HSM Configuration

Table of Contents

Configuring HSMs

The ADSS Server Installation Guide provides information on how to configure some of the supported HSMs. Generally, as long as there is a clear installation guide from the HSM manufacturer then the process is easy, simply enter the HSM specific PKCS#11 driver library name, Click "Fetch Slots", select the appropriate slot, enter the passphrase and the HSM should connect.

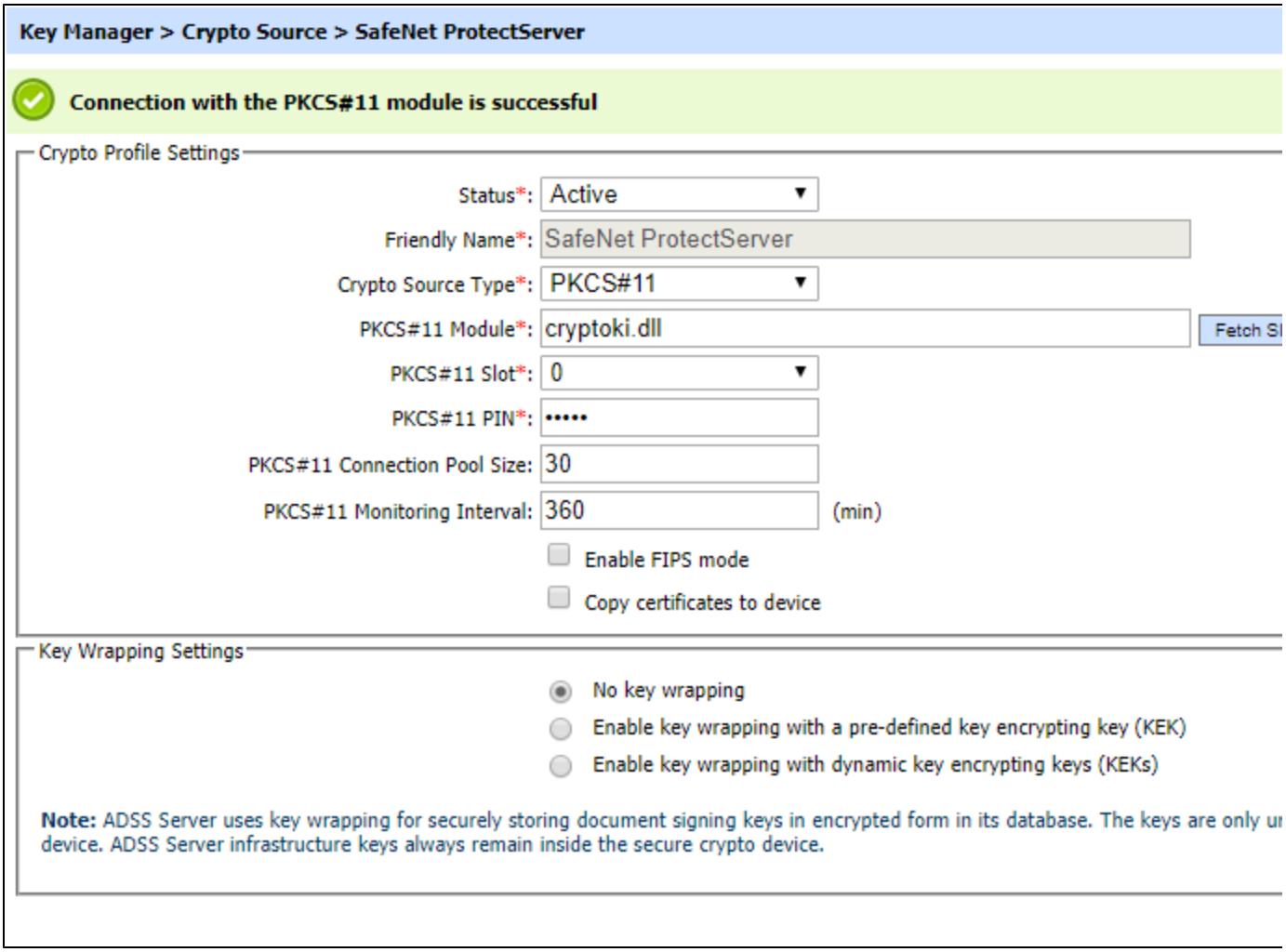
To check an HSM is functioning appropriately, a PKCS#11 interop utility is provided with ADSS Server to test PKCS#11 based interoperability and report on whether all possible algorithms and key lengths used by ADSS Server features are supported.

Gemalto ProtectServer - Important Notes on Key Generation

This FAQ describes the process of generating keys on a Gemalto ProtectServer HSM via ADSS Server. To generate keys the device must be in NORMAL mode. Note NORMAL mode represents a physical connection to a slot on the HSM as opposed to High Availability (HA) or Work Load Distribution (WLD). If you only have one HSM the mode will be NORMAL. Only when there are at least two HSMs in a cluster, you change the mode of operation to WLD or HA.

This guide will take you through the process of checking the current mode, changing the mode if required, replicating any generated keys and then finally placing the HSM mode of operation back to HA or WLD.

In this guide it is assumed that the ProtectServer HSM has already been added as a Crypto Source on ADSS Server, and the "Test Connection" was successful:



Note
If you are using ProtectServer software emulator some of these commands and processes are not applicable. The emulator cannot be used in HA or WLD mode.

ProtectServer supports both HA and WLD modes of operation. Although both provide clustering they have subtle differences in their functionality.Consult the ProtectServer administration guide to check the full details. The definition of the modes is outside the scope of this document. In addition, in order to replicate keys from one HSM to another, the two must have a trust relationship enabled. Consult the documentation for instructions on how to do this.

Step 1: Check if the HSM is in HA, WLD or NORMAL mode

WINDOWS
Right-click on your Start button and select "Run". Now type **regedit** and press enter to open the Registry Editor. To check if the HSM is in HA, WLD or NORMAL mode go to the following registry entry:
"HKEY_LOCAL_MACHINE\SOFTWARE\SafeNet\PTKC\WLD\".

- If the HSM is in HA mode it will read: "ET_PTKC_GENERAL_LIBRARY_MODE=**HA**".
- If the HSM is in NORMAL mode it will read "ET_PTKC_GENERAL_LIBRARY_MODE=**NORMAL**".
- If the HSM is in WLD mode it will read "ET_PTKC_GENERAL_LIBRARY_MODE=**WLD**"

LINUX
To check if the HSM is in HA, WLD or NORMAL mode open the following file:
"/etc/default/et_pktc".

- If the HSM is in HA mode it will read: "ET_PTKC_GENERAL_LIBRARY_MODE=**HA**".
- If the HSM is in NORMAL mode it will read "ET_PTKC_GENERAL_LIBRARY_MODE=**NORMAL**".
- If the HSM is in WLD mode it will read "ET_PTKC_GENERAL_LIBRARY_MODE=**WLD**"

If the HSM is in NORMAL mode, you can proceed with generating keys via ADSS Server. If your HSM is in either HA or WLD mode continue to step 2.

Note
If your HSM is in NORMAL mode and you have any issues generating keys run the PKCS#11 utility and contact support with results. Instructions can be found here - PKCS#11 Utility

Step 2: Set the HSM to NORMAL mode

WINDOWS
Right-click on your Start button and select "Run". Now type **regedit** and press enter to open the Registry Editor. Edit the following registry entry:

- "HKEY_LOCAL_MACHINE\SOFTWARE\SafeNet\PTKC\WLD\".
- Change String value "ET_PTKC_GENERAL_LIBRARY_MODE=**HA** or **WLD**"
- to "ET_PTKC_GENERAL_LIBRARY_MODE=**NORMAL**"

LINUX
Open the following file:

- "/etc/default/et_pktc".
- Change string value "ET_PTKC_GENERAL_LIBRARY_MODE=**HA** or **WLD**"
- to "ET_PTKC_GENERAL_LIBRARY_MODE=**NORMAL**"

Once you have made these changes you must restart the following Windows Services or Linux daemons:

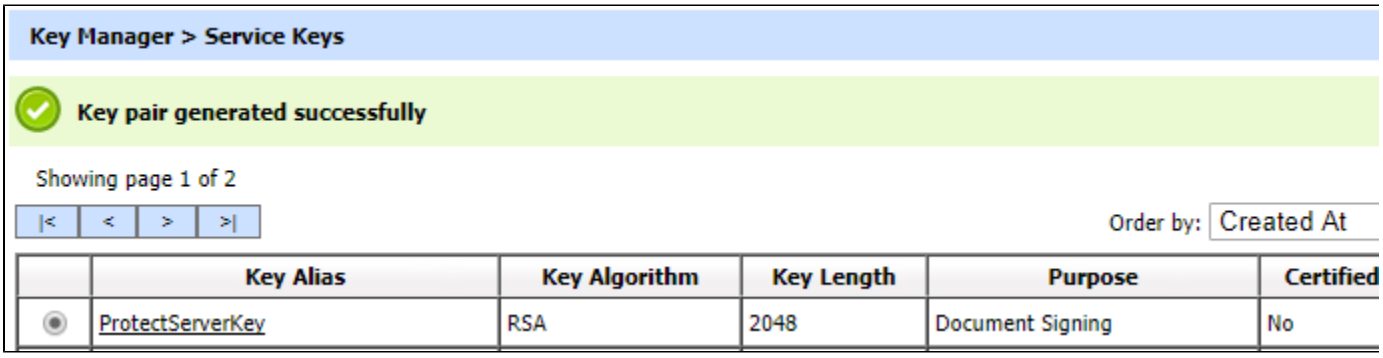
WINDOWS
Right-click on your Start button and select "Run". Type **services.msc** and press enter to open the **Services Manager**. Stop all services and restart them. Order for restart must be **Core** first followed by **Service** and **Console**:

- **Ascertia-ADSS-Console**
- **Ascertia-ADSS-Core**
- **Ascertia-ADSS-Service**

LINUX
Open a **Terminal** session (keyboard command **ctrl+alt+t**). Run the following commands:

- **service tomcatd-ADSS-core restart**
- **service tomcatd-ADSS-console restart**
- **service tomcatd-ADSS-service restart**

Now the HSM is now in NORMAL mode. You should now be able to generate a key pair successfully via ADSS Server.



If you require the HSM to run in HA or WLD mode you now need to go through the process of replicating that key pair to the other HSM(s), before putting the HSM back into HA or WLD mode. If this is required, continue to step 3.

Note
If you have any issues generating keys at this point run the PKCS#11 utility and contact support with results. Instructions can be found here - PKCS#11 Utility

Step 3: Replicate key(s)

Example: Replicate the key(s) from slot 0 to slot 3, Slot 0 being HSM #1 (Location of ADSS Server generated keys) and slot 3 being HSM #2 - the destination slot. Run the following command:

- ctkmu rt -s0 -d3
- rt=replicate token command
- -s0=slot to copy from

- -d3=slot to copy to

If you wish to view the keys(s) on slots to confirm they have been replicated correctly, use the following command:

View keys on Slot 0 and Slot 3:

- ctstat -s0 -j
- ctstat -s3 -j

Now the keys have been replicated from HSM #1 to HSM #2 place the HSM back into HA or WLD mode. To do this, follow Step 4:

Step 4: Put HSM back into HA or WLD mode

WINDOWS
Right-click on your Start button and select "Run". Now type **regedit** and press enter to open the Registry Editor. To check if the HSM is in HA, WLD or NORMAL mode go to the following registry entry:

- "HKEY_LOCAL_MACHINE\SOFTWARE\SafeNet\PTKC\WLD\".
- Change string value "ET_PTKC_GENERAL_LIBRARY_MODE=**NORMAL**"
- to "ET_PTKC_GENERAL_LIBRARY_MODE=**HA**"
- or "ET_PTKC_GENERAL_LIBRARY_MODE=**WLD**"

UNIX
Open the following file:

- "/etc/default/et_pktc".
- Change string value "ET_PTKC_GENERAL_LIBRARY_MODE=**NORMAL**"
- to "ET_PTKC_GENERAL_LIBRARY_MODE=**HA**"
- or "ET_PTKC_GENERAL_LIBRARY_MODE=**WLD**"

Once you have made these changes you must restart the Windows Services or Linux daemons:

WINDOWS
Right-click on your Start button and select "Run". Type **services.msc** and press enter to open the **Services Manager**. Stop all services and restart them. Order for restart must be **Core** first followed by **Service** and **Console**:

- **Ascertia-ADSS-Console**
- **Ascertia-ADSS-Core**
- **Ascertia-ADSS-Service**

LINUX
Open a **Terminal** session (keyboard command **ctrl+alt+t**). Run the following commands:

- **service tomcatd-ADSS-core restart**
- **service tomcatd-ADSS-console restart**
- **service tomcatd-ADSS-service restart**

How can I turn on debug level logging for the Gemalto SafeNet Luna SA HSM?

As a pre-requisite, the Gemalto SafeNet SDK must already be installed. Modify the **crystoki.ini** file to look like this:

```
[Chrystoki2]
LibNT=C:\Program Files\LunaSA\cklog201.dll
[CkLog2]
Enabled=1
NewFormat=1
File=c:\luna.txt
Error=c:\lunaerr.txt
LibNT=c:\program files\lunasa\cryptoki.dll
```

- Now login to the ADSS Server Console and go to **Key Manager > Crypto Devices** module
- Edit the respective crypto device and change the library name to **cklog201.dll** instead of **cryptoki.dll**
- Restart the ADSS Server Services from Windows Service Panel/Unix daemon for the changes to take effect.

How can I generate an HSM resident KEK?

KEKs are used by ADSS Server to support key export. Some HSMs do not support key wrapping and export, the Luna SA-CL (clone) version is one such example, the Luna SA Export version must be used.
Use the ADSS Server PKCS#11 utility to check the target HSM supports KEK generation and key export functionality.

Windows OS

1. Launch windows command prompt as administrator
2. Change Directory (cd) to [ADSS Server Home]\util\bin\
3. Type the following command to run the **genkek_pkcs11.bat** according to the syntax: genkek_pkcs11 [Name] [PKCS#11 module library] [Slot id] [PIN] [FIPS mode] [Log file path] [Key label]

genkek_pkcs11.bat

```
genkek_pkcs11.bat HSMalias cryptoki.dll 0 password false C:\adss\pkcs11.log
TestKEK
```

UNIX OS

1. Launch the Terminal.
2. Change Directory (cd) to [ADSS Server Home]\util\bin\

Type the following command to run the **genkek_pkcs11.sh** utility according to the syntax: **./**genkek_pkcs11 [Name] [PKCS#11 module library] [Slot id] [PIN] [FIPS mode] [Log file path] [Key label]

genkek_pkcs11.sh

```
./genkek_pkcs11.sh LunaHSM cryptoki.so 0 password false /home/adss/pkcs11.log TestKEK
```

How to import a PFX/PKCS#12 file into a Luna HSM for use by ADSS Server?

This technote describes how to import a PFX file into a Luna HSM for use by ADSS Server. This description only details the HSM operations. Once completed, refer to the standard ADSS Server manual for instructions on how to search and use the key and associated certificate.

To import a PFX into Luna HSM, there are three parts: (1) private key import, (2) certificate import, and (3) binding the two objects together.

A PFX contains a private key and associated public certificate. Once the information has been imported into Luna HSM, it is important to either securely delete or store the PFX. The HSM should have a suitable backup option.

1. **Import Private Key**

On the ADSS Server host, open a command prompt with suitable privileges and change the folder/directory to the SafeNet Luna Client home directory. On Windows the default is C:\Program Files\SafeNet\LunaClient, for example. Once you're at the SafeNet Luna Client home directory, run the following command:

Command Prompt

```
cmu importkey -PKCS12 -in <PFX file name>.pfx -keyalg RSA
```

2. Import the public certificate

First export the certificate from the PFX file. There are numerous ways to achieve this, which are not documented here. One example is to import the PFX into the Windows Security Store. Note upon doing this remember to securely delete the entry as to prevent unauthorized access to the private key.

From the same command prompt as used in point one above, run the following command:

Command Prompt

```
cmu import -inputFile=<Certificate file name>.cer
```

The import operation creates the new object inside the HSM partition with the “CKR_PRIVATE” attribute set to “true”. This must be set to “false”. To do this use the CKDEMO application (found in the same directory as cmu utility):

a. Start CKDEMO and open a session as a normal user (option 1)
b. Log in as the crypto officer (option 3)
c. Find objects (26) and search for the public key (option 4)
d. Copy the public key (option 21) with the handle from step 3
e. Select add attribute and change CKR_PRIVATE from 01 (true) to 00 (false)
f. Destroy the old object (option 22).

3. Bind the imported objects

To bind the two objects inside the HSM for ADSS Server, you must set the ID attribute of both the private key and public certificate. Remember to bind the new object created in step two above with CKDEMO.

First, check for a unique number to set this ID attribute to. You can do this by selecting a random number for use and checking no current object has this value assigned. Do this by running the command “*cmu list -id=<id number>*”. If the command returns no results, then the ID number tested is not currently used.

Proceed to bind the private key and public certificate. From the command line set the ID attribute to the random number just tested, by running following command for both objects

Command Prompt

```
cmu setAttribute -handle=<object handle number> -id=<id number>
```

The key and certificate are now ready for use by ADSS Server. Remember to add the necessary chain of trust of CA certificates to ADSS Server Trust Manager before the key is used.

Initializing the Utimaco CryptoServer 4 Simulator

The following are the instructions to configure and initialize the Utimaco CryptoServer 4 simulator on Windows:

1. Install the Utimaco CryptoServer 4 using its installer
2. Go to location **C:\Program Files\Utimaco\CryptoServer\Lib** and edit the **cs_pcs11_R2.cfg** file in a text editor:
    a. change the IP under **Device = 3001@192.178.213.201** (This is the IP where Utimaco is running. It is running on the localhost, you can even set the value 3001@127.0.01)
    b. change the **ConnectionTimeout = 5000**
    c. change the **KeepAlive = true**
3. Start/Restart the HSM by launching the **CryptoServer Simulator** for the changes to take effect
4. To initialize the HSM, launch cmd and go to **C:\Program Files\Utimaco\CryptoServer\Administration** and run the command **p11tool2 Login=ADMIN,ADMIN.key InitToken=654321** for initialization
5. To initialize the slot run the command: **p11tool2 LoginSO=654321 InitPIN=123456**
6. Restart the machine
7. Start the the **CryptoServer Simulator**
8. To configure it in the ADSS Server, use the library name **C:\Program Files\Utimaco\CryptoServer\Lib\cs_pkcs11_R2.dll**

Initializing the SafeNet Emulator 5.2

The following are the instructions to configure and initialize the SafeNet Emulator 5.2 on Windows:

1. Install the **PTKcpsdk.msi** from location **SafeNet Emulator\610-009981-014_SW_PTK_5.2_Client_RevA\SDKs\Win64\ptkc_sdk**
2. Additionally install the following applications if the Emulator is installed on a network:
    a. Install the **PTKnethsm.msi** from location **SafeNet Emulator\610-009981-014_SW_PTK_5.2_Client_RevA\SDKs\Win64\network_hsm_access_provider**
    b. Install the **PTKcprt.msi** from location **SafeNet Emulator\610-009981-014_SW_PTK_5.2_Client_RevA\SDKs\Win64\ptkc_runtime**

3. Once installation is completed, run the **cmd**
4. To create a new slot, run the command: **ctconf -c1** (where 1 is slot number)
5. To initialize any of the created slot, run the command: **ctconf -n1**
6. To re-initialize any of the created slots, run the command: **ctconf -r1**
7. To change the user password of any of the created slot, run the command: **ctkmu p -sx**
8. To see the details and status of the all created slots, run the command: **ctstat**
9. To configure it in the ADSS Server, use the library name **cryptoki.dll**

Configuring the Utimaco CP5 HSM Simulator on Linux

Follow these steps to configure the Utimaco CP5 HSM Simulator on Linux. It is assumed that the simulator is already installed:

Starting HSM

- Start the simulator using the following command: *~/Utimaco/simulator/Linux/sim5_linux/bin/cs_sim.sh*
  Note: The simulator by default runs on 3001 port

Configure the simulator in CC mode

- Export the HSM public key: *csadm Dev=3001@127.0.0.1 GetHSMAuthKey>/home/Linux2/x86-64/Administration/HsmAuthKey.key*
- Now this HSM public key & P11 config must be set inside the machine's environment variable using following commands:
  - *export CS_AUTH_KEYS=/home/Linux2/x86-64/Administration/HsmAuthKey.key*
  - Create a PKCS#11 **Configuration File** (see below for instructions) and set it to: *CS_PKCS11_R2_CFG=**<FILE_PATH>**/cs_pkcs11_R2.cfg i.e. export CS_PKCS11_R2_CFG=/opt/linux/x86-64/Crypto_APIs/pkcs11_r2/lib/cs_pkcs11_R2.cfg*

Creating Users

Create two users and these must use a password based authentication which is then set inside ADSS Server. Here two users are added with different rights because we need admin rights on the slot to allow ADSS Server to generate keys and use them for signing. **Note:** Use a strong password

- *./csadm Dev=3001@127.0.0.1 LogonSign=ADMIN,**<FILE_PATH>**/ADMIN.key AddUser=**SO_0000,00000200**{CXI_GROUP=SLOT_0000},hmacpwd,password*
- *./csadm Dev=3001@127.0.0.1 LogonSign=ADMIN,**<FILE_PATH>**/ADMIN.key AddUser=**USR_0000,000000**22{CXI_GROUP=SLOT_0000},hmacpwd,password* // This password is configured in ADSS Server
- Create a third user to allow key restore: *./csadm Dev=3001@127.0.0.1 LogonSign=ADMIN,**<FILE_PATH>**/ADMIN.key AddUser=**USR1_0000,000000**22{CXI_GROUP=SLOT_0000},hmacpwd,password*
  Note: The restore key function of Utimaco requires two login on the slot hence we create another user and assign admin rights to it. This user is also configured inside ADSS Server.

Testing the Simulator/HSM

Once the Simulator is running in CC mode you can test its commands:

- *./csadm Dev=3001@127.0.0.1 GetState* // Tally this with Appendix below
- *./csadm Dev=3001@127.0.0.1 ListUser* //The above 3 users will be shown here
- *./csadm Dev=3001@127.0.0.1 LogonSign=ADMIN, /home/Linux2/x86-64/Administration/key/ADMIN.key* **(Note:** This command confirms that the simulator or HSM is running in CC mode)

Troubleshooting

Some times the simulator gets stuck or not performing as intended. In this case run the following commands:

- *./csadm Dev=3001@127.0.0.1 **ResetToBL***
- *./csadm Dev=3001@127.0.0.1**RecoverOS***
- *./csadm Dev=3001@127.0.0.1 LogonSign=ADMIN,/home/Linux2/x86-64/Administration/key/ADMIN.key **ClearAuditLog***

To get the P11 log, see the CFG file **Logpath** parameter and check those logs. If required, change the Log level to get more details. Most of the time you can find the root cause looking at this file.

You can also get more logs from the simulator/HSM. To get the audit log for debugging run this command:

- *./csadm Dev=3001@127.0.0.1 LogonSign=ADMIN,./home/Linux2/x86-64/Administration/key/ADMIN.key **GetAuditLog** > ./auditlog.txt*
  Important: As HSM/Simulator runs, it generates audit logs. CP5 creates 10 files with 240000 bytes. Log rotation is off and can't be

turned ON. This means administrator must export these logs to avoid HSM returning error. See **CryptoServer Se-Series Gen2 CP5 Administration Manual** > 8.12 - see the 3rd command above

Configuration File

The cs_pkcs11_R2.cfg file is important for Utimaco P11 library to communicate with the HSM. This must be placed any where on the machine and referenced by pointed by **CS_PKCS11_R2_CFG** env variable, as mentioned above:

```
cs_pkcs11_R2.cfg
[Global]
# Path to the logfile (name of logfile is attached by the API)
# For unix:
Logpath = /tmp
# For windows:
#Logpath = D:/HSMLogs

# Loglevel (0 = NONE; 1 = ERROR; 2 = WARNING; 3 = INFO; 4 = TRACE)
Logging = 1
# Maximum size of the logfile in bytes (file is rotated with a backupfile if full)
Logsize = 100mb

# Created/Generated keys are stored in an external or internal database
KeysExternal = false

# If true, every session establishs its own connection
SlotMultiSession = true

# if true, key handles of created/generated keys are random
RandomizeKeyHandles = true

# Maximum number of slots that can be used
SlotCount = 10

# If true, leading zeroes of decryption operations will be kept
KeepLeadZeros = false

# Configures load balancing mode ( == 0 ) or failover mode ( > 0 )
FallbackInterval = 0

# Prevents expiring session after inactivity of 15 minutes
KeepAlive = true

# Timeout of the open connection command in ms
ConnectionTimeout = 120000

# Timeout of command execution in ms
CommandTimeout = 60000

[CryptoServer]
# Device specifier (here: CryptoServer is CSLAN with IP address 192.168.0.1)
Device = TCP:3001@127.0.0.1

#[Slot]
# Slotsection for slot with number 0
#SlotNumber = 0
```

Appendix

> ***administrator@localhost p11]$ ./p11tool2 Version***

```
p11tool2                                                    2.0.14
p11adm_R2                                                   2.1.14
CryptoServer PKCS#11 R2 CP5 #02 2.50 (Sep 4 2018)
cxi_api 1.7.8 (Sep 4 2018)
csadm_lib (global) 3.4.5
csapi 1.10.3 (Sep 4 2018)
csxapi 1.8.5 (Sep 4 2018)
pp_api 1.7.3 (Sep 4 2018)
yacl 1.10.3
sdb 2.1.4 (Sep 4 2018 14:42:31)
copa 1.1.4
sl 1.1.3
```

[administrator@localhost Administration]$ ./csadm ListUser

```
Name Permission Mechanism Attributes
ADMIN 22000000 RSA sign Z[0]
ADMIN01 00000030 HMAC passwd A[CXI_GROUP=SLOT_0000]
ADMIN1 03000000 HMAC passwd A[CXI_GROUP=SLOT_0000]
ADMIN3 02000000 HMAC passwd Z[0]A[CXI_GROUP=SLOT_0000]
ADMIN4 02000000 HMAC passwd Z[0]A[CXI_GROUP=SLOT_0000]
ADMIN_INST 22000000 RSA sign Z[0]
SO_0000 00000200 HMAC passwd A[CXI_GROUP=SLOT_0000]
USR1_0000 00000022 HMAC passwd A[CXI_GROUP=SLOT_0000]
USR_0000 00000022 HMAC passwd A[CXI_GROUP=SLOT_0000]
```

administrator@localhost Administration]$ ./csadm GetState

```
mode = Operational Mode
state = INITIALIZED (0x00100004)
temp = 44.1 [C]
alarm = OFF
bl_ver = 5.01.4.0 (Model: Se-Series Gen2)
hw_ver = 5.01.4.0
uid = 9f000001a aceed701 |
adm1 = 53653135 30302020 43533637 30313135 | Se1500 CS670115
adm2 = 43727970 746f5320 43503520 54657374 | CryptoS CP5 Test
adm3 = 494e5354 414c4c45 44202020 20202020 | INSTALLED
```

[root@localhost Administration]# ./csadm Version

```
csadm 2.3.2
csadm_lib (global) 3.4.5
csapi 1.10.3 (Sep 4 2018)
pp_api 1.7.3 (Sep 4 2018)
sl 1.1.3
yacl 1.10.3
```

[root@localhost Administration]# ./csadm Dev=/dev/cs2.0 GetBootLog

```
SMOS CP5 Ver. 5.5.9.2 (Sep 4 2018) started [0]
FPGA Ver. 5.1.0.8
Hardware Rev. 5.1.4.0
Compiler Ver. 7.4.8
AIS31 compliant TRNG [strict]
Sensory Controller Ver. 2.0.0.31 [0/0]
Real Random Number Generator initialized with:
RESEED_INTERVAL = 1000
PREDICTION_RESISTANCE = 0
REALRANDOM_SHARE = 3
Pseudo Random Number Generator initialized with:
RESEED_INTERVAL = 1000
PREDICTION_RESISTANCE = 0
REALRANDOM_SHARE = 0
Signed Licence File found: se1500.slf
CMDS: no TPS limit
module 0x89 (HASH) initialized successfully
module 0x83 (CMDS) initialized successfully
module 0x86 (UTIL) initialized successfully
module 0x81 (VDES) initialized successfully
module 0x0d (EXAR) initialized successfully
Exar Hardware Crypto Engine detected
module 0x0a (HCE) initialized successfully
```

```
module          0x8e          (LNA)          initialized          successfully
module          0x84          (VRSA)         initialized          successfully
module          0x91          (ASN1)         initialized          successfully
module          0x8f          (ECA)          initialized          successfully
module          0x8b          (AES)          initialized          successfully
module          0x88          (DB)           initialized          successfully
module          0x96          (MBK)          initialized          successfully
module          0x9c          (ECDSA)        initialized          successfully
module          0x68          (CXI)          initialized          successfully
module          0x04          (POST)         initialized          successfully
module          0x87          (ADM)          initialized          successfully
CMDS:                   verified                    approved              firmware
POST: DSA Cryptographic Algorithm Test skipped
```

[root@localhost Administration]# ./csadm Dev=/dev/cs2.0 ListFirmware

```
ID          name          type          version          initialization          level
-----------------------------------------------------
0           SMOS          C64           5.5.9.2          INIT_OK
4           POST          C64           1.0.0.1          INIT_OK
a           HCE           C64           2.2.2.3          INIT_OK
d           EXAR          C64           2.1.1.4          INIT_OK
68          CXI           C64           2.2.3.4          INIT_OK
81          VDES          C64           1.0.9.2          INIT_OK
83          CMDS          C64           3.6.0.8          INIT_OK
84          VRSA          C64           1.3.4.65         INIT_OK
86          UTIL          C64           3.0.5.0          INIT_OK
87          ADM           C64           3.0.25.4         INIT_OK
88          DB            C64           1.3.2.0          INIT_OK
89          HASH          C64           1.0.11.1         INIT_OK
8b          AES           C64           1.4.1.4          INIT_OK
8e          LNA           C64           1.2.3.4          INIT_OK
8f          ECA           C64           1.1.10.2         INIT_OK
91          ASN1          C64           1.0.3.4          INIT_OK
96          MBK           C64           2.2.7.3          INIT_OK
9c ECDSA C64 1.1.11.0 INIT_OK
```

Gemalto SafeNet Luna HSM connection problem with physical partition, RSA key generation mechanism with Luna 7.x

Followings are the changes required in crystoki.ini file to remove latency while working with Gemalto 7.x HSM in PED authenticated mode.

1. Change ProtectedAuthenticationPathFlagStatus equal to “1” from “0” if a PED is used.
2. Add RSAKeyGenMechRemap under the miscellaneous section and set to a value of “1”.

Advantages of Key Wrapping

Storage:
HSMs normally struggle with internal limits to store more than a few hundred or a few thousand user keys. The key wrapping approach makes it more practical to manage many thousands or even millions of user keys. This is the only practical way of supporting large user bases.

Cost:
The HSM are generally expensive, and if all the user keys are to be stored within the HSM, then multiple HSMs would be needed to work together, hence the overall solution cost would be much higher, plus it would be a very complex system to manage and administer many different HSMs. With the key wrapping approach, the dependence on HSM storage is minimized hence two HSMs for high-availability are sufficient. In addition the required storage capacity, management and administration effort for these HSMs will be lower.

Compatibility:
Not all HSMs support the key wrapping functionality. The ADSS Server’s built-in PKCS#11 test utility can be used to verify if the underlying HSM supports the key wrapping functionality. Ensure your HSM has the ability to export keys.

Security:
The security of wrapped user keys depends on the security of the Key Encryption Key (KEK), i.e. its key length and algorithm used. The industry standard is to use a AES 256-bit key. This is considered secure and the only option against this algorithm is a brute force attack, which is not

practical given the 256-bit key size. The KEK is generated, stored and used inside the HSM so there is no potential for an attacker to get it through any other means. Therefore this approach provides almost the same security level for the user keys.

Performance:
If the user key is wrapped and stored in the database, then at the time of signing, it requires additional calls to retrieve the key from database and unwrap in the HSM. Although database and HSM operations are very fast but it would still not provide the same performance in absolute terms when compared to if the user key was already in the HSM. However from a user perspective this small difference in time will not be visible – you can judge this in your test environment. Furthermore in a bulk signing scenario, for the first signing operation the key is retrieved from the database and unwrapped to HSM, but for subsequent signing operations the key is already in HSM hence the overall performance would almost be same for both modes.

Standards and Regulations:
It is important to look at whether the key wrapping approach is acceptable according to local regulations. Most countries we know of allow this approach. The most common example of this is in the EU and the new eIDAS Regulation. This allows for this approach as long as it is done in a secure manner, e.g. the keys are protected by a certified HSM which acts as a Qualified Signature Creation Device, and furthermore there is a front-end Signature Activation Module (SAM) which authenticates the user fully and seeks their authorization to use the key protected by the HSM for signing a particular identified data item. Within eIDAS this approach is allowed for both advanced and Qualified Signatures – which is the highest level of trust. In terms of standards, the key wrapping technique is well known and defined in PKCS#11 standard as being supported by HSM vendors – our solution follows the PKCS#11 standard approach rather than a particular HSM vendors proprietary approach.