OpenSSL

v3.4.1

Integration Guide

u.trust GP HSM Se-Series

SecurityServer 6.0.0 and 6.1.1

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-2025-0065

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.

1.2 Target Audience

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

1.3 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

Abbreviation	*Meaning*
IP	Internet Protocol
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 1: List of Abbreviations

1.4 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.

Convention	*Use*	*Example*
`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 2: 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.

2 Product 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 3.4.1	u.trust Anchor cHSM 6.0.0 and 6.1.1	u.trust Anchor Se-Series
RHEL 10	OpenSSL 3.4.1	u.trust Anchor cHSM 6.0.0 and 6.1.1	u.trust Anchor Se-Series
Ubuntu 24.04	OpenSSL 3.4.1	u.trust Anchor cHSM 6.0.0 and 6.1.1	u.trust Anchor Se-Series
Ubuntu 22.04	OpenSSL 3.4.1	u.trust Anchor cHSM 6.0.0 and 6.1.1	u.trust Anchor Se-Series
Windows Server 2022	OpenSSL 3.4.1	u.trust Anchor cHSM 6.0.0 and 6.1.1	u.trust Anchor Se-Series

Table 3: List of Tested Versions

3.2 Hardware and Software Requirements

Hardware	***Hardware Requirements***
Utimaco u.trust Anchor LAN HSM	u.trust Anchor Se-Series cHSM available with firmware SecurityServer 6.0.0 and 6.1.1

Table 4: List of Hardware Requirements

Software	***Software Requirements***
OpenSSL	OpenSSL 3.4.1
Libp11 (Linux)	0.4.14
Libp11 (Windows)	0.4.14
HSM Interface	SecurityServer PKCS#11 Provider

Table 5: List of Software Requirements

3.3 Prerequisites

Before you begin, please ensure that you have:

- A u.trust Anchor cHSM setup and configured. Refer to the u.trust Anchor Se-Series documentation to set up the HSM.
- An MBK created and stored on each cHSM. Refer to the u.trust Anchor Se-Series documentation to set up the MBK.
- The u.trust Anchor cHSM Default Admin replaced with a new admin user for production environments.
- An operating system listed in Tested Versions.
- A SecurityServer version listed in Tested Versions.
- Familiarized yourself with the OpenSSL documents and setup process.

4 Installation and Configuration

4.1 Setting Up u.trust Anchor HSM

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.

If you have purchased an HSM from Utimaco, you will need the associated product bundle containing the required Windows or Linux software packages. This bundle can be downloaded from the Utimaco Support Portal. Please ensure you request access to the product bundle beforehand through the support portal.

4.1.1 Installing SecurityServer Software (Linux)

To install the SecurityServer Software on a Linux system, the following steps must be followed.

1. Copy the downloaded software at the appropriate location on the OpenSSL Server.
2. Create an 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 ./u.trust_anchor_product_bundle-x.x.x/Software/Linux/Crypto_APIs/PKCS11_R3//
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 ./u.trust_anchor_product_bundle-x.x.x/Software/Linux/Administration
cp csadm p11tool2 /opt/utimaco/bin
chmod +x /opt/utimaco/bin/csadm /opt/utimaco/bin/p11tool2
```

4.1.2 Installing SecurityServer Software (Windows)

Install the latest version of the SecurityServer software following the instructions provided in the Section 4.1 of *u.trust Anchor Administration Manual* [UTAADMIN]. For integration with OpenSSL, only the PKCS#11 Application Interface is required, as well as the administration tools.

4.2 Setting Up OpenSSL

4.2.1 Installing OpenSSL and Libp11 (Linux)

If you are using an existing or pre-installed OpenSSL, then skip the first two steps.

1. Install the necessary dependencies.

›_ Console

```
dnf -y update && \
    dnf install -y make gcc wget tar git autoconf automake libtool \
    perl-IPC-Cmd perl-core perl-Test-Harness perl-Data-Dumper \
    which findutils diffutils openssl-devel
```

2. Install Openssl.

›_ Console

```
# From the SO repository
dnf install openssl
# From the official source
cd /tmp
wget https://www.openssl.org/source/openssl-3.4.1.tar.gz && \
    tar -xzvf openssl-3.4.1.tar.gz && \
    cd openssl-3.4.1 && \
    ./Configure linux-x86_64 --prefix=/usr/local/openssl-3.4.1 shared && \
    make -j$(nproc) && \
    make install_sw && \
    cd .. && rm -rf openssl-3.4.1
```

3. Verify the version of OpenSSL.

›_ Console

```
openssl version
OpenSSL 3.4.1 11 Feb 2025 (Library: OpenSSL 3.4.1 11 Feb 2025)
```

4. Install libp11 from the official source.

›_ Console

```
cd /tmp
wget https://github.com/OpenSC/libp11/releases/download/libp11-0.4.14/
libp11-0.4.14.tar.gz && \
    tar -xzvf libp11-0.4.14.tar.gz && \
    cd libp11-0.4.14 && \
    PKG_CONFIG_PATH=/usr/local/openssl-3.4.1/lib64/pkgconfig ./configure --
prefix=/usr/local/libp11 --with-openssl=/usr/local/openssl-3.4.1 && \
    make -j$(nproc) && \
    make install && \
    cd .. && rm -rf libp11-0.4.14*
```

5. Adjust the environment variables.

›_ Console

```
export PATH="/usr/local/openssl-3.4.1/bin:${PATH}"
export LD_LIBRARY_PATH="/usr/local/openssl-3.4.1/lib64:\
/usr/local/libp11/lib:${LD_LIBRARY_PATH}"
```

To ensure these environmental variables are set after a reboot, the last commands should be added to `~/.bashrc`.

4.2.2 Installing OpenSSL and Libp11 (Windows)

1. Download and install OpenSSL, refer to OpenSSL for Windows release page to obtain the OpenSSL installer.
2. Add the OpenSSL `bin/` directory to system `PATH`.
3. Verify OpenSSL version using the following command in a terminal.

›_ Console

```
openssl version
OpenSSL 3.4.1 11 Feb 2025 (Library: OpenSSL 3.4.1 11 Feb 2025)
```

4. Download libp11 from the official libp11 release page and extract its content.
5. Verify pkcs11.dll is available inside `<libp11>/` folder.

5 Integration Steps

5.1 Configuration on u.trust Anchor

5.1.1 Configuration of the PKCS#11 Provider

1. Create the directory `/etc/utimaco`.
2. Locate the Utimaco PKCS#11 configuration file in your SecurityServer directory, `./u.trust_anchor_product_bundle-x.x.x/Software/Linux/Crypto_APIs/PKCS11_R3/sample`. Copy the Utimaco PKCS#11 configuration file `cs_pkcs11_R3.cfg` into the `/etc/utimaco` directory.
3. 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 the HSM

# Device specifier
Device = <PORT@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 = 4001@<HSM IP address> Hardware (LAN) HSM

or

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

To make your testing easier, enable the PKCS#11 log file. You can enable it by editing the Logging Loglevel. Set the LogPath and Logging Loglevel to 1. For testing, you may want to increase them 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, change the Logging to 1 or 2. This will limit the logging to only critical and important messages.

4. Set up the `CS_PKCS11_R3_CFG` environment variable.

```
export CS_PKCS11_R3_CFG=/etc/utimaco/cs_pkcs11_R3.cfg
```

5.1.2 Initializing a PKCS#11 Slot

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

Using the following commands, the SO and User will be initialized.

```
./p11tool2 slot=<slot_no> Label=<token_label> Login=ADMIN,ADMIN.key InitToken=ask
./p11tool2 slot=<slot_no> LoginSO=ask InitPin=ask
```

5.2 Configuration on OpenSSL

5.2.1 Configuring OpenSSL Configuration File

1. Open the file `/etc/pki/tls/openssl.cnf` on Linux or `C:\Program Files\Common Files\SSL\openssl.cnf` on Windows 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.

›_ openssl.cnf

```
[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
```

The dynamic path and module path will be changed according to the user environment.

On Linux, `dynamic_path` must point to `pkcs11.so` . On Windows, it must point to `pkcs11.dll` .

In both environments, `dynamic_path` must point to the SecurityServer PKCS#11 provider.

5.2.2 Verify PKCS#11 Engine

Run the command below to verify if the OpenSSL Engine is available.

›_ Console

```
openssl engine pkcs11 -t
(pkcs11) pkcs11 engine
     [ available ]
```

6 Verification and Testing

6.1 Functional Testing

This section describes common cryptographic procedures for testing the integration between the OpenSSL PKCS#11 engine and the Utimaco HSM.

All the commands are written for a Linux environment. To execute them in a Windows environment, please adjust the paths to their Windows equivalent.

6.1.1 Creating a Key

1. Generate the key using p11tool2.

›_ Console

```
# For RSA
p11tool2 slot=<SLOT_NUMBER> LoginUser=ask PubKeyAttr=CKA_LABEL="CertKey"
 PrvKeyAttr=CKA_LABEL="CertKey" GenerateKeyPair=RSA

# For ECC
p11tool2 slot=<SLOT_NUMBER> LoginUser=ask PubKeyAttr=CKA_LABEL="TestECDSAKey"
 PrvKeyAttr=CKA_LABEL="TestECDSAKey",CKA_DERIVE=CK_TRUE GenerateKeyPair=ECC
```

The PKCS#11 PIN can be directly introduced in the `LoginUser` parameter, but this will be stored in plain text in the command history.

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

›_ Console

```
p11tool2 LoginUser=ask ListObjects
CKO_PUBLIC_KEY:
+ 1.1
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_UNIQUE_ID                 = C92DB9A3-9D2C-4DB7-B217-EDC44BB5966C
  CKA_LABEL                     = CertKey
  CKA_ID                        =
+ 1.2
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_UNIQUE_ID                 = 3B4C290B-46FE-4F66-8E95-C7771A112A45
  CKA_LABEL                     = TestECDSAKey
  CKA_ID                        =
CKO_PRIVATE_KEY:
+ 2.1
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_UNIQUE_ID                 = 02F66F14-0BB7-45B9-9100-74CDF5F71F46
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = CertKey
  CKA_ID                        =
+ 2.2
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_UNIQUE_ID                 = 6153C311-91E7-4CA5-875A-8CC43DF36732
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = TestECDSAKey
  CKA_ID                        =
```

6.1.2 Generate a Certificate from an Existing Key

1. Obtain the key label using p11tool2.

›_ Console

```
p11tool2 LoginUser=ask ListObjects
```

2. Generate a Certificate Signing Request (CSR).

›_ Console

```
openssl req -engine pkcs11 -new -key
"pkcs11:token=<token_label>;object=<key_label>" -keyform engine -out
TestRSACSR.csr
```

Here, `key_label` is the key label on the HSM. Provide the Cryptouser PIN and certificate details when prompted.

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

›_ Console

```
openssl req -engine pkcs11 -new -x509 -days 365 -key
"pkcs11:token=<token_label>;object=<key_label>" -keyform engine -out Test.cert
```

6.1.3 Sign and Verify a Message

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

›_ Console

```
echo "Sample message" > message.txt
```

2. Sign the message file.

›_ Console

```
openssl cms -engine pkcs11 -sign -in message.txt -signer TestRSA.cert - inkey
"pkcs11:token=<token_label>;object=<key_label>" -keyform engine -out
signedRSAmessage.txt
```

The result will be a signed message with the following format.

›_ Signed Message

```
MIME-Version: 1.0
Content-Type: multipart/signed; protocol="application/pkcs7-signature";
micalg="sha-256"; boundary="----4483E8F5EBA4258E73C09280E53170FE"

This is an S/MIME signed message

------4483E8F5EBA4258E73C09280E53170FE
Sample Message

------4483E8F5EBA4258E73C09280E53170FE
Content-Type: application/pkcs7-signature; name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"

MIIGFgYJKoZIhvcNAQcCoIIGBzCCBgMCAQExDTALBglghkgBZQMEAgEwCwYJKoZI
...
EwOn9N1Fh/Y+f4B4zkan/JvlsuCUhma630BdmQm03SGGvyWFkdN/lpLCP2zGPwAZ
JI8M/QyBddmJJHvy/K5zRrxHmkpEDhvYwf0=

------4483E8F5EBA4258E73C09280E53170FE--
```

3. Verify the signature of the message.

›_ Console

```
openssl cms -engine pkcs11 -verify -noverify -in signedRSAmessage.txt - CAfile
TestRSA.cert -out originalmessage.txt TestRSA.cert
Engine "pkcs11" set.
Warning: recipient certificate file parameters ignored for operation other than
-encrypt
Sample Message
CMS Verification successful
```

6.1.4 Encrypt and Decrypt a Message

1. Encrypt the message using the following command:

›_ Console

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

The encrypted message will have the following format:

›_ Encrypted Message

```
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

MIIMFAYJKoZIhvcNAQcDoIIMBTCCDAECAQAxggF5MIIBdQIBADBdMEUxCzAJBgNV
BAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEwHwYDVQQKDBhJbnRlcm5ldCBX
aWRnaXRzIFB0eSBMdGQCFAdGmeMIYrV4twsA8FLt3ZBvn5vdMA0GCSqGSIb3DQEB
...
4QTULf9IDxS7/pxXbOP63ij+jIPB8WRD5hm6uYna/niWxMlEfFlwyBSEK+5VtIRT
z5jrZEplI7a3qsdCYPICWSQWn5oxg1L7uskWBBtwyFrMGHIC9BtMx3wNATix9Ni7
HUoc0oXZyWhjn4jKqdmj2bSrfk2BqU5I
```

2. Decrypt the message with the following command:

›_ Console

```
openssl cms -engine pkcs11 -decrypt -in encryptedRSAsignedmessage.txt -inkey
"pkcs11:token=<token_label>;object=<key_label>" -keyform engine -out
decryptedRSAsignedmessage.txt
```

6.1.5 Creating a Local CA

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

openssl.cnf

```
dir = /localCA
new_certs_dir = $dir/newcerts
```

You can change `dir` to the directory of your choice, but make sure to use the correct path in the subsequent steps. Here, we have created directory `/localCA` under the 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
echo "01" > /localCA/serial
```

4. Create a key pair by using `p11tool2` for root CA.

›_ Console

```
# For RSA
p11tool2 slot=<SLOT_NUMBER> LoginUser=ask PubKeyAttr=CKA_LABEL="CAKey"
 PrvKeyAttr=CKA_LABEL="CAKey" GenerateKeyPair=RSA
# This generates RSA 2048 CA private and public keys on the HSM
# For ECC
p11tool2 slot=<SLOT_NUMBER> LoginUser=ask PubKeyAttr=CKA_LABEL="CAKey"
 PrvKeyAttr=CKA_LABEL="CAKey" GenerateKeyPair=ECC
# This generates ECC CA private and public keys on the HSM
```

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

›_ Console

```
p11tool2 Slot=<SLOT_NUMBER> LoginUser=ask ListObjects
CKO_PUBLIC_KEY:

+ 1.1
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_UNIQUE_ID                 = C92DB9A3-9D2C-4DB7-B217-EDC44BB5966C
  CKA_LABEL                     = CAKey
  CKA_ID                        =

+ 1.2
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_UNIQUE_ID                 = 3B4C290B-46FE-4F66-8E95-C7771A112A45
  CKA_LABEL                     = CAKeyECC
  CKA_ID                        =


CKO_PRIVATE_KEY:

+ 2.1
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_UNIQUE_ID                 = 02F66F14-0BB7-45B9-9100-74CDF5F71F46
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = CAKey
  CKA_ID                        =


+ 2.2
  CKA_KEY_TYPE                  = CKK_ECDSA
  CKA_UNIQUE_ID                 = 6153C311-91E7-4CA5-875A-8CC43DF36732
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_FALSE
  CKA_LABEL                     = CAKeyECC
  CKA_ID                        =
```

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

›_ Console

```
openssl req -engine pkcs11 -new -x509 -days 365 -key
“pkcs11:token=<token_name>;object=CAKey” -keyform engine -out /localCA/newcerts/
ca.cer
```

Here, `CAKey` is the `Object label` for the CA private key on the Utimaco HSM created in Step 5 and, `<token_name>` is the token label. Provide CryptoUser PIN when prompted.

6.1.6 Sign a Certificate using the Local CA

1. Generate a key pair using `p11tool2`.

›_ Console

```
# For RSA
p11tool2 slot=<slot_No.> LoginUser=ask PubKeyAttr=CKA_LABEL="<key_label>"
 PrvKeyAttr=CKA_LABEL="<key_label>" GenerateKeyPair=RSA
# This generates RSA 2048 private and public keys on the HSM
# For ECC
p11tool2 slot=<slot_No.> LoginUser=ask PubKeyAttr=CKA_LABEL="<key_label>"
 PrvKeyAttr=CKA_LABEL="<key_label>" GenerateKeyPair=ECC
# This generates ECC private and public keys on the HSM
```

2. Generate a certificate request.

›_ Console

```
openssl req -engine pkcs11 -new -key
"pkcs11:token=<token_label>;object=<key_label>" -keyform engine -out /localCA/
newcerts/request.csr
```

3. Sign the certificate request for sender by CA.

›_ Console

```
openssl ca -engine pkcs11 -policy policy_anything -cert /localCA/newcerts/ca.cer
-in /localCA/newcerts/request.csr -keyfile
"pkcs11:token=<token_label>;object=<key_label>" -keyform engine -out /localCA/
newcerts/certificate.crt
```

7 Troubleshooting

7.1 Common Issues and How to Resolve Them

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`
	Version incompatibility was found on RHEL 8 and on RHEL 9 it successfully worked.

`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`
	Check that the Key exists on the slot and providethe correct key name.

`11.01.2023 11:18:37.059 | [00016138:00016138] open_plugin| W: HSM::ConnectionException(Error::NO_DEVICE_AVAILABLE = 0xbe000007) thrown in select_device Error::NO_DEVICE_AVAILABLE`	Make the changes in the key storage section in the cs_pkcs11.cfg file and check that the HSM device is running and reachable from the host.

Error	*Diagnosis*
`20.06.2025 07:56:39: C_OpenSession returned Error 0x00000005 (CKR_GENERAL_ERROR)`	Check the PKCS#11 log (`/tmp/cs_pkcs11_R3.log` by default) for errors. If it doesn't exist or doesn't get updated, check if the `CS_PKCS11_R3_CFG` environment variable is set with the location of the PKCS#11 configuration file.

Table 6: List of Errors and their Diagnoses

7.2 Log Locations and Interpretation

The following logs can be reviewed to check for errors:

- Utimaco PKCS#11 provider log: `/tmp/cs_pkcs11_R3.log` by default, configured in the PKCS#11 configuration file.
    - This log contains all the PKCS#11 function calls and attributes.

Error	*Diagnosis*
`20.06.2025 07:51:25.108 | [00000332:00000332] C_OpenSession| E: HSM::ConnectionException(Error::NO_CONNECTION = 0xbe000015) thrown in open_session Error::NO_CONNECTION`	The provider couldn't connect to the HSM. Please check the PKCS#11 configuration file and ensure the Device section is correctly configured. See Configuration of the PKCS#11 Provider for details.

Error	***Diagnosis***
`20.06.2025 07:10:30.866 | [00000059:00000059] C_GetSlotInfo | E: Error CKR_DEVICE_REMOVED occurred.`	The device was disconnected during the cryptographic operation. This probably happens because of connection issues.

Table 7: List of Common Errors on the PKCS#11 Log File

8 Appendices

8.1 References

Reference	*Title*	*Document Number*
[CSADM]	u.trust Anchor – csadm Manual / Utimaco IS GmbH	2021-0037
[UTAADMIN]	u.trust Anchor - Administration Manual / Utimaco IS GmbH	2020-0035
[UTAP11Tool2]	u.trust Anchor - PKCS#11 p11tool2 Reference Manual / Utimaco IS GmbH	2021-0072

Table 8: References

8.2 Command Summary

Task	*Command*
Create a RSA Key	`p11tool2 slot=$PKCS11_SLOT LoginUser=$PKCS11_PIN PubKeyAttr=CKA_LABEL=$KEY_LABEL PrvKeyAttr=CKA_LABEL=$KEY_LABEL GenerateKeyPair=RSA`
Create an ECC Key	`p11tool2 slot=$PKCS11_SLOT LoginUser=$PKCS11_PIN PubKeyAttr=CKA_LABEL=$KEY_LABEL PrvKeyAttr=CKA_LABEL=$KEY_LABEL GenerateKeyPair=RSA`
Generate a Certificate Signing Request from a key	`openssl req -engine pkcs11 -new -key "pkcs11:token=$PKCS11_TOKEN;object=$KEY_LABEL;pin-value=$PKCS11_PIN" -keyform engine [-subj "/CN=.../"] -out test.csr`

Task	*Command*
Generate a certificate from a Certificate Signing Request	`openssl req -engine pkcs11 -new -x509 -days 365 -key "pkcs11:token=$PKCS11_TOKEN;object=$KEY_LABEL;pin-value=$PKCS11_PIN" -subj "/CN=.../" -keyform engine -out test.cert`
Generate an encrypted message from a file using a certificate	`openssl cms -engine pkcs11 -encrypt -in message.txt -out encrypted_message.txt test.cert`
Decrypt an encrypted message using a private key	`openssl cms -engine pkcs11 -decrypt -in encrypted_message.txt -inkey "pkcs11:token=$PKCS11_TOKEN;object=$KEY_LABEL;pin-value=$PKCS11_PIN" -keyform engine -out decrypted_message.txt`
Generate a signed message from a file using a private key	`openssl cms -engine pkcs11 -sign -in message.txt -signer test.cert -inkey "pkcs11:token=$PKCS11_TOKEN;object=$KEY_LABEL;pin-value=$PKCS11_PIN" -keyform engine -out signed_message.txt`
Verify a signed message using a certificate	`openssl cms -engine pkcs11 -verify -noverify -in signed_message.txt -CAfile test.cert -out verified_message.txt`
Generate a CA from a private key	`openssl req -engine pkcs11 -new -x509 -days 365 -key "pkcs11:token=$PKCS11_TOKEN;object=$KEY_LABEL;pin-value=$PKCS11_PIN" -subj "/CN=.../" -keyform engine -out ca.cer`

Task	***Command***
Sign a Certificate Signing Request using a CA private key	`openssl ca -batch -engine pkcs11 -policy policy_anything -cert ca.cer -in request.csr -keyfile "pkcs11:token=$PKCS11_TOKEN;object=$KEY_LABEL;pin-value=$PKCS11_PIN" -keyform engine -out certificate.crt`

Table 9: List of Commands