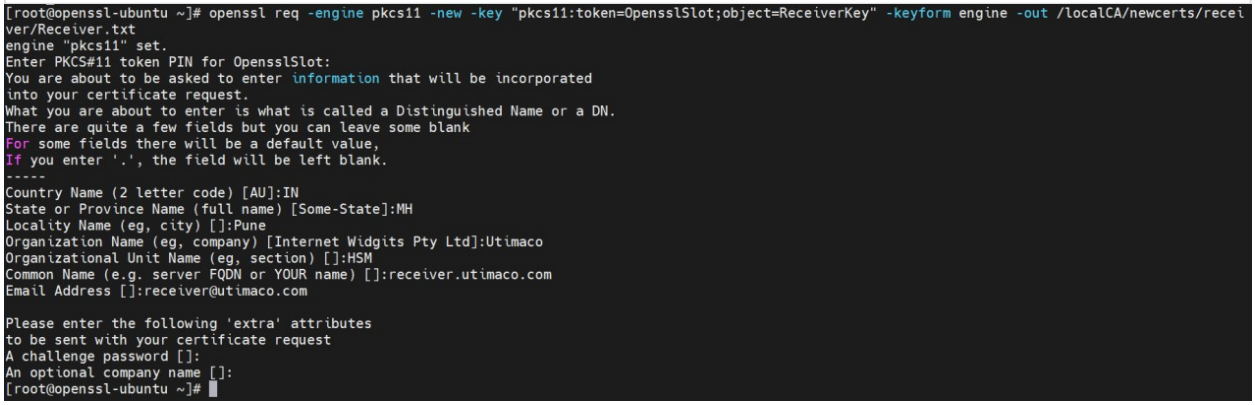

Figure 32: Receiver certificate request generation

Enter prompted value for "A challenge password" as blank.

Here, OpensslSlot is the token label and ReceiverKey is the key on the HSM. Provide Cryptouser PIN when prompted.

9. Sign the certificate request for receiver by CA

›_ Console

```
# openssl ca -engine pkcs11 -policy policy_anything -cert /localCA/newcerts/
ca.cer -in /localCA/newcerts/receiver/Receiver.txt - keyfile
"pkcs11:token=OpensslSlot;object=CAKey" -keyform engine -out/localCA/newcerts/
receiver/ReceiverSignedCertificate.cert
```

```
[root@openssl-ubuntu ~]# openssl ca -engine pkcs11 -policy policy_anything -cert /localCA/newcerts/ca.cer -in /localCA/newcerts/receiver/Receiver.txt -k
eyfile "pkcs11:token=OpensslSlot;object=CAKey" -keyform engine -out /localCA/newcerts/receiver/ReceiverSignedCertificate.cert
engine "pkcs11" set.
Using configuration from /usr/local/openssl/ssl/openssl.cnf
Enter PKCS#11 token PIN for OpensslSlot:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 2 (0x2)
        Validity
            Not Before: Aug 28 20:34:39 2022 GMT
            Not After : Aug 28 20:34:39 2023 GMT
        Subject:
            countryName               = IN
            stateOrProvinceName       = MH
            localityName              = Pune
            organizationName          = Utimaco
            organizationalUnitName    = HSM
            commonName                = receiver.utimaco.com
            emailAddress              = receiver@utimaco.com
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                E0:DB:8A:DD:08:89:D8:7B:02:2D:91:E7:CA:90:CF:8A:0F:BA:0C:BB
            X509v3 Authority Key Identifier:
                keyid:5F:51:53:AB:98:1A:6B:1C:6E:3E:38:D8:D9:32:98:C8:E6:EC:29:D6

Certificate is to be certified until Aug 28 20:34:39 2023 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
[root@openssl-ubuntu ~]#
```

Figure 33: Receiver certificate request signing by CA

Press y to sign and y again to commit.

Here, OpensslSlot is the token label and CAKey is the key on the HSM. Provide Cryptouser PIN when prompted.

4.4.6 Using OpenSSL to sign and encrypt a file

1. Go to /localCA directory and create a text file message.txt and enter any value in it.

›_ Console

```
# cd /localCA
# echo "Welcome to Utimaco Security World">message.txt
```

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

›_ Console

```
# openssl cms -engine pkcs11 -sign -in message.txt -signer /localCA/newcerts/
sender/SenderSignedCertificate.cert -inkey
"pkcs11:token=OpensslSlot;object=SenderKey" -keyform engine -out
signedmessage.txt
```

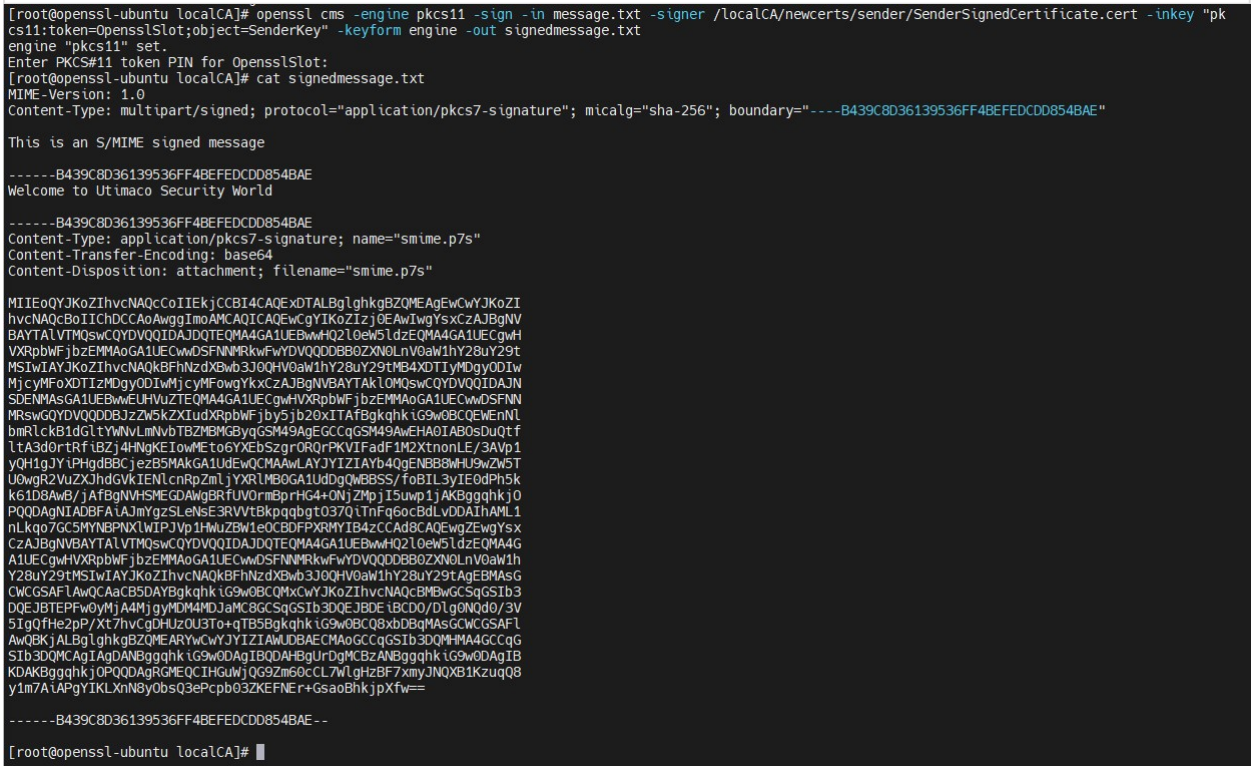

Figure 34: Openssl sign command output and content of signed message file

Here, OpensslSlot is the token label and SenderKey is the key on the HSM. Provide Cryptouser PIN when prompted.

3. Encrypt the signedmessage.txt using the receiver's public key, supplied with the receiver's certificate

›_ Console

```
# openssl cms -engine pkcs11 -encrypt -in signedmessage.txt -out
encryptedsignedmessage.txt/localCA/newcerts/receiver/
ReceiverSignedCertificate.cert
```

Figure 35: Openssl encrypt command output

4.4.7 Decrypt Sender's Message

1. Decrypt the encryptedsignedmessage.txt using the receiver's private key

›_ Console

```
# openssl cms -engine pkcs11 -decrypt -in encryptedsignedmessage.txt - inkey
"pkcs11:token=OpensslSlot;object=ReceiverKey" -keyform engine -out
decryptedsignedmessage.txt
```

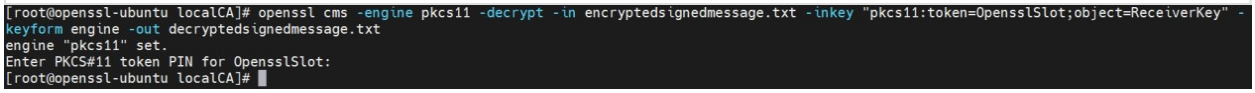

Figure 36: Openssl decrypt command output

Here, OpensslSlot is the token label and ReceiverKey is the key on the HSM. Provide Cryptouser PIN when prompted.

4.4.8 Verify the Signature

1. Verify the signature of decryptedsignedmessage.txt using the sender's certificate

›_ Console

```
# openssl cms -engine pkcs11 -verify -in decryptedsignedmessage.txt - CAfile /
localCA/newcerts/ca.cer -out originalmessage.txt/localCA/newcerts/sender/
SenderSignCertificate.cert
```

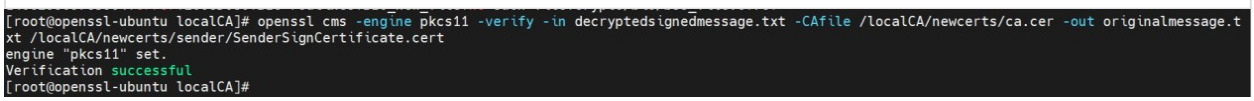

Figure 37: Openssl verify command output

2. Open the originalmessage.txt and verify the message which you have typed in the message.txt

```
[root@openssl-ubuntu localCA]# cat originalmessage.txt
Welcome to Utimaco Security World
[root@openssl-ubuntu localCA]#
```

Figure 38: Content of original message file

5 Integrating OpenSSL on Windows

5.1 PKCS#11 Configuration for Utimaco CryptoServer

On windows, as part of CryptoServer software installation, cs_pkcs11_R3.cfg will get automatically created and will be available under “C:\ProgramData\Utimaco\PKCS11_R3” folder.

Edit the cs_pkcs11_R3.cfg file and make the appropriate changes to the file.

cs_pkcs11_R3.cfg

```
[Global]
# For windows:
Logpath = C:\ProgramData\Utimaco\PKCS11_R3
# Loglevel (0 = NONE; 1 = ERROR; 2 = WARNING; 3 = INFO; 4 = TRACE)
Logging = 1
# Prevents expiring session after inactivity of 15 minutes KeepAlive = true
# Set the Device to connect with [CryptoServer]
# Device specifier Device = <HSM_IP>
```

For more information regarding the commands and command parameters, please check the Utimaco CryptoServer documentation. The device may be a CryptoServer (PCIe or LAN) device. The device line will follow one of these patterns, based on the HSM form-factor:
Device = 288@<HSM IP address> Hardware (LAN) HSM

OR

Device = /dev/cs2.0 Hardware (PCIe) HSM

To make your testing easier, it would be good to enable the PKCS#11 log file. That can be enabled by editing the Logging Loglevel. Set the LogPath and Logging Loglevel to 1. For testing, you may want to increase it to 4.

The added LogPath points to a writable directory, not to a file.

If you encounter problems, check the log file named cs_pkcs11_R3.log in the LogPath defined directory. When you are done testing, you should change Logging to 1 or 2. This will limit the logging to only critical and important messages.

5.1.1 Create SO User and Initialize a Slot:

You should initialize a slot with a custom label using p11tool2.

First using p11tool2, create the SO or Security Officer, and then using p11tool2 command, initialize the Slot 0 User as shown below.

Use PKCS#11 CryptoServer Administration Tool (CAT) to create SO user and initializing the slot.

5.2 OpenSSL and Libp11 Installation

1. Download and install OpenSSL, refer https://slproweb.com/products/Win32OpenSSL.html for windows build. Here OpenSSL is installed in C:\OpenSSL-Win64
2. Download libp11 from https://github.com/OpenSC/libp11/releases
3. Download and Install Visual Studio, refer https://visualstudio.microsoft.com/vs/older-downloads/
4. Open the Native x64 VS Command Prompt and go to the directory where libp11 is extracted
5. Run the following command to build and install libp11 with openssl

›_ Console

```
nmake /f Makefile.mak OPENSSL_DIR=c:\OpenSSL-Win64 BUILD_FOR=WIN64
```

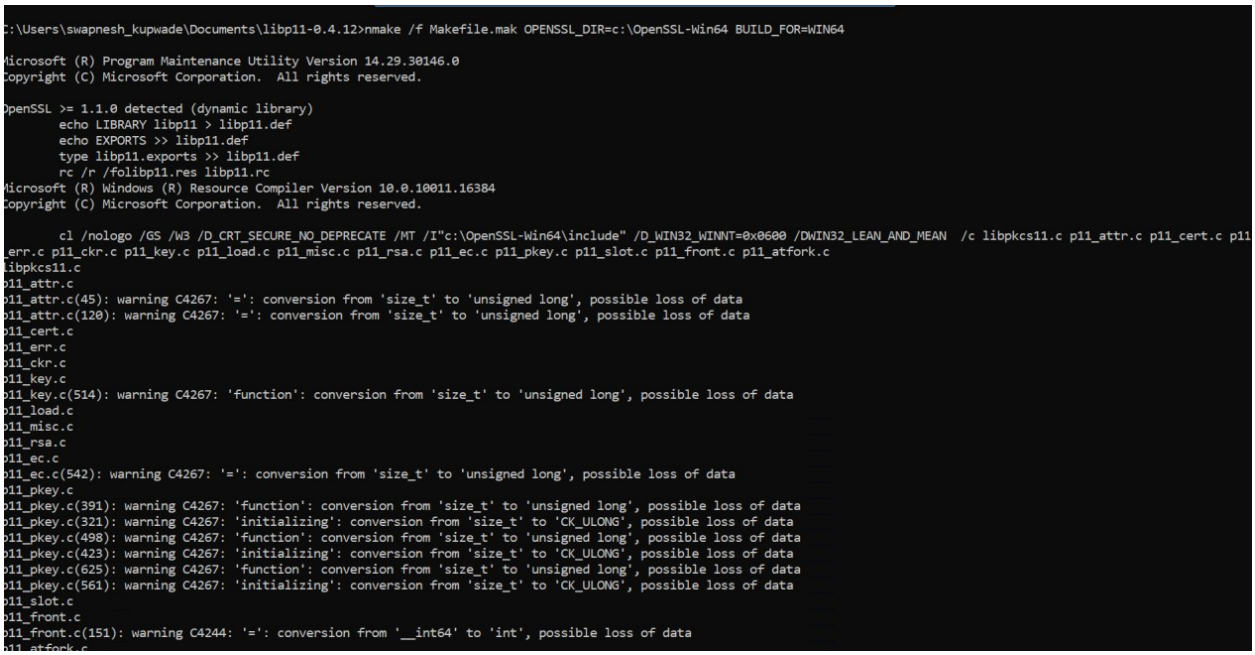

Figure 39: Output of nmake command

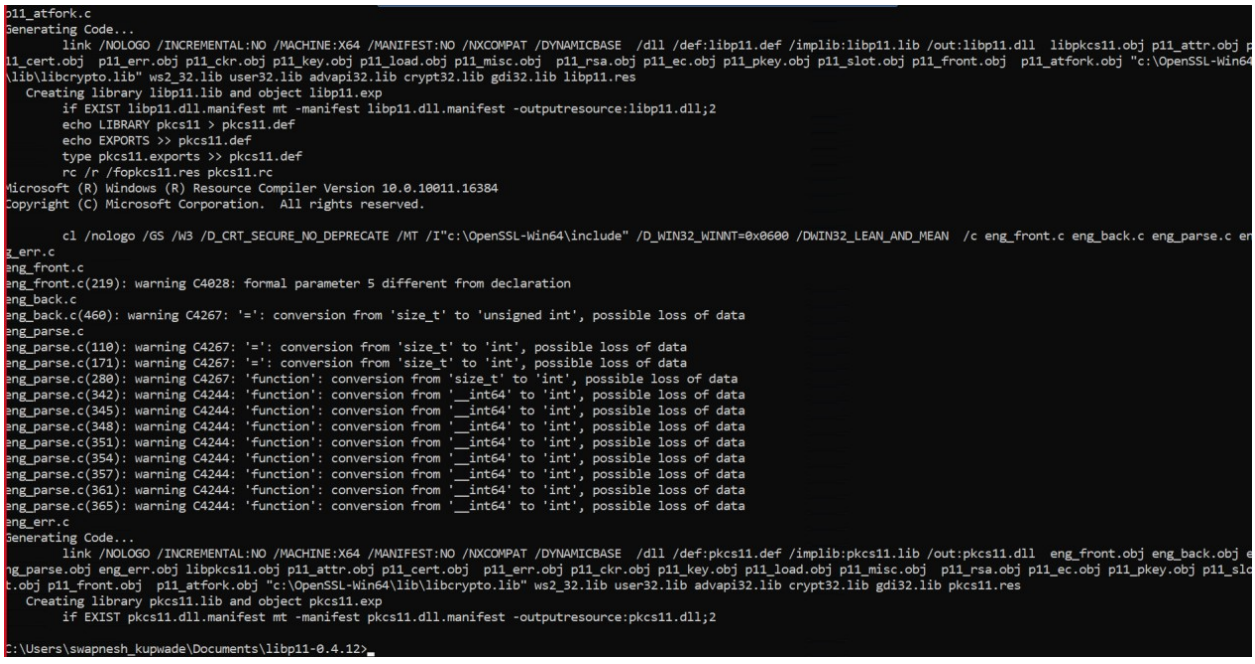

Figure 40: Output of nmake command (continued)

6. Verify pkcs11.dll is available inside <libp11>/src/ folder

5.3 Configuring OpenSSL to Use Utimaco HSM:

5.3.1 Setting up Utimaco CryptoServer library in OpenSSL Configuration File:

1. Edit openssl.cnf from C:\Program Files\Common Files\SSL\openssl.cnf and add the following line to the first line of the file.


example.file

```
openssl_conf = openssl_init
```

2. Enter the following lines under last section of openssl.cnf file


example.file

```
[openssl_init] engines=engine_section [engine_section]
pkcs11 = pkcs11_section [pkcs11_section] engine_id = pkcs11
dynamic_path = C:\\Users\\openssl-user\\Downloads\\libp11- 0.4.12\\src\
\pkcs11.dll
MODULE_PATH = C:\\Program Files\\Utimaco\\SecurityServer\\Lib\\cs_pkcs11_R3.dll
init = 0
```

Dynamic path and Module path will get changed according to the user environment.


example.file

```
[openssl_init] engines=engine_section [engine_section]
pkcs11 = pkcs11_section [pkcs11_section] engine_id = pkcs11
dynamic_path = C:\\Users\\openssl-user\\Downloads\\libp11- 0.4.12\\src\
\pkcs11.dll
MODULE_PATH = C:\\Program Files\\Utimaco\\SecurityServer\\Lib\\cs_pkcs11_R3.dll
init = 0
```

Dynamic path and Module path will get changed according to the user environment.

5.3.2 Verify PKCS#11 Engine:

Run the command below to verify whether whether the OpenSSL Engine is available or not.

›_ Console

```
# openssl engine pkcs11 -t
```

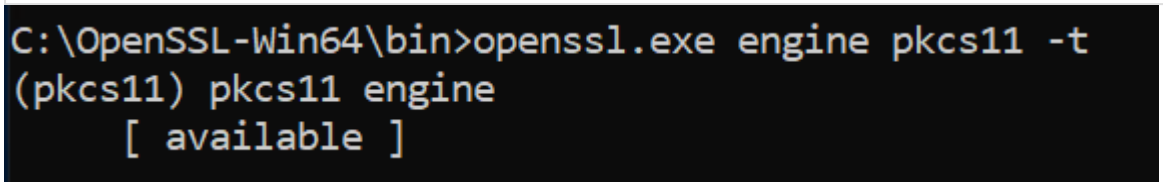

Figure 41: Verification of pkcs11 engine

5.3.3 Creating a local CA(Certificate Authority) and performing cryptographic operation with OpenSSL

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

›_ Console

```
[ CA_default ]
dir = C:\\localCA   # Where everything is kept
certs   = $dir/certs    # Where the issued certs are kept crl_dir   = $dir/crl
# Where the issued crl are kept database   = C:\\localCA\\index.txt    #
database index file. #unique_subject  = no    # Set to 'no' to allow creation of
# several certs with same subject. new_certs_dir    = C:\\localCA\\newcerts #
default place for new certs.
certificate = $dir/cacert.pem   # The CA certificate
serial  = C:\\localCA\\serial.txt   # The current serial number crlnumber   =
$dir/crlnumber # the current crl number
# must be commented out to leave a V1 CRL crl   = $dir/crl.pem  # The current
CRL
private_key = $dir/private/cakey.pem# The private key
```

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

Here, We have created directory C:\localCA and new_certs_dir= $dir\newcerts

2. Create the text files C:\localCA\index.txt and C:\localCA\serial.txt
3. Create a directory C:\localCA\newcerts
4. Open the C:\localCA\serial.txt file and write 01 at the top and click Enter. Save the file
5. Create a key pair using pkcs11tool2

For RSA

›_ Console

```
C:\Program Files\Utimaco\SecurityServer\Administration>p11tool2 slot=7
 LoginUser=123456 PubKeyAttr=CKA_LABEL="CAKey" PrvKeyAttr=CKA_LABEL="CAKey"
 GenerateKeyPair=RSA
```

This generates RSA 2048 CA private keys on the HSM

For ECDSA

›_ Console

```
C:\Program Files\Utimaco\SecurityServer\Administration>p11tool2 slot=7
 LoginUser=123456 PubKeyAttr=CKA_LABEL="CAKey" PrvKeyAttr=CKA_LABEL="CAKey"
 GenerateKeyPair=ECC
```

Once key generation is complete, then add CKA_ID for both public and private ECDSA keys using PKCS11# CryptoServer Administration tool.

6. Verify the key gets generated onto the HSM using following command

›_ Console

```
C:\Program Files\Utimaco\SecurityServer\Administration>p11tool2 slot=7
 LoginUser=<hsm_password> ListObjects
```

For RSA

```
C:\Program Files\Utimaco\SecurityServer\Administration>p11tool2 slot=7 LoginUser=123456 ListObjects


CKO_PUBLIC_KEY:

+ 1.1
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_LABEL                     = CAKey
  CKA_ID                        =



CKO_PRIVATE_KEY:

+ 2.1
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = CAKey
  CKA_ID                        =
```

Figure 42: CA RSA Key list

For ECDSA:

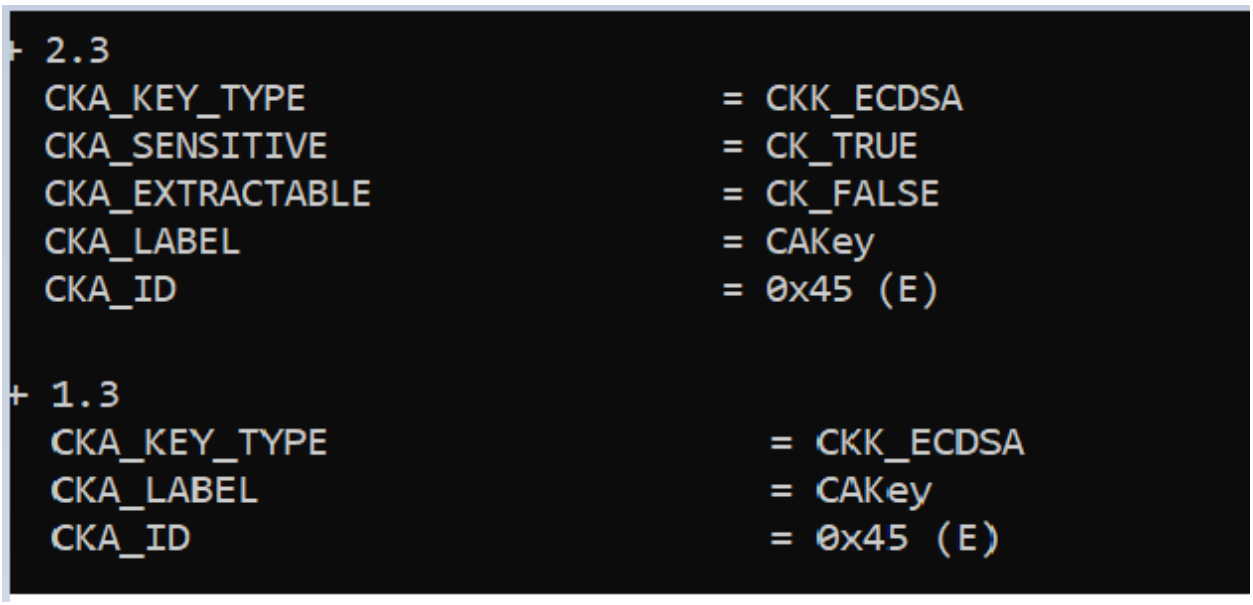

Figure 43: CA ECDSA Key list

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

›_ Console

```
C:\OpenSSL-Win64\bin>openssl req -engine pkcs11 -new -x509 -days 365 -key
"pkcs11:token=OPENSSLWINSLOT;object=CAKey" -keyform engine -out C:
\localCA\newcerts\ca.cer
```

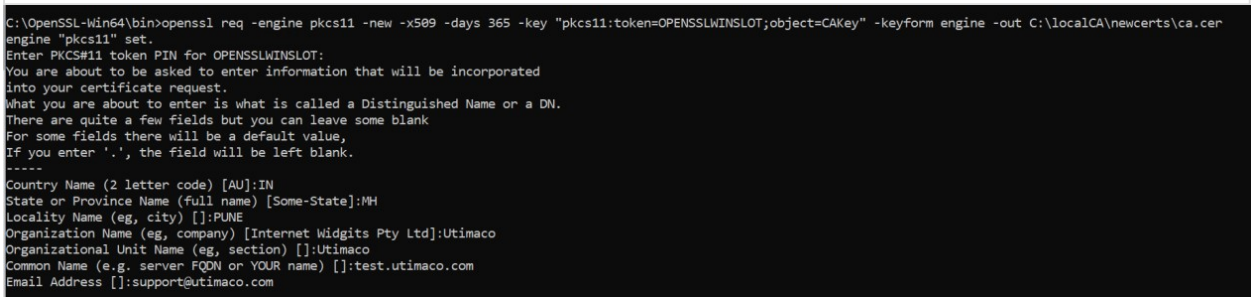

Figure 44: CA certificate generation output

Where CAKey is the object label for the CA private key on the Utimaco HSM created in Step 5 and OPENSSLWINSLOT is the token label. Provide Cryptouser PIN when prompted.

5.3.4 Generate Certificate Request for Sender and Receiver:

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

›_ Console

```
# mkdir C:\localCA\newcerts\sender
# mkdir C:\localCA\newcerts\receiver
```

2. Generate a sender key using p11tool2

For RSA

›_ Console

```
C:\Program Files\Utimaco\SecurityServer\Administration>p11tool2 slot=7
 LoginUser=123456
 PubKeyAttr=CKA_LABEL="SenderKey"PrvKeyAttr=CKA_LABEL="SenderKey"
 GenerateKeyPair=RSA
```

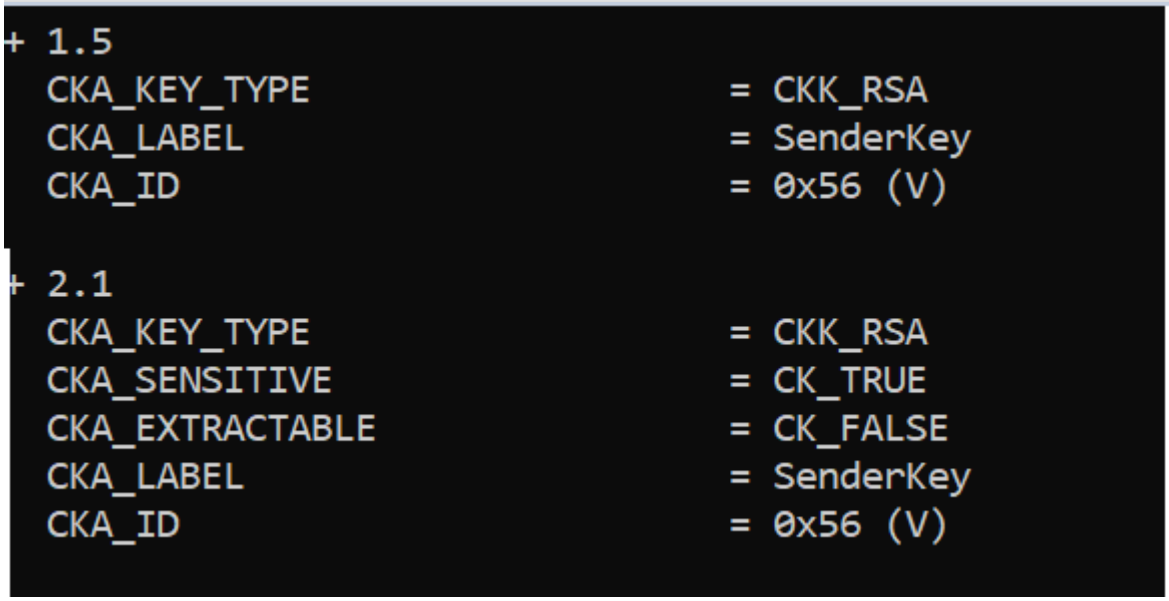

Figure 45: Sender RSA Key list

For ECDSA

›_ Console

```
C:\Program Files\Utimaco\SecurityServer\Administration>p11tool2 slot=7
 LoginUser=123456 PubKeyAttr=CKA_LABEL="SenderKey"
 PrvKeyAttr=CKA_LABEL="SenderKey" GenerateKeyPair=ECC
```

Once key generation is completed then add CKA_ID for both public and private ECDSA keys using PKCS11# CryptoServer Administration tool.

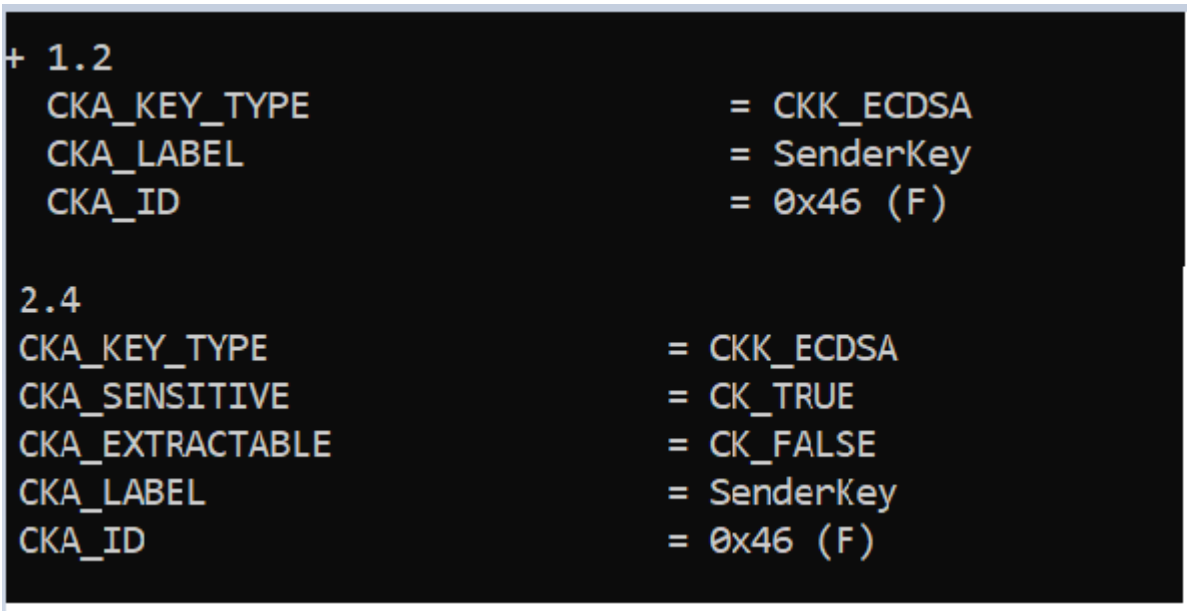

Figure 46: Sender ECDSA Key list

3. Generate a certificate request for the sender

›_ Console

```
C:\OpenSSL-Win64\bin>openssl req -engine pkcs11 -new -key
"pkcs11:token=OPENSSLWINSLOT;object=SenderKey" -keyform engine -out C:
\localCA\newcerts\sender\senderNew.txt
```

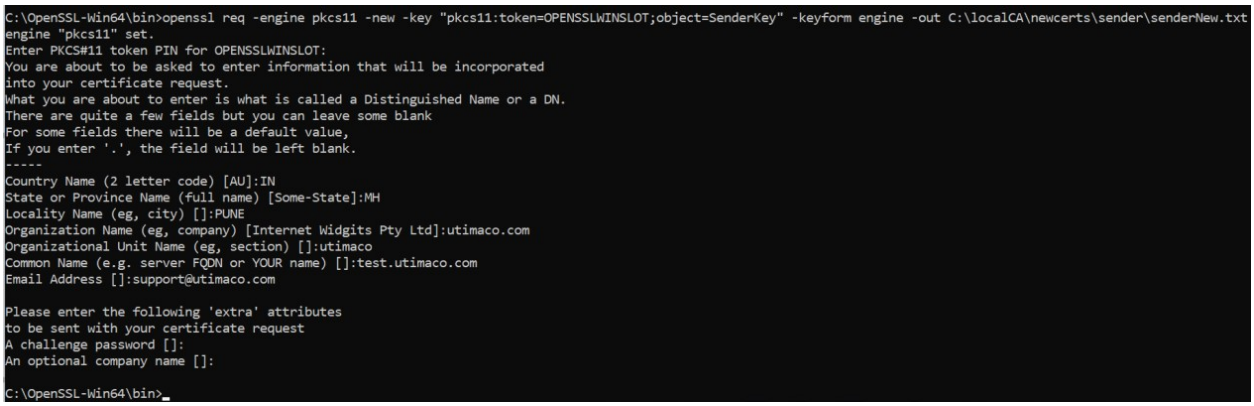

Figure 47: Sender certificate request generation output

Enter the prompted value for "A challenge password" as blank.

Here, OPENSSLWINSLOT is the token label and SenderKey is the key on the HSM. Provide Cryptouser PIN when prompted.

4. Sign the certificate request for the Sender by CA

›_ Console

```
C:\OpenSSL-Win64\bin>openssl ca -engine pkcs11 -policy policy_anything - cert C:
\localCA\newcerts\ca.cer -in C:\localCA\newcerts\sender\senderNew.txt -keyfile
"pkcs11:token=OPENSSLWINSLOT;object=CAKey" -keyform engine -out C:
\localCA\newcerts\sender\SenderSignedCertificate.cer
```

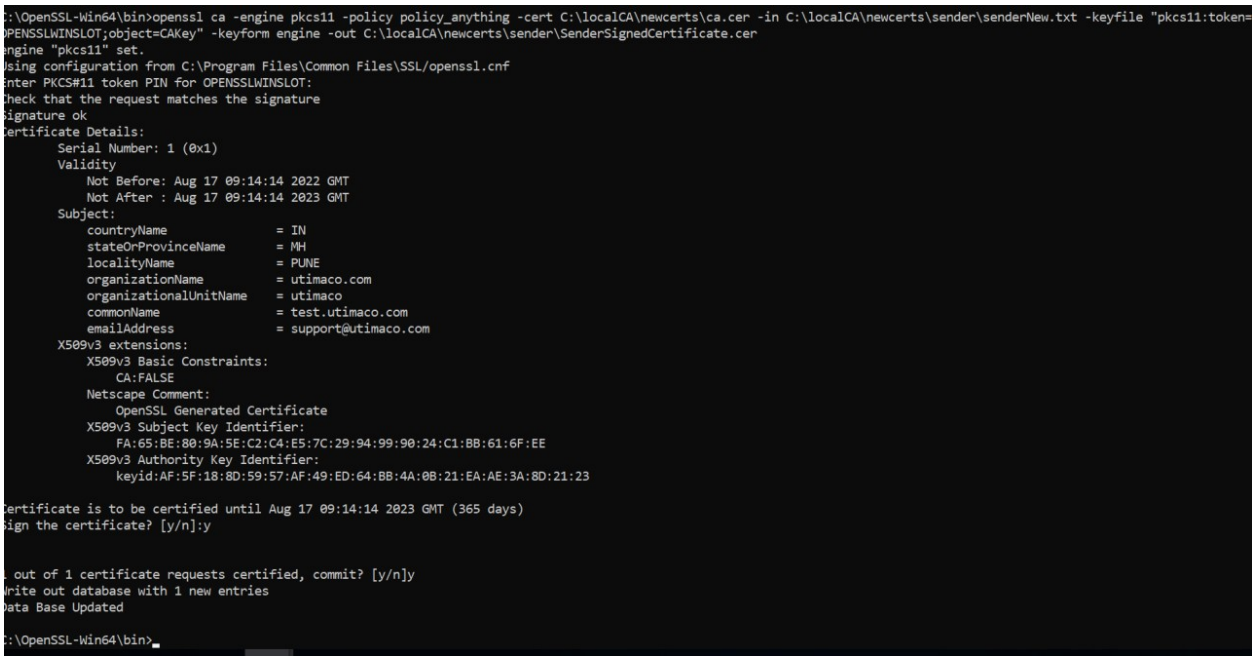

Figure 48: Sender certificate request signing by CA

Press y to sign and y again to commit.

Here, OPENSSLWINSLOT is the token label and CAKey is the key on the HSM. Provide Cryptouser PIN when prompted.

5. Generate key for the receiver using p11tool2

For RSA

›_ Console

```
C:\Program Files\Utimaco\SecurityServer\Administration> p11tool2 slot=7
 LoginUser=123456 PubKeyAttr=CKA_LABEL="ReceiverKey"
 PrvKeyAttr=CKA_LABEL="ReceiverKey" GenerateKeyPair=RSA
```

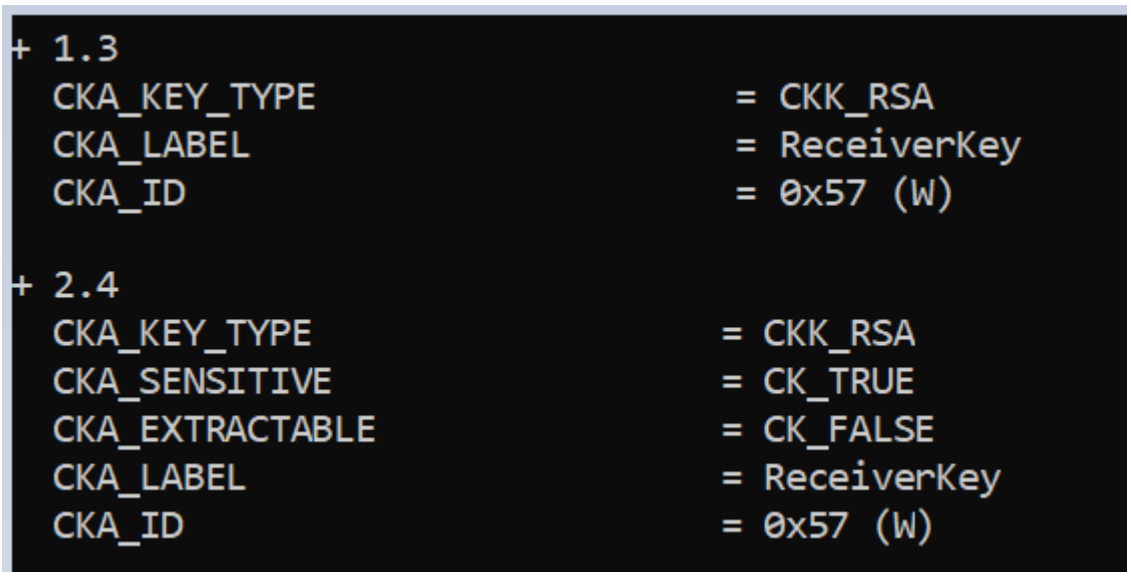

Figure 49: Receiver RSA Key list

For ECDSA

›_ Console

```
C:\Program Files\Utimaco\SecurityServer\Administration> p11tool2 slot=7
 LoginUser=123456 PubKeyAttr=CKA_LABEL="ReceiverKey"
 PrvKeyAttr=CKA_LABEL="ReceiverKey",CKA_DERIVE=CK_TRUE GenerateKeyPair=ECC
```

Once key generation is completed then add CKA_ID for both public and private ECDSA keys using PKCS11# CryptoServer Administration tool.

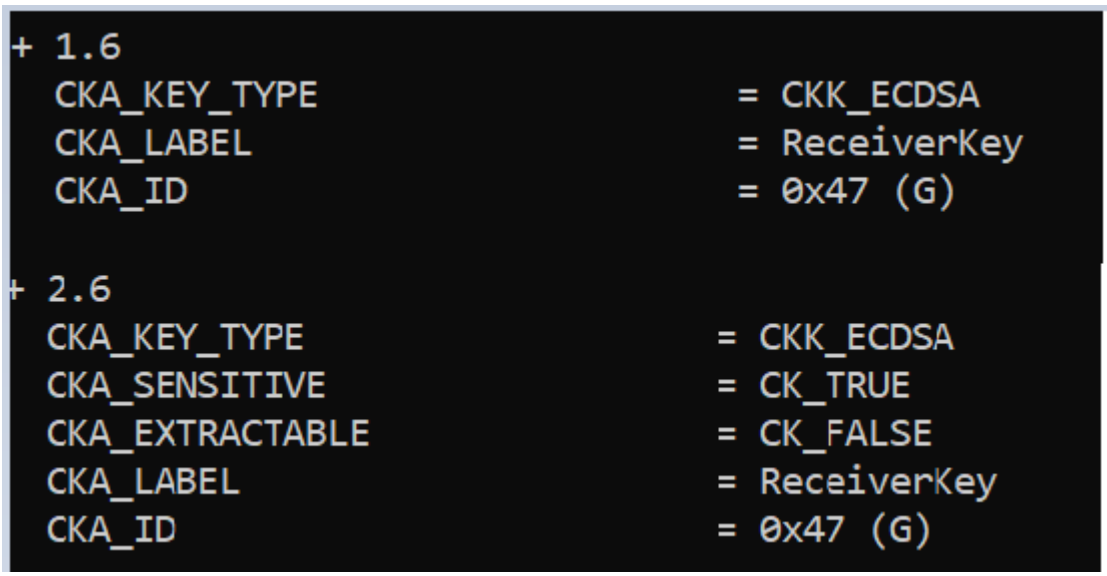

Figure 50: Receiver ECDSA Key list

6. Generate a certificate request for the receiver

›_ Console

```
C:\OpenSSL-Win64\bin>openssl req -engine pkcs11 -new -key
"pkcs11:token=OPENSSLWINSLOT;object=ReceiverKey" -keyform engine -out C:
\localCA\newcerts\receiver\ReceiverNew.txt
```

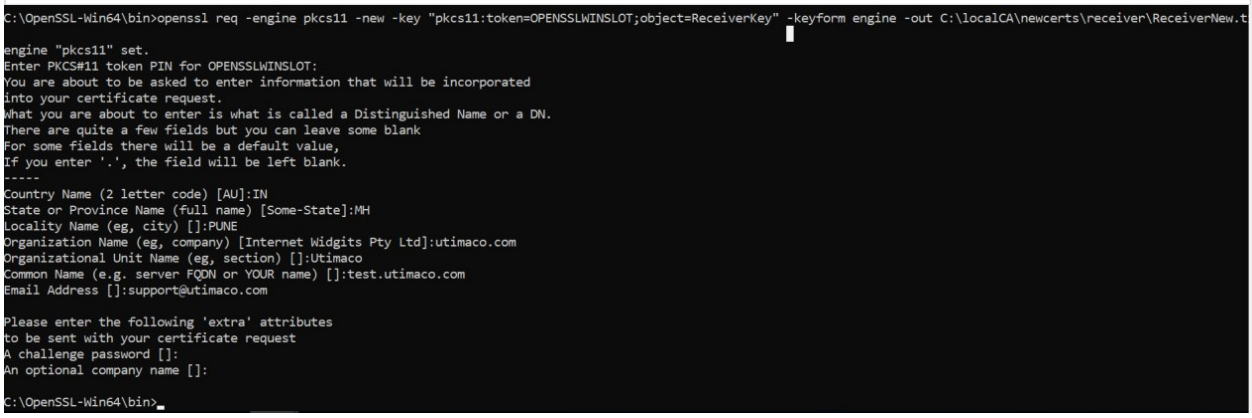

Figure 51: Receiver certificate request generation output

Here, OPENSSLWINSLOT is the token label and ReceiverKey is the key on the HSM. Provide Cryptouser PIN when prompted.

7. Sign the certificate request for the receiver by CA

›_ Console

```
C:\OpenSSL-Win64\bin>openssl ca -engine pkcs11 -policy policy_anything - cert C:
\localCA\newcerts\ca.cer -in C:\localCA\newcerts\receiver\ReceiverNew.txt
-keyfile "pkcs11:token=OPENSSLWINSLOT;object=CAKey" -keyform engine -out C:
\localCA\newcerts\receiver\receiverNew.cer
```

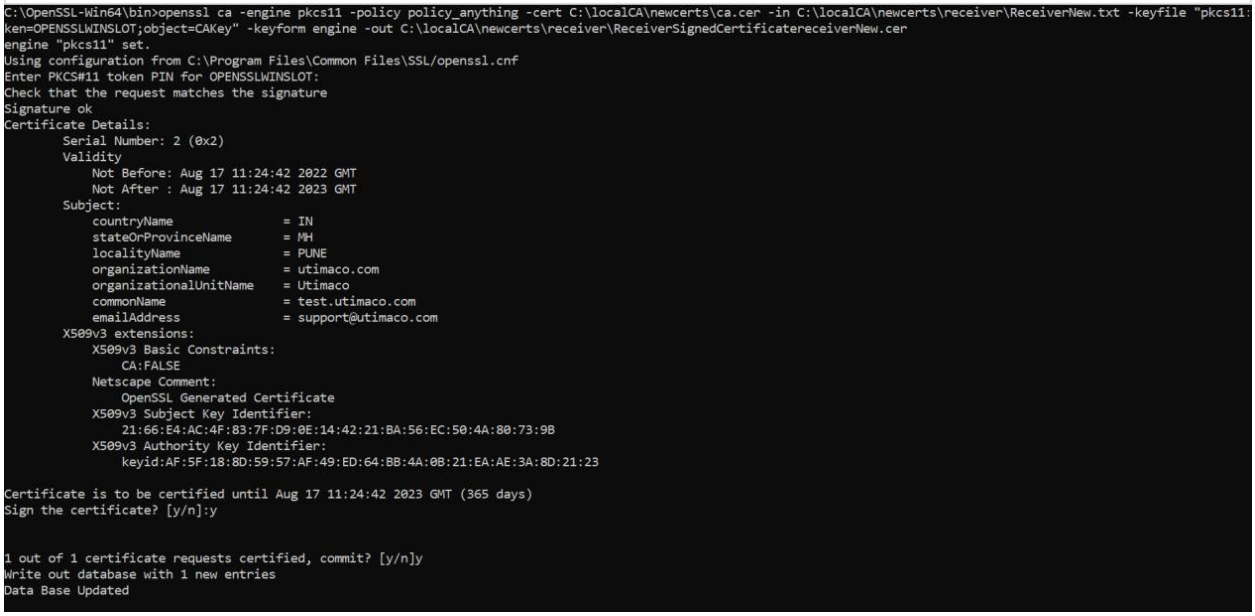

Figure 52: Receiver certificate request signing by CA

Press y to sign and y again to commit.

Here, OPENSSLWINSLOT is the token label and CAKey is the key on the HSM. Provide Cryptouser PIN when prompted.

5.3.5 Using OpenSSL to sign and encrypt a file:

1. Create a text file message.txt under C:\localCA directory and enter any value in it

›_ Console

```
Welcome to Utimaco Security World
```

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

›_ Console

```
C:\openssl cms -engine pkcs11 -sign -in C:\localCA\message.txt -signer C:
\localCA\newcerts\sender\SenderSignedCertificate.cer -inkey
"pkcs11:token=OPENSSLWINSLOT;object=SenderKey" -keyform engine -out C:
\localCA\signedmessage.txt
```

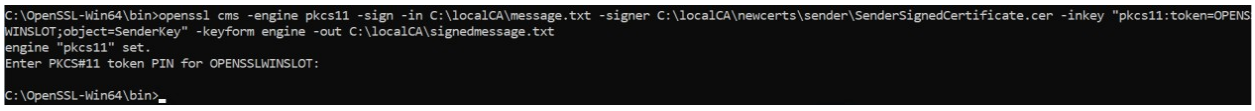

Figure 53: Openssl sign command output

Here, OPENSSLWINSLOT is the token label and SenderKey is the key on the HSM. Provide Cryptouser PIN when prompted.

3. Encrypt the signedmessage.txt using the receiver's public key, supplied with the receiver's certificate

›_ Console

```
C:\OpenSSL-Win64\bin>openssl cms -engine pkcs11 -encrypt -in C:
\localCA\signedmessage.txt -out C:\localCA\encryptedsignedmessage.txt C:
\localCA\newcerts\receiver\ReceiverSignedCertificate.cer
```

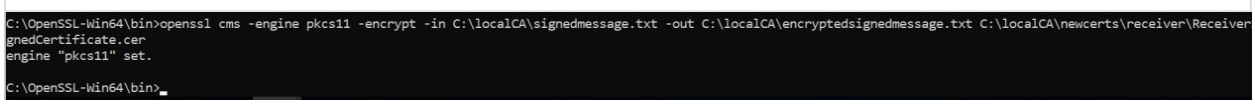

Figure 54: Openssl encrypt command output

5.3.6 Decrypt Sender's encrypted message:

Decrypt the encryptedsignedmessage.txt using the receiver's private key

›_ Console

```
C:\OpenSSL-Win64\bin>openssl cms -engine pkcs11 -decrypt -in C:\localCA\
encryptedsignedmessage.txt -inkey
"pkcs11:token=OPENSSLWINSLOT;object=ReceiverKey" -keyform engine -out C:
\localCA\decryptedsignedmessage.txt
```

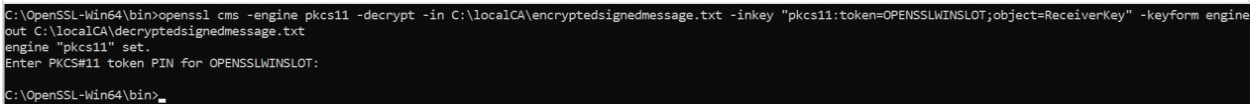

Figure 55: Openssl decrypt command output

Here, OPENSSLWINSLOT is the token label and ReceiverKey is the key on the HSM. Provide Cryptouser PIN when prompted.

5.3.7 Verify the Signature:

1. Verify the signature of decryptedsignedmessage.txt using the sender's certificate

›_ Console

```
C:\OpenSSL-Win64\bin>openssl cms -engine pkcs11 -verify -in C:
\localCA\decryptedsignedmessage.txt -CAfile C:\localCA\newcerts\ca.cer - out
originalmessage.txt C:\localCA\newcerts\sender\SenderSignCertificate.cer
```

Figure 56: Openssl verify command output

2. Open the originalmessage.txt and verify the message which you have typed in the

message.txt

This completes the Integration of OpenSSL with Utimaco HSM.

6 Troubleshooting

Error	*Diagnosis*
LoginUser= failed: 05.12.2021 23:45:45 src/p11adm_R2.c[429] p11_login: C_Login [type=1] returned Error 0x00000102 (CKR_USER_PIN_NOT_INITIALIZED)	PKCS#11 Slot is not initialized. Refer to Initialize a Slot
The CryptoServer PKCS#11 Library R3 is not initialized. Error CKR_CRYPTOKI_NOT_INITIALIZED occurred	PKCS#11 Slot is not initialized. Refer to Initialize a Slot
[root@openssl]# openssl ca -engine pkcs11 -policy policy_anything -cert /localCA/newcerts/ca.cer -in /localCA/newcerts/sender/sender.txt -keyfile openssl12345 -keyform engine -out /localCA/newcerts/sender/sender.cer engine "pkcs11" set. Using configuration from /usr/local/ssl/openssl.cnf Enter PKCS#11 token PIN for Hashicorp: ca: ./localCA/newcerts is not a directory ./localCA/newcerts: No such file or directory	Make sure to use the same path which is mentioned in Openssl.cnf file which is available under /usr/local/ssl/Openssl.cnf Check the value dir and certs under [CA_default] section use the same or add proper path to avoid the above error.

OpenSSL> engine -t dynamic -pre SO_PATH:/usr/lib64/openssl/engines/pkcs11.so -pre ID:pkcs11 -pre LIST_ADD:1 -pre LOAD -pre MODULE_PATH:/opt/utimaco/lib/libcs_pkcs11_ R3.soengine: Cannot mix flags and engine names. engine: Use -help for summary. error in engine	Install the updated libp11 library on the host machine
openssl req -engine pkcs11 -new -key 4F70656E73736C4B6579 -keyform engine -out req.pem -text -x509 -subj "CN=Utimaco" invalid engine "pkcs11" 139703122831248:error:25066067:DSO support routines:DLFCN_LOAD:could not load the shared library:dso_dlfcn.c:187:filename(/usr/lib64/openssl/engines/libpkcs11.so): libcrypto.so.1.1:	Export the below value of LD_LIBRARY_PATH and Path for Openssl to avoid the above error. export LD_LIBRARY_PATH=/usr/local/lib64 export PATH=/usr/local/bin:$PATH

cannot open shared object file: No such file or directory 139703122831248:error:25070067:DSO support routines:DSO_load:could not load the shared library:dso_lib.c:233: 139703122831248:error:260B6084:engine routines:DYNAMIC_LOAD:dso not found:eng_dyn.c:467: 139703122831248:error:2606A074:engine routines:ENGINE_by_id:no such engine:eng_list.c:392:id=pkcs11 139703122831248:error:25066067:DSO support routines:DLFCN_LOAD:could not load the shared library:dso_dlfcn.c:187:filename(libpkcs11.so): libpkcs11.so: cannot open shared object file: No such file or directory 139703122831248:error:25070067:DSO support routines:DSO_load:could not load the shared library:dso_lib.c:233: 139703122831248:error:260B6084:engine routines:DYNAMIC_LOAD:dso not found:eng_dyn.c:467: no engine specified unable to load Private Key	

Table 6: List of Error and its Diagnosis

7 Further Information

This document forms a part of the information and support which is provided by the Utimaco IS GmbH. Additional documentation can be found on the product CD in the Documentation directory.

All CryptoServer product documentation is also available at the Utimaco IS GmbH website:

http://hsm.utimaco.com

8 References

Reference	*Title/Company*	*Document No.*
[CSADMIN]	CryptoServer – csadm Manual/Utimaco IS GmbH	2009-0003
[CSTrSh]	CryptoServer Troubleshooting/Utimaco IS GmbH	M011-0008-en
[CSADMIN2]	CryptoServer_csadm_Manual_Systemadministrators.pdf	2009-0003
[CSP11Tool2]	CryptoServer_p11tool2_Manual.pdf	2012-0004
[CSPKCSM]	CryptoServer - PKCS#11 P11CAT Manual	M013-0001-en
[CSLAN5]	CryptoServerLAN_Manual_Systemadministrators.pdf	2018-0004

8.1 Contact

Utimaco IS GmbH

Krefelder Straße 220, 52070 Aachen, Germany

Phone: AMERICAS: +1-844-UTIMACO (+1 844-884-6226)

EMEA: +49 800-627-3081

APAC: +81 800-919-1301

Web: https://support.utimaco.com Email: support@utimaco.com