OpenSSL

v3.0

Integration Guide

u.trust GP HSM Se-Series

SecurityServer v4.50.0.2

utimaco®

Imprint

Copyright 2026	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	0.1.0
Date	2026-02-04
Status	PUBLISHED
Document No.	IG-2025-0031

Table of Contents

1 Introduction

This guide is part of the information and support provided by Utimaco. Additional documentation produced to support your Utimaco u.trust Anchor product can be found in the document directory of the Utimaco u.trust Anchor product bundle. All Utimaco u.trust Anchor product documentation is available from Utimaco's web site at https://utimaco.com/.

1.1 About This Guide

This guide explains how to integrate an Utimaco u.trust Anchor HSM with OpenSSL through the SecurityServer PKCS#11 provider.

1.1.1 Target Audience

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

1.1.2 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 *u.trust Anchor - csadm Manual* or [CSADM].

Table 1: Document conventions

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

This message marks the result expected after the successful execution of an instruction.

Here you find additional notes or supplementary information.

Here you find important safety information that should be followed.

1.2 Abbreviations

Abbreviation	***Meaning***
API	Application Programming Interface
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

Abbreviation	*Meaning*
LAN	Local Area Network
PCIe	PCI Express Interface
PIN	Personal Identification Number
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 key generation, as well as 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 the "engine," which can delegate cryptographic operations to other software or hardware.

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

2.2 Utimaco u.trust Anchor HSM

The u.trust Anchor is a next-generation hardware security module developed by Utimaco IS GmbH. It is a multi-tenant, physically protected, and tamper-resistant cryptographic appliance designed to perform high-assurance cryptographic operations and manage cryptographic keys securely. The u.trust General Purpose HSM is built on a modern, container-based design inspired by cloud technology. With support for up to 31 containers and multiple PKCS #11 partitions per cHSM, it ensures seamless application separation and key partitioning, making it an ideal choice for all types of cryptographic applications. It’s also upgradeable for specific use cases like blockchain and 5G, and offers flexibility for custom solutions, including proprietary algorithms and customer key derivations via the Software Development Kit.

3 Integration Requirements and Prerequisites

Ensure that 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 the 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 Anchor cHSM Version***	***Utimaco HSM***
RHEL 9	OpenSSL 3.0.1	SecurityServer 4.50.0.2	CryptoServer CSe-Series/Se-Series
Windows 2019	OpenSSL 3.0.5	SecurityServer 4.50.0.2	CryptoServer CSe-Series/Se-Series

Table 3: List of Tested Versions

3.2 Hardware and Software Requirements

Hardware	***Hardware Requirements***
Utimaco LAN HSM	CryptoServer CSe-Series/Se-Series LAN with firmware SecurityServer 4.50.0.2 or higher
Utimaco PCI-e HSM	CryptoServer CSe-Series/Se-Series PCI-e with firmware SecurityServer 4.50.0.2 or higher

Table 4: List of Hardware Requirements

Software	*Software Requirements*
OpenSSL	OpenSSL 3.0.1 and 3.0.5
Libp11 (Linux)	0.4.12
Libp11 (Windows)	0.4.12
HSM Interface	SecurityServer PKCS#11 Provider

Table 5: List of Software Requirements

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

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

3.3 Prerequisites

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

- Operating system listed in Tested Versions
- SecurityServer listed in Tested Versions
- SecurityServer Default Admin should be replaced with a new admin user
- SecurityServer is setup and configured. Refer the SecurityServer documentations to setup the HSM
- Admin access is required to install the software
- PKCS#11 library is setup and configured as per the environment. Refer the SecurityServer documentations to setup and configure the PKCS#11 library for SecurityServer
- Familiarize yourself with the OpenSSL documents and setup process

4 Integrating OpenSSL 3.0 with SecurityServer on Linux

4.1 Installing and Configuring Utimaco SecurityServer Software (Linux)

4.1.1 Download and Install Utimaco Software (Linux)

If you have not already done so, please create and request an Utimaco Support Portal Account. This will allow you to download the software components needed for this installation.

1. Copy the downloaded software at the appropriate location on the OpenSSL Server
2. Create utimaco folder under /opt directory and further create 2 directories

/opt/utimaco/bin and /opt/utimaco/lib

›_ Console

```
# mkdir -p /opt/utimaco/bin # mkdir -p /opt/utimaco/lib
```

3. Copy pkcs11 library file libcs_pkcs11_R3.so from Utimaco SecurityServer software to the /opt/utimaco/lib directory

›_ Console

```
# cp ~/path_to_application_folder/lib/libcs_pkcs11_R3.so /opt/utimaco/lib
```

4. Copy the csadm and p11tool2 files from Utimaco SecurityServer software to

/opt/utimaco/bin directory and make both the files executable

›_ Console

```
# cd ~/path_to_application_folder

# cp csadm p11tool2 /opt/utimaco/bin

# chmod +x /opt/utimaco/bin/csadm /opt/utimaco/bin/p11tool2
```

4.1.2 Create SO User and Initialize a Slot (Linux)

You must 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=ask

# ./p11tool2 slot=<slot_no> LoginSO=ask InitPin=ask
```

4.1.3 SecurityServer PKCS#11 Configuration (Linux)

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 Keepalive = false

# 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 SecurityServer documentation. The device may be a SecurityServer (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.

4.2 Installing OpenSSL (Linux)

If you are using existing or pre-installed openssl then skip this section.

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

›_ Console

```
# dnf update
```

2. Install openssl

›_ Console

```
# dnf install openssl
```

3. Verify the version of openssl

```
[root@Openssl-RHEL9 /]# openssl version -a
OpenSSL 3.0.1 14 Dec 2021 (Library: OpenSSL 3.0.1 14 Dec 2021)
built on: Wed Oct 26 00:00:00 2022 UTC
platform: linux-x86_64
options:  bn(64,64)
compiler: gcc -fPIC -pthread -m64 -Wa,--noexecstack -Wall -O3 -O2 -flto=auto -ffat-lto-objects -fexceptions -g -grecord-gcc-swit
ches -pipe -Wall -Werror=format-security -Wp,-D_FORTIFY_SOURCE=2 -Wp,-D_GLIBCXX_ASSERTIONS -specs=/usr/lib/rpm/redhat/redhat-har
dened-cc1 -fstack-protector-strong -specs=/usr/lib/rpm/redhat/redhat-annobin-cc1 -m64 -march=x86-64-v2 -mtune=generic -fasynchro
nous-unwind-tables -fstack-clash-protection -fcf-protection -Wa,--noexecstack -Wa,--generate-missing-build-notes=yes -specs=/usr
/lib/rpm/redhat/redhat-hardened-ld -specs=/usr/lib/rpm/redhat/redhat-annobin-cc1 -DOPENSSL_USE_NODELETE -DL_ENDIAN -DOPENSSL_PIC
 -DOPENSSL_BUILDING_OPENSSL -DZLIB -DNDEBUG -DPURIFY -DDEVRANDOM="\"/dev/urandom\"" -DREDHAT_FIPS_VERSION="\"3.0.1-ff3fa868c239d
690\"" -DSYSTEM_CIPHERS_FILE="/etc/crypto-policies/back-ends/openssl.config"
OPENSSLDIR: "/etc/pki/tls"
ENGINESDIR: "/usr/lib64/engines-3"
MODULESDIR: "/usr/lib64/ossl-modules"
Seeding source: os-specific
CPUINFO: OPENSSL_ia32cap=0xfeda32034f8bffff:0xd09e2fb9
[root@Openssl-RHEL9 /]#
```

Figure 1: OpenSSL version output

4.3 Installing Libp11 (Linux)

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 libp11using the following commands

›_ Console

```
# cd libp11-0.4.12

# ./configure prefix="/usr/local/libp11/" # make

# make install

# export LD_LIBRARY_PATH= /usr/local/libp11/lib/:$LD_LIBRARY_PATH
```

```
[root@Openssl-RHEL9 libp11-0.4.12]# ./configure 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 ... /bin/install -c
checking whether build environment is sane ... yes
checking for a thread-safe mkdir -p ... /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 ... /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 ... /bin/grep
checking for egrep ... /bin/grep -E
checking for ANSI C header files ... yes
checking for sys/types.h ... yes
checking for sys/stat.h ... yes
checking for stdlib.h ... yes
checking for string.h ... yes
```

```
config.status: creating Makefile
config.status: creating src/Makefile
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:       /usr/lib64/engines-3
Default PKCS11 module:  /usr/lib64/p11-kit-proxy.so
API doc support:        no

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

OPENSSL_CFLAGS:
OPENSSL_LIBS:           -lcrypto

[root@Openssl-RHEL9 libp11-0.4.12]#
```

Figure 2: Output of configure command for libp11

4.4 Configuring OpenSSL to Use Utimaco SecurityServer (Linux)

4.4.1 Setting up Utimaco SecurityServer Library in OpenSSL Configuration File (Linux)

1. Open the file /etc/pki/tls/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/lib64/engines-3/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 (Linux)

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

›_ Console

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

```
[root@Openssl-RHEL9 libp11-0.4.12]# openssl engine pkcs11 -t
(pkcs11) pkcs11 engine
     [ available ]
[root@Openssl-RHEL9 libp11-0.4.12]#
```

Figure 3: Verification of pkcs11 engine

4.4.3 Test OpenSSL Functionalities with Utimaco SecurityServer (Linux)

4.4.3.1 Testing with RSA Key (Linux)

1. Generate the RSA key using p11tool2

›_ Console

```
# p11tool2 slot=2 LoginUser=12345678 PubKeyAttr=CKA_LABEL="CertKey"
 PrvKeyAttr=CKA_LABEL="CertKey" GenerateKeyPair=RSA
```

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

›_ Console

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

Example:

›_ Console

```
[root@Openssl-RHEL9 bin]# ./p11tool2 slot=2 LoginUser=ask ListObjects Enter
normal user PIN:

CKO_PUBLIC_KEY:

+ 1.1

CKA_KEY_TYPE    = CKK_RSA

CKA_UNIQUE_ID   = F9818668-8663-497F-B41F-D47A3A069970

CKA_LABEL   = CertKey

CKA_ID  =

CKO_PRIVATE_KEY:

+ 2.1

CKA_KEY_TYPE    = CKK_RSA

CKA_UNIQUE_ID   = 1DF1BDD0-A073-4743-96CB-DCFB986B9F0A

CKA_SENSITIVE   = CK_TRUE

CKA_EXTRACTABLE = CK_FALSE

CKA_LABEL   = CertKey

CKA_ID  =

[root@Openssl-RHEL9 bin]#
```

3. Generate a certificate signing request (CSR)

›_ Console

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

Here SSLCert is the token label and CertKey is the key on the HSM. Provide Cryptouser PIN when prompted.

```
[root@Openssl-RHEL9 home]# openssl req -engine pkcs11 -new -key "pkcs11:token=SSLCert;object=CertKey" -keyform engine -out TestR
SACSR.csr
Engine "pkcs11" set.
Enter PKCS#11 token PIN for SSLCert:
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) [XX]:IN
State or Province Name (full name) []:MH
Locality Name (eg, city) [Default City]:Pune
Organization Name (eg, company) [Default Company Ltd]:Utimaco
Organizational Unit Name (eg, section) []:BU
Common Name (eg, your name or your server's hostname) []:SSL
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
[root@Openssl-RHEL9 home]#
```

Figure 4: Certificate request output

```
[root@Openssl-RHEL9 home]# cat TestRSACSR.csr
-----BEGIN CERTIFICATE REQUEST-----
MIICmzCCAYMCAQAwVjELMAkGA1UEBhMCSU4xCzAJBgNVBAgMAk1IMQ0wCwYDVQQH
DARQdW5lMRAwDgYDVQQKDAdVdGltYWNvMQswCQYDVQQLDAJCVTEMMAoGA1UEAwwD
U1NMMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEApTrp7nZsitAFtXj8
RLGLLH00FAFnz1Jp3h0J6Cja1gz+XVonoV+m0Ro61FzLOKFjSmfK3CgvaKX8o2W4
PgPOsu2rIF9vgwS/QK64ruEFInyeJWsyWhKkWP/XxQ3mx1f9YuhelB/UEpmP6iJG
5FyHQ2tRUsRnQ6YTOu3hzcvrN5rIhZSAsMYAwjZARm1iNYUxH3uRRnvXYswtngpi
hWaUZynA1KwJ0CEH3tX67bCg83JbnTIpCU9WXy/I8uKdUH04jAJDh1Dvzs0pGa/B
Wi2amGK8JXCLmF2Rcd6zhTUSh7U7le5fJRloN5ilEgvowTKa5B6JQxVxQ5gMmqv6
GCeKQwIDAQABoAAwDQYJKoZIhvcNAQELBQADggEBAFNpODr75t06nG6e44GGCFtE
Si01G4qWhngFGjrQXng7fWYcfWOdo8ydkooWYPR5RVzqOQUABSyZKDCFQfdbwabV
n2QJy3S8Bp6r8idAxTfU7E0rrr75meOX2Hz3gs3940pcsESLF4QwrFZ6iD7/5NUk
kzH7aD+fDGfB1X2Ia6oJXaDZHYJIOKszoBWHTn926A/vv3Asvz49AL+miOs+ylMk
Tmo3mtFSwNuqRkOHnaINrCSiT5mu0zUrJzfCKFJ2SbZmUbtkRgSiCMk9acussO5T
mUQgKG8OzKuIlrrUY79k0v12Nax9MKVgUuSJEJwYruuCMnD0ORwvwMzEfJft5Cw=
-----END CERTIFICATE REQUEST-----
[root@Openssl-RHEL9 home]#
```

Figure 5: 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=SSLCert;object=CertKey" -keyform engine -out TestRSA.cert
```

```
[root@Openssl-RHEL9 home]# openssl req -engine pkcs11 -new -x509 -days 365 -key "pkcs11:token=SSLCert;object=CertKey" -keyform e
ngine -out TestRSA.cert
Engine "pkcs11" set.
Enter PKCS#11 token PIN for SSLCert:
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) [XX]:CN
State or Province Name (full name) []:MH
Locality Name (eg, city) [Default City]:NSK
Organization Name (eg, company) [Default Company Ltd]:Utimaco
Organizational Unit Name (eg, section) []:BU
Common Name (eg, your name or your server's hostname) []:SSL
Email Address []:
[root@Openssl-RHEL9 home]#
```

Here SSLCert is the token label and CertKey is the key on the HSM. Provide Cryptouser PIN when prompted.

Figure 6: Self signed certificate generation output

```
[root@Openssl-RHEL9 home]# cat TestRSA.cert
-----BEGIN CERTIFICATE-----
MIIDizCCAnOgAwIBAgIUK622UzKgI2Qy3szVyEdSOk4F5l8wDQYJKoZIhvcNAQEL
BQAwVTELMAkGA1UEBhMCQ04xCzAJBgNVBAgMAk1IMQwwCgYDVQQHDANOU0sxEDAO
BgNVBAoMB1V0aW1hY28xCzAJBgNVBAsMAkJVMQwwCgYDVQQDDANTU0wwHhcNMjMw
MTA5MTEyMjU3WhcNMjQwMTA5MTEyMjU3WjBVMQswCQYDVQQGEwJDTjELMAkGA1UE
CAwCTUgxDDAKBgNVBAcMA05TSzEQMA4GA1UECgwHVXRpbWFjbzELMAkGA1UECwwC
QlUxDDAKBgNVBAMMA1NTTDCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEB
AKU66e52bIrQBbV4/ESxiyx9NBQBZ89Sad4dCego2tYM/l1aJ6FfptEaOtRcyzih
Y0pnytwoL2il/KNluD4DzrLtqyBfb4MEv0CuuK7hBSJ8niVrMloSpFj/18UN5sdX
/WLoXpQf1BKZj+oiRuRch0NrUVLEZ0OmEzrt4c3L6zeayIWUgLDGAMI2QEZtYjWF
MR97kUZ712LMLZ4KYoVmlGcpwNSsCdAhB97V+u2woPNyW50yKQlPVl8vyPLinVB9
OIwCQ4dQ787NKRmvwVotmphivCVwi5hdkXHes4U1Eoe1O5XuXyUZaDeYpRIL6MEy
muQeiUMVcUOYDJqr+hgnikMCAwEAAaNTMFEwHQYDVR0OBBYEFDb730xQO5fZDWmt
/ebjU7BN3esUMB8GA1UdIwQYMBaAFDb730xQO5fZDWmt/ebjU7BN3esUMA8GA1Ud
EwEB/wQFMAMBAf8wDQYJKoZIhvcNAQELBQADggEBAJPD8CE7LJ4PItVfPTEVz57o
D+Nr3RR5kuS6s0sQuGt+oQBW9L2F912VAjV0ruEvGcgwEbCrKTnIo786UCPUOEA2
9MetJVzQVyoOXV7vYWg+xrKmYhVUsHslcLMs6Rt7Eba1HbwQB0M4UI6cnVedB+4B
dFq7Ga8AAeiwti48TMxxBiPZiJVaIFDyD0doA776PWaD2lczsrd/iw74jxt59jvr
K+mrMbxd+AnRbhSwSUHvgke4ROftF+zkS35FFAZx39Rf5AzHhCu50GnAmsRaofBb
VxmAqto5F1iG9LRufGgqNaEuE2wOPh/FImNCnNcjBRrneEbIzKa6iaukb+Glr6g=
-----END CERTIFICATE-----
[root@Openssl-RHEL9 home]#
```

Figure 7: Content of self-signed certificate file

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

›_ Console

```
# touch message.txt
```

```
[root@Openssl-RHEL9 libp11-0.4.12]# cat message.txt
Welcome to Utimaco Team!!!
[root@Openssl-RHEL9 libp11-0.4.12]#
```

Figure 8: Content of message.txt

5. Sign the message file

›_ Console

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

```
[root@Openssl-RHEL9 home]# openssl cms -engine pkcs11 -sign -in message.txt -signer TestRSA.cert -inkey "pkcs11:token=SSLCert;ob
ject=CertKey" -keyform engine -out signedRSAmessage.txt
Engine "pkcs11" set.
Enter PKCS#11 token PIN for SSLCert:
[root@Openssl-RHEL9 home]#
```

Here SSLCert is the token label and CertKey is the key on the HSM. Provide Cryptouser PIN when prompted.

```
yKO/OlAj1DhANvTHrSVc0FcqDl1e72FoPsaypmIVVLB7JXCzLOkbexG2tR28EAdD
OFCOnJ1XnQfuAXRauxmvAAHosLYuPEzMcQYj2YiVWiBQ8g9HaAO++j1mg9pXM7K3
f4sO+I8befY76yvpqzG8XfgJ0W4UsElB74JHuETn7Rfs5Et+RRQGcd/UX+QMx4Qr
udBpwJrEWqHwW1cZgKraORdYhvS0bnxoKjWhLhNsDj4fxSJjQpzXIwUa53hGyMym
uomrpG/hpa+oMYICfTCCAnkCAQEwbTBVMQswCQYDVQQGEwJDTjELMAkGA1UECAwC
TUgxDDAKBgNVBAcMA05TSzEQMA4GA1UECgwHVXRpbWFjbzELMAkGA1UECwwCQlUx
DDAKBgNVBAMMA1NTTAIUK622UzKgI2Qy3szVyEdSOk4F5l8wCwYJYIZIAWUDBAIB
oIHkMBgGCSqGSIb3DQEJAzELBgkqhkiG9w0BBwEwHAYJKoZIhvcNAQkFMQ8XDTIz
MDEwOTExMjUyMFowLwYJKoZIhvcNAQkEMSIEIFGjJMKehCx8wScjC9442jsx9BzW
uHgCwrazEDnlaPYkMHkGCSqGSIb3DQEJDzFsMGowCwYJYIZIAWUDBAEqMAsGCWCG
SAFlAwQBFjALBglghkgBZQMEAQIwCgYIKoZIhvcNAwcwDgYIKoZIhvcNAwICAgCA
MA0GCCqGSIb3DQMCAgFAMAcGBSsOAwIHMA0GCCqGSIb3DQMCAgEoMA0GCSqGSIb3
DQEBAQUABIIBAEJrq//MMVLwJbu+HbCh6Gks1GtbFuyDMDUz0GigE1jxBjtCM6Sl
j8nbE3qrk5h4/RWB1M222cpt7YzsoNZkOGyNEeBLyF20Jdx9ujUWdbkQg9sdfEEV
uTaTte+yLTl3zpCD1zr3qT6ENTcujojyeGrhOkgde5oI3rev3uM17FVaA69cdJCy
L1aPvffnJJmGU5hhsfr2rOq8l+kw/6ovJrGEwU/DEUUZ5fDmx/3RapOQ8inDpkAP
8ipMQzzQBQEzUNPPAFRhBMFz0Kb8QLn+PngohPOp34XVRvFUI8cTfAPY2Zr4tnJJ
m6imY0wthBXRROXxCh17srU04/VMnu9kvy4=

------7CED1F97C9932FB9776F3313F117664D--

[root@Openssl-RHEL9 home]#
```

```
[root@Openssl-RHEL9 home]# cat signedRSAmessage.txt
MIME-Version: 1.0
Content-Type: multipart/signed; protocol="application/pkcs7-signature"; micalg="sha-256"; boundary="----7CED1F97C9932FB9776F3313
F117664D"

This is an S/MIME signed message

------7CED1F97C9932FB9776F3313F117664D
Welcome to Utimaco !!!

------7CED1F97C9932FB9776F3313F117664D
Content-Type: application/pkcs7-signature; name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"

MIIGRgYJKoZIhvcNAQcCoIIGNzCCBjMCAQExDTALBglghkgBZQMEAgEwCwYJKoZI
hvcNAQcBoIIDjzCCA4swggJzoAMCAQICFCuttlMyoCNkMt7M1chHUjpOBeZfMA0G
CSqGSIb3DQEBCwUAMFUxCzAJBgNVBAYTAkNOMQswCQYDVQQIDAJNSDEMMAoGA1UE
BwwDTlNLMRAwDgYDVQQKDAdVdGltYWNvMQswCQYDVQQLDAJCVTEMMAoGA1UEAwwD
U1NMMB4XDTIzMDEwOTExMjI1N1oXDTI0MDEwOTExMjI1N1owVTELMAkGA1UEBhMC
Q04xCzAJBgNVBAgMAk1IMQwwCgYDVQQHDANOU0sxEDAOBgNVBAoMB1V0aW1hY28x
CzAJBgNVBAsMAkJVMQwwCgYDVQQDDANTU0wwggEiMA0GCSqGSIb3DQEBAQUAA4IB
DwAwggEKAoIBAQClOunudmyK0AW1ePxEsYssfTQUAWfPUmneHQnoKNrWDP5dWieh
X6bRGjrUXMs4oWNKZ8rcKC9opfyjZbg+A86y7asgX2+DBL9Arriu4QUifJ4lazJa
EqRY/9fFDebHV/1i6F6UH9QSmY/qIkbkXIdDa1FSxGdDphM67eHNy+s3msiFlICw
xgDCNkBGbWI1hTEfe5FGe9dizC2eCmKFZpRnKcDUrAnQIQfe1frtsKDzcludMikJ
T1ZfL8jy4p1QfTiMAkOHUO/OzSkZr8FaLZqYYrwlcIuYXZFx3rOFNRKHtTuV7l8l
GWg3mKUSC+jBMprkHolDFXFDmAyaq/oYJ4pDAgMBAAGjUzBRMB0GA1UdDgQWBBQ2
+99MUDuX2Q1prf3m41OwTd3rFDAfBgNVHSMEGDAWgBQ2+99MUDuX2Q1prf3m41Ow
Td3rFDAPBgNVHRMBAf8EBTADAQH/MA0GCSqGSIb3DQEBCwUAA4IBAQCTw/AhOyye
DyLVXz0xFc+e6A/ja90UeZLkurNLELhrfqEAVvS9hfddlQI1dK7hLxnIMBGwqyk5
```

Figure 9: Content of signed message file

6. Encrypt the signed message file

›_ Console

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

```
[root@Openssl-RHEL9 home]# 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

MIIMbAYJKoZIhvcNAQcDoIIMXTCCDFkCAQAxggGJMIIBhQIBADBtMFUxCzAJBgNV
BAYTAkNOMQswCQYDVQQIDAJNSDEMMAoGA1UEBwwDTlNLMRAwDgYDVQQKDAdVdGlt
YWNvMQswCQYDVQQLDAJCVTEMMAoGA1UEAwwDU1NMAhQrrbZTMqAjZDLezNXIR1I6
TgXmXzANBgkqhkiG9w0BAQEFAASCAQBXDy1XhWkSRO4/PQudSJ6aV8+IX9NFaTya
JTIztMypXQ0YPp90g5Nz7tv/T2I9FQCCKhE8Mu1n2hXRWmQVuDHFGV5q4BqWcrxs
GnMzhxFIAY/oECGQIDV2mkZdAFvSoKRVawFtiwKdvu7dBces7eiojUsZt6AHLNAw
SmwtbAVuW95rEANCGA7ciC/8QeRKpqd4GKlW2OW2Cj15KVbmpxrCX6GUXb0czBi8
k2TLWA1Bud/RQEeWsyNVkC17CR7Nn/aRkbz2Bv+BW6CbmEd1vdExzDfRQf+bc2OB
e5VeBij7CJFmmb3GxXbhcZrKaho64a+JJkhESrpVEpN+uUlMs0WHMIIKxQYJKoZI
hvcNAQcBMBQGCCqGSIb3DQMHBAjIHI8yjsmpcoCCCqB7v98E26Nihdzm6AmPZIqY
+sPXBqiNCL8cgTLbub5aFeYRLQu6yh8LfiFwCIYSgXLVI32MgiD6HzOZmSv5dUHG
6sf4HufJweCUXZExuHboHHgXT0Hq//Dorq354QH5tLLDzWaKlNkwQpFZ2ciW7JS4
uBvHSNj480G1awv0S6AEsm2Jaf5uqXjQjypIblUdeZic7nj3wBHAzKoRnKDX54Q1
FnZWDnD0c2y1pzh8Jq3GrY33hk6Hvw6Z48dsX1UXccRtCgfcSFu2nbvlkd8MbUD5
lYDpJOF8pEVntqZB5/9TBF2Dr9Po44eadbRMJ/ojjE1cFodvaINzPx24s53cIPoG
kZbgmB8w34SaxJfAkYpYDSpHc1gAipZ6zjhKVR9H69UZsNEs3S7MiMSt7cbW5lkj
epjN5NjG1rSrmObHDL6iRgke5h8BtLFF+7+URU+1CESRhednuAWnPcP47rwBeoLh
7y7orkWCeYsbPzOWU0skpkF1PEL9uYJxEaKJhXF7Qt9ohEibAdSh36s7qjmq/qsi
SMVKE1o8r43pTqwZ6hk7gpJ20tMFlJ2DAw9Kdu+qYCm/RkkDqRVzLehsyH1GnoSt
+by5lUfPV8Kdh5hWNi+3NGw1pLCS2Fn+TeNyXsJPzWoNlykiuQeLgIDQUimtQGP/
yad+IRb2BLWvt4zYhzbRxHwZ2cx+Lc0NjgyVgpppRlzqlVDG94HdN4+b+H6djKtU
CkHCECx/BFKiuXmfOZU/Belcffuit6musD9P/MVGOx1D5KzbKdZoKwI2mdRze7f6
glCQ1r0FT7/UE22LmauL2dhVIDcKIDQGATIqNrEmgIuKidbFdL1KzJjPuEs3ov5E
5WFE6DRSKcXIiDqKjfrJLjXPa+ahC7VdaOT5CA7UURbXmWL0WXFAEVaeMMqjZz3Y
8MjULWznf8ovpYcZH+IapRWTbWlUxLcEd1CTJgfEr944GqE0QYmQPU23IbYIVJLq
vpSuk1VjENww+UaU1OmrpL+6QklCJAE7cC8NW7gojJN0JPe2vOP/1K4xwA25u0Nx
```

Figure 10: Encrypted message file content

7. Decrypt the encrypted signed message file

›_ Console

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

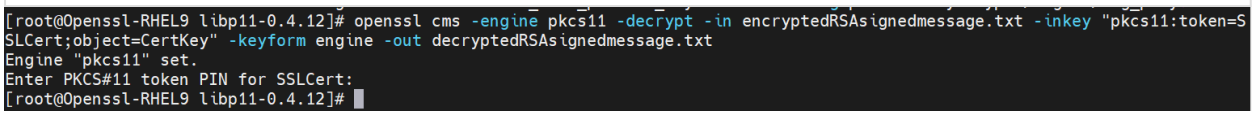

Figure 11: Decrypt Sign message

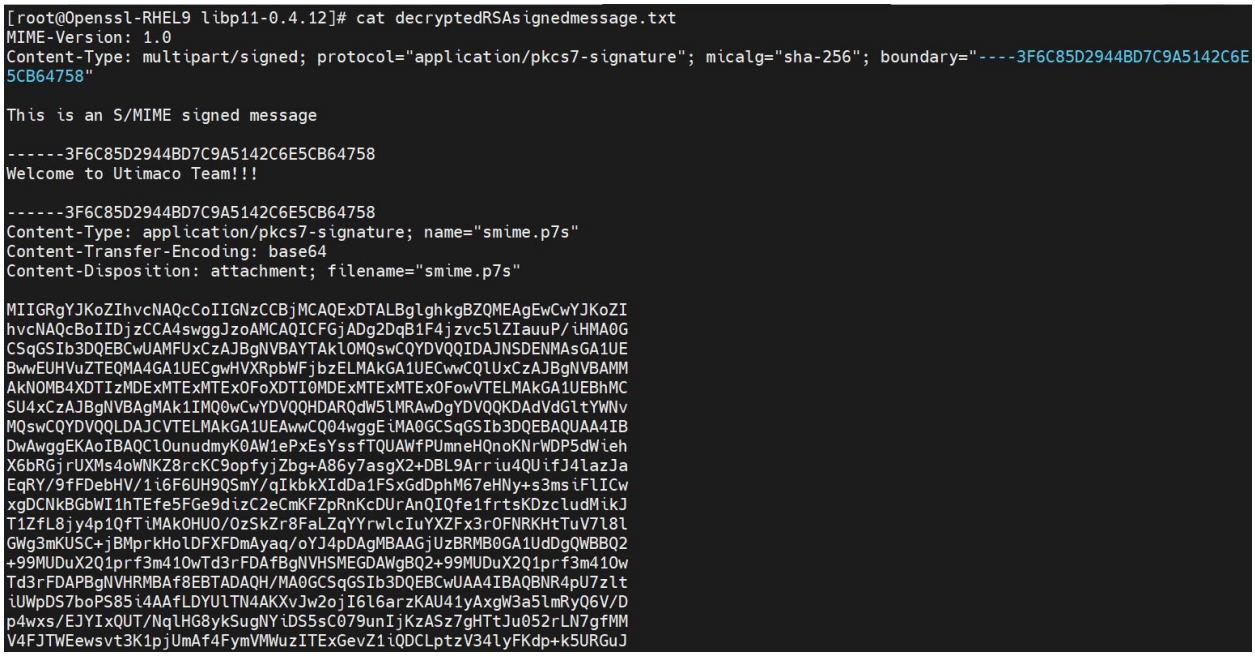

Figure 12: Content of decrypted signed message file

Here SSLCert is the token label and CertKey is the key on the HSM. Provide Cryptouser PIN when prompted.

8. Verify the decrypted signed message file

›_ Console

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

Figure 13: Output of openssl verification command

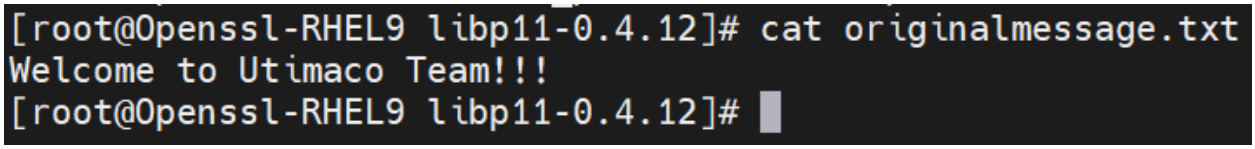

Figure 14: Output of original message content

4.4.3.2 Testing with ECC Key (Linux)

1. Generate the ECC 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 complete then add CKA_ID for both public and private ECC 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 following command

›_ Console

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

Example

›_ Console

```
[root@Openssl-RHEL9 bin]# ./p11tool2 slot=1 LoginUser=ask ListObjects Enter
normal user PIN:

CKO_PUBLIC_KEY:

+ 1.1

CKA_KEY_TYPE    = CKK_ECDSA

CKA_UNIQUE_ID   = 87E13F2D-A49F-497C-B49C-CFEB9725EBEB

CKA_LABEL   = TestECDSAKey

CKA_ID  =

CKO_PRIVATE_KEY:

+ 2.1

CKA_KEY_TYPE    = CKK_ECDSA

CKA_UNIQUE_ID   = F1C1638F-41B3-4447-BF77-22E9EC04F2E5

CKA_SENSITIVE   = CK_TRUE

CKA_EXTRACTABLE = CK_FALSE

CKA_LABEL   = TestECDSAKey

CKA_ID  =
```

3. Generate a certificate request

›_ Console

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

```
[root@Openssl-RHEL9 bin]# openssl req -engine pkcs11 -new -key "pkcs11:token=SSLCertNew;object=TestECDSAKey" -keyform engine
 -out TestECDSACSR.csr
Engine "pkcs11" set.
Enter PKCS#11 token PIN for SSLCertNew:
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) [XX]:IN
State or Province Name (full name) []:CP
Locality Name (eg, city) [Default City]:UK
Organization Name (eg, company) [Default Company Ltd]:Utimaco
Organizational Unit Name (eg, section) []:CS
Common Name (eg, your name or your server's hostname) []:Security
Email Address []:Test@gmail.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
[root@Openssl-RHEL9 bin]#
```

```
[root@Openssl-RHEL9 bin]# cat TestECDSACSR.csr
-----BEGIN CERTIFICATE REQUEST-----
MIIBNDCB2gIBADB4MQswCQYDVQQGEwJJTjELMAkGA1UECAwCQ1AxCzAJBgNVBAcM
AlVLMRAwDgYDVQQKDAdVdGltYWNvMQswCQYDVQQLDAJDUzERMA8GA1UEAwwIU2Vj
dXJpdHkxHTAbBgkqhkiG9w0BCQEWDlRlc3RAZ21haWwuY29tMFkwEwYHKoZIzj0C
AQYIKoZIzj0DAQcDQgAErwps9+W15gA42jErk9SeUV8OOFX6ecLyu5/EhI3q3dyW
ozKjQOOb11JBW0WlYffhHGljQk04cZ0H6d3jQjm0/KAAMAoGCCqGSM49BAMCA0kA
MEYCIQC7+L8i8pqHqxSc4bIB3YJShRdI2eNkMDCiLsYVYf3i5wIhAM4N1VXaYphQ
mXaAgpFbo4LJjvQ8e3nXvWGfDlS3gALJ
-----END CERTIFICATE REQUEST-----
[root@Openssl-RHEL9 bin]#
```

Figure 15: Generate certificate request and Content of generated certificate request

Here SSLCertNew 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=SSLCertNew;object=TestECDSAKey" -keyform engine -out
TestECDSA.cert
```

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

```
[root@Openssl-RHEL9 bin]# openssl req -engine pkcs11 -new -x509 -days 365 -key "pkcs11:token=SSLCertNew;object=TestECDSAKey"
 -keyform engine -out TestECDSA.cert
Engine "pkcs11" set.
Enter PKCS#11 token PIN for SSLCertNew:
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) [XX]:UK
State or Province Name (full name) []:MN
Locality Name (eg, city) [Default City]:VB
Organization Name (eg, company) [Default Company Ltd]:Utimaco
Organizational Unit Name (eg, section) []:CS
Common Name (eg, your name or your server's hostname) []:Security
Email Address []:gtsdf@gmail.com
[root@Openssl-RHEL9 bin]#
```

```
[root@Openssl-RHEL9 bin]# cat TestECDSA.cert
-----BEGIN CERTIFICATE-----
MIICSDCCAe2gAwIBAgIUWCCLiuGheiKa0XEPuoLbOG3YMoowCgYIKoZIzj0EAwIw
eTELMAkGA1UEBhMCVUsxCzAJBgNVBAgMAk1OMQswCQYDVQQHDAJWQjEQMA4GA1UE
CgwHVXRpbWFjbzELMAkGA1UECwwCQ1MxETAPBgNVBAMMCFNlY3VyaXR5MR4wHAYJ
KoZIhvcNAQkBFg9ndHNkZkBnbWFpbC5jb20wHhcNMjMwMzIwMTEwODA5WhcNMjQw
MzE5MTEwODA5WjB5MQswCQYDVQQGEwJVSzELMAkGA1UECAwCTU4xCzAJBgNVBAcM
AlZCMRAwDgYDVQQKDAdVdGltYWNvMQswCQYDVQQLDAJDUzERMA8GA1UEAwwIU2Vj
dXJpdHkxHjAcBgkqhkiG9w0BCQEWD2d0c2RmQGdtYWlsLmNvbTBZMBMGByqGSM49
AgEGCCqGSM49AwEHA0IABK8KbPflteYAONoxK5PUnlFfDjhV+nnC8rufxISN6t3c
lqMyo0Djm5dSQVtFpWH34RxpY0JNOHGdB+nd40I5tPyjUzBRMB0GA1UdDgQWBBSq
v26xMtKttT9S/hWVMtoaICxAXDAfBgNVHSMEGDAWgBSqv26xMtKttT9S/hWVMtoa
ICxAXDAPBgNVHRMBAf8EBTADAQH/MAoGCCqGSM49BAMCA0kAMEYCIQC0iWe6rqoQ
dEGetd3BJLKeXa1twJom5WQy6t098hyQagIhAO1VrH1s5MkmMPj4eLadpJxrypkH
Y0rJsG+SEfA6tiwU
-----END CERTIFICATE-----
[root@Openssl-RHEL9 bin]#
```

Figure 16: Content of self-signed certificate

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

›_ Console

```
# touch message.txt
```

```
[root@Openssl-RHEL9 libp11-0.4.12]# cat message.txt
Welcome to Utimaco Team!!!
[root@Openssl-RHEL9 libp11-0.4.12]#
```

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=SSLCertNew;object=TestECDSAKey" -keyform engine -out
signedECDSAmessage.txt
```

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

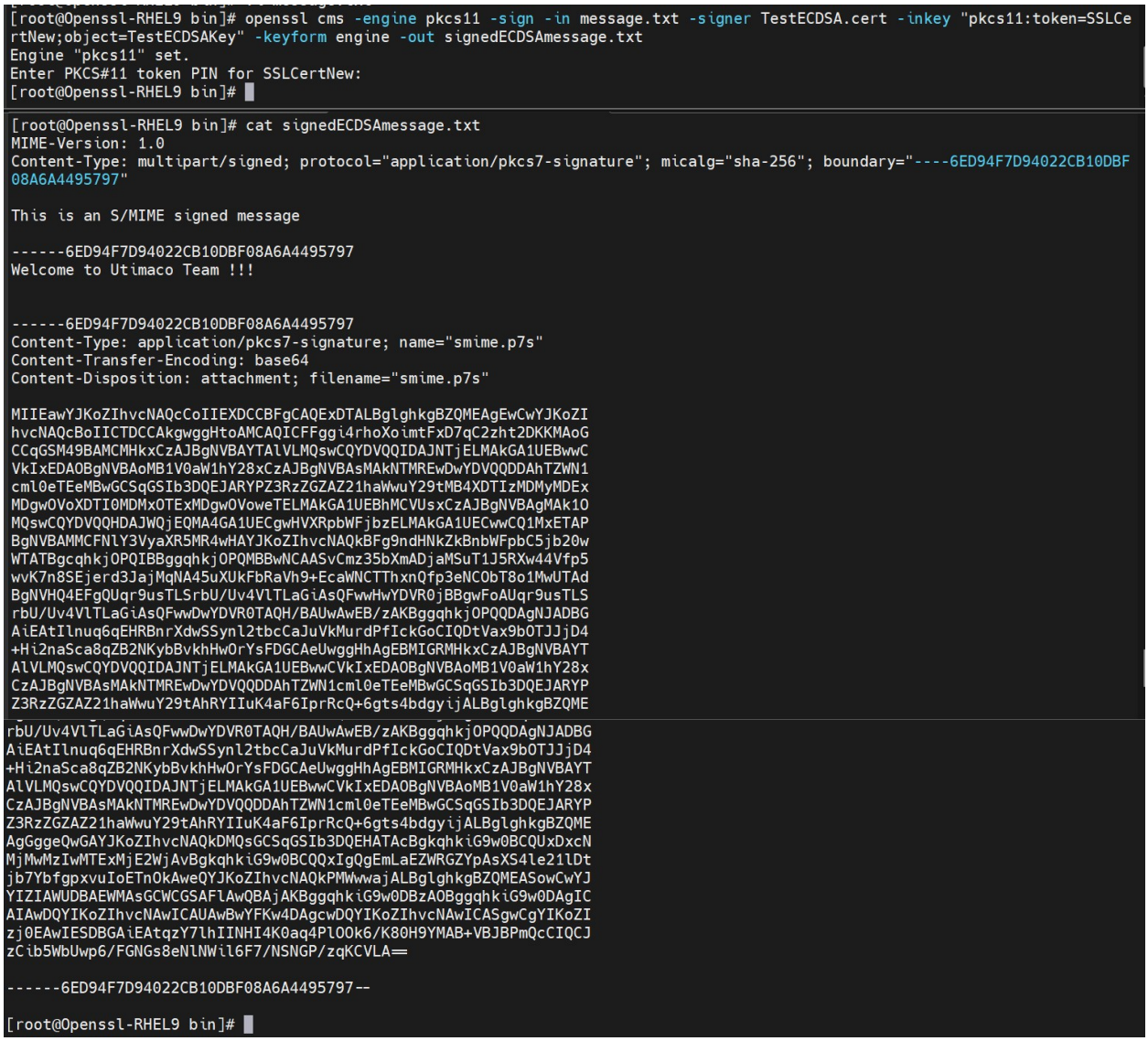

Figure 18: Content of signed message file

7. Verify the signed message file

›_ Console

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

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

```
[root@Openssl-RHEL9 libp11-0.4.12]# cat originalmessage.txt
Welcome to Utimaco Team!!!
[root@Openssl-RHEL9 libp11-0.4.12]#
```

Figure 19: Content of original message file

4.4.4 Creating a Local CA and Performing Cryptographic Operation with OpenSSL (Linux)

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/newcerts`

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 p11tool2 for root CA For RSA

›_ Console

```
# p11tool2 slot=2 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 ECC

›_ Console

```
# p11tool2 slot=2 LoginUser=123456 PubKeyAttr=CKA_LABEL="CAKey"
 PrvKeyAttr=CKA_LABEL="CAKey" GenerateKeyPair=ECC
```

This generates ECC CA private and public keys on the HSM

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

›_ Console

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

```
[root@Openssl-RHEL9 bin]# ./p11tool2 slot=0 LoginUser=12345678 ListObjects


CKO_PUBLIC_KEY:

+ 1.1
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_UNIQUE_ID                 = ECE4D623-EDC7-4250-A675-69517F52C2FA
  CKA_LABEL                     = CAKey
  CKA_ID                        =



CKO_PRIVATE_KEY:

+ 2.1
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_UNIQUE_ID                 = C9FFADE9-C714-4797-A1E0-1A5EABF086F4
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = CAKey
  CKA_ID                        =

[root@Openssl-RHEL9 bin]#
```

Figure 20: CA RSA Key list

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

›_ Console

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

```
[root@Openssl-RHEL9 bin]# ./p11tool2 slot=2 LoginUser=ask ListObjects
Enter normal user PIN:


CKO_PUBLIC_KEY:

+ 1.1
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_UNIQUE_ID                 = 5274D14E-E336-4B3A-9437-FDF08C39726F
  CKA_LABEL                     = TestECDSAKey
  CKA_ID                        =


+ 1.2
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_UNIQUE_ID                 = A8357051-2FCE-4A2C-A7BB-A492BD7436AA
  CKA_LABEL                     = CAKey
  CKA_ID                        =



CKO_PRIVATE_KEY:

+ 2.1
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_UNIQUE_ID                 = 2D45F73E-5DC3-4ACA-9B8D-598E663534A1
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = TestECDSAKey
  CKA_ID                        =


+ 2.2

+ 2.2
  CKA_KEY_TYPE                   = CKK_ECDSA
  CKA_UNIQUE_ID                  = 22A04B15-27C6-4310-9D1D-D2E5984B4871
  CKA_SENSITIVE                  = CK_TRUE
  CKA_EXTRACTABLE                = CK_FALSE
  CKA_LABEL                      = CAKey
  CKA_ID                         =

[root@Openssl-RHEL9 bin]#
```

Figure 21: List ECC key

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=SSLCert1;object=CAKey” -keyform engine -out

/localCA/newcerts/ca.cer
```

```
[root@Openssl-RHEL9 libp11-0.4.12]# openssl req -engine pkcs11 -new -x509 -days 365 -key "pkcs11:token=SSLCert1;object=CAKey" -k
eyform engine -out /localCA/newcerts/ca.cer
Engine "pkcs11" set.
Enter PKCS#11 token PIN for SSLCert1:
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) [XX]:IN
State or Province Name (full name) []:MH
Locality Name (eg, city) [Default City]:PN
Organization Name (eg, company) [Default Company Ltd]:Utimaco
Organizational Unit Name (eg, section) []:BU
Common Name (eg, your name or your server's hostname) []:CS
Email Address []:
[root@Openssl-RHEL9 libp11-0.4.12]#
```

Figure 22: CA certificate generation output

Here CAKey is the Object label for the CA private key on the Utimaco HSM created in Step 5 and SSLCert1 is token label. Provide Cryptouser PIN when prompted.

4.4.5 Generate Certificate Request for Sender and Receiver (Linux)

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=2 LoginUser=123456 PubKeyAttr=CKA_LABEL=”SenderKey”
PrvKeyAttr=CKA_LABEL=”SenderKey” GenerateKeyPair=RSA
```

For ECC

›_ Console

```
# p11tool2 slot=2 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 ECC keys using PKCS11# CryptoServer Administration tool.

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

```
[root@Openssl-RHEL9 bin]# ./p11tool2 slot=2 LoginUser=ask ListObjects
Enter normal user PIN:


CKO_PUBLIC_KEY:

+ 1.1
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_UNIQUE_ID                 = 5274D14E-E336-4B3A-9437-FDF08C39726F
  CKA_LABEL                     = TestECDSAKey
  CKA_ID                        =


+ 1.2
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_UNIQUE_ID                 = A8357051-2FCE-4A2C-A7BB-A492BD7436AA
  CKA_LABEL                     = CAKey
  CKA_ID                        =


+ 1.3
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_UNIQUE_ID                 = CA926885-B571-416D-B5DB-E5245C906E99
  CKA_LABEL                     = SenderKey
  CKA_ID                        =



CKO_PRIVATE_KEY:
```

```
CKO_PRIVATE_KEY:

+ 2.1
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_UNIQUE_ID                 = 2D45F73E-5DC3-4ACA-9B8D-598E663534A1
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = TestECDSAKey
  CKA_ID                        =


+ 2.2
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_UNIQUE_ID                 = 22A04B15-27C6-4310-9D1D-D2E5984B4871
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = CAKey
  CKA_ID                        =


+ 2.3
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_UNIQUE_ID                 = FA3D06F7-58A4-4116-B64C-3A693BF02E89
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = SenderKey
  CKA_ID                        =

[root@Openssl-RHEL9 bin]#
```

Figure 23: CA certificate generation output Sender ECC Key List

For RSA

›_ Console

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

```
[root@Openssl-RHEL9 bin]# ./p11tool2 slot=5 LoginUser=12345678 ListObjects

CKO_PUBLIC_KEY:

+ 1.1
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_UNIQUE_ID                 = EFD36AB5-3468-4E4A-A43B-EDF7BE812065
  CKA_LABEL                     = SenderKey
  CKA_ID                        =

+ 1.2
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_UNIQUE_ID                 = BFFE6D15-48C9-42B5-8CDB-C03588356D24
  CKA_LABEL                     = CAKey
  CKA_ID                        =


CKO_PRIVATE_KEY:

+ 2.1
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_UNIQUE_ID                 = F091D971-827A-42C2-9FD4-0FA083D99E6C
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = SenderKey
  CKA_ID                        =

+ 2.2
  CKA_KEY_TYPE                  = CKK_RSA
```

Figure 24: Sender RSA Key list

4. Generate a certificate request for sender.

›_ Console

```
# openssl req -engine pkcs11 -new -key "pkcs11:token=SSLCert1;object=SenderKey"
 -keyform engine -out

/localCA/newcerts/sender/sender.txt
```

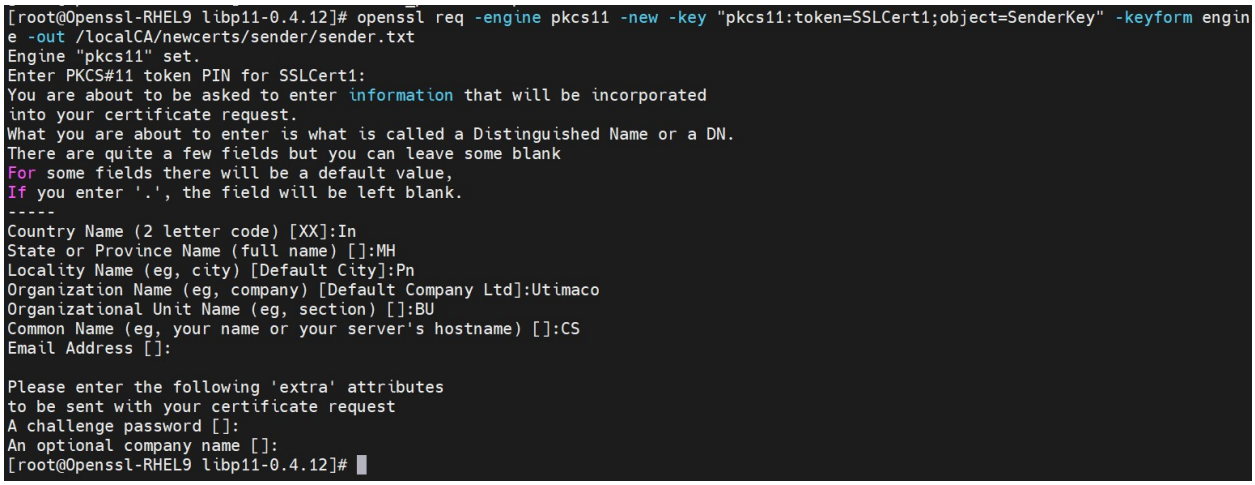

Figure 25: Sender certificate request generation

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

Here SSLCert1 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=SSLCert1;object=CAKey" -keyform engine -out

/localCA/newcerts/sender/SenderSignedCertificate.cert
```

```
[root@Openssl-RHEL9 libp11-0.4.12]# openssl ca -engine pkcs11 -policy policy_anything -cert /localCA/newcerts/ca.cer -in /localC
A/newcerts/sender/sender.txt -keyfile "pkcs11:token=SSLCert1;object=CAKey" -keyform engine -out /localCA/newcerts/sender/SenderS
ignedCertificate.cert
Engine "pkcs11" set.
Using configuration from /etc/pki/tls/openssl.cnf
Enter PKCS#11 token PIN for SSLCert1:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 1 (0x1)
        Validity
            Not Before: Feb  7 12:31:20 2023 GMT
            Not After : Feb  7 12:31:20 2024 GMT
        Subject:
            countryName               = In
            stateOrProvinceName       = MH
            localityName              = Pn
            organizationName          = Utimaco
            organizationalUnitName    = BU
            commonName                = CS
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Subject Key Identifier:
                85:B3:B0:C5:7B:20:A3:01:5B:DB:B8:94:76:13:56:1A:B3:F8:5E:86
            X509v3 Authority Key Identifier:
                F4:75:2F:5B:F5:AF:97:3B:DA:D8:6E:E9:0F:87:24:44:3E:B3:98:9F
Certificate is to be certified until Feb  7 12:31:20 2024 GMT (365 days)
Sign the certificate? [y/n]:y


Certificate Details:
        Serial Number: 1 (0x1)
        Validity
            Not Before: Feb  7 12:31:20 2023 GMT
            Not After : Feb  7 12:31:20 2024 GMT
        Subject:
            countryName               = In
            stateOrProvinceName       = MH
            localityName              = Pn
            organizationName          = Utimaco
            organizationalUnitName    = BU
            commonName                = CS
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Subject Key Identifier:
                85:B3:B0:C5:7B:20:A3:01:5B:DB:B8:94:76:13:56:1A:B3:F8:5E:86
            X509v3 Authority Key Identifier:
                F4:75:2F:5B:F5:AF:97:3B:DA:D8:6E:E9:0F:87:24:44:3E:B3:98:9F
Certificate is to be certified until Feb  7 12:31:20 2024 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-RHEL9 libp11-0.4.12]#
```

Figure 26: Sender certificate request signing by CA

Press y to sign and y again to commit.

Here SSLCert1 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 receiver only RSA keys are generated. OpenSSL 3 does not support encryption and decryption with ECC key.

For RSA

›_ Console

```
./p11tool2 slot=<Slot_No.> LoginUser=123456 PubKeyAttr=CKA_LABEL="ReceiverKey"
 PrvKeyAttr=CKA_LABEL="ReceiverKey" GenerateKeyPair=RSA
```

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

›_ Console

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

For RSA

```
[root@Openssl-RHEL9 bin]# ./p11tool2 slot=5 LoginUser=12345678 ListObjects


CKO_PUBLIC_KEY:

+ 1.1
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_UNIQUE_ID                 = 4A58B012-920C-4332-B73D-8E2E00B8FDA8
  CKA_LABEL                     = ReceiverKey
  CKA_ID                        =


+ 1.2
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_UNIQUE_ID                 = EFD36AB5-3468-4E4A-A43B-EDF7BE812065
  CKA_LABEL                     = SenderKey
  CKA_ID                        =


+ 1.3
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_UNIQUE_ID                 = BFFE6D15-48C9-42B5-8CDB-C03588356D24
  CKA_LABEL                     = CAKey
  CKA_ID                        =



CKO_PRIVATE_KEY:

+ 2.1
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_UNIQUE_ID                 = F091D971-827A-42C2-9FD4-0FA083D99E6C
  CKA_SENSITIVE                 = CK_TRUE
```

```
CKO_PRIVATE_KEY:

+ 2.1
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_UNIQUE_ID                 = F091D971-827A-42C2-9FD4-0FA083D99E6C
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = SenderKey
  CKA_ID                        =


+ 2.2
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_UNIQUE_ID                 = D9622BDC-397F-453C-86A6-937BE7D7534A
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = CAKey
  CKA_ID                        =


+ 2.3
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_UNIQUE_ID                 = 73D9829D-C534-42D1-AE45-21110E01BBCA
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = ReceiverKey
  CKA_ID                        =

[root@Openssl-RHEL9 bin]#
```

Figure 27: Receiver RSA Key list

7. Generate a certificate request for receiver

›_ Console

```
# openssl req -engine pkcs11 -new -key
"pkcs11:token=SSLCert1;object=ReceiverKey" -keyform engine -out

/localCA/newcerts/receiver/Receiver.txt
```

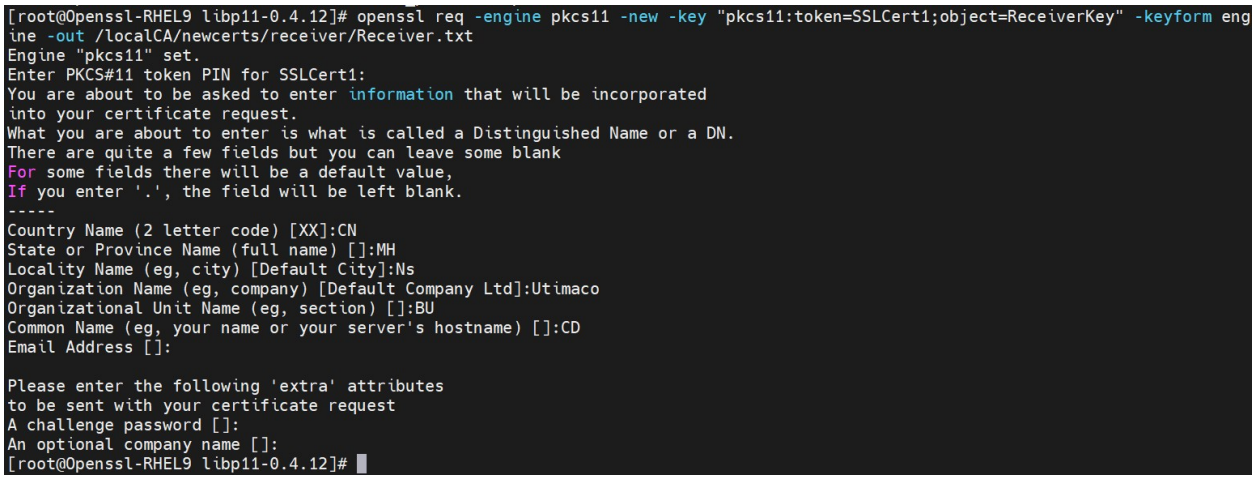

Figure 28: Receiver certificate request generation

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

Here SSLCert1 is the token label and ReceiverKey is the key on the HSM. Provide Cryptouser PIN when prompted.

8. 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=SSLCert1;object=CAKey" -keyform engine -out

/localCA/newcerts/receiver/ReceiverSignedCertificate.cert
```

```
[root@Openssl-RHEL9 libp11-0.4.12]# openssl ca -engine pkcs11 -policy policy_anything -cert /localCA/newcerts/ca.cer -in /localC
A/newcerts/receiver/Receiver.txt -keyfile "pkcs11:token=SSLCert1;object=CAKey" -keyform engine -out /localCA/newcerts/receiver/R
eceiverSignedCertificate.cert
Engine "pkcs11" set.
Using configuration from /etc/pki/tls/openssl.cnf
Enter PKCS#11 token PIN for SSLCert1:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 2 (0x2)
        Validity
            Not Before: Feb  7 12:35:01 2023 GMT
            Not After : Feb  7 12:35:01 2024 GMT
        Subject:
            countryName               = CN
            stateOrProvinceName       = MH
            localityName              = Ns
            organizationName          = Utimaco
            organizationalUnitName    = BU
            commonName                = CD
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Subject Key Identifier:
                86:17:99:D1:66:A1:53:5A:EF:2D:36:55:AB:ED:53:99:36:D5:3D:BC
            X509v3 Authority Key Identifier:
                F4:75:2F:5B:F5:AF:97:3B:DA:D8:6E:E9:0F:87:24:44:3E:B3:98:9F
Certificate is to be certified until Feb  7 12:35:01 2024 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Certificate Details:
        Serial Number: 2 (0x2)
        Validity
            Not Before: Feb  7 12:35:01 2023 GMT
            Not After : Feb  7 12:35:01 2024 GMT
        Subject:
            countryName               = CN
            stateOrProvinceName       = MH
            localityName              = Ns
            organizationName          = Utimaco
            organizationalUnitName    = BU
            commonName                = CD
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Subject Key Identifier:
                86:17:99:D1:66:A1:53:5A:EF:2D:36:55:AB:ED:53:99:36:D5:3D:BC
            X509v3 Authority Key Identifier:
                F4:75:2F:5B:F5:AF:97:3B:DA:D8:6E:E9:0F:87:24:44:3E:B3:98:9F
Certificate is to be certified until Feb  7 12:35:01 2024 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-RHEL9 libp11-0.4.12]#
```

Figure 29: Receiver certificate request signing by CA

Press y to sign and y again to commit.

Here SSLCert1 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 the File (Linux)

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 Team!!!">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=SSLCert1;object=SenderKey" -keyform engine -out signedmessage.txt
```

```
[root@Openssl-RHEL9 libp11-0.4.12]# openssl cms -engine pkcs11 -sign -in message.txt -signer /localCA/newcerts/sender/SenderSign
edCertificate.cert -inkey "pkcs11:token=SSLCert1;object=SenderKey" -keyform engine -out signedmessage.txt
Engine "pkcs11" set.
Enter PKCS#11 token PIN for SSLCert1:
[root@Openssl-RHEL9 libp11-0.4.12]#
```

```
[root@Openssl-RHEL9 libp11-0.4.12]# cat signedmessage.txt
MIME-Version: 1.0
Content-Type: multipart/signed; protocol="application/pkcs7-signature"; micalg="sha-256"; boundary="----431DB26E65692D7585ED1CE2
409DDA38"

This is an S/MIME signed message

------431DB26E65692D7585ED1CE2409DDA38
Welcome to Utimaco Team!!!

------431DB26E65692D7585ED1CE2409DDA38
Content-Type: application/pkcs7-signature; name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"

MIIGFAYJKoZIhvcNAQcCoIIGBTCCBgECAQExDTALBglghkgBZQMEAgEwCwYJKoZI
hvcNAQcBoIIDcjCCA24wggJWoAMCAQICAQEwDQYJKoZIhvcNAQELBQAwUzELMAkG
A1UEBhMCSU4xCzAJBgNVBAgMAk1IMQswCQYDVQQHDAJQTjEQMA4GA1UECgwHVXRp
bWFjbzELMAkGA1UECwwCQlUxCzAJBgNVBAMMAkNTMB4XDTIzMDIwNzEyMzEyMFoX
DTI0MDIwNzEyMzEyMFowUzELMAkGA1UEBhMCSW4xCzAJBgNVBAgMAk1IMQswCQYD
VQQHDAJQbjEQMA4GA1UECgwHVXRpbWFjbzELMAkGA1UECwwCQlUxCzAJBgNVBAMM
AkNTMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA2VNDsGM/JRvxToen
vJPbXTOaScS2zi1Ghy4OYNmd35pJAj36RkaNNyAanxgwkqtQ8mP03s0b1YoLXNxJ
T/ixStgjwXnfKg5+58Dmb1vXCTo/xKADpCwAQLi09rse1LCmkJv6K0hiOGkMDKgd
zo96ffNR431PAU/3M7Qif/rfgZH3fIFAy2KyO2LjtjjlVQ2DIAFdCPVdROXPoEDf
bK4KNrP4cmsqlwrCK7rUypybhpYNT38f/bDeYHL7Ff/xtucM6StTPTYo+EKTOaqA
jOFdUCKr5sDKY3Y6vSmeJsewIoWTxj3j/qRmhWdXiCJdNko7ys7dQl3vM6DZlyaZ
bQTJAQIDAQABo00wSzAJBgNVHRMEAjAAMB0GA1UdDgQWBBSFs7DFeyCjAVvbuJR2
E1Yas/hehjAfBgNVHSMEGDAWgBT0dS9b9a+XO9rYbukPhyREPrOYnzANBgkqhkiG
9w0BAQsFAAOCAQEAL9Lp4anXERGh61snNKH74aD0S2lXSt13byR8TR2dvGUXkkVq
atigsUWm7d2TP2INRH67arUcfFqHEbFM9RYIqYPQy0LBFyJJSeSDRcOqtJ+ONibv
cxTJ2UrpIl74dESWxkqn8YfH93tNwU5Ds12D/GlsU5OyogRL7vmj0+u0vdsXLKkJ
HMt53Q+RSvyyjtWCriailFHuOzOkR+xtTXTtp64FXcSMJ2fVXLP6q8vy8DwfRq2S
0CaDEaMdSXixbwWJJdK2JetjLH/64jQF1rSdX7xn2dcyd5QybrO+S7hT1MV8vAbZ
+GMpMTig5FmaNFnGk+0z+9DJgZEgUDxCiS169jGCAmgwggJkAgEBMFgwUzELMAkG
A1UEBhMCSU4xCzAJBgNVBAgMAk1IMQswCQYDVQQHDAJQTjEQMA4GA1UECgwHVXRp
bWFjbzELMAkGA1UECwwCQlUxCzAJBgNVBAMMAkNTAgEBMAsGCWCGSAFlAwQCAaCB
5DAYBgkqhkiG9w0BCQMxCwYJKoZIhvcNAQcBMBwGCSqGSIb3DQEJBTEPFw0yMzAy
MDcxMjM4MThaMC8GCSqGSIb3DQEJBDEiBCCn6poeZg/AMa/zcpO2Tr0hNFhifm5g
2eH8O/yxiQv+szB5BgkqhkiG9w0BCQ8xbDBqMAsGCWCGSAFlAwQBKjALBglghkgB
ZQMEARYwCwYJYIZIAWUDBAECMAoGCCqGSIb3DQMHMA4GCCqGSIb3DQMCAgIAgDAN
BggqhkiG9w0DAgIBQDAHBgUrDgMCBzANBggqhkiG9w0DAgIBKDANBgkqhkiG9w0B
AQEFAASCAQBGcWJXcPjc3wxEidqJssesvF6xaYo4F2fBd+ry9TTmUtsYYdGK4UVM
1774HExmrBNE1PpIk+P7E+ypmY8Jdq3iM2nBIWHaTb6YKi1WWo1yaWTReso1I2as
8shXPAEUWCrHVfRnouvpb/GB6xk+TTSykoKo5UhzWWetdGm91RwYa9kk125E5cvE
6OiARaDfNA8tRgHs5+lMZLHxB3KS+BpHat/RiqUw32nHutN6/40o6D6h+i/7TLQ4
qkGFq50D1cwM2bAf92ehuqSqBIJgt7vErSAb9DybqbI05iYRKrI5c3uA281XkD3+
WX2wMqcXR0PVyLg542Yf69GVT3D7q9Gi

------431DB26E65692D7585ED1CE2409DDA38--

[root@Openssl-RHEL9 libp11-0.4.12]#
```

Figure 30: Openssl sign command output and content of signed message file

Here SSLCert1 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
```

```
[root@Openssl-RHEL9 libp11-0.4.12]# openssl cms -engine pkcs11 -encrypt -in signedmessage.txt -out encryptedsignedmessage.txt /l
ocalCA/newcerts/receiver/ReceiverSignedCertificate.cert
Engine "pkcs11" set.
[root@Openssl-RHEL9 libp11-0.4.12]#
```

Figure 31: Openssl encrypt command output

Using ECC key, only sign and verify operations can be performed with OpenSSL3.Encryption and decryption operation are not supported in this version for ECC Key.

4.4.7 Decrypt Sender´s Message (Linux)

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

›_ Console

```
# openssl cms -engine pkcs11 -decrypt -in encryptedsignedmessage.txt - inkey
"pkcs11:token=SSLCert1;object=ReceiverKey" -keyform engine -out
decryptedsignedmessage.txt
```

```
[root@Openssl-RHEL9 libp11-0.4.12]# openssl cms -engine pkcs11 -decrypt -in encryptedsignedmessage.txt -inkey "pkcs11:token=SSLC
ert1;object=ReceiverKey" -keyform engine -out decryptedsignedmessage.txt
Engine "pkcs11" set.
Enter PKCS#11 token PIN for SSLCert1:
[root@Openssl-RHEL9 libp11-0.4.12]#
```

Figure 32: Openssl decrypt command output

Here SSLCert1 is the token label and ReceiverKey is the key on the HSM. Provide Cryptouser PIN when prompted.

Using ECC key, only sign and verify operations can be performed with OpenSSL3.Encryption and decryption operation are not supported in this version for ECC Key.

4.4.8 Verify the Signature (Linux)

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

›_ Console

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

/localCA/newcerts/sender/SenderSignCertificate.cert
```

```
[root@Openssl-RHEL9 libp11-0.4.12]# openssl cms -engine pkcs11 -verify -in decryptedsignedmessage.txt -CAfile /localCA/newcerts/
ca.cer -out originalmessage.txt /localCA/newcerts/sender/SenderSignCertificate.cert
Engine "pkcs11" set.
CMS Verification successful
[root@Openssl-RHEL9 libp11-0.4.12]#
```

Figure 33: Openssl verify command output

2. Verify the signature of signedmessage.txt using the sender's certificate for ECC

›_ Console

```
[root@Openssl-RHEL9 bin]# openssl cms -engine pkcs11 -verify -in
signedmessage.txt -CAfile /localCA/newcerts/ca.cer -out originalmessage.txt /
localCA/newcerts/sender/SenderSignCertificate.cert
```

```
[root@Openssl-RHEL9 bin]# openssl cms -engine pkcs11 -sign -in message.txt -signer /localCA/newcerts/sender/SenderSignedCertif
icate.cert -inkey "pkcs11:token=SSLCertNew;object=SenderKey" -keyform engine -out signedmessage.txt^C
[root@Openssl-RHEL9 bin]# openssl cms -engine pkcs11 -verify -in signedmessage.txt -CAfile /localCA/newcerts/ca.cer -out origi
nalmessage.txt /localCA/newcerts/sender/SenderSignCertificate.cert
Engine "pkcs11" set.
CMS Verification successful
[root@Openssl-RHEL9 bin]#
```

Figure 34: Openssl verify command output

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

```
[root@Openssl-RHEL9 libp11-0.4.12]# cat originalmessage.txt
Welcome to Utimaco Team!!!
[root@Openssl-RHEL9 libp11-0.4.12]#
```

Figure 35: Content of original message file

5 Integrating OpenSSL 3.0 with SecurityServer on Windows

5.1 Installing and Configuring Utimaco SecurityServer Software (Windows)

5.1.1 SecurityServer PKCS#11 Configuration (Windows)

On windows, as part of SecurityServer 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 = false

# 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 SecurityServer documentation. The device may be a SecurityServer (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.*

5.1.2 Create SO User and Initialize a Slot (Windows)

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.

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

5.2 Configuration OpenSSL to use Utimaco SecurityServer (Windows)

5.2.1 Setting up Utimaco SecurityServer Library in OpenSSL Configuration File (Windows)

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

openssl.cnf

```
openssl_conf = openssl_init
```

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

openssl.cnf

```
[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.

5.2.2 Verify PKCS#11 Engine (Windows)

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

›_ Console

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

```
C:\OpenSSL-Win64\bin>openssl.exe engine pkcs11 -t
(pkcs11) pkcs11 engine
     [ available ]

C:\OpenSSL-Win64\bin>openssl version
OpenSSL 3.0.5 5 Jul 2022 (Library: OpenSSL 3.0.5 5 Jul 2022)

C:\OpenSSL-Win64\bin>_
```

Figure 37: Verification of pkcs11 engine and OpenSSL version

5.2.3 Creating a local CA 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/CertKey.pem# The private key
```

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 p11tool2 For RSA

›_ Console

```
C:\Program Files\Utimaco\SecurityServer\Administration>p11tool2 slot=0
 LoginUser=12345678 PubKeyAttr=CKA_LABEL="CertKey"
 PrvKeyAttr=CKA_LABEL="CertKey" GenerateKeyPair=RSA
```

This generates RSA 2048 CA private keys on the HSM

Figure 38: Generating RSA Key

For ECC

›_ Console

```
C:\Program Files\Utimaco\SecurityServer\Administration>p11tool2 slot=<slot_no.>
LoginUser=12345678 PubKeyAttr=CKA_LABEL="CertKey" PrvKeyAttr=CKA_LABEL="CertKey"
 GenerateKeyPair=ECC
```

This generates ECC CA private keys on the HSM

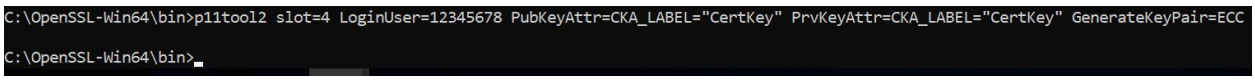

Figure 39: Generating ECC Key

Once key generation is completed then add CKA_ID for both public and private ECC 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=0
 LoginUser=<slot_PIN> ListObjects
```

```
C:\Program Files\Utimaco\SecurityServer\Administration>p11tool2 slot=0 LoginUser=12345678 ListObjects


CKO_PUBLIC_KEY:

+ 1.1
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_UNIQUE_ID                 = B1E4ADAF-693D-44E4-8767-DFA187BBA35D
  CKA_LABEL                     = SSLKey1
  CKA_ID                        =


+ 1.2
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_UNIQUE_ID                 = 837E1CEF-6E6A-482A-9703-36A41C614B70
  CKA_LABEL                     = CertKey
  CKA_ID                        =



CKO_PRIVATE_KEY:

+ 2.1
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_UNIQUE_ID                 = 1A2F6B3C-1457-4A46-AB1C-1E2B75DA62F5
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = CertKey
  CKA_ID                        =


+ 2.2
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_UNIQUE_ID                 = 6382B1FE-8E16-40D3-BA00-1C18D0A58441
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = SSLKey1
  CKA_ID                        =


C:\Program Files\Utimaco\SecurityServer\Administration>
```

Figure 40: CA RSA Key list

For ECC

```
C:\OpenSSL-Win64\bin>p11tool2 slot=4 LoginUser=ask ListObjects
Enter normal user PIN:


CKO_PUBLIC_KEY:

+ 1.1
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_UNIQUE_ID                 = E5873C5C-104F-40F3-BAAE-78EA1112EAFE
  CKA_LABEL                     = TestECDSAKey
  CKA_ID                        =


+ 1.2
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_UNIQUE_ID                 = 3743CC3C-0CC2-4941-B16C-22FD36B78480
  CKA_LABEL                     = CertKey
  CKA_ID                        =



CKO_PRIVATE_KEY:

+ 2.1
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_UNIQUE_ID                 = 70049D97-F8BD-4B13-9968-8AE115589D9E
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = CertKey
  CKA_ID                        =


+ 2.2
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_UNIQUE_ID                 = 66B1B1C1-8E86-4BAE-B2EF-1B65EE4D2061
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = TestECDSAKey
  CKA_ID                        =
```

Figure 41: ECC Key

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=SSLCert;object=CertKey" -keyform engine -out C:
\localCA\newcerts\ca.cer
```

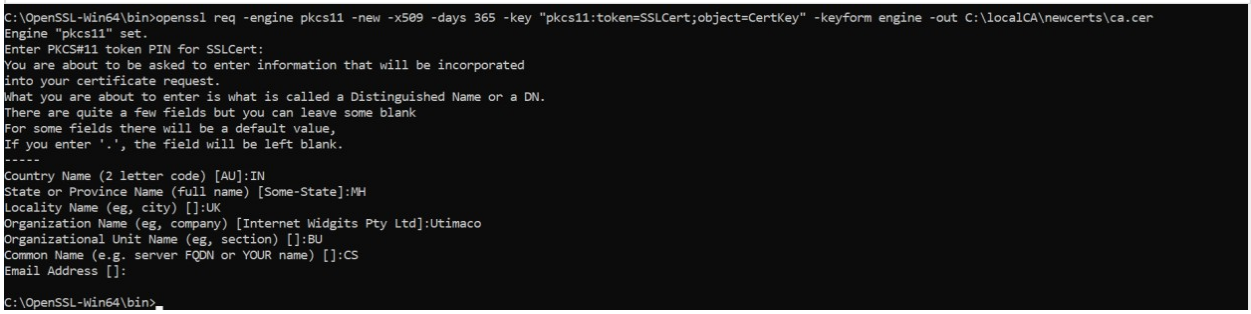

Figure 42: CA certificate generation output

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

5.2.4 Generate Certificate Request for Sender and Receiver (Windows)

1. Create a directory to generate the certificate request for 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=0
 LoginUser=12345678 PubKeyAttr=CKA_LABEL="SenderKey"
 PrvKeyAttr=CKA_LABEL="SenderKey" GenerateKeyPair=RSA
```

For ECC

›_ Console

```
C:\Program Files\Utimaco\SecurityServer\Administration>p11tool2 slot=4
 LoginUser=12345678 PubKeyAttr=CKA_LABEL="SenderKey"
 PrvKeyAttr=CKA_LABEL="SenderKey" GenerateKeyPair=ECC
```

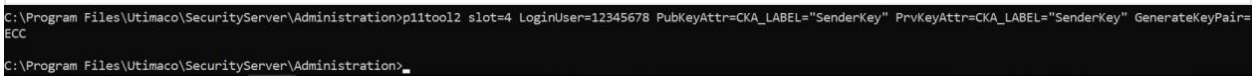

Figure 43: Generate ECC key for Sender

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

Verify the ECC key

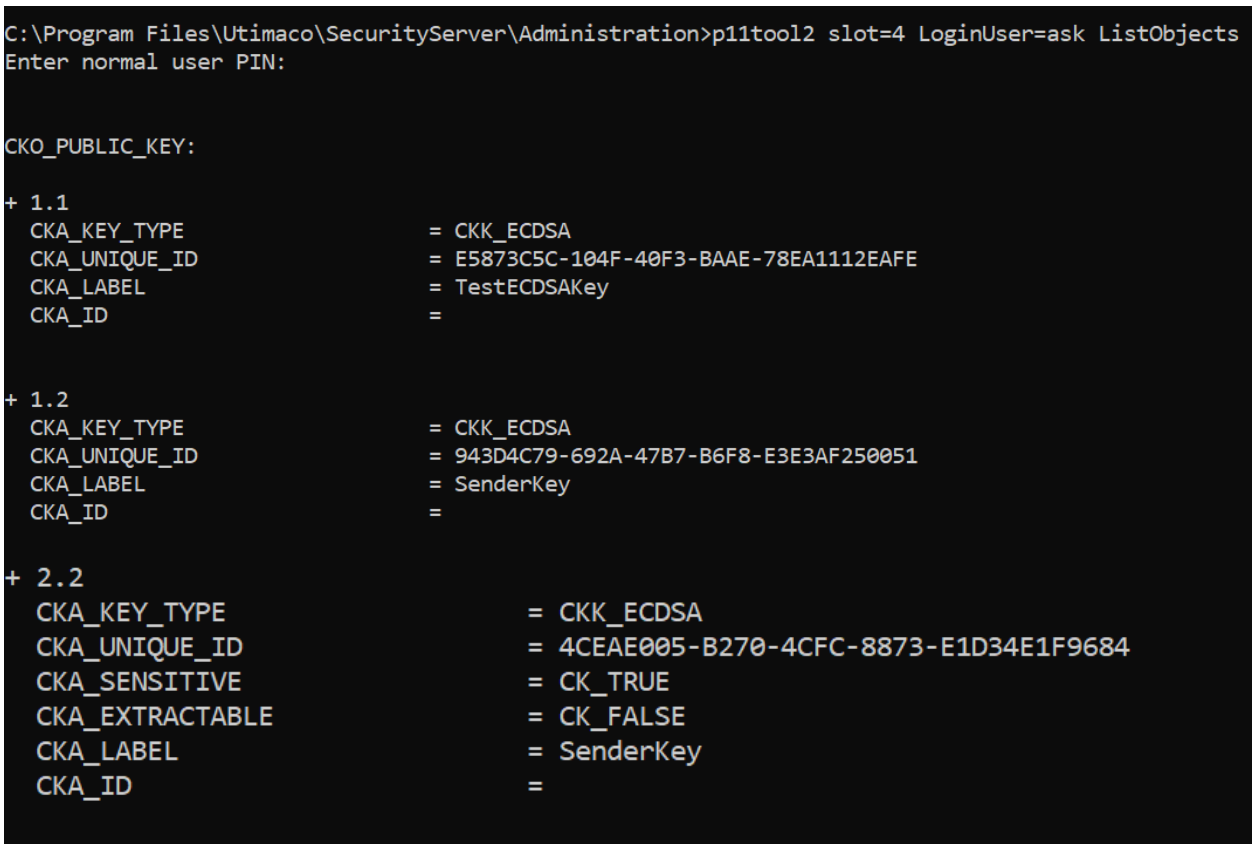

Figure 44: List Sender ECC Key

3. Generate a certificate request for sender

›_ Console

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

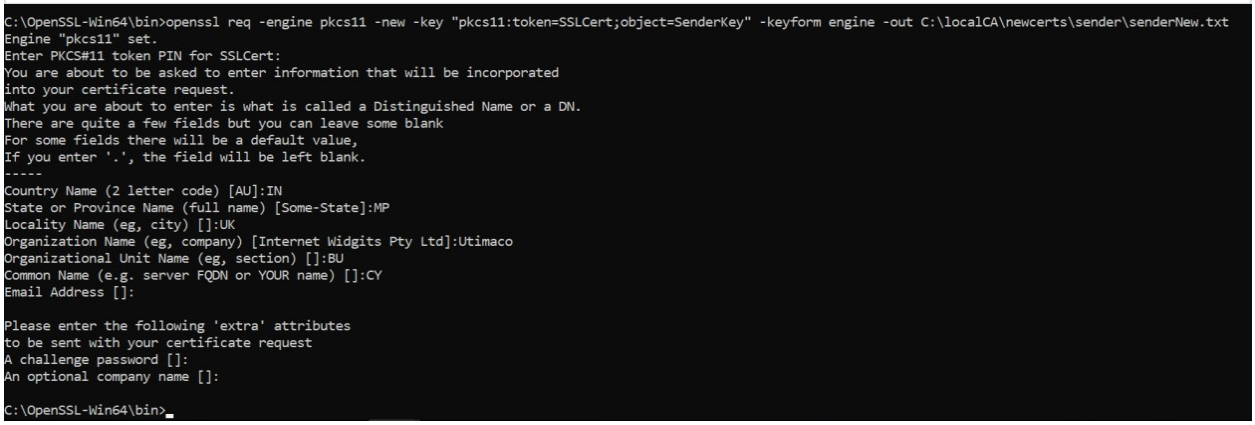

Figure 45: Sender certificate request generation output

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

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

4. Sign the certificate request for 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=SSLCert;object=CertKey" -keyform engine -out C:
\localCA\newcerts\sender\SenderSignedCertificate.cer
```

```
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=
SSLCert;object=CertKey" -keyform engine -out C:\localCA\newcerts\sender\SenderSignedCertificate.cer
Engine "pkcs11" set.
Using configuration from C:\Program Files\Common Files\SSL/openssl.cnf
Enter PKCS#11 token PIN for SSLCert:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 1 (0x1)
        Validity
            Not Before: Jan 31 06:23:38 2023 GMT
            Not After : Jan 31 06:23:38 2024 GMT
        Subject:
            countryName               = IN
            stateOrProvinceName       = MP
            localityName              = UK
            organizationName          = Utimaco
            organizationalUnitName    = BU
            commonName                = CY
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Subject Key Identifier:
                A3:B3:5F:4F:F2:91:6E:E0:1A:AB:CB:5A:DC:9B:05:0D:BA:B4:16:DF
            X509v3 Authority Key Identifier:
                73:2D:7B:2B:F9:71:2F:EC:FD:85:AA:4D:A7:C1:45:B6:83:F1:66:98
Certificate is to be certified until Jan 31 06:23:38 2024 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

C:\OpenSSL-Win64\bin>
```

Figure 46: Sender certificate request signing by CA

Press y to sign and y again to commit.

Here SSLCert is the token label and CertKey is the key on the HSM. Provide Cryptouser PIN when prompted.

5. Generate key for receiver using p11tool2 for RSA

›_ Console

```
C:\Program Files\Utimaco\SecurityServer\Administration>

p11tool2 slot=0 LoginUser=12345678 PubKeyAttr=CKA_LABEL="ReceiverKey"
 PrvKeyAttr=CKA_LABEL="ReceiverKey" GenerateKeyPair=RSA
```

```
C:\Program Files\Utimaco\SecurityServer\Administration>p11tool2 slot=0 LoginUser=12345678 ListObjects


CKO_PUBLIC_KEY:

+ 1.1
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_UNIQUE_ID                 = 1A910E61-C482-4494-80E7-5662346A34F2
  CKA_LABEL                     = ReceiverKey
  CKA_ID                        =
```

```
CKO_PRIVATE_KEY:

+ 2.1
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_UNIQUE_ID                 = E46ED062-8A83-4F14-8995-ACA3328BF688
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = SenderKey
  CKA_ID                        = 0x46 (F)

+ 2.2
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_UNIQUE_ID                 = 9A33687F-FE64-43FF-ADB4-7799E0CC1DCE
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = CertKey
  CKA_ID                        = 0x45 (E)

+ 2.3
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_UNIQUE_ID                 = 50B2406F-265E-4DAD-9EA7-92EA9CD302E3
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = ReceiverKey
  CKA_ID                        =
```

Figure 47: List Receiver key output

For receiver only RSA keys are generated. OpenSSL 3 does not support encryption and decryption with ECC key.

6. Generate a certificate request for receiver using RSA key

›_ Console

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

```
C:\OpenSSL-Win64\bin>openssl req -engine pkcs11 -new -key "pkcs11:token=SSLCert;object=ReceiverKey" -keyform engine -out C:\localCA\newcerts\receiver\ReceiverNew.txt
Engine "pkcs11" set.
Enter PKCS#11 token PIN for SSLCert:
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) []:UK
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Utimaco
Organizational Unit Name (eg, section) []:BU
Common Name (e.g. server FQDN or YOUR name) []:CF
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:

C:\OpenSSL-Win64\bin>
```

Figure 48: Receiver certificate request generation output

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

7. Sign the certificate request for 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=SSLCert;object=CertKey" -keyform engine -out C:
\localCA\newcerts\receiver\receiverNew.cer
```

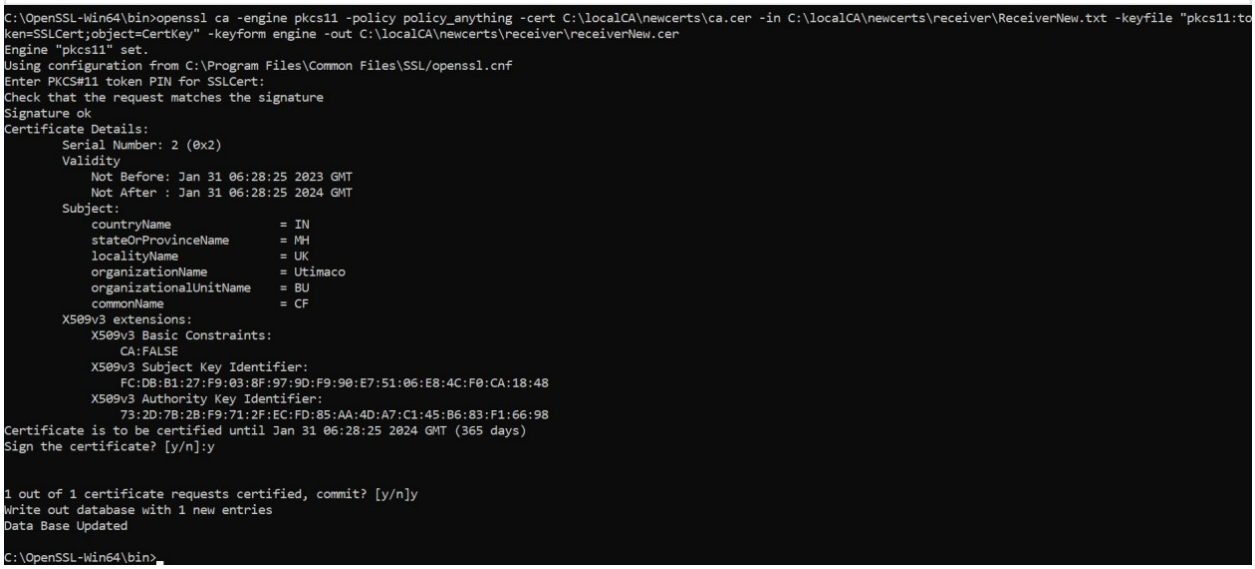

Figure 49: Receiver certificate request signing by CA

Press y to sign and y again to commit.

Here SSLCert is the token label and CertKey is the key on the HSM. Provide Cryptouser PIN when prompted.

5.2.5 Using OpenSSL to Sign and Encrypt the File (Windows)

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

›_ message.txt

```
Welcome to Utimaco !!!
```

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=SSLCert;object=SenderKey" -keyform engine -out C:
\localCA\signedmessage.txt
```

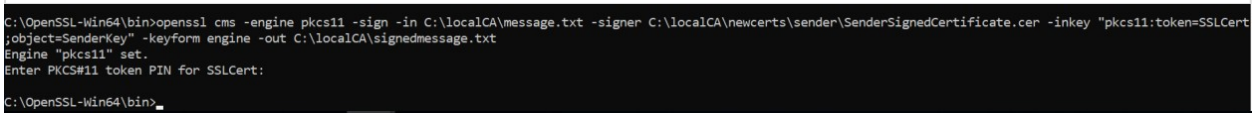

Figure 50: Openssl sign command output

Here SSLCert 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\receiverNew.cer
```

Figure 51: Openssl encrypt command output

Using ECC key, only sign and verify operations can be performed with OpenSSL3.Encryption and decryption operation are not supported in this version for ECC Key.

5.2.6 Decrypt Sender´s Encrypted Message (Windows)

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=SSLCert;object=ReceiverKey" -keyform engine -out C:
\localCA\decryptedsignedmessage.txt
```

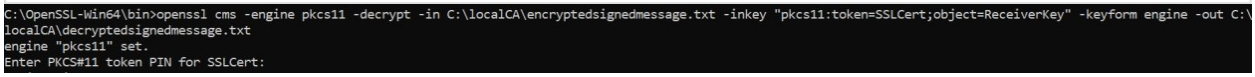

Figure 52: Openssl decrypt command output

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

Using ECC key, only sign and verify operations can be performed with OpenSSL3.Encryption and decryption operation are not supported in this version for ECC Key.

5.2.7 Verify the Signature (Windows)

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

›_ 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 53: Openssl verify command output

2. Verify the signature of signedmessage.txt using the sender's certificate for ECC

›_ Console

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

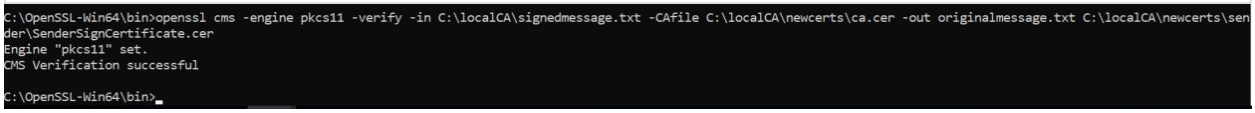

Figure 54: Openssl verify command output

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

This completes the Integration of OpenSSL 3.0 with Utimaco SecurityServer.

5.3 OpenSSL 3 and Libp11 Installation (Windows)

1. Download and install OpenSSL 3, 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
```

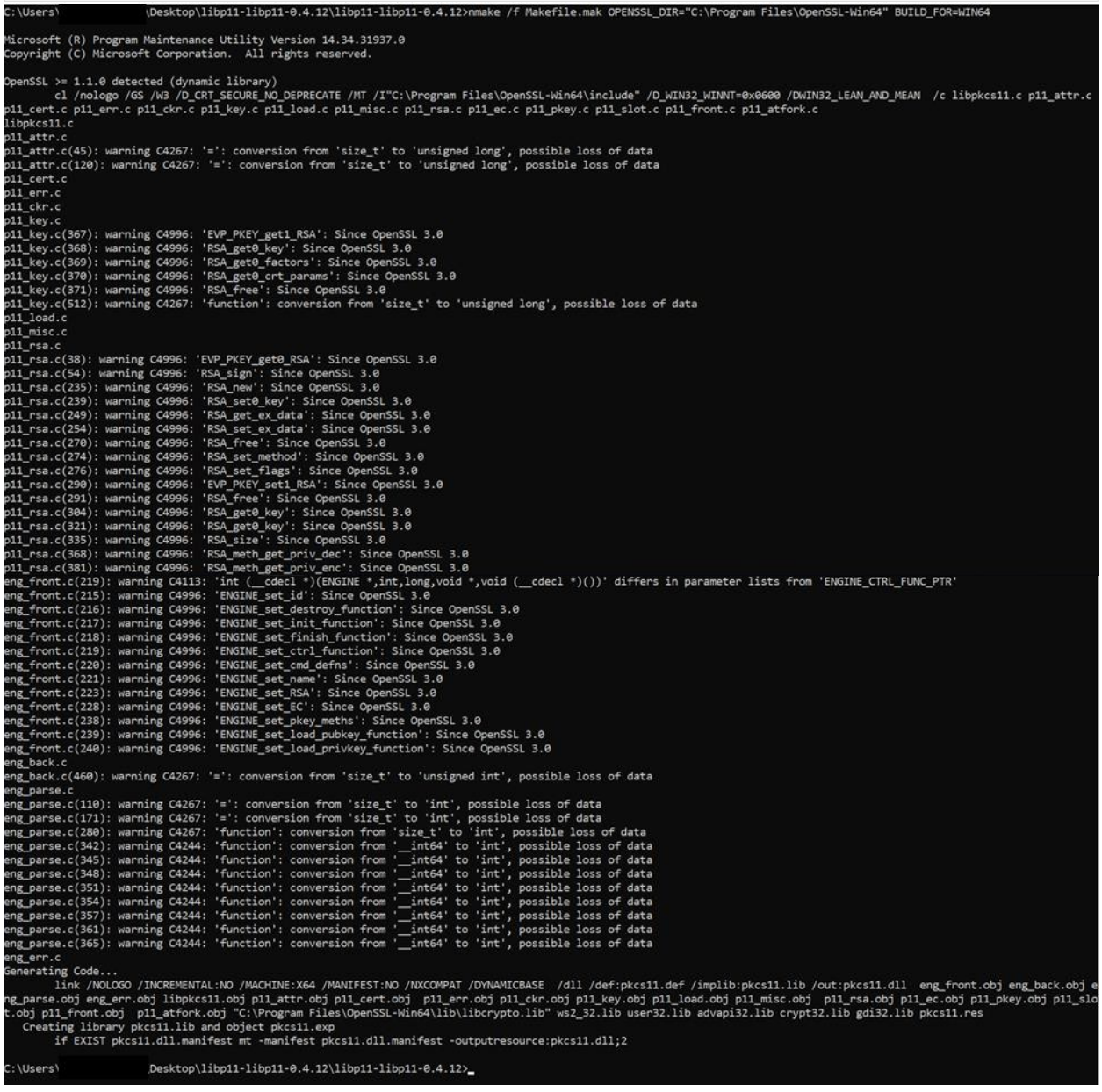

Figure 36: Output of nmake command

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

6 Troubleshooting

Error	*Diagnosis*
openssl engine pkcs11 -v FATAL: Startup failure (dev note: apps_startup()) for openssl 00219D7BFB7F0000:error:13000091:engine routines:dynamic_load:version incompatibility:crypto/engine/eng_dyn.c:481: 00219D7BFB7F0000:error:13000066:engine routines:int_engine_configure:engine configuration error:crypto/engine/eng_cnf.c:139:section=pkcs11_section, name=dynamic_path,value=/usr/local/libp11/lib/pkcs11.so 00219D7BFB7F0000:error:0700006D:configuration file routines:module_run:module initializationerror:crypto/conf/conf_mod.c:270:module=engines,value=engine_section retcode=-1	Version incompatibility found on RHEL 8 and on RHEL 9 it successfully worked

Error	***Diagnosis***
openssl req -engine pkcs11 -new -key "pkcs11:token=FedoraCert;object= CertKey1" -keyform engine -out TestRSACSR.csr Engine "pkcs11" set. Enter PKCS#11 token PIN for FedoraCert: The private key was not found at: pkcs11:token=FedoraCert;object= CertKey1 PKCS11_get_private_key returned NULL Could not read private key from org.openssl.engine:pkcs11:pkcs11:token=FedoraCert;object= CertKey1 002EBC7E1E7F0000:error:40000065:pkcs11 engine:ERR_ENG_error:object not found:eng_back.c:887: 002EBC7E1E7F0000:error:13000080:engine routines:ENGINE_load_private_key:failed loading private key:crypto/ engine/eng_pkey.c:79: Segmentationfault(coredumped)	Check Key exist on the slot and provide correct key name

Error	***Diagnosis***
11.01.2023 11:18:37.059 | [00016138:00016138] open_plugin | I: Opening KeyStorePlugin 'Ephemeral Storage' (config:) 11.01.2023 11:18:37.059 | [00016138:00016138] set_default_plugin_id | I: Set new default KeyStorePlugin 'Internal Storage' 11.01.2023 11:18:37.059 | [00016138:00016138] exec_loadbalanced | W: HSM::ConnectionException(Error::NO_DEVICE_AVAILABLE = 0xbe000007) thrown in select_device Error::NO_DEVICE_AVAILABLE 11.01.2023 11:18:37.059 | [00016138:00016138] enter_failover | I: Entering failover mode (fallback interval: 0 seconds) 11.01.2023 11:18:37.059 | [00016138:00016138] exec_distributed | W: HSM::ConnectionException(Error::NO_DEVICE_AVAILABLE = 0xbe000007)throwninexec_failoverError::NO_DEVICE_AVAILABLE 11.01.2023 11:18:37.059 | [00016138:00016138] reconnect_failed_devices | W: Reconnection attempt failed:Utimaco::HSM::ConnectionException(error_code=0xb901306f)	Make the changes in keystorage section in cs_pkcs11.cfg file and check that the HSM device is running up and reachable from the host.

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

https://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