OpenSSL

v3.0

Integration Guide

CryptoServer HSM

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	2026-08-06
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 CryptoServer HSM on Linux

4.1 Installing and Configuring CryptoServer HSM (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

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
```

Figure 2: Output of configure command for libp11

4.4 Configuring OpenSSL to Use CryptoServer HSM (Linux)

4.4.1 Setting up CryptoServer HSM 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
```

Figure 3: Verification of pkcs11 engine

4.4.3 Test OpenSSL Functionalities with CryptoServer HSM (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.

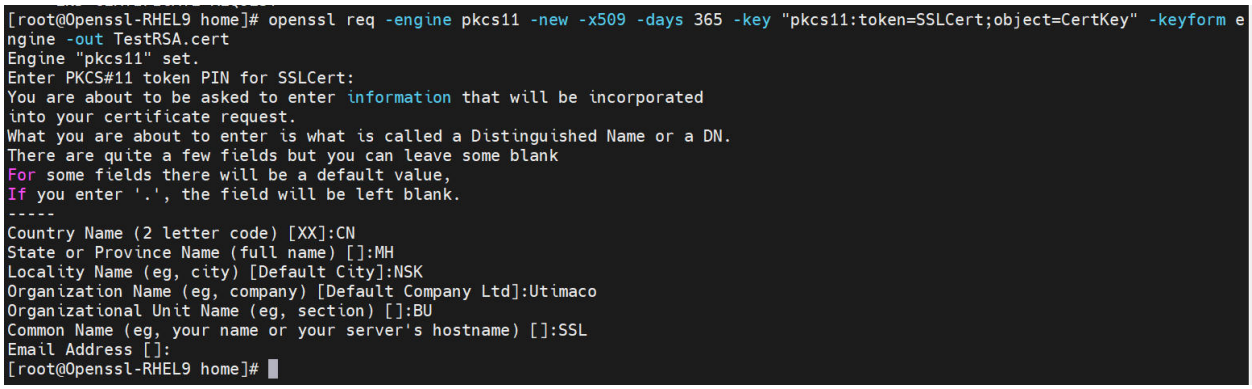
Figure 4: Certificate request output

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
```



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

Figure 7: Content of self-signed certificate file

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

```
›_ Console
```

```
# touch message.txt
```

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.

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
```

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

```
[root@Openssl-RHEL9 libp11-0.4.12]# cat decryptedRSAsignedmessage.txt
MIME-Version: 1.0
Content-Type: multipart/signed; protocol="application/pkcs7-signature"; micalg="sha-256"; boundary="----3F6C85D2944BD7C9A5142C6E
5CB64758"

This is an S/MIME signed message

------3F6C85D2944BD7C9A5142C6E5CB64758
Welcome to Utimaco Team!!!

------3F6C85D2944BD7C9A5142C6E5CB64758
Content-Type: application/pkcs7-signature; name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"

MIIGRgYJKoZIhvcNAQcCoIIGNzCCBjMCAQExDTALBglghkgBZQMEAgEwCwYJKoZI
hvcNAQcBoIIDjzCCA4swggJzoAMCAQICFGjADg2DqB1F4jzvc5lZIauuP/iHMA0G
CSqGSIb3DQEBCwUAMFUxCzAJBgNVBAYTAklOMQswCQYDVQQIDAJNSDENMAsGA1UE
BwwEUHVuZTEQMA4GA1UECgwHVXRpbWFjbzELMAkGA1UECwwCQlUxCzAJBgNVBAMM
AkNOMB4XDTIzMDExMTExMTExOFoXDTI0MDExMTExMTExOFowVTELMAkGA1UEBhMC
SU4xCzAJBgNVBAgMAk1IMQ0wCwYDVQQHDARQdW5lMRAwDgYDVQQKDAdVdGltYWNv
MQswCQYDVQQLDAJCVTELMAkGA1UEAwwCQ04wggEiMA0GCSqGSIb3DQEBAQUAA4IB
DwAwggEKAoIBAQClOunudmyK0AW1ePxEsYssfTQUAWfPUmneHQnoKNrWDP5dWieh
X6bRGjrUXMs4oWNKZ8rcKC9opfyjZbg+A86y7asgX2+DBL9Arriu4QUifJ4lazJa
EqRY/9fFDebHV/1i6F6UH9QSmY/qIkbkXIdDa1FSxGdDphM67eHNy+s3msiFlICw
xgDCNkBGbWI1hTEfe5FGe9dizC2eCmKFZpRnKcDUrAnQIQfe1frtsKDzcludMikJ
T1ZfL8jy4p1QfTiMAkOHUO/OzSkZr8FaLZqYYrwlcIuYXZFx3rOFNRKHtTuV7l8l
GWg3mKUSC+jBMprkHolDFXFDmAyaq/oYJ4pDAgMBAAGjUzBRMB0GA1UdDgQWBBQ2
+99MUDuX2Q1prf3m41OwTd3rFDAfBgNVHSMEGDAWgBQ2+99MUDuX2Q1prf3m41Ow
Td3rFDAPBgNVHRMBAf8EBTADAQH/MA0GCSqGSIb3DQEBCwUAA4IBAQBNR4pU7zlt
iUWpDS7boPS85i4AAfLDYUlTN4AKXvJw2ojI6l6arzKAU41yAxgW3a5lmRyQ6V/D
p4wxs/EJYIxQUT/NqlHG8ykSugNYiDS5sC079unIjKzASz7gHTtJu052rLN7gfMM
V4FJTWEewsvt3K1pjUmAf4FymVMWuzITExGevZ1iQDCLptzV34lyFKdp+k5URGuJ
```

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
```

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.

Figure 16: Content of self-signed certificate

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

```
›_ Console
```

```
# touch message.txt
```

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.

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
```

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
```

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
```

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

Figure 23: CA certificate generation output Sender ECC Key List

For RSA

›_ Console

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

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
```

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

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
```

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
```

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
```

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
```

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
```

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
```

Figure 34: Openssl verify command output

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

Figure 35: Content of original message file

5 Integrating OpenSSL 3.0 with CryptoServer HSM on Windows

5.1 Installing and Configuring CryptoServer HSM 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 CryptoServer HSM (Windows)

5.2.1 Setting up CryptoServer HSM 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
```

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
```

Figure 40: CA RSA Key list

For ECC

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
```

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
```

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
```

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