<http://docs.oracle.com/cd/E11882_01/network.112/e10746/asotrans.htm#CHDCJFEJ>

<http://docs.oracle.com/cd/B28359_01/network.111/b28530/asotrans.htm>

A hardware security module (HSM) is a physical device that provides secure storage for encryption keys. It also provides secure computational space (memory) to perform encryption and decryption operations. HSM is a more secure alternative to the Oracle wallet.

<http://www.jasypt.org/hibernate.html>

Luna PCI-E is the fastest, most secure, cryptographic PCI accelerator card in the industry, and is widely used by major governments, financial institutions and large enterprises around the world. The PCI Express bus on Luna PCI-E easily plugs into the host computer and provides reliable protection for data, applications, and digital identities to reduce risk and ensure regulatory compliance.

<http://geekcredential.wordpress.com/tag/luna-sa/>

a code example for accessing credentials stored on a Luna SA HSM for use with digital signatures. This example is implemented as a Grails service.

package com.geekcredential.crypto

// The EnvironmentContext class reads out environment variables that are set in the Tomcat

// conf/context.xml of whatever environment this class is running in. This enables us to

// connect to different HSM partitions for development or production servers.

import com.geekcredential.common.ui.EnvironmentContext

import java.security.cert.Certificate

import java.security.PrivateKey

import java.security.Security

import java.security.KeyStore

import com.safenetinc.luna.LunaSlotManager

/\*

\* The KeyStoreService provides a layer of abstraction between e-signing

\* resources and the code that uses them. It manages connections to an

\* external certificate storage device and caches certificates in memory to

\* reduce the number of times it is necessary to retrieve them. Other

\* information also used in the e-signing process is also made available

\* through the KeyStoreService as a convenience.

\*/

class KeyStoreService {

boolean transactional = true

// Represents a connection to a hardware device. The Luna SA has 4 "slots", but three

// of those are ports where a USB PED can be plugged in. Slot #1 is a handle to the

// cryptographic acceleration &amp; storage device inside the Luna SA server.

private LunaSlotManager hsmConnection

private String getEnvironmentVariable(String variableName) {

String value = EnvironmentContext.getEnv(variableName)

if (!value || value == 'null') {

throw new Exception("Environment variable '${variableName}' is not set. Please add it to the environment configuration (Tomcat conf/context.xml if this application is running as a .war) and restart this application.")

}

return value

}

private String getPartitionName() {

return getEnvironmentVariable("esignaturePartitionName")

}

private String getPartitionPassword() {

return getEnvironmentVariable("esignaturePartitionPassword")

}

private String getPrivateKeyLabel() {

return getEnvironmentVariable("esignaturePrivateKeyLabel")

}

private String[] getCaCertLabels() {

return getEnvironmentVariable("esignatureCaCertLabels").split(";")

}

private String getCertLabel() {

return getEnvironmentVariable("esignatureCertLabel")

}

String getContactEmail() {

return getEnvironmentVariable("esignatureContactEmail")

}

String getTimestampUrl() {

return getEnvironmentVariable("esignatureTimestampUrl")

}

String getOcspUrl() {

return getEnvironmentVariable("esignatureOcspUrl")

}

String getPdfOwnerPassword() {

return getEnvironmentVariable("esignaturePdfOwnerPassword")

}

void refreshHsmConnection() {

try {

resolveLunaSlotManagerInstance()

hsmConnectionLogin()

} catch (Throwable t) {

log.fatal("Unable to login to the Hardware Storage Module (HSM). E-signing can't be completed without access to a certificate", t)

throw new Exception("Unable to login to the Hardware Storage Module (HSM). E-signing can't be completed without access to a certificate", t)

}

}

private void hsmConnectionLogin() {

synchronized (hsmConnection) {

if (!hsmConnection.loggedIn) {

hsmConnection.login(partitionPassword)

}

}

}

private void resolveLunaSlotManagerInstance() {

if (!hsmConnection) {

hsmConnection = LunaSlotManager.getInstance()

}

if (!hsmConnection) {

throw new Exception("LunaSlotManager did not return an instance.")

}

}

Map getCredentials() {

try {

if (!Security.getProvider("LunaProvider")) {

Security.addProvider(new com.safenetinc.luna.provider.LunaProvider())

}

refreshHsmConnection()

KeyStore ks = KeyStore.getInstance("Luna")

ks.load(null, null)

Map credentials = [:]

credentials.put("privateKey", (PrivateKey) ks.getKey(privateKeyLabel, null))

// We need to assemble the certificate chain manually because the HSM doesn't support the

// getCertificateChain method. The array of certificates in the chain should be ordered

// starting with our cert and then proceeding to any intermediate certificate authority certs

// up to the original issuer.

List chain = [ks.getCertificate(certLabel)]

caCertLabels.each {label -&gt;

chain &lt;&lt; ks.getCertificate(label)

}

credentials.put("certificateChain", chain.toArray(new Certificate[chain.size()]))

return credentials

} catch (Throwable t) {

log.fatal("Unable to retrieve resources from the Hardware Storage Module (HSM). E-signing can't be completed without a private key and certificate chain", t)

throw new Exception("Unable to retrieve resources from the Hardware Storage Module (HSM). E-signing can't be completed without a private key and certificate chain", t)

}

}

}

|  |  |  |
| --- | --- | --- |
| package com.geekcredential.crypto | | |
| 002 |  |

|  |  |
| --- | --- |
| 003 | // The EnvironmentContext class reads out environment variables that are set in the Tomcat |
| 004 | // conf/context.xml of whatever environment this class is running in. This enables us to |

|  |  |  |
| --- | --- | --- |
| 005 | // connect to different HSM partitions for development or production servers. | |
| 006 | import com.geekcredential.common.ui.EnvironmentContext |

|  |  |  |
| --- | --- | --- |
| 007 | import java.security.cert.Certificate | |
| 008 | import java.security.PrivateKey |

|  |  |
| --- | --- |
| 009 | import java.security.Security |
| 010 | import java.security.KeyStore |

|  |  |  |
| --- | --- | --- |
| 011 | import com.safenetinc.luna.LunaSlotManager | |
| 012 |  |

|  |  |
| --- | --- |
| 013 | /\* |
| 014 | \* The KeyStoreService provides a layer of abstraction between e-signing | |

|  |  |
| --- | --- |
| 015 | \* resources and the code that uses them. It manages connections to an |
| 016 | \* external certificate storage device and caches certificates in memory to | |

|  |  |
| --- | --- |
| 017 | \* reduce the number of times it is necessary to retrieve them. Other |
| 018 | \* information also used in the e-signing process is also made available | |

|  |  |  |
| --- | --- | --- |
| 019 | \* through the KeyStoreService as a convenience. | |
| 020 | \*/ |

|  |  |  |
| --- | --- | --- |
| 021 | class KeyStoreService { | |
| 022 |  |

|  |  |  |
| --- | --- | --- |
| 023 | boolean transactional = true | |
| 024 |  |

|  |  |
| --- | --- |
| 025 | // Represents a connection to a hardware device. The Luna SA has 4 "slots", but three |
| 026 | // of those are ports where a USB PED can be plugged in. Slot #1 is a handle to the |

|  |  |  |
| --- | --- | --- |
| 027 | // cryptographic acceleration &amp; storage device inside the Luna SA server. | |
| 028 | private LunaSlotManager hsmConnection |

|  |  |
| --- | --- |
| 029 |  |
| 030 | private String getEnvironmentVariable(String variableName) { | |

|  |  |  |
| --- | --- | --- |
| 031 | String value = EnvironmentContext.getEnv(variableName) | |
| 032 | if (!value || value == 'null') { |

|  |  |  |
| --- | --- | --- |
| 033 | throw new Exception("Environment variable '${variableName}' is not set. Please add it to the             environment configuration (Tomcat conf/context.xml if this application is running as a .war) and restart this application.") | |
| 034 | } |

|  |  |  |
| --- | --- | --- |
| 035 | return value | |
| 036 | } |

|  |  |
| --- | --- |
| 037 |  |
| 038 | private String getPartitionName() { | |

|  |  |  |
| --- | --- | --- |
| 039 | return getEnvironmentVariable("esignaturePartitionName") | |
| 040 | } |

|  |  |
| --- | --- |
| 041 |  |
| 042 | private String getPartitionPassword() { | |

|  |  |  |
| --- | --- | --- |
| 043 | return getEnvironmentVariable("esignaturePartitionPassword") | |
| 044 | } |

|  |  |
| --- | --- |
| 045 |  |
| 046 | private String getPrivateKeyLabel() { | |

|  |  |  |
| --- | --- | --- |
| 047 | return getEnvironmentVariable("esignaturePrivateKeyLabel") | |
| 048 | } |

|  |  |
| --- | --- |
| 049 |  |
| 050 | private String[] getCaCertLabels() { | |

|  |  |  |
| --- | --- | --- |
| 051 | return getEnvironmentVariable("esignatureCaCertLabels").split(";") | |
| 052 | } |

|  |  |
| --- | --- |
| 053 |  |
| 054 | private String getCertLabel() { | |

|  |  |  |
| --- | --- | --- |
| 055 | return getEnvironmentVariable("esignatureCertLabel") | |
| 056 | } |

|  |  |
| --- | --- |
| 057 |  |
| 058 | String getContactEmail() { | |

|  |  |  |
| --- | --- | --- |
| 059 | return getEnvironmentVariable("esignatureContactEmail") | |
| 060 | } |

|  |  |
| --- | --- |
| 061 |  |
| 062 | String getTimestampUrl() { | |

|  |  |  |
| --- | --- | --- |
| 063 | return getEnvironmentVariable("esignatureTimestampUrl") | |
| 064 | } |

|  |  |
| --- | --- |
| 065 |  |
| 066 | String getOcspUrl() { | |

|  |  |  |
| --- | --- | --- |
| 067 | return getEnvironmentVariable("esignatureOcspUrl") | |
| 068 | } |

|  |  |
| --- | --- |
| 069 |  |
| 070 | String getPdfOwnerPassword() { | |

|  |  |  |
| --- | --- | --- |
| 071 | return getEnvironmentVariable("esignaturePdfOwnerPassword") | |
| 072 | } |

|  |  |
| --- | --- |
| 073 |  |
| 074 | void refreshHsmConnection() { | |

|  |  |
| --- | --- |
| 075 | try { |
| 076 | resolveLunaSlotManagerInstance() | |

|  |  |  |
| --- | --- | --- |
| 077 | hsmConnectionLogin() | |
| 078 | } catch (Throwable t) { |

|  |  |
| --- | --- |
| 079 | log.fatal("Unable to login to the Hardware Storage Module (HSM). E-signing can't be completed without access to a certificate", t) |
| 080 | throw new Exception("Unable to login to the Hardware Storage Module (HSM). E-signing can't be completed without access to a certificate", t) |

|  |  |  |
| --- | --- | --- |
| 081 | } | |
| 082 | } |

|  |  |
| --- | --- |
| 083 |  |
| 084 | private void hsmConnectionLogin() { | |

|  |  |
| --- | --- |
| 085 | synchronized (hsmConnection) { |
| 086 | if (!hsmConnection.loggedIn) { | |

|  |  |  |
| --- | --- | --- |
| 087 | hsmConnection.login(partitionPassword) | |
| 088 | } |

|  |  |  |
| --- | --- | --- |
| 089 | } | |
| 090 | } |

|  |  |
| --- | --- |
| 091 |  |
| 092 | private void resolveLunaSlotManagerInstance() { | |

|  |  |
| --- | --- |
| 093 | if (!hsmConnection) { |
| 094 | hsmConnection = LunaSlotManager.getInstance() | |

|  |  |
| --- | --- |
| 095 | } |
| 096 | if (!hsmConnection) { | |

|  |  |  |
| --- | --- | --- |
| 097 | throw new Exception("LunaSlotManager did not return an instance.") | |
| 098 | } |

|  |  |  |
| --- | --- | --- |
| 099 | } | |
| 100 |  |

|  |  |  |
| --- | --- | --- |
| 101 | Map getCredentials() { | |
| 102 | try { |

|  |  |
| --- | --- |
| 103 | if (!Security.getProvider("LunaProvider")) { |
| 104 | Security.addProvider(new com.safenetinc.luna.provider.LunaProvider()) | |

|  |  |
| --- | --- |
| 105 | } |
| 106 | refreshHsmConnection() | |

|  |  |  |
| --- | --- | --- |
| 107 | KeyStore ks = KeyStore.getInstance("Luna") | |
| 108 | ks.load(null, null) |

|  |  |
| --- | --- |
| 109 |  |
| 110 | Map credentials = [:] | |

|  |  |  |
| --- | --- | --- |
| 111 | credentials.put("privateKey", (PrivateKey) ks.getKey(privateKeyLabel, null)) | |
| 112 |  |

|  |  |
| --- | --- |
| 113 | // We need to assemble the certificate chain manually because the HSM doesn't support the |
| 114 | // getCertificateChain method. The array of certificates in the chain should be ordered |

|  |  |  |
| --- | --- | --- |
| 115 | // starting with our cert and then proceeding to any intermediate certificate authority certs | |
| 116 | // up to the original issuer. |

|  |  |  |
| --- | --- | --- |
| 117 | List chain = [ks.getCertificate(certLabel)] | |
| 118 | caCertLabels.each {label -&gt; |

|  |  |  |
| --- | --- | --- |
| 119 | chain &lt;&lt; ks.getCertificate(label) | |
| 120 | } |

|  |  |  |
| --- | --- | --- |
| 121 | credentials.put("certificateChain", chain.toArray(new Certificate[chain.size()])) | |
| 122 |  |

|  |  |
| --- | --- |
| 123 | return credentials |
| 124 | } catch (Throwable t) { | |

|  |  |
| --- | --- |
| 125 | log.fatal("Unable to retrieve resources from the Hardware Storage Module (HSM). E-signing can't be completed without a private key and certificate chain", t) |
| 126 | throw new Exception("Unable to retrieve resources from the Hardware Storage Module (HSM). E-signing can't be completed without a private key and certificate chain", t) |

|  |  |  |
| --- | --- | --- |
| 127 | } | |
| 128 | } |

|  |  |
| --- | --- |
| 129 |  |
| 130 | } | |

<http://stackoverflow.com/questions/12981977/how-do-i-integrate-hsm-encryption-with-java>

In JAVA you can just use JCE/JCA. Ask you provider for the implementation, you will need some jar files, and you r ready.

Secure Socket Layer (SSL)

iaik pkcs11wrapper.jar

This means that Java applications calling standard JCA and JCE APIs can, without modification, take advantage of algorithms offered by the underlying PKCS#11 implementations, such as, for example,

* Cryptographic Smartcards,
* Hardware cryptographic accelerators, and
* High performance software implementations.

<https://www.opensc-project.org/opensc>

pkcs11-helper

**Features**

pkcs11-helper allows using multiple PKCS#11 providers at the same time, enumerating available token certificates, or selecting a certificate directly by serialized id, handling card removal and card insert events, handling card re-insert to a different slot, supporting session expiration and much more all using a simple API.

pkcs11-helper is not designed to manage card content, since object attributes are usually vendor specific, and 99% of application need to access existing objects in order to perform signature and decryption.

**Compatibility**

* The pkcs11-helper library is available in POSIX and WIN32 compliant systems.
* The pkcs11-helper library should work with almost any PKCS#11 provider, since it uses the minimum required features of the PKCS#11 standard, a list of WorkingProviders? is available.

<http://stackoverflow.com/questions/12937138/how-does-jca-jce-and-pkcs11-work-together?rq=1>

1. **What is meant with a key handle in JCE?**  
   A key handle (in JCE, PKCS#11, or most other cryptographic APIs) is simply a reference that enables you to use a key without seeing its actual value. That is good: you can have the key permanently stored in a secure place (e.g. an HSM) with the assurance that nobody will be able to copy it and run away with it - as it may happen if the key is the application space. Unlike a physical safe though, you can still perform cryptographic operation without running any security risk of key leakage.
2. **Does the PKCS#11 standard define a way so that the signature is generated in the HSM?**  
   PKCS#11 is a C API for cryptographic tokens. A token is a PKCS#11 abstraction for any device or program that offers services described by such API. The API defines which operations you can perform using the objects inside the PKCS#11 token: some objects are non sensitive, and can be extracted (e.g. public keys); some others are sensitive and can only be used, via handles.  
   If you have a handle to an object that supports signing, you can use the C function [C\_Sign](http://www.cryptsoft.com/pkcs11doc/v220/pkcs11__all_8h.html#aC_Sign) to ask the token to authenticate some data provided by your application. The key does not leave the HSM.
3. **The featurelist of my HSM states JCE and PKCS#11 separately. What does that mean?**  
   Your HSM supports JCE in the sense that it comes with a native library that qualifies as a [Cryptographic Service Provider](http://docs.oracle.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html#ProviderArch).  
   It supports PKCS#11 in the sense that it comes with a native library that offers a C PKCS#11 API.
4. **I thought PKCS#11 is a standard, and JCE defines classes to use that standard. Does JCE specify its own protocols?**  
   Indeed PKCS#11 is a standard; but it is not directly usable by languages other than C. You need a mapping layer that translates it into something compatible to your language. A PKCS#11 library (and the physical tokens that it abstracts) can be mapped to [a JCE provider](http://docs.oracle.com/javase/1.5.0/docs/guide/security/p11guide.html).

However, a JCE provider may have nothing to do with PKCS#11.

## [HSM](https://wiki.opendnssec.org/display/DOCREF/HSM) Vendors

### [AEP Networks](http://www.aepnetworks.com/)

* Keyper

### [Athena Smartcard Solutions](http://www.athena-scs.com/)

* IDProtect

### [OpenSC](http://www.opensc-project.org/)

* OpenSC PKCS#11 Provider

### [Oracle](http://www.oracle.com/)

* Sun Crypto Accelerator 6000 (SCA/6000)

### [RealSec](http://www.realsec.com/)

* Cryptosec LAN

### [SafeNet](http://www.safenet-inc.com/)

* eToken PRO
* Luna SA 4
* Luna SA 5
* Luna G5

### [SmartCard-HSM](http://www.smartcard-hsm.com/)

### [Thales](http://www.thales-esecurity.com/)

* nCshield Connect

### [Utimaco](http://www.utimaco.com/)

* HSM-Series

## Download softHSM

<https://wiki.opendnssec.org/download/attachments/590262/A-Review-of-Hardware-Security-Modules-Fall-2010.pdf?version=1&modificationDate=1317976607000>

# Installing and Configuring a LunaSA Hardware Security Module (HSM) with FIM CM 2010

http://social.technet.microsoft.com/wiki/contents/articles/1731.installing-and-configuring-a-lunasa-hardware-security-module-hsm-with-fim-cm-2010.aspx

<http://stackoverflow.com/questions/8881536/pkcs11-pkcs11-dll?rq=1>

[connecting to softhsm java](http://stackoverflow.com/questions/7009931/connecting-to-softhsm-java)

http://stackoverflow.com/questions/7009931/connecting-to-softhsm-java

* 1. **For Windows:**

Copy these files from the Program Files\LunaSA\JSP\lib folder into your JAVA\_HOME\jre\lib\ext folder:

* LunaAPI.dll
* LunaJCASP.jar
* LunaJCESP.jar

1. Add .dll file in the path of the operating system.

String configName = "J:\\jars\\softHSM\\softhsm.cfg";

Provider p = **new** SunPKCS11(configName);

1. softhsm.conf to create the slots over the softHSM server.

#### SYNOPSIS

**softhsm.conf**

#### DESCRIPTION

In PKCS#11 you need tokens in order to do cryptographic operations.

Tokens can be viewed as object stores where you can store e.g. private

and public keys. A token must then be attached to a slot so that you

can use it.

Slots and tokens are handled by the SoftHSM configuration file. The

given paths in the configuration file are just an indication to SoftHSM

on where it should store the information for each token. The token

databases will be created when the tokens gets initialized.

#### FILE FORMAT

Each pair of slot and token are configured on one line. Starting with

an unsigned integer as the slot ID and then a path where SoftHSM can

create a SQLite database. These parameters are separated by a

semicolon. It is OK to have extra space between the parameters, since

these will be ignored.

<Slot\_ID>:<Path\_to\_the\_token\_database>

It is also possible to add comments in the file by using the hash sign.

Anything after the hash sign will be ignored.

#<text>

Any line that does not have the correct format will be ignored.

#### EXAMPLE

0:/var/softhsm/slot0.db

1:/home/user/token.database # My own token

#### ENVIRONMENT

SOFTHSM\_CONF

When defined, the value will be used as path to the

configuration file.

#### FILES

/etc/softhsm/softhsm.conf

default location of the SoftHSM configuration file

/etc/softhsm/softhsm.conf.sample

an example of a SoftHSM configuration file

1. softhsm.cfg contains the

name = SoftHSM

library = C:\SoftHSM\lib\libsofthsm.dll

slot = 0

attributes(generate, \*, \*) = {

CKA\_TOKEN = true

}

sun.security.pkcs11.wrapper.PKCS11Exception: CKR\_TOKEN\_NOT\_RECOGNIZED

to fix this use command prompt with below commands.

C:\SoftHSM\bin>softhsm --init-token --slot 0 --label "TokenA"

The SO PIN must have a length between 4 and 255 characters.

Enter SO PIN:

The user PIN must have a length between 4 and 255 characters.

Enter user PIN:

The token has been initialized.

* 1. In your Java SDK directory, open the file java.security in the jre/lib/security directory and add the two lines in **boldface** below to the list of security providers, *after* all Sun providers:

# List of providers and their preference orders (see above):  
security.provider.1=sun.security.provider.Sun  
security.provider.2=sun.security.rsa.SunRsaSign  
security.provider.3=com.sun.net.ssl.internal.ssl.Provider  
security.provider.4=com.sun.crypto.provider.SunJCE  
security.provider.5=sun.security.jgss.SunProvider  
security.provider.6=com.sun.security.sasl.Provider  
security.provider.7=org.jcp.xml.dsig.internal.dom.XMLDSigRI  
security.provider.8=sun.security.smartcardio.SunPCSC  
security.provider.9=sun.security.mscapi.SunMSCAPI  
security.provider.10=com.chrysalisits.crypto.LunaJCAProvider  
security.provider.11=com.chrysalisits.cryptox.LunaJCEProvider

* 1. Save and close the java.security file.
  2. Add the tokens to the slots.   
     The default location of the config file is /etc/softhsm.conf.  This location can be change by setting the environment variable.

|  |
| --- |
| export SOFTHSM\_CONF=/home/user/config.file |

* 1. Open the config file and add the slots (note there should be no whitespace at the beginning of the lines)

|  |  |
| --- | --- |
| > pico /home/user/config.file    0:/home/user/my.db  # Comments can be added  4:/home/user/token.database | |
| https://wiki.opendnssec.org/images/icons/emoticons/warning.png | The token databases does not exist at this stage. The given paths are just an indication to SoftHSM on where it should store the information for each token. |

* Each token is now treated as uninitialized.  
  1. Initialize your tokens.  
       
     Use either the softhsm tool or the PKCS#11 interface. The SO PIN can e.g. be used to re-initialize the token and the user PIN is handed out to the application so it can interact with the token.

Check your understanding with the John:

SoftHSM contains the key and the oracle database contains the encrypted data. The key contained by the SoftHSM will be used to encrypt/decrypt the data in the oracle database column.

Checked with Lawrence and it’s true.

**Architecture**

A typical Cryptoki-based system architecture is depicted in Figure 1.

****

**Figure 1**

The cryptographic device (aka *token*) is connected to the system via a *slot*. Typically, a slot corresponds to a

smart card reader or a specific card terminal. However, because *Cryptoki* offers a purely logical view of the

system it could happens that different slots point to the same physical reader device or, viceversa, a single

slot could have more than one device.

**The logical structure of a Token**

A specific Cryptoki implementation maps the token’s physical structure, typically composed by memory

zones in which data, cryptographic keys and their digital certificates are stored, into a logical structure that

adheres to the hierarchical model shown in Figure 2.



**Figure 2**

The specifications define three main object classes:

*- Data* objects host generic data which semantics is defined by the application who created them;

*- Certificate* objects store digital certificates;

*- Key* objects contain a public, private or secret cryptographic key.

Cryptoki’s objects are classified depending on their visibility in *public objects* (i.e. accessible by all

applications), and *private objects* (visible only after granting access permissions typically performed via PINverification

as described later), and on their persistency in: *token objects* which persist when the token is

plugged-out from the slot and in *session objects* which don’t persist.

For each class of objects the specifications define a set of attributes (as described later) characterizing all

instances of the class, which, are inherited by derived classes, similarly to an object-oriented model (for

example, the Private Key class inherits all attributes from the Key class etc.)

# All other Cryptoki function return values

<http://www.cryptsoft.com/pkcs11doc/v220/group__SEC__11__1__6__ALL__OTHER__CRYPTOKI__FUNCTION__RETURN__VALUES.html>

# Java ™ Cryptography Architecture (JCA) Reference Guide

<http://docs.oracle.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html#ProviderInstalling>

# JavaTM PKCS#11 Reference Guide

<http://docs.oracle.com/javase/6/docs/technotes/guides/security/p11guide.html>

Sun Java System Application Server 9.1 Administration Guide

<http://docs.oracle.com/cd/E19159-01/819-3671/gcsoc/index.html>

# IBM(R) JavaTM PKCS 11 Implementation Provider 1.1

# API Specification & Reference

<http://publib.boulder.ibm.com/infocenter/javasdk/v5r0/index.jsp?topic=%2Fcom.ibm.java.security.component.doc.50%2Fsecguides%2Fpkcs11implDocs%2FIBMJavaPKCS11ImplementationProvider.html>

# [Java Sun PKCS#11 provider, HSM token LOGIN REQUIRED flag not set and empty list of aliases](http://stackoverflow.com/questions/8218566/java-sun-pkcs11-provider-hsm-token-login-required-flag-not-set-and-empty-list)

Or you can get your patch accepted, I would certainly be interested in an option to login before the available private keys are looked up using PKCS#11 FindObjects. If you file a bug report I will certainly vote for it.

<http://stackoverflow.com/questions/8218566/java-sun-pkcs11-provider-hsm-token-login-required-flag-not-set-and-empty-list?rq=1>

**use of command prompt for the softHSM**

Initialize your tokens.

softhsm --init-token --slot 0 --label “TokenA”

------------------------------------------------------------------------------------------------------

C:\SoftHSM\bin>softhsm --init-token --slot 0 --label "TokenA"

The SO PIN must have a length between 4 and 255 characters.

Enter SO PIN:

The user PIN must have a length between 4 and 255 characters.

Enter user PIN:

The token has been initialized.

C:\SoftHSM\bin>softhsm --show-slots

Available slots:

Slot 0

Token present: yes

Token initialized: yes

User PIN initialized: yes

Token label: ôTokenAö

-------------------------------------------------------------------------------------------

create one .pem file and export it to create a key in the softhsm server.

softhsm --import key1.pem --slot 1 --label "Token1" --id A1B2 --pin 123456

C:\SoftHSM\bin>softhsm --import C:\SoftHSM\bin\key.pem --slot 0 --label "TokenA"

--id 123456 --pin 123456

Botan: Decoding error: PKCS #8 private key decoding failed

Error: Perhaps wrong path to file, wrong file format, or wrong PIN to file (--fi

le-pin <PIN>).

<http://stackoverflow.com/questions/13671348/cant-configure-jasypt-hibernate>

Exception in thread "main" org.jasypt.exceptions.EncryptionOperationNotPossibleException: Encryption raised an exception. A possible cause is you are using strong encryption algorithms and you have not installed the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files in this Java Virtual Machine

<http://stackoverflow.com/questions/10685093/jasypt-with-hibernate-registering-the-encryptor-in-hibernateutil?rq=1>

set provider in the encyptor

StandardPBEStringEncryptor