July 2009

Cryptographic Hardware Use Cases for Web Servers on Linux on IBM System z

Technical description

Table of Contents

Use Case 1: Configuring Apache2 to use Cryptographic Hardware Devices on Novell SLES 9

Apache2 built with mod_ssl, as it is with Novell SLES 9 distributions, can be configured to use a **cryptographic** card to accelerate the SSL handshakes that occur when clients make https requests. When configured to use **cryptographic** hardware, Apache will use the CPACF instructions to accelerate the encryption and decryption of messages sent during an SSL session, when certain ciphers are used. The following steps show how the default Apache2 configuration, as shipped with SLES 9, can be updated to enable the use of cryptographic hardware on Linux® on IBM System z®.

Step 1: Load the z90crypt device driver

a) The z90crypt device driver must be loaded before attempting to start servers that have been configured to use cryptographic hardware devices. On Novell SLES, use the rcz90crypt initialization script provided by the libica rpm. Issue the ‘rcz90crypt start’ command:

```
rcz90crypt start
Loading z90crypt module                                        done
```

b) The z90crypt driver reads the file /etc/sysconfig/z90crypt at start time.

This file can be used to specify a domain number. By default, the file contains:

Z90CRYPT_DOMAIN=-1 (autodetect).

Domain numbers are not an issue under z/VM with CRYPTO APVIRT in the user directory, because a virtual domain is used. Therefore, the default: -1 works fine.

c) Check file /var/log/messages for errors. These messages indicate success:

```
Jul 18 18:50:41 mg8lnx04 kernel: z90crypt: Version 1.3.3 loaded, built on May 15 2006
16:08:30
Jul 18 18:50:41 mg8lnx04 kernel: z90crypt: z90main.o ($Revision: 1.31.2.12 $/
$Revision: 1.8.2.5 $/$Revision: 1.2.2.4 $)
Jul 18 18:50:41 mg8lnx04 kernel: z90crypt: z90hardware.o ($Revision: 1.19.2.8 $/
$Revision: 1.8.2.5 $/$Revision: 1.2.2.4
```

These messages mean that you do not have a **cryptographic** card available:

```
Jul 18 17:44:57 mg8lnx06 kernel: z90crypt: query_online -> Exception testing device 0
Jul 18 17:44:57 mg8lnx06 kernel: z90crypt: helper_scan_devices -> exception taken!
Jul 18 17:44:57 mg8lnx06 kernel: z90crypt: z90crypt_config_task -> Error 34 detected
in refresh_z90crypt.
```

d) To have z90crypt loaded automatically at boot time on Novell SUSE distributions, update file /etc/sysconfig/kernel:

```
MODULES_LOADED_ON_BOOT="z90crypt"
```

e) To have z90crypt loaded automatically at boot time on Red Hat distributions, run this command:

```
-mkinitrd -v -with=z90crypt -f <name-of-initrd> `uname -r`
```

and then run the command: zipl

f) z90crypt writes status to a file in the /proc file system: /proc/driver/z90crypt

To check status, issue this command:

```
cat /proc/driver/z90crypt
z90crypt version: 1.3.3
Cryptographic domain: 0
Total device count: 1
PCICA count: 0
PCICC count: 0
PCIXCC MCL2 count: 0
PCIXCC MCL3 count: 0
CEX2C count: 0
CEX2A count: 1
requestq count: 0
pendingq count: 2     <--- requests queued
Total open handles: 2 <--- 2 users Note: SLES 9 SP3 ssh sessions open a
handle

Online devices: 1=PCICA 2=PCICC 3=PCIXCC(MCL2) 4=PCIXCC(MCL3) 5=CEX2C 6=CEX2A
           0000000000006000 0000000000000000 0000000000000000 0000000000000000

Waiting work element counts
           0000000000002000 0000000000000000 0000000000000000 0000000000000000

Per-device successfully completed request counts
    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000 00000213 00000000 00000000 00000000
    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

In this example, there is one virtual cryptographic device available – a CEX2A. It shows as device number 12 in the 'Online devices' section of the display. The 'waiting work elements counts' shows that there are 2 requests waiting for device 12. The 'Per-device successfully completed request counts' shows that device number 12 has completed x'213' requests. Note that under z/VM, with CRYPTO APVIRT specified in its directory entry, a Linux guest will, at most, see one virtual **cryptographic** device.

Step 2: Add an SSLCryptoDevice statement in file /etc/apache2/ssl-global.conf specified to use 'ibmca'.

On the blank line under the statement `<IfModule mod_ssl.c>`, add the line:

SSLCryptoDevice ibmca

Step 3: Create an SSL virtual host

Create an ssl virtual host conf file from the template – /etc/apache2/vhosts.d/vhost-ssl.template

a) **cd /etc/apache2/vhosts.d**

b) **cp vhost-ssl.template vhost-ssl443.conf**

Apache will read the *.conf files in the vhosts.d directory at start time to create virtual hosts.

Step 4: Update the Cipher Suite for the virtual host to use high-strength ciphers AES-128 or triple-DES

a) Edit **vhost-ssl443.conf**

b) Delete the line:

```
SSLCipherSuite ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:
+eNULL
```

c) And replace that line with the following lines:

```
SSLCipherSuite  AES128-SHA
SSLCipherSuite  DES-CBC3-SHA
```

d) Note the following lines in the vhost-ssl443.conf file:

```
SSLCertificateFile /etc/apache2/ssl.crt/server.crt
SSLCertificateKeyFile /etc/apache2/ssl.key/server.key
```

These directives point to the certificate that Apache will send to clients requesting an SSL connection, and the private key that Apache will use during the SSL handshake. The files shown are just placeholders, you need to create your own certificate and private key.

Step 5: Generate a new (self-signed) certificate and keys using OpenSSL

a) `cd /etc/ssl`

b) Issue the openssl command to generate a new key and certificate request:

```
openssl req -config openssl.cnf -new -nodes -keyout apachenodes.key -out
    apache.csr
```

The certificate request dialog is started. You are prompted for information used to generate the certificate:

```
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:CA
Locality Name (eg, city) []:SanDiego
Organization Name (eg, company) [Internet Widgits Pty Ltd]:SHARE
Organizational Unit Name (eg, section) []:hit enter  (skip entering a value)
Common Name (eg, YOUR name) []:lncrpt01.endicott.ibm.com   (enter your hostname)
Email Address []: hit enter    (skip entering a value)
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []: linux390
An optional company name []: hit enter   (skip entering a value)
```

After answering the prompts to create the contents of the certificate, a certificate request and keys are created. Using the -nodes parameter leaves the private key unencrypted. Encrypt the key with DES3. A pass phrase is entered, which will be required to start Apache using the encrypted key -

c) Issue the command:

```
openssl rsa -in apachenodes.key -des3 -out apache.key
writing RSA key
Enter PEM pass phrase: linux390
Verifying - Enter PEM pass phrase: linux390
```

d) Sign the certificate request – Issue the command:

```
openssl x509 -in apache.csr -out apache.crt -req -signkey apache.key -days 999
Signature ok
subject=/C=US/ST=CA/L=SanDiego/O=SHARE/CN=lncrpt01.endicott.ibm.com
Getting Private key
Enter pass phrase for apache.key: linux390
```

You now have a PEM-encoded self-signed certificate (with public key) and encrypted private key that the Apache server can use to establish SSL sessions with clients. The certificate will look similar to this:

```
-----BEGIN CERTIFICATE-----
MIICMzCCAZwCAQAwDQYJKoZIhvcNAQEEBQAwYjELMAkGA1UEBhMCVVMxEDAOBgNV
BAgTB0Zsb3JpZGExDjAMBgNVBAcTBVRhbXBhMQ4wDAYDVQQKEwVTSEFSRTEhMB8G
A1UEAxMYbWc4bG54MDQucGRsLnBvay5pYm0uY29tMB4XDTA3MDExMTE2Mjc0MloX
DTA5MTAwNjE2Mjc0MlowYjELMAkGA1UEBhMCVVMxEDAOBgNVBAgTB0Zsb3JpZGEx
DjAMBgNVBAcTBVRhbXBhMQ4wDAYDVQQKEwVTSEFSRTEhMB8GA1UEAxMYbWc4bG54
MDQucGRsLnBvay5pYm0uY29tMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDY
d6hqJphGSZA5mWqB53+frcU+yXzJ62VLNWaKEsRAfeH0EHb5PULHfE6LXmexmGhL
YoYnhWF9yM9cJV6jdKSIsJZknL2V3nus5YlX1t74lwujPkAhBafuCJ442e6eOkW+
4YcxldWGbQQwaF9WPmlpfYAwsXfRkjnioIpYPNo8bQIDAQABMA0GCSqGSIb3DQEB
BAUAA4GBAD8Fw7e+R8+VPfVnzHD8cpAHPZRN89AcYczJR4LLxt3wMLd3m7VH7cOR
shDMyTns+9VT2XCxVmF7hTkXmx8Nfx6/WW6bZmGPaBAy2C0SKusZKtVc4+71GsS9
02De1KubNZR6dIqMHYOPrwRnNpGiu8h7RwM/6POrE/l7fy87xFnP
-----END CERTIFICATE-----
```

Step 6: Copy the certificate and key to the files pointed to in the vhost-ssl443.conf file:

```
cp apache.crt  /etc/apache2/ssl.crt/server.crt
cp apache.key  /etc/apache2/ssl.key/server.key
```

Step 7: Create the file /etc/apache2/sysconfig.d/include.conf.

This file is referenced by the default SLES 9 Apache2 configuration, but does not exist:

`touch /etc/apache2/sysconfig.d/include.conf` (This will create an empty file)

Step 8: Start Apache with SSL. Because an encrypted key file is used, there is a pass phrase prompt

Issue the command: `apache2ctl startssl`

```
Apache/2.0.49 mod_ssl/2.0.49 (Pass Phrase Dialog)
Some of your private key files are encrypted for security reasons.
In order to read them you have to provide us with the pass phrases.
Server mg8lnx04.pdl.pok.ibm.com:443 (RSA)
Enter pass phrase: linux390
```

(This is the pass phrase used to encrypt the server's private key in Step 5)

```
Ok: Pass Phrase Dialog successful.
```

At this point in this use case, the set up is done and you can verify that it works.

Step 9: Verify that Apache has allocated the cryptographic cards, display the z90crypt driver information and check the open handles.

Issue the command:

```
cat /proc/driver/z90crypt

z90crypt version: 1.3.3
Cryptographic domain: 0
Total device count: 1
PCICA count: 0
PCICC count: 0
PCIXCC MCL2 count: 0
PCIXCC MCL3 count: 0
CEX2C count: 0
CEX2A count: 1
requestq count: 0
pendingq count: 0
Total open handles: 1   <-------- open handles increments after starting Apache2
```

Step 10: Verify that the configuration is working by opening a browser, and access your Apache2 server using https.

Type this in the URL field of your browser:

```
https://Linux_system_hostname
```

Step 11: Verify that the request to Apache used your specified Cipher Suite, by displaying Apache's ssl_request_log:

```
cat /var/log/apache2/ssl_request_log
```

This will show a line similar to the one below, or with cipher DES-CBC3-SHA

```
[19/Jan/2007:12:09:19 -0400] 9.56.164.156 TLSv1 AES128-SHA "GET / HTTP/1.1" 44
```

Step 12: Verify that the cryptographic cards are being used.

Access your Apache2 server from a browser using https, Display the z90crypt driver status information by issuing the following command:

```
cat /proc/driver/z90crypt
```

Every successful SSL handshake results in the counter being incremented:

```
Per-device successfully completed request counts
    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000 00000006 00000000 00000000 00000000
```

How do you verify that the CPACF instructions are being used? You know that they are being run by libica. The source is available at the Web site: http://sourceforge.net/projects/opencryptoki

The CPACF instructions can be found in the source:

KM - Cipher Message (encrypt/decrypt)

KMC - Cipher Message with Chaining (encrypt/decrypt)

KIMD - Compute Intermediate Message Digest (generate SHA hash values)

KLMD - Compute Last Message Digest (generate SHA hash values)

KMAC - Compute Message Authentication Code

You can add printf statements in front of the CPACF instructions in icalinux.c, in the 'z' routines – zDes, zAes, zSha1, zSha256, zPrng. For example:

```
printf("In zDes doing KMC\n");
rv = KMC(function_code,
         keybuff,
         pDes->outputdata,
         pDes->inputdata,
         pDes->inputdatalength);
```

Then configure, make, and replace the original /usr/lib64/libica.so file with the modified one.

To view the messages from the modified libica.so, you have to start Apache2 in the foreground.

To start Apache2 in the foreground, issue the command:

```
httpd2 -X -DSSL
```

After responding to the prompt for the pass phrase, Apache initializes. HTTPS requests from a browser, when using the AES-128 cipher suite, result in the print messages are displayed similar to:

```
In zSha1 doing KLMD
In zAes doing KMC
```

Or, when using TDES:

```
In zSha1 doing KLMD
In zDes doing KMC
```

To stop Apache2 running in the foreground – type `CTRL+C`

Use Case 2: Configuring Tivoli Access Manager for eBusiness WebSEAL 6.1 to use Cryptographic Hardware on Linux for IBM System z

Use cases 2a through 2c show examples of configuring IBM Tivoli® Access Manager for eBusiness WebSEAL to use cryptographic hardware, on Novell SLES 9, RHEL 5, and Novell SLES 10. Included in these examples are the steps required to do the basic PKCS#11 setup on each of the Linux distributions. Although WebSEAL is the specific user of the PKCS#11 API in these examples, the PKCS#11 setup steps apply to any application that uses the PKCS#11 API on Linux for IBM System z, such as the IBM HTTP Server or WebSphere® MQSeries®.

Use Case 2a: Configuring Tivoli Access Manager for eBusiness WebSEAL 6.1 to use Cryptographic Hardware on Novell SLES 9 for IBM System z

Assumptions:

- Linux is running under z/VM® in an LPAR on a z9. The z/VM directory entry for the Linux Guest Userid has 'CRYPTO APVIRT' specified.
- One Crypto Express2 feature with each device configured as an accelerator is online to z/VM, which means that z/VM will make the two CEX2A accelerator devices available to Linux, and Linux will 'see' one virtual CEX2A device.
- The CPACF feature is enabled
- The SLES 9 level is SP3, 64-bit
- The libica, libica-32bit, openCryptoki, openCryptoki-32bit,compat-2004.7.1-1.2, and compat-32bit-9-200407011411 packages are all installed. The 32-bit packages are required because WebSEAL runs in 31-bit mode, even on the 64-bit Linux for IBM System z distributions.
- The openCryptoki package level is 2.1.6_rc6-0.6. Note that this level is included with SLES 9 SP4, or is available from the Novell maintenance Web site. This level fixes a segfault problem with the pkcsslotd daemon at the openCryptoki level included with SLES 9 SP3. (Novell Bugzilla 192665)
- AMeB WebSEAL 6.1 is installed, along with the GSKit package included with AMeB 6.1: gsk7bas-7.0-4.11.s390.rpm
- The Java™ Package included with AMeB 6.1 is installed: ibm-java2-s390-sdk-5.0-5.0.s390.rpm, and is the Java specified in the JAVA_HOME and PATH environment variables.

Step 1: Load z90crypt

a) The z90crypt **cryptographic** device driver must be loaded before attempting to start servers that have been configured to use cryptographic hardware. On SLES 9, the rcz90crypt initialization script provided by the libica rpm can be used. Issue the 'rcz90crypt start' command:

```
rcz90crypt start
Loading z90crypt module                                     done
```

(other rcz90crypt commands: stop|status|restart…….)

b) In the file /etc/sysconfig/z90crypt, read by z90crypt at load time, you can specify a cryptographic domain number, for the case where multiple domains are available. Multiple domains are not an issue under z/VM with CRYPTO APVIRT specified in the user directory. In this case, the default: Z90CRYPT_DOMAIN=-1 (autodetect) should be used.

c) Check file /var/log/messages – you should see messages similar to these:

```
Jul 18 18:50:41 mg8lnx04 kernel: z90crypt: Version 1.3.3 loaded, built on May
15 2006 16:08:30
Jul 18 18:50:41 mg8lnx04 kernel: z90crypt: z90main.o ($Revision: 1.31.2.12 $/
$Revision: 1.8.2.5 $/$Revision: 1.2.2.4 $)
Jul 18 18:50:41 mg8lnx04 kernel: z90crypt: z90hardware.o ($Revision: 1.19.2.8
$/$Revision: 1.8.2.5 $/$Revision: 1.2.2.4
```

d) To display the current status of the z90crypt driver, issue the command:

```
cat /proc/driver/z90crypt
```

The status of the online **cryptographic** devices (Adjunct Processors, or AP's) is shown:

```
z90crypt version: 1.3.3
Cryptographic domain: 0
Total device count: 1
PCICA count: 0
PCICC count: 0
PCIXCC MCL2 count: 0
PCIXCC MCL3 count: 0
CEX2C count: 0
CEX2A count: 1          <----- Only one cryptographic device under z/VM with CRYPTO APVIRT
requestq count: 0
pendingq count: 0       <----- Number of Requests queued
Total open handles: 0   <----- Number of users Note: SLES 9 SP3 ssh sessions open a handle

Online devices: 1=PCICA 2=PCICC 3=PCIXCC(MCL2) 4=PCIXCC(MCL3) 5=CEX2C 6=CEX2A
     0000000000006000 0000000000000000 0000000000000000 0000000000000000

Waiting work element counts
    0000000000000000 0000000000000000 0000000000000000 0000000000000000

Per-device successfully completed request counts
    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

The Online device array shows that AP 12 is online – a CEX2A device. The Waiting work element counters show that AP 12 has no work waiting, and the request counters show that AP 12 has completed no requests.

Step 2: Initialize the PKCS#11 token

a) Start the openCryptoki slot daemon (z90crypt must be loaded)

For PKCS#11 applications (such as WebSEAL) to access cryptographic hardware, the openCryptoki slot daemon, named pkcsslotd, must be started. An /etc/init.d script is provided, with a /usr/sbin/rcpkcsslotd symbolic link. Issue the command 'rcpkcsslotd start':

```
rcpkcsslotd start
Starting pkcsslotd daemon:usermod: `root' is primary group
name.                                              done
```

The first time that the pkcsslotd daemon is started, the group pkcs11 is created, and root is added as a member. PKCS#11 users must be members of this group. Also, the directory /etc/pkcs11 is created and initialized, with default token information.

b) Initialize the token. To initialize the PKCS#11 token and assign a label, login with root and run the pkcsconf utility with the –c <slot_number> and –I (upper case letter i) options. When prompted, type the default Security Officer PIN (87654321). The pkcsconf utility is located in the /usr/lib/pkcs11/methods directory:

1. `cd /usr/lib/pkcs11/methods`
2. `./pkcsconf -c 0 -I` (Initialize the token for slot #0)

```
   Enter the SO PIN: ********    <----------------- 87654321
   Enter a unique token label: share
```

c) Display the token. Use the pkcsconf –t command to display the token:

```
lncrpt01:/usr/lib/pkcs11/methods # ./pkcsconf -t
Token #0 Info:
Label: share
Manufacturer: IBM Corp.
Model: IBM ICA
Serial Number: 123
Flags: 0x880445
Sessions: -1/-1
R/W Sessions: -1/-1
PIN Length: 4-8.................
```

d) Set the SO (Security Officer) PIN. Change the SO PIN from the default value (87654321):

```
lncrpt01:/usr/lib/pkcs11/methods # ./pkcsconf -c 0 -P
Enter the SO PIN: ********           <---------------  87654321
Enter the new SO PIN: ********       <---------------  01234567 (for example)
Re-enter the new SO PIN: ********    <---------------  01234567
```

e) Set the User PIN

```
lncrpt01:/usr/lib/pkcs11/methods # ./pkcsconf -c 0 -u
Enter the SO PIN: ********           <---------------- 01234567
Enter the new user PIN: ********     <---------------- 87654321  (for example)
Re-enter the new user PIN: ********  <---------------- 87654321
```

f) Change the User PIN (The User PIN is expired after its initial setting)

```
lncrpt01:/usr/lib/pkcs11/methods # ./pkcsconf -c 0 -p
Enter user PIN: ********             <---------------- 87654321
Enter the new user PIN: ********     <---------------- 01234567 (for example)
Re-enter the new user PIN: ********  <---------------- 01234567
```

Notes on PINs:

1. Applications must specify the User PIN to access the token
2. PINs can be 4 through 8 characters in length - numbers, letters, or graphic symbols
3. Warning: If you don't change the User PIN after its initial setting, and try to open a token using an expired User PIN with utility gsk7ikm, it will fail with a java core file, on SLES 9 SP3.
4. Warning: Avoid the User PIN 12345678 – there is a hard coded check in openCryptoki 2.2 which will fail requests with that PIN
5. If you forget the PINs, you can remove the contents of file /etc/pkcs11 (or /var/lib/opencryptoki at openCryptoki level 2.2). Then a restart of the pkcsslotd daemon will reinitialize this directory, and then you can start over. However, removing that directory will remove certificates and keys created using the PKCS#11 token.

Step 3: Create a certificate using the PKCS#11 token

To manage certificates using the PKCS#11 token, the GSKit-supplied gsk7ikm utility can be used. gsk7ikm is a Java X-Windows application. Note the following considerations:

a) If using a 31-bit GSKit package (such as gsk7bas-7.0-4.11.s390.rpm) then a 31-bit Java Runtime is required (such as ibm-java2-s390-sdk-5.0-5.0.s390.rpm)

b) To use the gsk7ikm utility, your workstation must be enabled as an XWindows Server for your Linux system.

c) If using a 31-bit gsk7ikm utility, the Linux system must have the packages installed to support running a 31-bit XWindows application. For example:

```
expat-32bit-9-200407011411
zlib-32bit-9-200507230921
freetype2-32bit-9-200407011411
fontconfig-32bit-9-200407011411
XFree86-Mesa-32bit-9-200506070135
XFree86-libs-32bit-9-200512021711
```

d) GSKit level 7.0-4 supplies a version of libcrypto.so.0.9.7 that is incompatible with the version supplied by OpenSSL. Although GSKit loads its version for local use, there is a problem on SLES 9, which allows other libraries to access the incompatible libcrypto.so.0.9.7. This problem causes undefined symbol errors. This problem prevents the gsk7ikm utility from being able to open a PKCS#11 token, and, prevents WebSEAL from starting, when configured to use PKCS#11. When the error occurs, this message is seen in file /var/log/messages:

```
 openCryptokiModule[4100]: DL_Load: dlload of
[/usr/lib/pkcs11/stdll/PKCS11_ICA.so] failed; dlerror = [/usr/lib/libica.so:
undefined symbol: AES_set_decrypt_key]
```

The problem is only seen on SLES 9. It does not occur on SLES 10, RHEL4, or RHEL5.

Workaround: Specify LD_PRELOAD when starting the gsk7ikm utility to manage PKCS#11 tokens, or when starting WebSEAL configured to use PKCS#11. For example:

```
# LD_PRELOAD=/usr/lib/libcrypto.so.0.9.7 gsk7ikm
```

Note that on a 64-bit SLES 9 system, several of these messages will appear, which can be ignored:

```
ERROR: ld.so: object '/usr/lib/libcrypto.so.0.9.7' from LD_PRELOAD cannot be
preloaded: ignored.
```

The following technote describes the problem and offers an additional workaround when using the IBM HTTP Server:

```
http://www.ibm.com/support/docview.wss?
rs=177&context=SSEQTJ&uid=swg21313367&loc=en_US&cs=UTF-8&lang=en
```

e) Run the gsk7ikm utility from root

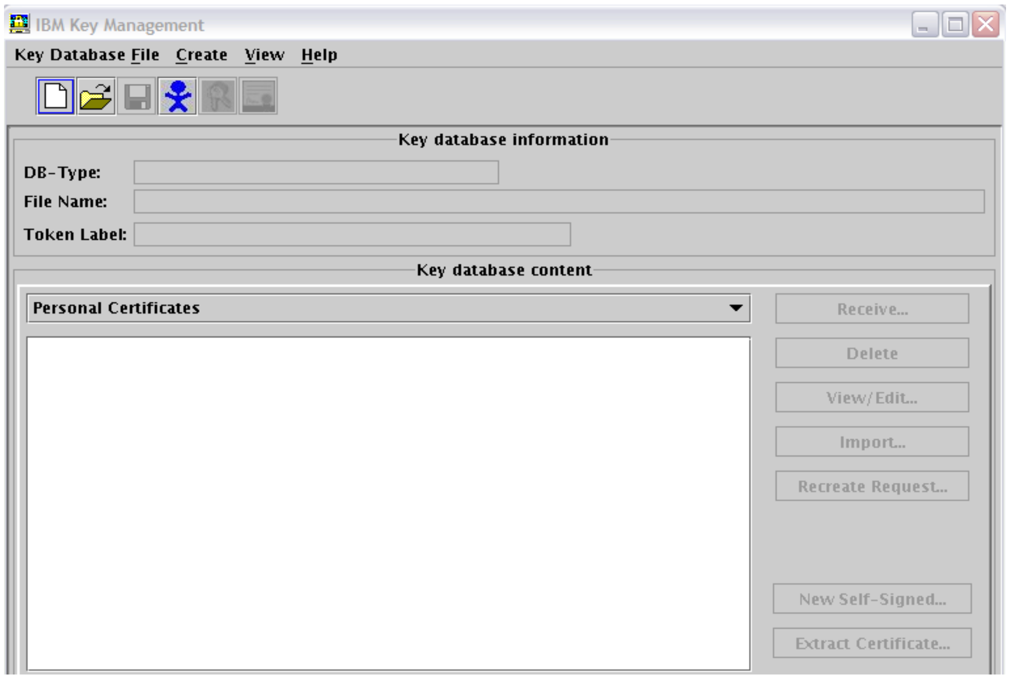
f) Set the JAVA_HOME environment variable to point to your Java Runtime, and ensure that Java is in the PATH. For example, issue the commands:

```
export JAVA_HOME=/opt/ibm/java2-s390-50/jre
export PATH=/opt/ibm/java2-s390-50/jre/bin:$PATH
```

g) Start the gsk7ikm utility:

```
# LD_PRELOAD=/usr/lib/libcrypto.so.0.9.7 gsk7ikm
```

When the utility has initialized, you see a panel similar to the following:



crypto7.jpg

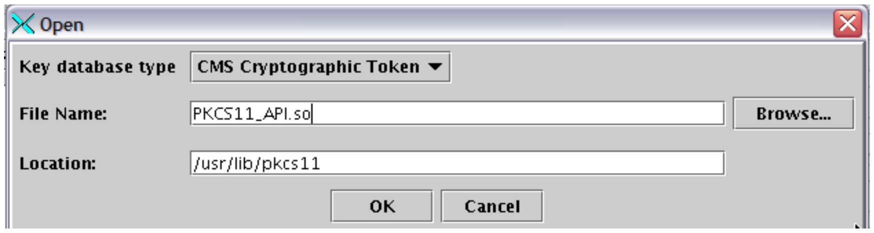
h) Click on Key Database File -> Open

Select Key database type of CMS Cryptographic Token

Type in the File name: **PKCS11_API.so** (this is the PKCS#11 driver provided by openCryptoki)

Type in the Location: **/usr/lib/pkcs11**

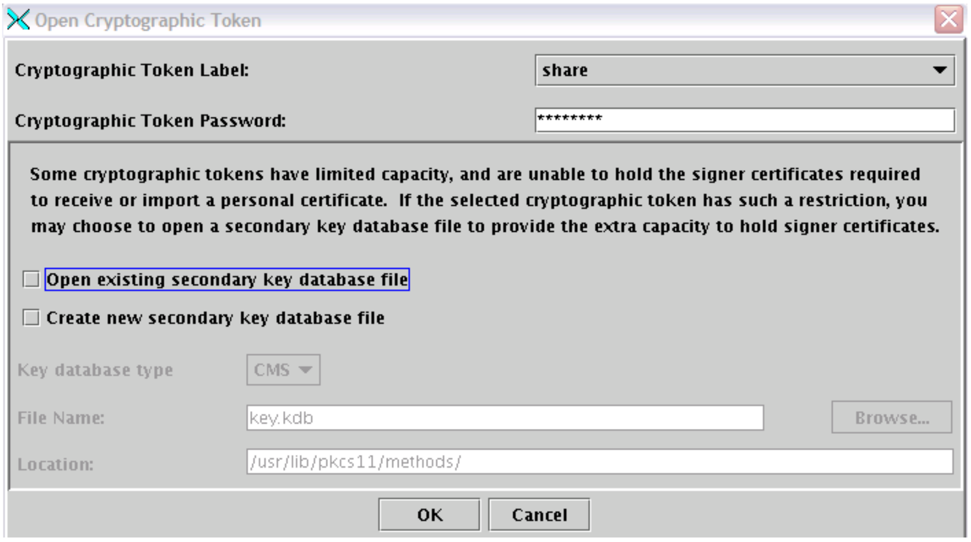
Then click OK:



Key database type: CMS Cryptographic Token
File Name: PKCS11_API.so
Location: /usr/lib/pkcs11

crypto8.jpg

i) Type in the Cryptographic Token Password – This is the User PIN set in Step 2. In this example the value is **01234567**
Also, clear the 'Open existing secondary key database file' box, then click OK:



Cryptographic Token Label: share

Cryptographic Token Password: ********

Some cryptographic tokens have limited capacity, and are unable to hold the signer certificates required to receive or import a personal certificate. If the selected cryptographic token has such a restriction, you may choose to open a secondary key database file to provide the extra capacity to hold signer certificates.

☐ Open existing secondary key database file

☐ Create new secondary key database file

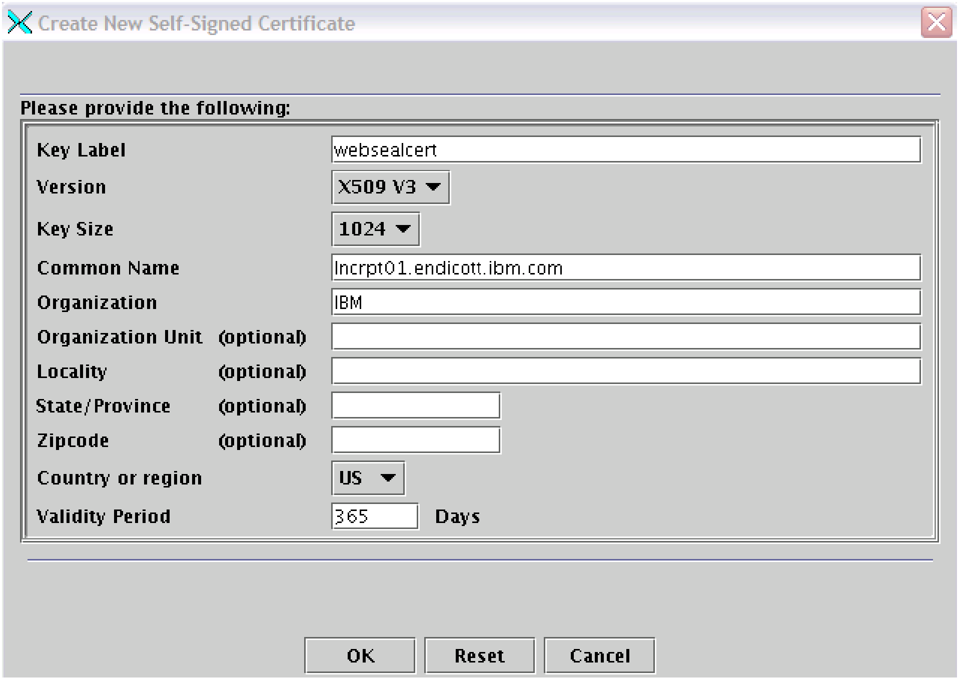
Key database type: CMS

File Name: key.kdb

Location: /usr/lib/pkcs11/methods/

crypto9.jpg

j) Click New Self-Signed
Type in a Key Label – **websealcert** (for example)
Type in a Common Name - **<your-hostname>**
Type in an Organization – **IBM** (for example)
Then click OK:



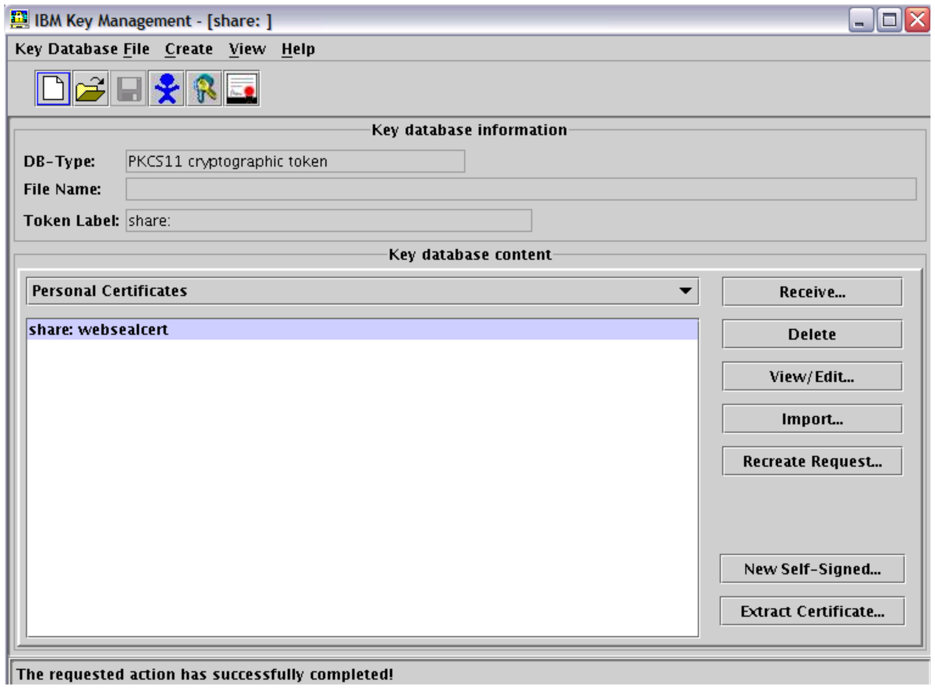
Please provide the following:

Key Label	websealcert
Version	X509 V3
Key Size	1024
Common Name	lncrpt01.endicott.ibm.com
Organization	IBM
Organization Unit (optional)	
Locality (optional)	
State/Province (optional)	
Zipcode (optional)	
Country or region	US
Validity Period	365 Days

crypto10.jpg

The newly-created self-signed certificate will be listed. It has the format token_label:cert_label.

To exit: Key Database File -> Exit



DB-Type: PKCS11 cryptographic token
File Name:
Token Label: share:

share: websealcert

The requested action has successfully completed!

crypto11.jpg

Another way to create a self-signed certificate using the PKCS#11 Token is to use the gsk7cmd command line utility instead of the gsk7ikm GUI: For example:

```
LD_PRELOAD=/usr/lib/libcrypto.so.0.9.7 gsk7cmd -cert -create -size 1024 -dn
"CN=lncrpt03.endicott.ibm.com,O=IBM,C=US" -label websealcert
-crypto /usr/lib/pkcs11/PKCS11_API.so -tokenlabel share -pw 01234567
```

Step 4: Configure WebSEAL to use PKCS#11

a) The userid of the WebSEAL server (named ivmgr) must be added to group pkcs11. Enter the usermod -G command as shown:

```
usermod -G ivmgr,tivoli,pkcs11 ivmgr
usermod: `ivmgr' is primary group name.
```

b) Make these updates to the WebSEAL server conf file (for example - /opt/pdweb/etc/webseald default.conf):

```
[server]
unix-group = pkcs11

[ssl]
webseal-cert-keyfile-label = share:websealcert <---certificate created using the
                                                    PKCS#11 token
pkcs11-driver-path = /usr/lib/pkcs11/PKCS11_API.so
pkcs11-token-label = share
pkcs11-token-pwd = 01234567              <----- User PIN set using the pkcsconf utility
pkcs11-symmetric-cipher-support = yes    <----- This directive enables the use of CPACF
```

```
[ssl-qop]
ssl-qop-mgmt = yes      <--------  enables 'quality of protection' management
default = DES-168       <--------  Sets cipher spec – also DES-56 or AES-128 will use CPACF
```

These parameters will allow WebSEAL to use the certificate generated earlier, using the PKCS#11 Token.

c) Restart WebSEAL (z90crypt must be loaded and the pkcsslotd daemon started)

Enter the pdweb restart command:

```
LD_PRELOAD=/usr/lib/libcrypto.so.0.9.7 pdweb restart
```

If WebSEAL does not start successfully check the log: /var/pdweb/log/msg__webseald-default.log

After successfully starting WebSEAL, the setup is now complete and you can verify it worked.

d) Issue the following command to verify the open handles count was incremented:

```
cat /proc/driver/z90crypt
z90crypt version: 1.3.3
Cryptographic domain: 0
Total device count: 1
....
CEX2C count: 0
CEX2A count: 1
requestq count: 0
pendingq count: 0
Total open handles: 1
```

e) Access WebSEAL from a browser using https:

```
https://<Linux-system's-hostname>
```

An https request which establishes an SSL session between the browser and WebSEAL results in the request counter in the output of the 'cat /proc/driver/z90crypt ' command being incremented:

```
Per-device successfully completed request counts
00000000 00000000 00000000 00000000 00000000 00000000 00000000 0000000
00000000 00000000 00000000 00000000 00000119 00000000 00000000 00000000
```

Use Case 2b: Configuring Tivoli Access Manager for eBusiness WebSEAL 6.1 to use Cryptographic Hardware on RHEL 5 for IBM System z

Assumptions:

- Linux is running under z/VM in an LPAR on an IBM System z9®. The z/VM directory entry for the Linux Guest Userid has 'CRYPTO APVIRT' specified.
- One Crypto Express2 feature with each device configured as an accelerator is online to z/VM, which means that z/VM will make the two CEX2A accelerator devices available to Linux, and Linux will 'see' one virtual CEX2A device.
- The CPACF feature is enabled
- Red Hat Enterprise Linux 5 GA is installed
- The following libica and openCryptoki packages are installed:
  ```
  libica-1.3.7-5.el5.s390x.rpm
  libica-1.3.7-5.el5.s390.rpm
  openCryptoki-2.2.4-15.el5.s390x.rpm
  openCryptoki-2.2.4-15.el5.s390.rpm
  ```
- The s390 (31-bit) packages are required because WebSEAL runs in 31-bit mode, even on the 64-bit Linux for IBM System z distributions.
- TAMeB WebSEAL 6.1 is installed, along with the GSKit package included with AMeB 6.1: gsk7bas-7.0-4.11.s390.rpm
- The Java Package included with AMeB 6.1 is installed: ibm-java2-s390-sdk-5.0-5.0.s390.rpm, and is the Java specified in the JAVA_HOME and PATH environment variables. This provides a 31-bit JRE.
- Both 64-bit and 31-bit compat rpms are installed:
  ```
  compat-libstdc++-295-2.95.3-85.s390.rpm
  compat-libstdc++-295-2.95.3-85.s390x.rpm
  compat-libstdc++-33-3.2.3-61.s390.rpm
  compat-libstdc++-33-3.2.3-61.s390x.rpm
  ```
- 31-bit X11 libraries are installed. For example, here is a partial list:
  ```
  zlib-1.2.3-3.s390.rpm                libXdmcp-1.0.1-2.1.s390.rpm
  expat-1.95.8-8.2.1.s390.rpm          libXau-1.0.1-3.1.s390.rpm
  freetype-2.2.1-16.el5.s390.rpm       libX11-1.0.3-8.el5.s390.rpm
  fontconfig-2.4.1-6.el5.s390.rpm      .............
  ```

Step 1: Load z90crypt

a) The z90crypt cryptographic device driver must be loaded before attempting to start servers that have been configured to use cryptographic hardware.

On RHEL5 use modprobe:

```
modprobe z90crypt
```
or:
```
modprobe z90crypt domain=x
```

Where domain is set to a value of -1, or a number between 0 and 15. A value of -1 means autodetect, which is the default. If multiple domains are available, you can specify a domain number. Multiple domains are not an issue under z/VM with CRYPTO APVIRT specified in the user directory, in which case, Linux “sees” one virtual domain, and the default (autodetect) works fine.

b) Issue the command: `cat /proc/driver/z90crypt`

The current status of the z90crypt driver is displayed similar to:

```
z90crypt version: 1.3.3
Cryptographic domain: 12
Total device count: 1
PCICA count: 0
PCICC count: 0
PCIXCC MCL2 count: 0
PCIXCC MCL3 count: 0
CEX2C count: 0
CEX2A count: 1           <--- Only one cryptographic device under z/VM with CRYPTO APVIRT

requestq count: 0
pendingq count: 0        <---  requests queued
Total open handles: 0    <---  Number of users

Online devices: 1=PCICA 2=PCICC 3=PCIXCC(MCL2) 4=PCIXCC(MCL3) 5=CEX2C 6=CEX2A
0000000000000000 0000000000000000 0060000000000000 0000000000000000
(Adjunct Processor  #34 is online - a CEX2A )

Waiting work element counts
0000000000000000 0000000000000000 0000000000000000 0000000000000000
(0 requests waiting for AP #34)

Per-device successfully completed request counts
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
(0 requests have been completed by AP #34)
```

Step 2: Initialize the PKCS#11 Token

a) Start the openCryptoki slot daemon (z90crypt must be loaded)

For PKCS#11 applications (such as WebSEAL) to access cryptographic hardware. The openCryptoki slot daemon, named pkcsslotd, must be started. Run the /etc/init.d/pkcsslotd script with the start parameter:

```
# /etc/init.d/pkcsslotd start
# [ OK ]
```

The first time that pkcsslotd is started, the group pkcs11 is created, and root is added as a member. PKCS#11 users need to be members of this group. Also, the directory /var/lib/opencryptoki is created and initialized with default token information.

b) Initialize the token

To initialize the PKCS#11 token and assign a label, login with root and run the pkcsconf utility with the –c <slot_number> and –I (upper case letter i) options. When prompted, enter the default Security Officer PIN (87654321). The pkcsconf utility is located in the /usr/sbin directory:

```
# pkcsconf -c 0 -I                 (Initialize the token for slot #0)
   Enter the SO PIN: ********  <-------------------------- 87654321
   Enter a unique token label: share
```

c) Display the token

Use the pkcsconf –t command to display the token:

```
# pkcsconf –t      (dot-slash pkcsconf dash t)
Token #0 Info:
Label: share
Manufacturer: IBM Corp.
Model: IBM ICA
Serial Number: 123
Flags: 0x880445
Sessions: -1/-1
R/W Sessions: -1/-1
PIN Length: 4-8.................
```

d) Set the SO (Security Officer) PIN

Change the SO PIN from the default value (87654321):

```
# pkcsconf -c 0 –P
Enter the SO PIN: ********          <---------------- 87654321
Enter the new SO PIN: ********      <---------------- 01234567 (for example)
Re-enter the new SO PIN: ********   <---------------- 01234567
```

e) Set the User PIN

```
# pkcsconf -c 0 -u
Enter the SO PIN: ********          <---------------- 01234567
Enter the new user PIN: ********    <---------------- 87654321  (for example)
Re-enter the new user PIN: ******** <---------------- 87654321
```

f) Change the User PIN (The User PIN is expired after its initial setting)

```
# pkcsconf -c 0 -p
Enter user PIN: ********            <---------------- 87654321
Enter the new user PIN: ********    <---------------- 01234567 (for example)
Re-enter the new user PIN: ******** <---------------- 01234567
```

Notes on PINs:

1. Applications must specify the User PIN to access the token
2. PINs can be 4 through 8 characters in length - numbers, letters, or graphic symbols
3. Warning: Avoid the User PIN 12345678 – there is a hard coded check in openCryptoki 2.2 which will fail requests with that PIN
4. If you forget the PINs, you can remove the contents of file /var/lib/opencryptoki. Then, a restart of the pkcsslotd daemon will reinitialize this directory, and you can start over. However, removing that directory will also remove any certificates and keys created using the PKCS#11 token.

Step 3: Create a certificate using the PKCS#11 Token

To manage certificates using the PKCS#11 Token, the GSKit-supplied gsk7ikm utility can be used. gsk7ikm is a Java X-Windows application. Note the following considerations:

a) If using a 31-bit GSKit package (such as gsk7bas-7.0-4.11.s390.rpm) then a 31-bit Java Runtime is required (such as ibm-java2-s390-sdk-5.0-5.0.s390.rpm)

b) To use the gsk7ikm utility, your workstation must be enabled as an XWindows Server for your Linux system.

c) If using a 31-bit gsk7ikm, the Linux system must have the packages installed to support running a 31-bit XWindows application. For example, here is a partial list of rpms required for 31-bit XWindows support:

```
zlib-1.2.3-3.s390.rpm                  libXfixes-4.0.1-2.1.s390.rpm
expat-1.95.8-8.2.1.s390.rpm            libXft-2.1.10-1.1.s390.rpm
freetype-2.2.1-16.el5.s390.rpm         libXinerama-1.0.1-2.1.s390.rpm
fontconfig-2.4.1-6.el5.s390.rpm        libXi-1.0.1-3.1.s390.rpm
libXau-1.0.1-3.1.s390.rpm              libXrandr-1.1.1-3.1.s390.rpm
libXcursor-1.1.7-1.1.s390.rpm          libXrender-0.9.1-3.1.s390.rpm
libXdamage-1.0.3-2.1.s390.rpm          libXres-1.0.1-3.1.s390.rpm
libXdmcp-1.0.1-2.1.s390.rpm            libXScrnSaver-1.1.0-3.1.s390.rpm
libXext-1.0.1-2.1.s390.rpm             libXt-1.0.2-3.1.fc6.s390.rpm
libXevie-1.0.1-3.1.s390.rpm            libXtst-1.0.1-3.1.s390.rpm
libXxf86misc-1.0.1-3.1.s390.rpm        libX11-1.0.3-8.el5.s390.rpm
```

d) Run the gsk7ikm utility from root

e) Set the JAVA_HOME environment variable to point to your Java Runtime, and ensure that Java is in the PATH. For example, enter the commands:

```
export JAVA_HOME=/opt/ibm/java2-s390-50/jre
export PATH=/opt/ibm/java2-s390-50/jre/bin:$PATH
```

f) Start the gsk7ikm utility:

```
# gsk7ikm
```

The steps to create a self-signed certificate using gsk7ikm are identical to those in **Use Case #2a: Configuring Tivoli Access Manager for eBusiness WebSEAL 6.1 to use cryptographic hardware on Novell SUSE Linux Enteprise Server 9 for IBM System z.** Refer to the steps in Use Case #2a showing the creation of the self-signed certificate.

Step 4: Configure WebSEAL to use PKCS#11

The steps to configure WebSEAL to use PKCS#11 on Red Hat Enterprise Linux 5 are identical to those in **Use Case #2a: Configuring Tivoli Access Manager for eBusiness WebSEAL 6.1 to use cryptographic hardware on Novell SUSE Linux Enteprise Server 9 for IBM System z,** with one exception: WebSEAL on REL 5 does not require `LD_PRELOAD=/usr/lib/libcrypto.so.0.9.7` to be specified when starting. Refer to the steps in Use Case #2a showing how to configure WebSEAL to use PKCS#11

Use Case 2c: Configuring Tivoli Access Manager for eBusiness WebSEAL 6.1 to use Cryptographic Hardware on Novell SLES 10 for IBM System z

Assumptions:

- Linux is running under z/VM in an LPAR on an IBM System z9. The z/VM directory entry for the Linux Guest Userid has 'CRYPTO APVIRT' specified.
- One Crypto Express2 feature with each device configured as an accelerator is online to z/VM, which means that z/VM will make the two CEX2A accelerator devices available to Linux, and Linux will 'see' one virtual CEX2A device.
- The CPACF feature is enabled
- SLES 10 SP1 is installed
- The following libica and openCryptoki packages are installed:

  ```
  libica-32bit-1.3.7-0.17
  libica-1.3.7-0.17
  openCryptoki-2.2.2-24.14
  openCryptoki-32bit-2.2.2-24.14
  ```

  And optionally, the 64-bit openCryptoki package: openCryptoki-64bit-2.2.2-24.14

  The s390 (32-bit) packages are required because WebSEAL runs in 31-bit mode, even on the 64-bit Linux for IBM System z distributions.
- Note that the openCryptoki level shown is the level shipped with SLES 10-SP1. With this openCryptoki-64bit level, the following rpm is a prerequisite: xcryptolinzGA-3.28-rc08.s390x.rpm

  The xcryptolinzGA-3.28-rc08.s390x.rpm provides support for secure-key **cryptographic functions**.

  This rpm can be obtained from this Web site:
  http://www.ibm.com/security/cryptocards/pcixcc/ordersoftware.shtml

  The xcryptolinzGA-3.28-rc8.s390x.rpm is not required when installing the 64-bit openCryptoki rpm at the SLES 10 SP2 level: openCryptoki-64bit-2.2.4-0.7.s390x.rpm
- AMeB WebSEAL 6.1 is installed, along with the GSKit package included with AMeB 6.1: gsk7bas-7.0-4.11.s390.rpm
- The Java Package included with AMeB 6.1 is installed: ibm-java2-s390-sdk-5.0-5.0.s390.rpm, and is the Java specified in the JAVA_HOME and PATH environment variables. This provides a 31-bit JRE.
- The following compat packages are installed (note the '32bit' versions) -

  ```
  compat-2006.1.25-11.2                compat-32bit-2006.1.25-11.2
  compat-libstdc++-5.0.7-22.2          compat-openssl097g-0.9.7g-13.5
  compat-openssl097g-32bit-0.9.7g-13.5
  ```
- The following 32-bit rpms are installed – these are required to run the GSKit gsk7ikm utility, which is an X-Windows application. Because the GSKit package provided with WebSEAL 6.1 is a 31-bit package, the 31-bit X-Windows packages must be installed on SLES 10:

  ```
  expat-32bit-2.0.0-13.2.s390x.rpm
  zlib-32bit-1.2.3-15.2.s390x.rpm
  freetype2-32bit-2.1.10-18.11.3.s390x.rpm
  ```

```
fontconfig-32bit-2.3.94-18.16.s390x.rpm
xorg-x11-libs-32bit-6.9.0-50.45.s390x.rpm
```

Step 1: Load z90crypt

a) The z90crypt cryptographic device driver must be loaded before attempting to start servers that have been configured to use cryptographic hardware.

b) On SLES 10, the rcz90crypt initialization script provided by the libica rpm can be used. Issue the 'rcz90crypt start' command:

```
rcz90crypt start
Loading z90crypt module                    done
```

(other rcz90crypt commands: stop|status|restart…….)

c) In the file: /etc/sysconfig/z90crypt, read by z90crypt at load time, a cryptographic domain number can be specified, in case multiple domains are available. Multiple domains are not an issue under z/VM with CRYPTO APVIRT specified in the user directory, in which case the default: Z90CRYPT_DOMAIN=- 1 (autodetect) works fine.

d) To check the status of the z90crypt driver, issue the command:

```
cat /proc/driver/z90crypt
```

The response will look similar to the following:

```
zcrypt version: 2.1.0
Cryptographic domain: 9
Total device count: 1
PCICA count: 0
..........................
CEX2C count: 0
CEX2A count: 1
```

Linux on IBM System z sees one CEX2A device, when one or more are online to z/VM

```
requestq count: 0
pendingq count: 0
Total open handles: 0

Online devices: 1=PCICA 2=PCICC 3=PCIXCC(MCL2) 4=PCIXCC(MCL3) 5=CEX2C 6=CEX2A
0060000000000000 0000000000000000 0000000000000000 0000000000000000

Waiting work element counts
0000000000000000 0000000000000000 0000000000000000 0000000000000000

Per-device successfully completed request counts
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

In the Online device array, the third position is a '6', meaning that Adjunct Processor (AP) number 2 is online, and is a CEX2A device (start counting with AP #0).

In the 'Waiting work element counts' array, the third position is '0', meaning that AP number 2 has no work waiting. In the 'Per-device successfully completed request counts' array, the third counter position is zeros, meaning that AP number 2 has not completed any requests.

Step 2: Initialize the PKCS#11 Token

a) Start the openCryptoki slot daemon (z90crypt must be loaded)

For PKCS#11 applications (such as WebSEAL) to access cryptographic hardware. The openCryptoki slot daemon, named pkcsslotd, must be started. An /etc/init.d script is provided – pkcsslotd, with a symbolic link - /usr/sbin/rcpkcsslotd

Run the rcpkcsslotd script with the start parameter:

```
# rcpkcsslotd start
    Starting pkcsslotd daemon:usermod: `root' is primary group name.
                                                                     done
```

The first time that pkcsslotd is started, the group pkcs11 is created, and root is added as a member. PKCS#11 users need to be members of this group. Also, the directory /var/lib/opencryptoki is created and initialized with default token information.

b) Initialize the token

To initialize the PKCS#11 token and assign a label, login with root and run the pkcsconf utility with the –c <slot_number> and –I (upper case letter i) options. When prompted, enter the default Security Officer PIN (87654321). The pkcsconf utility is located in the /usr/sbin directory:

```
# pkcsconf -c 0 -I                          (Initialize the token for slot 0)
Enter the SO PIN: ******** <-------------------------------------- 87654321
Enter a unique token label: share
```

c) Display the token

Use the pkcsconf –t command to display the token:

```
# pkcsconf -t
Token #0 Info:
Label: share
Manufacturer: IBM Corp.
Model: IBM ICA
Serial Number: 123
Flags: 0x880445
Sessions: -1/-1
R/W Sessions: -1/-1
PIN Length: 4-8..................
```

d) Set the SO (Security Officer) PIN

Change the SO PIN from the default value (87654321):

```
# pkcsconf -c 0 -P
Enter the SO PIN: ********             <--------------- 87654321
Enter the new SO PIN: ********         <--------------- 01234567  (for example)
Re-enter the new SO PIN: ********      <--------------- 01234567
```

e) Set the User PIN

```
# pkcsconf -c 0 -u
Enter the SO PIN: ********             <--------------- 01234567
Enter the new user PIN: ********       <--------------- 87654321  (for example)
Re-enter the new user PIN: ********    <--------------- 87654321
```

f) Change the User PIN (The User PIN is expired after initial setting)

```
# pkcsconf -c 0 -p
Enter user PIN: ********            <--------------- 87654321
Enter the new user PIN: ********    <--------------- 01234567  (for example)
Re-enter the new user PIN: ******** <--------------- 01234567
```

Notes on PINs:

1) Applications must specify the User PIN to access the token
2) PINs can be 4 through 8 characters in length: numbers, letters, or graphic symbols
3) Warning: Avoid the User PIN 12345678 – there is a hard coded check in openCryptoki 2.2 which will fail requests with that PIN.
4) If you forget the PINs, you can remove the contents of file /var/lib/opencryptoki. Then, a restart of the pkcsslotd daemon will reinitialize this directory, and you can start over. However, removing that directory will also remove any certificates and keys created using the PKCS#11 token.

Step 3: Create a certificate using the PKCS#11 token

To manage certificates using the PKCS#11 token, the GSKit-supplied gsk7ikm utility can be used. gsk7ikm is a Java X-Windows application. Note the following considerations:

a) If using a 31-bit GSKit package (such as gsk7bas-7.0-4.11.s390.rpm), a 31-bit Java Runtime is required (such as ibm-java2-s390-sdk-5.0-5.0.s390.rpm)

b) To use the gsk7ikm utility, your workstation must be enabled as an XWindows Server for your Linux system.

c) If using the 31-bit gsk7ikm, the Linux system must have the packages installed to support running a 31-bit XWindows application. For example, the following rpms had to be installed on SLES 10 SP1 to run the gsk7ikm utility:

```
expat-32bit-2.0.0-13.2.s390x.rpm
zlib-32bit-1.2.3-15.2.s390x.rpm
freetype2-32bit-2.1.10-18.11.3.s390x.rpm
fontconfig-32bit-2.3.94-18.16.s390x.rpm
xorg-x11-libs-32bit-6.9.0-50.45.s390x.rpm
```

d) Run the gsk7ikm utility from root

e) Set the JAVA_HOME environment variable to point to your Java Runtime, and ensure that Java is in the PATH. For example, issue the commands:

```
export JAVA_HOME=/opt/ibm/java2-s390-50/jre
export PATH=/opt/ibm/java2-s390-50/jre/bin:$PATH
```

f) Start the gsk7ikm utility:

```
# gsk7ikm
```

The steps to create a self-signed certificate using gsk7ikm are identical to those in **Use Case #2a: Configuring Tivoli Access Manager for eBusiness WebSEAL 6.1 to use cryptographic hardware on Novell SUSE Linux Enteprise Server 9 for IBM System z.** Refer to the steps in Use Case #2a showing the creation of the self-signed certificate.

Step 4: Configure WebSEAL to use PKCS#11

The steps to Configure WebSEAL to use PKCS#11 on Novell SLES 10 are identical to those in **Use Case #2a: Configuring Tivoli Access Manager for eBusiness WebSEAL 6.1 to use cryptographic hardware on Novell SUSE Linux Enteprise Server 9 for IBM System z,** with one exception: WebSEAL on Novell SLES 10 does not require `LD_PRELOAD=/usr/lib/libcrypto.so.0.9.7` to be specified when starting. Refer to the steps in Use Case #2a showing how to configure WebSEAL to use PKCS#11.

Use Case 3: Creating a certificate request and receiving a signed certificate using a PKCS#11 cryptographic token

This example uses the key management utility supplied with GSKit: named gsk7ikm, to create a certificate request for a personal certificate using the PKCS#11 Token. Send the certificate request to be signed by a Certificate authority on one of your own test systems. Then 'Add' the Certificate Authority certificate (CA cert) to the PKCS#11 token, and, receive the signed personal certificate into the PKCS#11 token. Use the personal certificate as the server certificate for WebSEAL. This allows WebSEAL to use the IBM System z Cryptographic hardware to accelerate both asymmetric and symmetric cryptographic operations.

This example uses WebSEAL 6.1, and the GSKit (7.0-4.11 31- bit) and Java JRE (5.0-5.0 31-bit) which come with Tivoli Access Manager for e-Business.

Assumptions:

- The Linux System is Novell SLES 10 SP2
- The openCryptoki rpms installed are:
  `openCryptoki-64bit-2.2.4-0.7`
  `openCryptoki-2.2.4-0.7`
  `openCryptoki-32bit-2.2.4-0.7`
- The libica rpms installed are:
  `libica-1.3.8-0.6`
  `libica-32bit-1.3.8-0.6`
- There is a Crypto-Express2 feature configured with both devices in accelerator mode online to z/VM, which means the Linux guest will see one CEX2A device.
- The z90crypt **cryptographic** device driver has been loaded, and the pkcsslotd daemon has been started.
- The PCKS11 (OpenCryptoki) setup has already been completed - the hardware token has been initialized, and the User PIN set to 01234567.

Step 1: start the gsk7ikm utility and open the Cryptographic Token

a) Prepare to start the gsk7ikm utility by setting the JAVA_HOME and PATH variables:

```
# export JAVA_HOME=/opt/ibm/java2-s390-50/jre
# export PATH=/opt/ibm/java2-s390-50/jre/bin:$PATH
```

b) Start the gsk7ikm utility:

```
# gsk7ikm
```

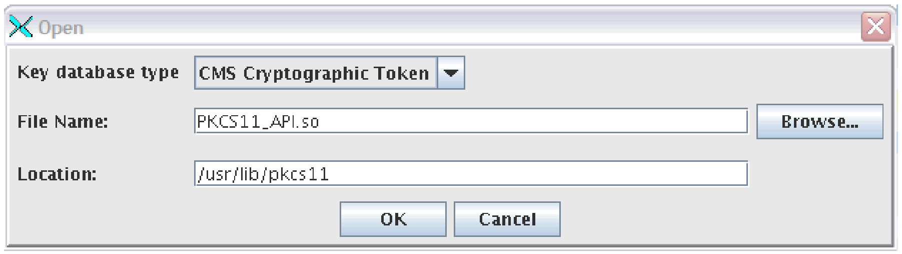
c) When the utility has initialized, select Key database File -> Open

d) Select Key database type of CMS Cryptographic Token

Type in the File Name: `PKCS11_API.so`

Type in the Location: `/usr/lib/pkcs11`

Click OK



Open

Key database type: CMS Cryptographic Token

File Name: PKCS11_API.so

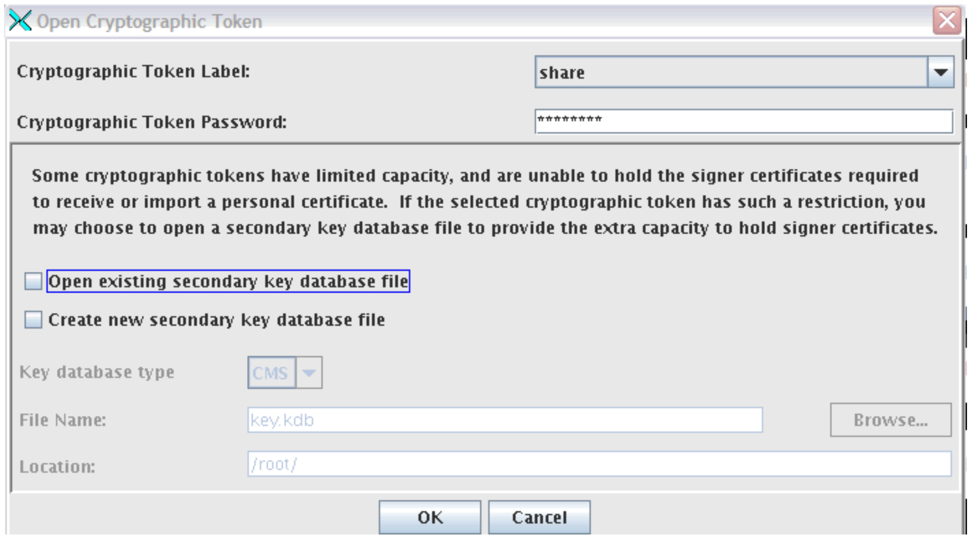
Location: /usr/lib/pkcs11

crypto12.jpg

e) On the 'Open Cryptographic Token Panel:

Specify the Cryptographic Token Password: `01234567` (for this example)

Clear the box - 'Open existing secondary key database file' and click OK



Open Cryptographic Token

Cryptographic Token Label: share

Cryptographic Token Password: *********

Some cryptographic tokens have limited capacity, and are unable to hold the signer certificates required to receive or import a personal certificate. If the selected cryptographic token has such a restriction, you may choose to open a secondary key database file to provide the extra capacity to hold signer certificates.

☐ Open existing secondary key database file

☐ Create new secondary key database file

Key database type: CMS

File Name: key.kdb

Location: /root/

crypto13.jpg

Why clear the box 'Open existing secondary key database file'?

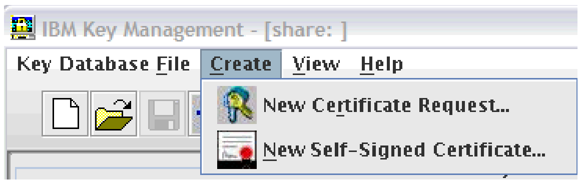
- When using clear-key cryptography, even though we store keys & certificates in a 'Hardware Token', the objects are not actually being stored in the hardware device, but in the Linux file system. As long as there is space in the Linux file system, and as long as the openCryptoki limitation of 2048 public and 2048 private token objects is not exceeded, then a secondary key database is not needed.
- In order to receive a certificate that has been created using the Hardware token, signed by a Certificate Authority, and then returned, the Certificate Authority root certificate, and any intermediate certificates, must be present in the Hardware token. It is not sufficient to have the CA root certificate and intermediate certificates in a secondary key database. An attempt to receive a certificate, when the certificate request was generated using the Hardware token, but the CA root certificate or intermediate certificates are not in the Hardware token as trusted signers, might result in the error message:

```
An error occurred while receiving the certificate from the given file.
```

```
The certificate request created for the certificate is not in the key database
```

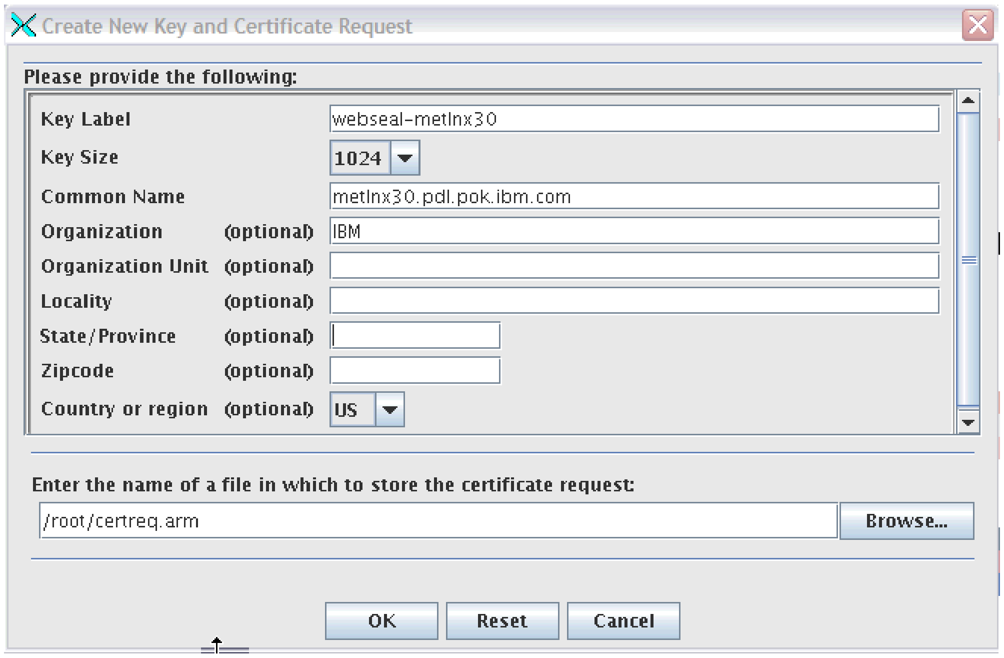
Step 2: Create a Personal Certificate Request

a) At the top of the 'IBM Key Management' panel, with the PKCS#11 cryptographic token open, select Create, → New Certificate Request:


IBM Key Management - [share:]
Key Database File
Create
View
Help
New Certificate Request...
New Self-Signed Certificate...

crypto14.jpg

b) On the 'Create New Key and Certificate Request' panel, Fill in a Key Label, and the values to be used in the Certificate, such as Common Name, Organization, and so forth... Note the location of the output file that will contain the certificate request (certreq.arm in this example):



Create New Key and Certificate Request

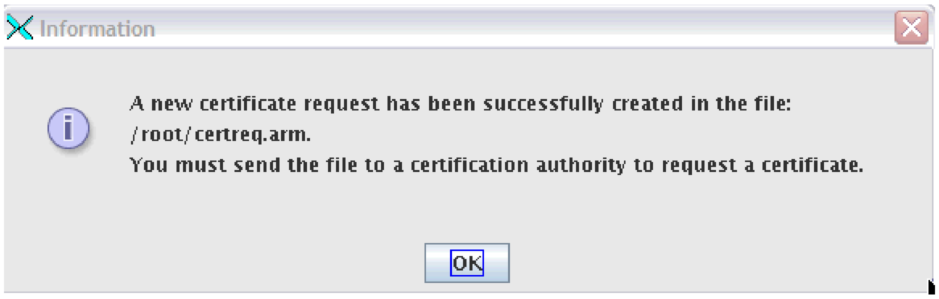
Please provide the following:

Key Label		webseal-metlnx30
Key Size		1024
Common Name		metlnx30.pdl.pok.ibm.com
Organization	(optional)	IBM
Organization Unit	(optional)	
Locality	(optional)	
State/Province	(optional)	
Zipcode	(optional)	
Country or region	(optional)	US

Enter the name of a file in which to store the certificate request:
/root/certreq.arm

crypto15.jpg

c) After filling in the Certificate request information, click OK. The following Information panel is displayed similar to:



Information

A new certificate request has been successfully created in the file:
/root/certreq.arm.
You must send the file to a certification authority to request a certificate.

crypto16.jpg

Step 3: Send the Certificate Request to be signed

Send the file certreq.arm, containing the request, to a Certificate Authority to be signed.

For this example, a Certificate Authority has been set up on a Linux system using OpenSSL. Send the certificate request file to that system (having renamed it to a more descriptive name) - webseal-metlnx30.csr. And, sign the certificate using the openssl 'ca' command:

```
/etc/ssl  # openssl ca -config openssl.cnf -in webseal-metlnx30.csr -out
webseal-metlnx30.crt
Using configuration from openssl.cnf
Enter pass phrase for .//mg8lnx07.key: password
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 17 (0x11)
        Validity
            Not Before: Jul  8 21:10:28 2008 GMT
            Not After : Jul  8 21:10:28 2009 GMT
        Subject:
            countryName               = US
            organizationName          = IBM
            commonName                = metlnx30.pdl.pok.ibm.com
        X509v3 extensions:
            ............................................
        X509v3 Authority Key Identifier:
                DirName:/C=US/O=IBM/CN=mg8lnx07.pdl.pok.ibm.com
                serial:00

Certificate is to be certified until Jul  8 21:10:28 2009 GMT (365 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
/etc/ssl #
```

The output file is sent from the signing operation: webseal-metlnx30.crt, as well as the certificate containing the public key of the signer, back to the WebSEAL system.

Step 4: Add the Root CA Certificate of the signer to the PKCS#11 token

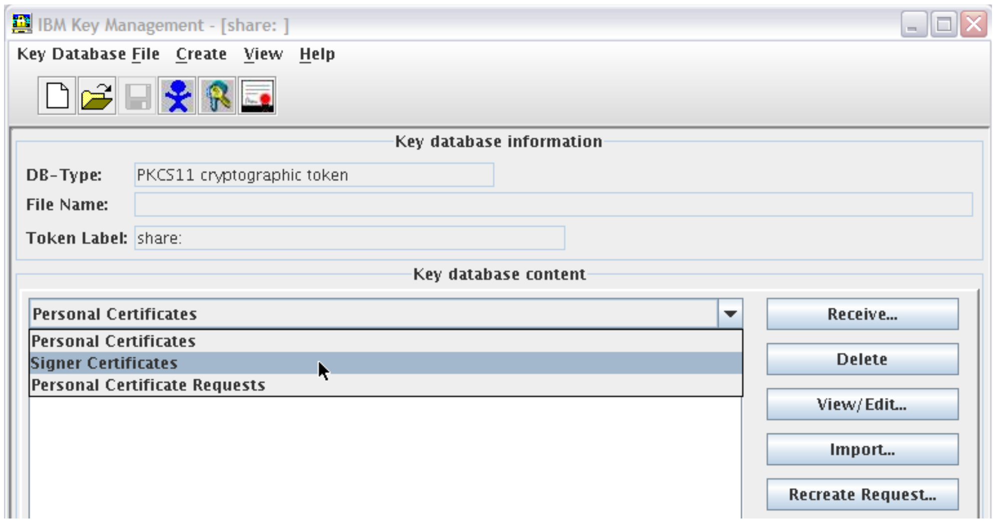
Before you receive the signed certificate, you have to receive the certificate with the public key that corresponds to the private key used to sign the certificate request.

In this example, the 'Certificate Authority' or CA certificate has been stored in a file named mg8lnx07.crt, which contains the certificate and public key in PEM (Privacy Enhanced Mail) format:

```
-----BEGIN CERTIFICATE-----
MIIB6zCCAVQCAQAwDQYJKoZIhvcNAQEEBQAwPjELMAkGA1UEBhMCVVMxDDAKBgNV
BAoTA0lCTTEhMB8GA1UEAxMYbWc4bG54MDcucGRsLnBvay5pYm0uY29tMB4XDTA3
MDUxNzE5MDUyNVoXDTEwMDIwOTE5MDUyNVowPjELMAkGA1UEBhMCVVMxDDAKBgNV
BAoTA0lCTTEhMB8GA1UEAxMYbWc4bG54MDcucGRsLnBvay5pYm0uY29tMIGfMA0G
CSqGSIb3DQEBAQUAA4GNADCBiQKBgQDEWt53T+sIU0E8/Mkj7vm/g4g91Qi1Cs1l
RxlWIBCBeio2eKpYm5xmeB+Dwe+w0mVdw1+vXkAmF99VUTWCTnI6ewksy3mBmZo+
Mz6LF5c6k8nUOflYWvQ7qx4pHa/1c6oI08VYf6joppkGHDBXD34Baq141iXXRU1q
```

```
TBfWgWCSOwIDAQABMA0GCSqGSIb3DQEBBAUAA4GBAJBgwjXkEs87TaqK9mvhe1SU
isaV+8cpIVVdqmew60P8iPYfnGosq6k2gLRA9yzV2NVs350z1U638eefwZescWyB
e3aM79r9UHF89rVD23yNWv0ET0msEIfGeARR4IOqqyP5nbl29e6riVJr6nVvoZlT
I74j6zTkITXPYp7pTspR
-----END CERTIFICATE-----
```

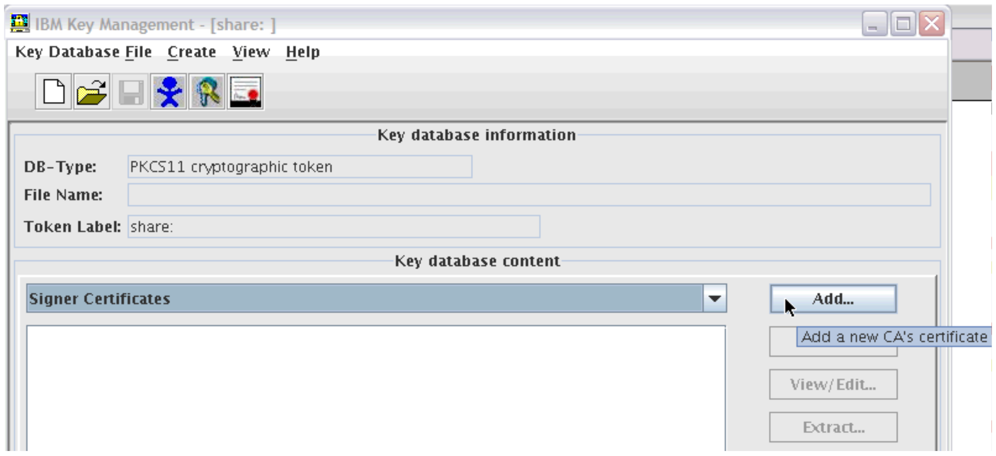
a) In the Key Management utility, after opening the CMS Cryptographic token, on the IBM Key Management panel, click on the drop-down arrow under 'Key database content', and choose Signer Certificates:



DB-Type: PKCS11 cryptographic token
File Name:
Token Label: share:

crypto17.jpg

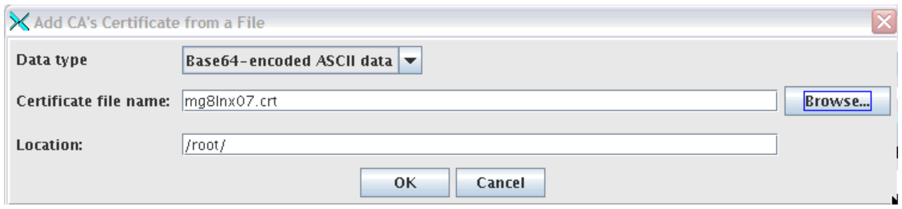
b) Click Add,



DB-Type: PKCS11 cryptographic token
File Name:
Token Label: share:

crypto18.jpg

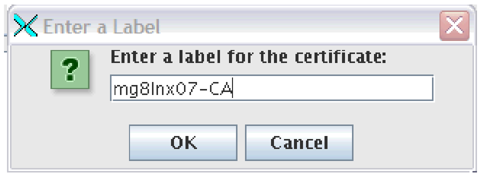
c) Select the Data type of Base64-encoded ASCII data, and the file with the CA certificate to be added, and click OK:



Data type: Base64-encoded ASCII data
Certificate file name: mg8lnx07.crt
Location: /root/

crypto19.jpg

d) Specify a label, and click OK:

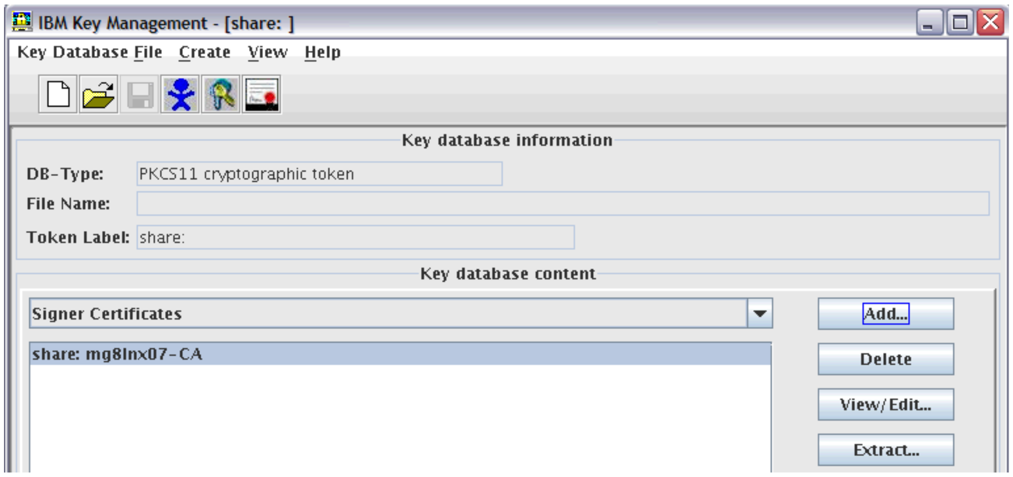

Enter a Label

Enter a label for the certificate:

mg8lnx07-CA

crypto20.jpg

e) The CA certificate is now displayed similar to this list of Signer certificates:


IBM Key Management - [share:]

Key database information

DB-Type: PKCS11 cryptographic token

File Name:

Token Label: share:

Key database content

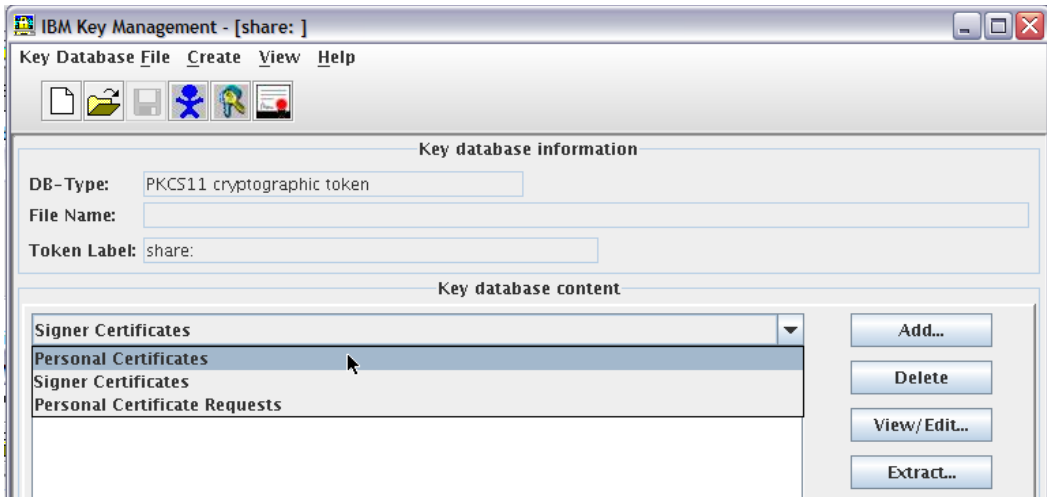
Signer Certificates

share: mg8lnx07-CA

crypto21.jpg

Step 5: Receive the signed personal certificate into the PKCS#11 token

a) Now that the signer certificate has been added, you can receive the signed certificate – choose 'Personal Certificates' from the Key database content drop-down:


IBM Key Management - [share:]

Key database information

DB-Type: PKCS11 cryptographic token

File Name:

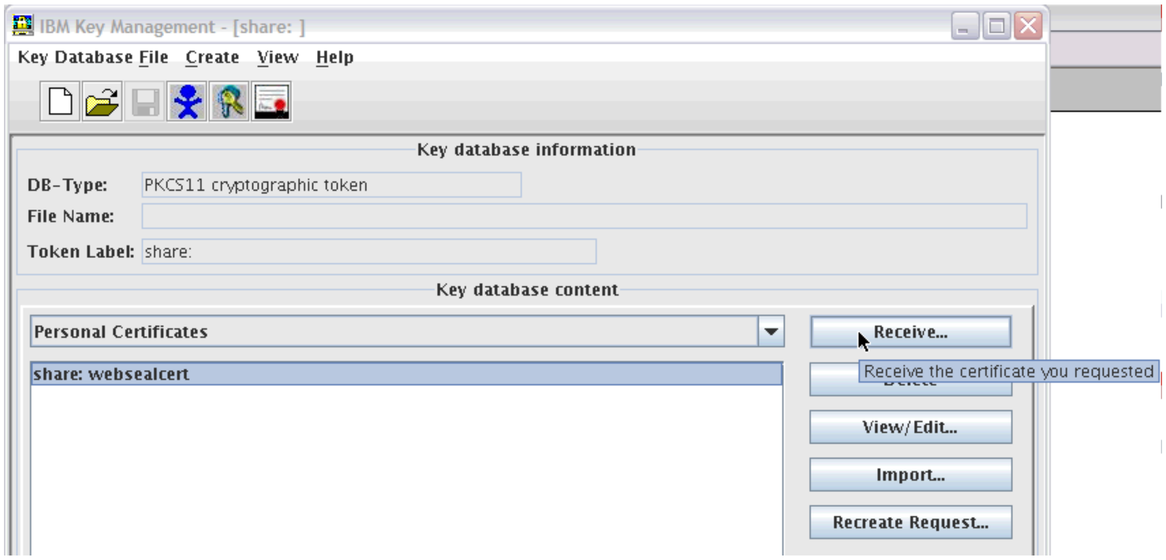
Token Label: share:

Key database content

Signer Certificates

crypto22.jpg

b) Click Receive

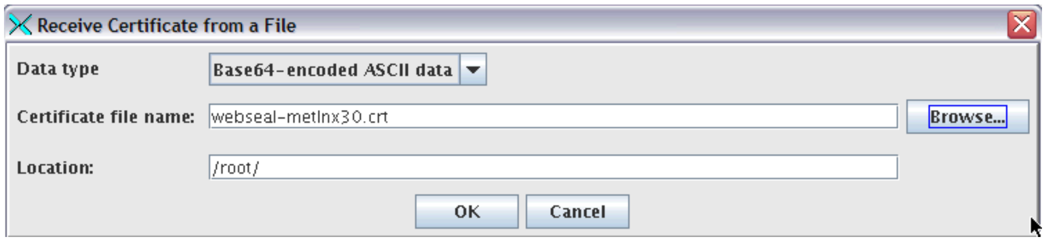

IBM Key Management - [share:]
Key Database File Create View Help
Key database information
Key database content
Receive...
Receive the certificate you requested
View/Edit...
Import...
Recreate Request...

DB-Type: PKCS11 cryptographic token
File Name:
Token Label: share:

Personal Certificates
share: websealcert

crypto23.jpg

c) Specify Data type of Base64-encoded ASCII data, and specify the file containing the signed certificate request:

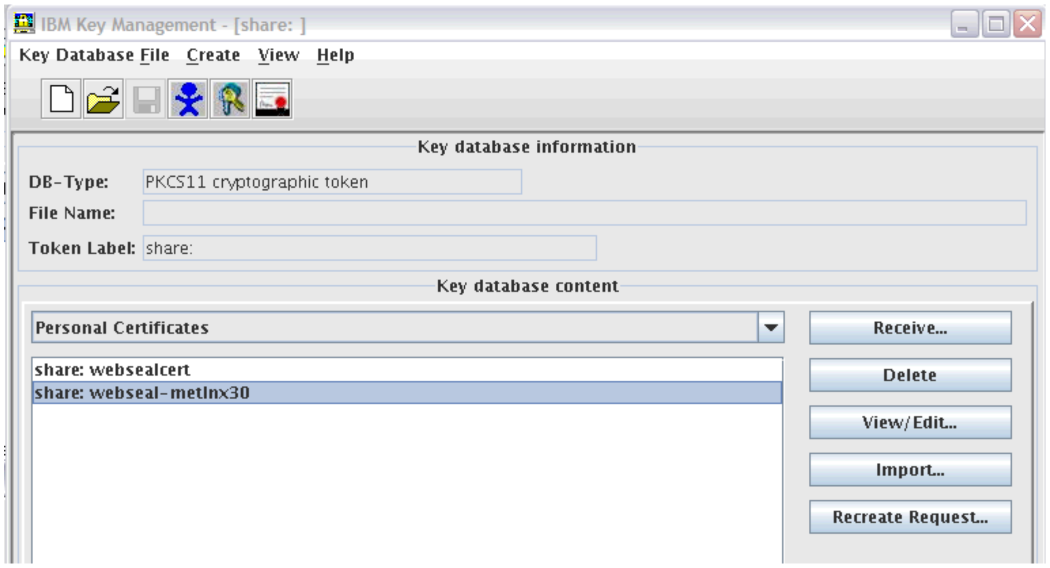


Data type: Base64-encoded ASCII data
Certificate file name: webseal-metlnx30.crt
Location: /root/

crypto24.jpg

d) Click OK

The received certificate is then displayed similar to this list of Personal Certificates:



DB-Type: PKCS11 cryptographic token
File Name:
Token Label: share:

Personal Certificates
share: websealcert
share: webseal-metlnx30

crypto25.jpg

Note that the certificate label has the format: Token_Label:Certificate_Label, where this Token_Label is 'share' and the Certificate_Label is 'webseal-metlnx30'.

To use this CA-signed certificate, you update your WebSEAL configuration file. The label of the server certificate that WebSEAL will send when clients issue https requests is specified as share:webseal-metlnx30:

```
webseal-cert-keyfile-label = share:webseal-metlnx30
```

You enable the PKCS#11 options in the WebSEAL configuration file:

```
pkcs11-driver-path = /usr/lib/pkcs11/PKCS11_API.so
pkcs11-token-label = share
pkcs11-token-pwd = 01234567  <---------------- User PIN
pkcs11-symmetric-cipher-support = yes
```

And restart WebSEAL -

```
# pdweb restart
```

Access the WebSEAL server from a browser using https: https://<Linux-system’s-host name>

An https request to WebSEAL increments the counter in the `cat /proc/driver/z90crypt` display,showing that the SSL handshake has resulted in a successful request completed by the CEX2A device:

```
Per-device successfully completed request counts
00000000 00000138 00000000 00000000 00000000 00000000 00000000 00000000
```

Use Case 4: Using a Test certificate from Thawte with the PKCS#11 Hardware token and the IBM HTTP Server on Novell SLES 10

Assumptions:

- The Linux System is Novell SLES 10 SP2
- The IBM HTTP Server is 6.1.0.17 (32-bit executable)
- The openCryptoki rpms installed are:
  ```
  openCryptoki-64bit-2.2.4-0.7
  openCryptoki-2.2.4-0.7
  openCryptoki-32bit-2.2.4-0.7
  ```
- The libica rpms installed are:
  ```
  libica-1.3.8-0.6
  libica-32bit-1.3.8-0.6
  ```
- There is a Crypto-Express2 feature configured with both devices in accelerator mode online to z/VM, which means that the Linux guest will see one CEX2A device.
- The z90crypt **cryptographic** device driver has been loaded, and the pkcsslotd daemon has been started.
- The PKCS#11 (OpenCryptoki) setup has already been completed - the PKCS#11 token has been initialized, and the User PIN set to 01234567.

Planning Considerations

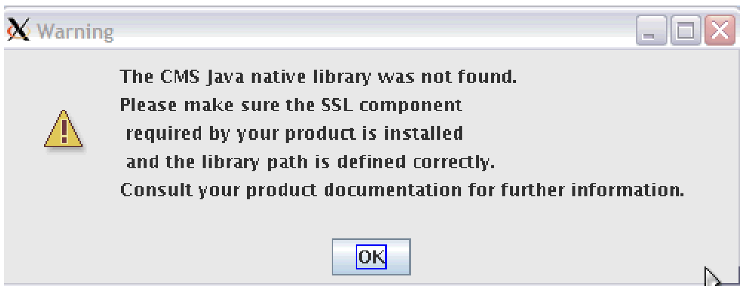
Note the following considerations when planning to configure the IBM HTTP Server with Cryptographic Hardware on Linux on IBM System z:

1) Install Fixpack 17 to resolve ikeyman issues

The base 6.1.0.0 IBM HTTP Server installed from the WebSphere 6.1 Network Deployment 64-bit

Supplement CD image for Linux on IBM System z provides a 32-bit HTTP Server, a 32-bit GSKit package, and a 64-bit Java Runtime.

The IBM HTTP Server 6.1 Info Center instructs that Key Management be performed by running ikeyman from the /opt/IBM/HTTPServer/bin directory. This version of ikeyman sets the Java environment to use the 64-bit Java Runtime located in the /opt/IBM/HTTPServer/java directory. When the ikeyman utility is run from /opt/IBM/HTTPServer/bin, and an attempt is made to open the CMS Cryptographic token, the following error is received:



The CMS Java native library was not found.
Please make sure the SSL component
required by your product is installed
and the library path is defined correctly.
Consult your product documentation for further information.

crypto26.jpg

The recommendation is to install IBM HTTP Server Fixpack 17 or later. Fixpack 17 provides an updated GSKit package: both 32-bit and 64-bit versions. The 32-bit version is required for the 32-bit HTTP server.

The 64-bit version is required to run ikeyman with the 64-bit Java Runtime.

2) Fixpack 3 or later is required for the IBM HTTP Server to use cryptographic hardware.

In the base level of IHS 6.1.0, the use of cryptographic hardware would fail with this error in the error_log:

```
SSL0227E: SSL Handshake Failed, Specified label could not be found in the key
file.
```

See APAR PK28359 for more information. The problem is resolved in Fixpack 3 or later.

3) Set environment variable LD_PRELOAD required on Novell SLES 9

Although this section deals with running IHS on SLES 10, it is worth noting the following problem when running on SLES 9: On Novell SLES 9, with GSKit 7.0-4 installed, it is necessary to set LD_PRELOAD libcrypto.so.0.9.7, either when starting ikeyman, or, when starting the IBM HTTP Server. For example, to start ikeyman from the /opt/IBM/HTTPServer/bin directory, when using IHS installed from the 64-bit Supplemental CD, it is necessary to LD_PRELOAD the 64-bit library:

```
/opt/IBM/HTTPServer/bin # LD_PRELOAD=/usr/lib64/libcrypto.so.0.9.7 ./ikeyman
```

To start the IBM HTTP Server, which is a 32-bit executable, it is necessary to LD_PRELOAD the 32-bit library:

```
/opt/IBM/HTTPServer/bin # LD_PRELOAD=/usr/lib/libcrypto.so.0.9.7 ./apachectl
  start
```

After issuing a command with LD_PRELOAD, one or more of these messages could be observed, which is OK:

```
ERROR: ld.so: object '/usr/lib/libcrypto.so.0.9.7' from LD_PRELOAD cannot be
   preloaded: ignored.
```

To determine the level of GSKit installed, issue the command:

```
# rpm -qa | grep gsk7
  gsk7bas64-7.0-4.14
  gsk7bas-7.0-4.14
```

If the response shows a 7.0-4 level of the gsk7bas rpm, as shown above, then the LD_PRELOAD is needed when running on Novell SLES 9.

LD_PRELOAD is not required on Novell SLES 10 or on Red Hat distributions.

The following technote describes the problem and offers additional workarounds:

```
http://www.ibm.com/support/docview.wss?
rs=177&context=SSEQTJ&uid=swg21313367&loc=en_US&cs=UTF-8&lang=en
```

4) Support for IBM System z Cryptographic Hardware

IBM HTTP Server 6.1 supports using a Cryptographic coprocessor/accelerator card such as the Crypto Express2 to accelerate the RSA part of the SSL handshake on Linux for IBM System z. IBM HTTP Server 6.1 does not at this time support the use of the CPACF instructions.

Step 1: Start the Key Management tool and open the PKCS#11 token

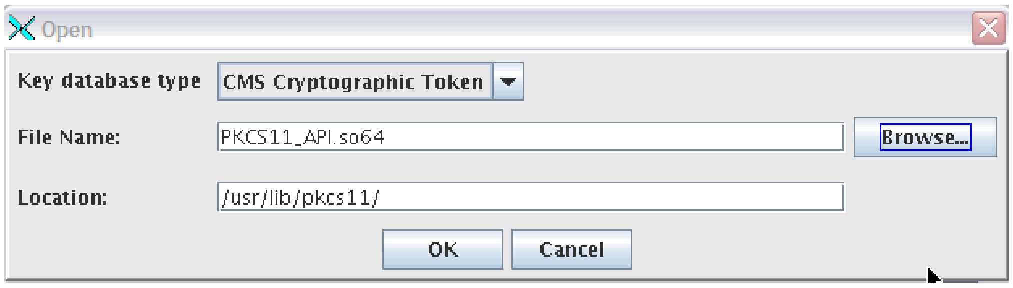
a) The /opt/IBM/HTTPServer/bin/ikeyman script sets the Java environment prior to invoking the ikeyman utility:

```
/opt/IBM/HTTPServer/bin # ./ikeyman
```

b) Note that on Novell SUSE Enterprise Server 9, the LD_PRELOAD has to be specified if GSKit 7.0-4 is installed:

```
/opt/IBM/HTTPServer/bin # LD_PRELOAD=/usr/lib64/libcrypto.so.0.9.7 ./ikeyman
```

c) When the key management utility starts, select Key Database File -> Open
Select the Key database type: CMS Cryptographic token
Specify File Name: `PKCS11_API.so64` (Need the 64-bit PKCS#11 driver because you are using a 64-bit JRE)
Specify Location: `/usr/lib/pkcs11/`

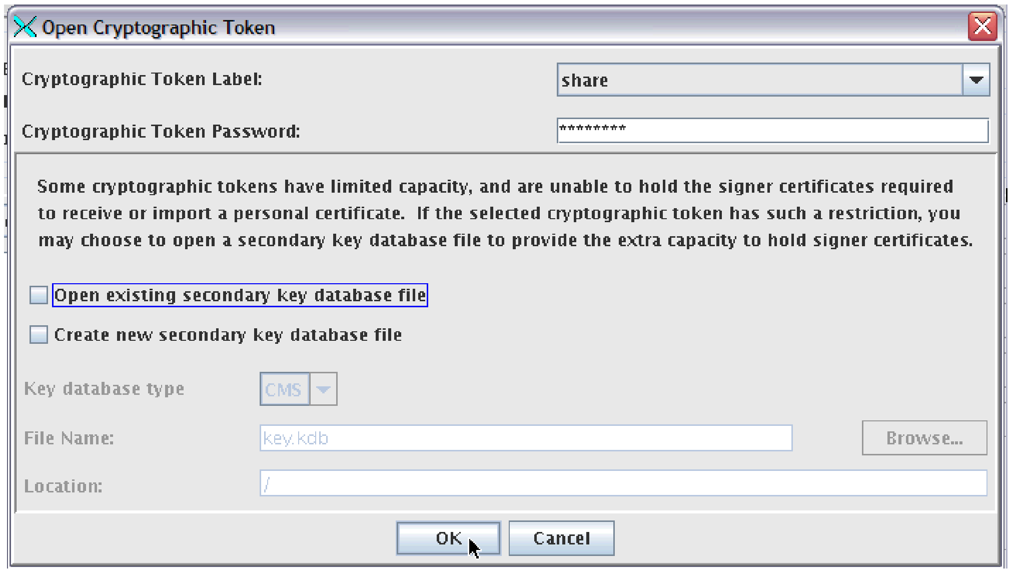


Key database type: CMS Cryptographic Token
File Name: PKCS11_API.so64
Location: /usr/lib/pkcs11/

crypto27.jpg

d) Click OK

e) On the 'Open Cryptographic Token' panel that follows, the Token label is filled in. Type the Cryptographic Token Password (the User Pin which you have set to 01234567).
Clear the box 'Open existing secondary key database file', and click OK:



Cryptographic Token Label: share
Cryptographic Token Password: ********

Some cryptographic tokens have limited capacity, and are unable to hold the signer certificates required to receive or import a personal certificate. If the selected cryptographic token has such a restriction, you may choose to open a secondary key database file to provide the extra capacity to hold signer certificates.

☐ Open existing secondary key database file
☐ Create new secondary key database file

Key database type: CMS
File Name: key.kdb
Location: /

crypto28.jpg

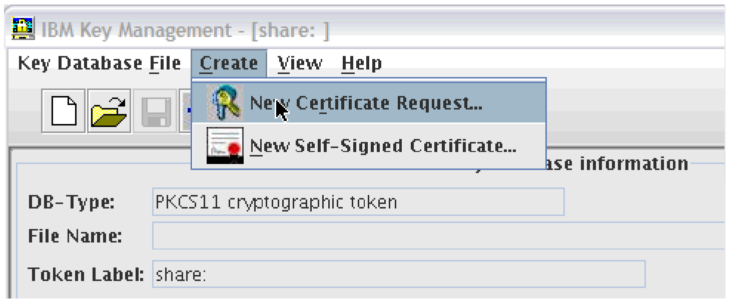
Why clear the box 'Open existing secondary key database file"?

- When using clear-key cryptography, even though we store keys & certificates in a 'Hardware Token', the objects are not actually being stored in the hardware device, but in the Linux file system. As long as there is space in the Linux file system, and as long as the openCryptoki limitation of 2048 public and 2048 private token objects is not exceeded, then a secondary key database is not needed.
- In order to receive a certificate that has been created using the Hardware token, signed by a Certificate Authority, and then returned, the Certificate Authority root certificate, and any intermediate certificates, must be present in the Hardware token. It is not sufficient to have the CA root certificate and intermediate certificates in a secondary key database. An attempt to receive a certificate, when the certificate request was generated using the Hardware token, but the CA root certificate or intermediate certificates are not in the Hardware token as trusted signers, might result in the error message:

```
An error occurred while receiving the certificate from the given file.
The certificate request created for the certificate is not in the key database
```

Step 2: Create a new certificate request

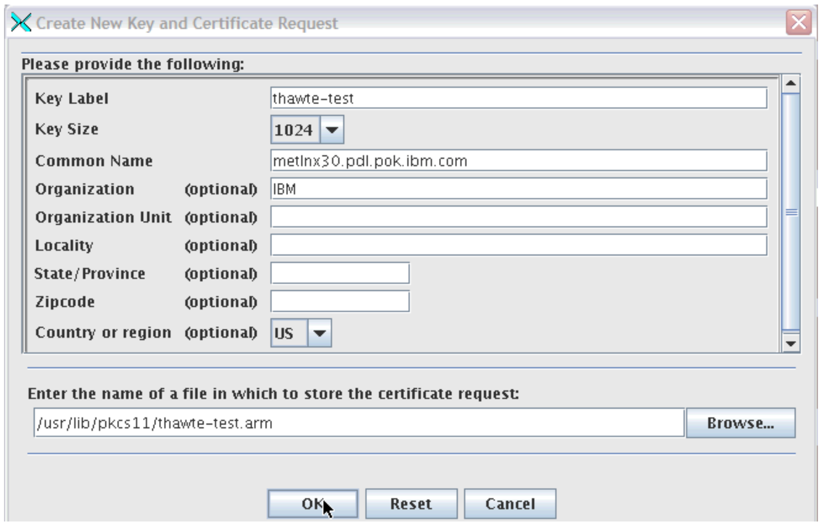
a) With the PKCS#11 token open, go to the top of the IBM Key Management panel, and select Create -> New Certificate Request:


IBM Key Management - [share:]
Key Database File
Create
View
Help
New Certificate Request...
New Self-Signed Certificate...
ase information

DB-Type: PKCS11 cryptographic token
File Name:
Token Label: share:

crypto29.jpg

b) Fill in the request panel. The Key Label is an arbitrary name that is assigned to this certificate. For the Common Name, you use the host name of the system. After filling in the appropriate values, click OK:



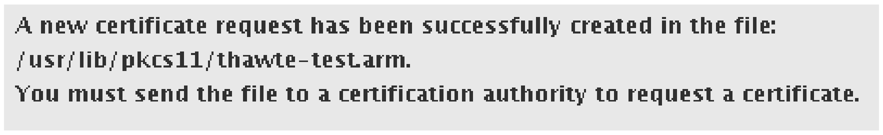
Please provide the following:

Key Label		thawte-test
Key Size		1024
Common Name		metlnx30.pdl.pok.ibm.com
Organization	(optional)	IBM
Organization Unit	(optional)	
Locality	(optional)	
State/Province	(optional)	
Zipcode	(optional)	
Country or region	(optional)	US

Enter the name of a file in which to store the certificate request:
/usr/lib/pkcs11/thawte-test.arm

crypto30.jpg

c) The certificate request is saved in a file -


A new certificate request has been successfully created in the file:
/usr/lib/pkcs11/thawte-test.arm.
You must send the file to a certification authority to request a certificate.

crypto31.jpg

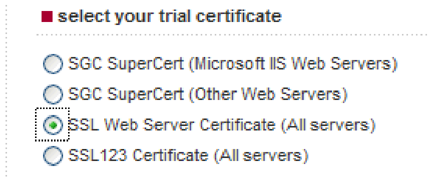
The content of the file is a PEM-format certificate request, similar to this:

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBfTCB5wIBADA+MQswCQYDVQQGEwJVUzEMMAoGA1UEChMDSUJNMSEwHwYDVQQD
ExhtZXRsbngzMC5wZGwucG9rLmlibS5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0A
MIGJAoGBAJ3kf/ZNEAKsNb7B6po5/zIe6J1bhrlY9yIXI5MB4jCjD+mIsqdU5LSS
+ZN/w4/2XlANS8sU75Tokznzs8Vip52+h9EJUYLxx7Hlp9fK+m9hP6ECOR9xtTZG
DcaiY+o4pyDkXrafBROEJ6nuu250HgJkHwuHdNKWicnjSYpzbD/3AgMBAAGgADAN
BgkqhkiG9w0BAQQFAAOBgQBcPWvXBR6EguQoHk4budXGOaUjprlFLrtLOEYGGJLb
0FcH7Hcm7indfRBb3W0u5X/TnvMJvCZdmMk3D+ck1bZT5Q8MTN9h9dj+F+BrG+6J
huSmswGfsGyN+nkIjA4rVhq+E01lT4JhkK119IN7uIRks4/d+V3PKUiD7v+YgIdv
Zg==
-----END NEW CERTIFICATE REQUEST-----
```

Step 3: Have the certificate signed

On the Thawte web site, there is a link to request a free SSL Trial Certificate.

The process requires you to select a type of certificate: you select 'SSL Web Server Certificate (All servers)'


select your trial certificate
SGC SuperCert (Microsoft IIS Web Servers)
SGC SuperCert (Other Web Servers)
[✓ SSL Web Server Certificate (All servers)]
SSL123 Certificate (All servers)

crypto32.jpg

In the space provided, you paste in the contents of your thawte-test.arm file, and then select 'continue'.

The signed trial certificate is displayed. Its contents are similar to:

```
-----BEGIN CERTIFICATE-----
MIIC8jCCAlugAwIBAgIQLZmXs9zIrP65kzZM8d2owTANBgkqhkiG9w0BAQUFADCB
hzELMAkGA1UEBhMCWkExIjAgBgNVBAgTGUZPUiBURVNUSU5HIFBVUlBPU0VTIE9O
TFkxHTAbBgNVBAoTFFRoYXd0ZSBDZXJ0aWZpY2F0aW9uMRcwFQYDVQQLEw5URVNU
IFRFU1QgVEVTVDEcMBoGA1UEAxMTVGhhd3RlIFRlc3QgQ0EgUm9vdDAeFw0wOTAx
MjYxNjIzMDhaFw0wOTAyMTYxNjIzMDhaMD4xCzAJBgNVBAYTAlVTMQwwCgYDVQQK
EwNJQk0xITAfBgNVBAMTGG1ldGxueDMwLnBkbC5wb2suaWJtLmNvbTCBnzANBgkq
hkiG9w0BAQEFAAOBjQAwgYkCgYEAneR/9k0QAqw1vsHqmjn/Mh7onVuGuVj3Ihcj
kwHiMKMP6Yiyp1TktJL5k3/Dj/ZeUA1LyxTvlOiTOfOzxWKnnb6H0QlRgvHHseWn
18r6b2E/oQI5H3G1NkYNxqJj6jinIORetp8FE4Qnqe67bnQeAmQfC4d00paJyeNJ
inNsP/cCAwEAAaOBpjCBozAMBgNVHRMBAf8EAjAAMB0GA1UdJQQWMBQGCCsGAQUF
BwMBBggrBgEFBQcDAjBABgNVHR8EOTA3MDWgM6Axhi9odHRwOi8vY3JsLnRoYXd0
ZS5jb20vVGhhd3RlUHJlbWl1bVNlcnZlckNBLmNybDAyBggrBgEFBQcBAQQmMCQw
IgYIKwYBBQUHMAGGFmh0dHA6Ly9vY3NwLnRoYXd0ZS5jb20wDQYJKoZIhvcNAQEF
BQADgYEAk/h5hiUMfhV8xh7JvTxgiwAkbAKCYtuSUjknY9b2D4dHHcJcqcpvW/kZ
sZpiLD4cntPjalGg6wLBtEErjBniNsImFi5po50F6rscDLksYLTnsh9FYsdzP4Av
```

```
65pzx8utnv+PkvXxUJLhyKXneEZdOPMG6fhsynFs4XtDHAMaC/k=
-----END CERTIFICATE-----
```

Copy and paste this into a file on your IBM HTTP Server system named thawte-test.crt.

You also need to obtain the Thawte Test CA Root Certificate.

It is available from a link on the Web page displaying your trial certificate.

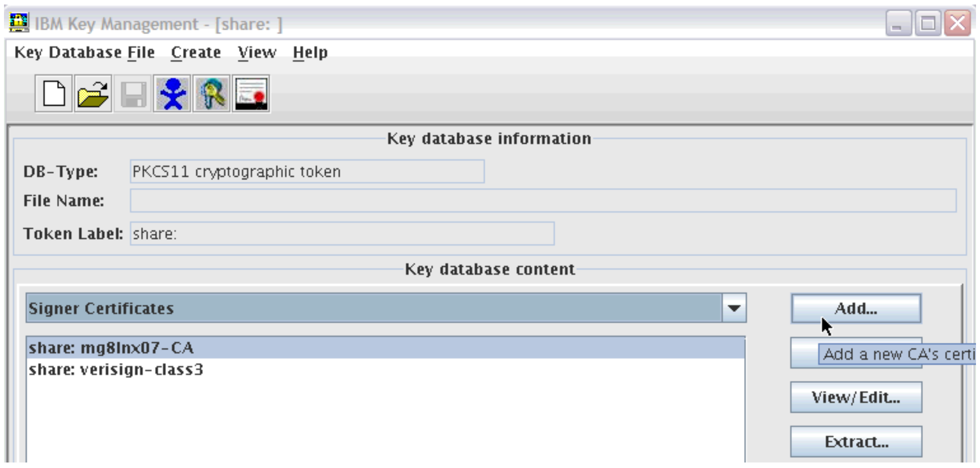
Download the file thawte-roots.zip to a workstation and unzip it.

Copy and paste the contents of the file 'Thawte Test Root_TEXT.txt'

into a file on your IBM HTTP Server system named Thawte_Test_Root.crt

Step 4: Add/Receive the CA and signed certificate to the PKCS#11 token

a) In the Key Management utility, with the PKCS#11 cryptographic token still open, you Select 'Signer Certificates' from the drop-down list under the 'Key database content' header, and click Add:



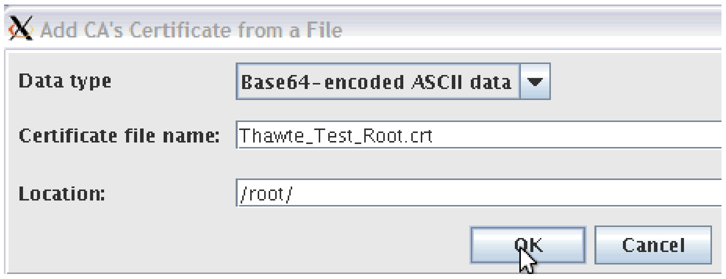
DB-Type: PKCS11 cryptographic token
File Name:
Token Label: share:

Signer Certificates

share: mg8lnx07-CA
share: verisign-class3

crypto33.jpg

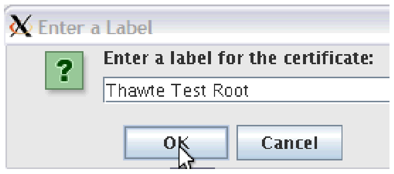
b) On the 'Add' panel, you select a Data type of 'Base64-encoded ASCII data', and fill in the name and location of the file containing the Test Root Certificate:



Data type: Base64-encoded ASCII data
Certificate file name: Thawte_Test_Root.crt
Location: /root/

crypto34.jpg

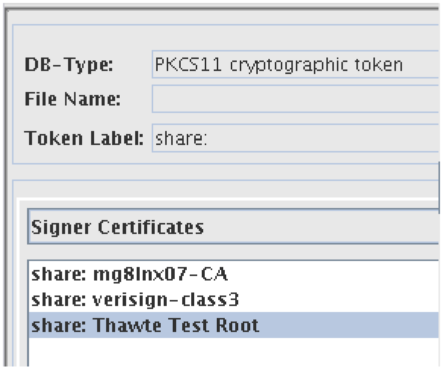
c) Type a label and click OK:



Enter a label for the certificate: Thawte Test Root

crypto35.jpg

d) The Thawte Test CA Root certificate is then shown among the signer certificates in the PKCS#11 token:



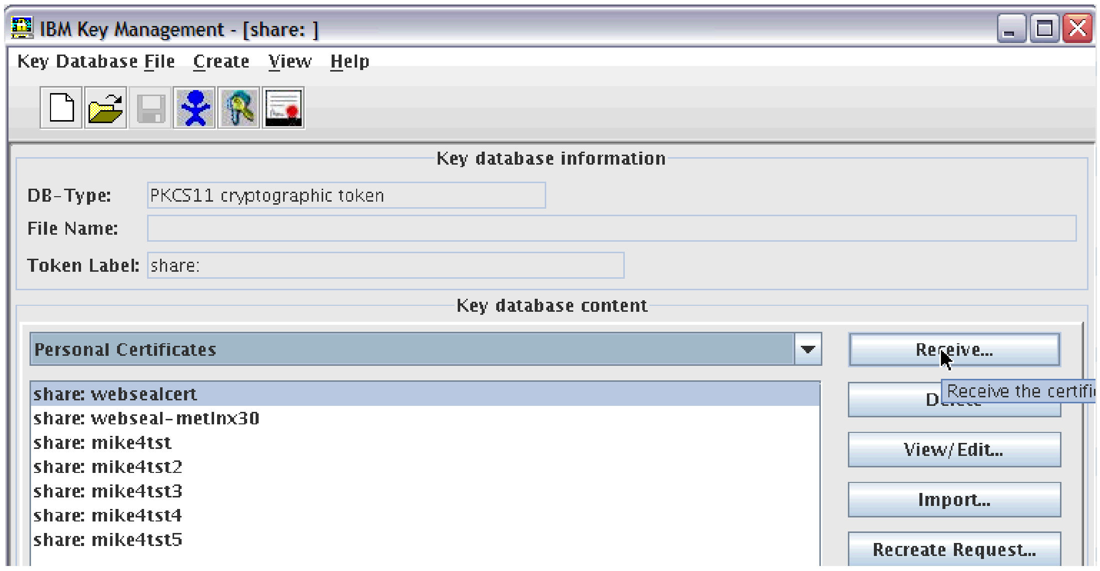
DB-Type: PKCS11 cryptographic token
File Name:
Token Label: share:

Signer Certificates

share: mg8lnx07-CA
share: verisign-class3
share: Thawte Test Root

crypto36.jpg

e) Receive the trial server certificate obtained from Thawte:
Select 'Personal Certificates' from the drop-down bar under 'Key database content', and click Receive



IBM Key Management - [share:]

Key database information

DB-Type: PKCS11 cryptographic token
File Name:
Token Label: share:

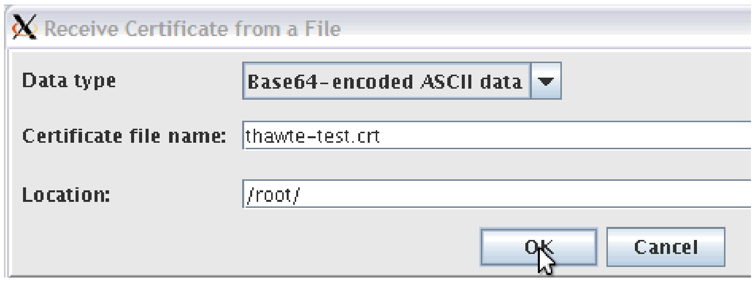
Key database content

Personal Certificates

share: websealcert
share: webseal-metlnx30
share: mike4tst
share: mike4tst2
share: mike4tst3
share: mike4tst4
share: mike4tst5

crypto37.jpg

f) On the 'Receive Certificate from a File' page, type the name of the file containing the trial server certificate, and click OK:

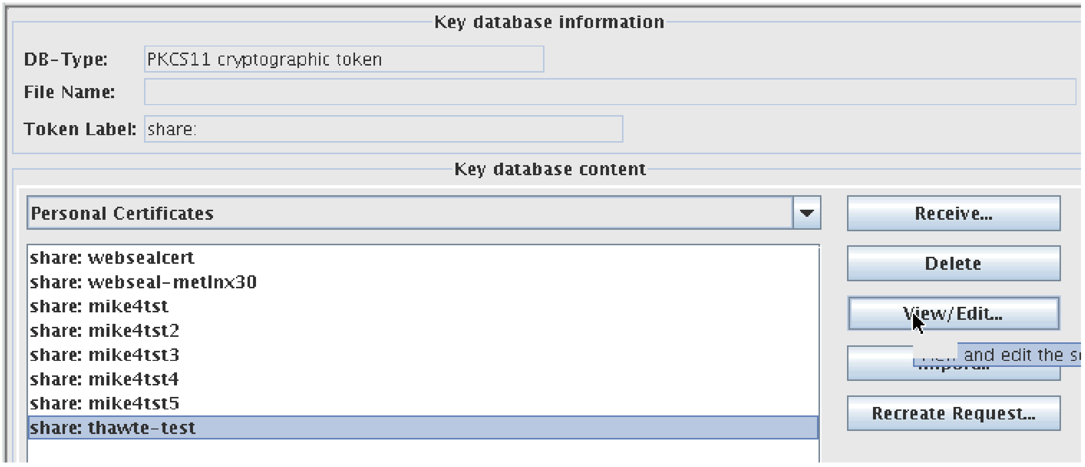


Receive Certificate from a File

Data type: Base64-encoded ASCII data
Certificate file name: thawte-test.crt
Location: /root/

crypto38.jpg

g) The signed trial certificate then shows in the list of personal certificates in the PKCS#11 token:



Key database information

DB-Type: PKCS11 cryptographic token
File Name:
Token Label: share:

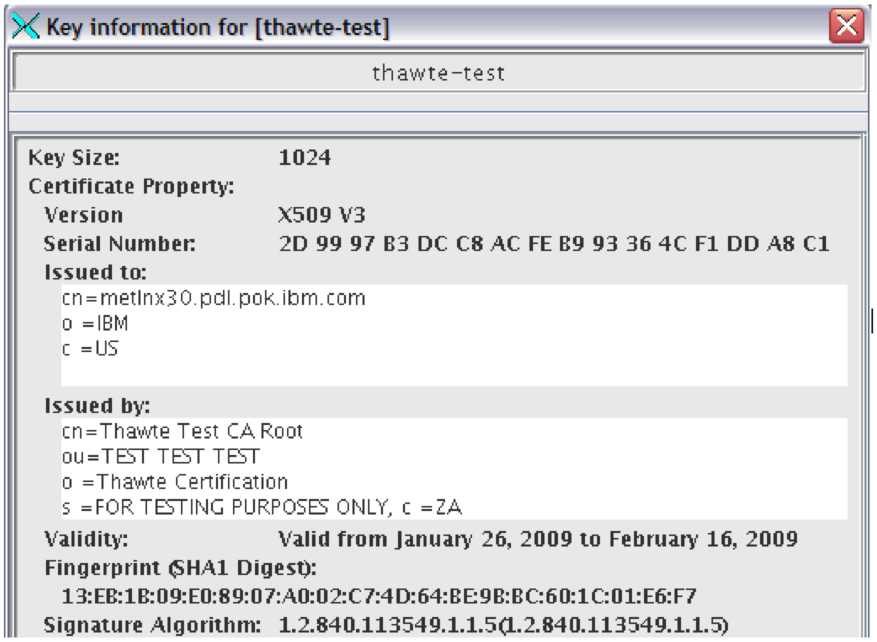
Key database content

Personal Certificates

share: websealcert
share: webseal-metlnx30
share: mike4tst
share: mike4tst2
share: mike4tst3
share: mike4tst4
share: mike4tst5
share: thawte-test

crypto39.jpg

h) By clicking View/Edit, you can confirm that the trial certificate was indeed issued (signed) by the Thawte Test CA Root -



Key information for [thawte-test]

thawte-test

Key Size: 1024
Certificate Property:
Version X509 V3
Serial Number: 2D 99 97 B3 DC C8 AC FE B9 93 36 4C F1 DD A8 C1
Issued to:
cn=metlnx30.pdl.pok.ibm.com
o =IBM
c =US

Issued by:
cn=Thawte Test CA Root
ou=TEST TEST TEST
o =Thawte Certification
s =FOR TESTING PURPOSES ONLY, c =ZA
Validity: Valid from January 26, 2009 to February 16, 2009
Fingerprint (SHA1 Digest):
13:EB:1B:09:E0:89:07:A0:02:C7:4D:64:BE:9B:BC:60:1C:01:E6:F7
Signature Algorithm: 1.2.840.113549.1.1.5(1.2.840.113549.1.1.5)

crypto40.jpg

Note, when a non-trial, production certificate is ordered from a Certificate Authority, there is likely to be more then one signer certificate in the Trust Chain. The production server certificate is likely to have been signed by an Intermediate Certificate, which is, in turn, signed by the Vendor's Root Certificate. Every Certificate in the

Trust Chain is required to be added to the PKCS#11 token as a trusted Signer, in order to successfully receive the production server certificate.

Step 5: Configure the IBM HTTP Server to use the trial certificate in the PKCS#11 token

a) Use the sslstash utility to stash the User PIN in a file:

```
/opt/IBM/HTTPServer/bin # ./sslstash -c /opt/IBM/HTTPServer/ssl/pkcs11stash
crypto 01234567
/opt/IBM/HTTPServer/bin #
```

b) Update the virtual host stanza of the SSL host that will use the Hardware token and trial certificate:

```
<VirtualHost metlnx30.pdl.pok.ibm.com:443>
ServerName metlnx30.pdl.pok.ibm.com
SSLEnable
DocumentRoot /opt/IBM/HTTPServer/htdocs/en_US
KeyFile /opt/IBM/HTTPServer/Plugins/etc/plugin-key.kdb
SSLServerCert share:thawte-test
SSLStashfile /opt/IBM/HTTPServer/ssl/pkcs11stash
SSLPKCSDriver /usr/lib/pkcs11/PKCS11_API.so
</VirtualHost>
SSLDisable
SSLV2Timeout 100
SSLV3Timeout 1000
```

c) Add the userid that is used for running the HTTP Server to the group pkcs11 (in this case 'nobody'):

```
# usermod -G nobody,pkcs11,ewlm_armusers,nogroup nobody
  usermod: `nobody' is primary group name.
#
```

d) Start the HTTP Server

At this point, the set up is done and you can verify that it works.

e) Issue the command:

```
cat /proc/driver/z90crypt
```

The open handles counter should have incremented by 1:

```
Total open handles: 1
```

f) Make an https request to the IBM HTTP Server, and then issue the command:

```
cat /proc/driver/z90crypt
```

The request counter field corresponding to the online **cryptographic** device (the fourth word) should increment by one:

```
Per-device successfully completed request counts
        00000000 00000000 00000000 00000001 00000000 00000000 00000000
00000000
```

The counter will increment for each SSL handshake. Hitting Reload on a browser will not result in an SSL handshake: the browser must be closed and restarted, and the Web server accessed again to cause a new SSL session to be established, requiring an SSL handshake.

Use Case 5: Importing a PKCS12 file containing a personal certificate and private key, plus intermediate and root CA certificates, into a PKCS#11 token

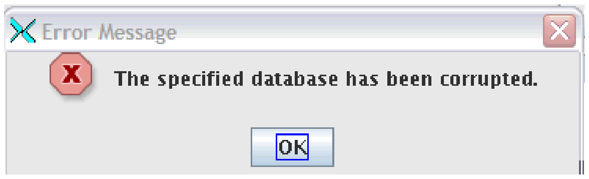
Assumptions:

- The Linux System is Novell SLES 10 SP2
- The openCryptoki rpms installed are:
  ```
  openCryptoki-64bit-2.2.4-0.7
  openCryptoki-2.2.4-0.7
  openCryptoki-32bit-2.2.4-0.7
  ```
- The libica rpms installed are:
  ```
  libica-1.3.8-0.6
  libica-32bit-1.3.8-0.6
  ```
- There is a Crypto-Express2 feature configured with both devices in accelerator mode online to z/VM, which means the Linux guest will see one CEX2A device.
- The z90crypt **cryptographic** device driver has been loaded, and the pkcsslotd daemon has been started.
- The PKCS#11 (openCryptoki) setup has already been completed – the PKCS#11 token has been initialized, and the User PIN set to 01234567.

a) The PKCS12 file containing the personal certificate and private key, plus the intermediate and Root CA certificates is on your Linux system at this location:

```
/etc/ssl/ticlsrv.p12
```

b) Import the certificates and keys from this file into the PKCS#11 token. First, you must add the unlimited jurisdiction policy jar files to the jre/lib/security directory of the Java Runtime you are using to run the Key Management utility named ikeyman/gsk7ikm. The unlimited jurisdiction policy files are required in order to import a PKCS12 file. Without these files, an attempt to import a PKCS12 file will fail with the message:



The specified database has been corrupted.

crypto41.jpg

c) A link to the IBM SDK Policy files: http://www-128.ibm.com/developerworks/java/jdk/security/50/

d) Download and unzip the package. Move the original policy files from JAVA_HOME/jre/lib/security:

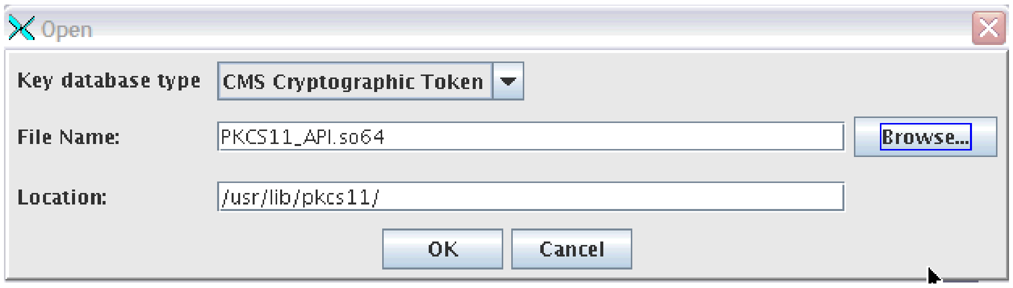
```
/opt/ibm/java2-s390x-50/jre/lib/security # mv US_export_policy.jar /tmp
/opt/ibm/java2-s390x-50/jre/lib/security # mv local_policy.jar  /tmp
```

e) Copy the downloaded versions of the jar files US_export_policy.jar and local_policy.jar to the JAVA_HOME/jre/lib/security directory. Then restart the key management utility.

f) This example shows a 64-bit Java Runtime installed, as well as the 64-bit GSKit rpm: gsk7bas64-7.0-4.14. Set the JAVA_HOME and PATH variables specifying the JRE, and start the key management utility by issuing the gsk7ikm_64 command:

```
# export JAVA_HOME=/opt/ibm/java2-s390x-50/jre/
# export PATH=/opt/ibm/java2-s390x-50/jre/bin:$PATH
# gsk7ikm_64
```

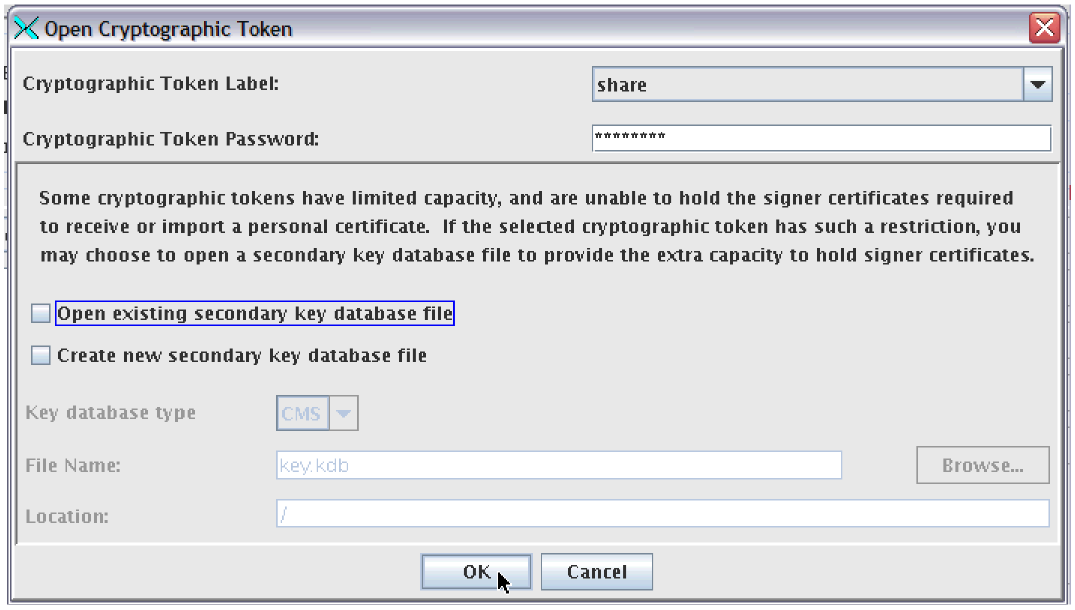
g) When the utility has initialized,
Select: Key database -> Open
Key database type -> CMS Cryptographic Token.
Fill in the location: `/usr/lib/pkcs11`
Fill in the File Name: `PKCS11_API.so64` (the 64-bit PKCS#11 driver is needed because you are using a 64-bit JRE) and click OK:



Key database type: CMS Cryptographic Token
File Name: PKCS11_API.so64
Location: /usr/lib/pkcs11/

crypto42.jpg

h) The Token label is filled in. Type the Cryptographic Token Password (User PIN).
Clear the box 'Open existing secondary key database file', and click OK:



Cryptographic Token Label: share
Cryptographic Token Password: *********

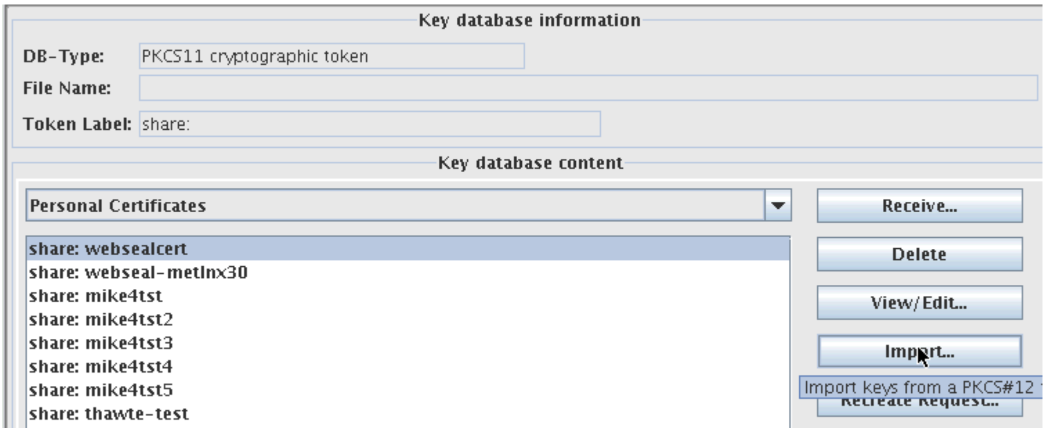
Some cryptographic tokens have limited capacity, and are unable to hold the signer certificates required to receive or import a personal certificate. If the selected cryptographic token has such a restriction, you may choose to open a secondary key database file to provide the extra capacity to hold signer certificates.

☐ Open existing secondary key database file
☐ Create new secondary key database file

Key database type: CMS
File Name: key.kdb
Location: /

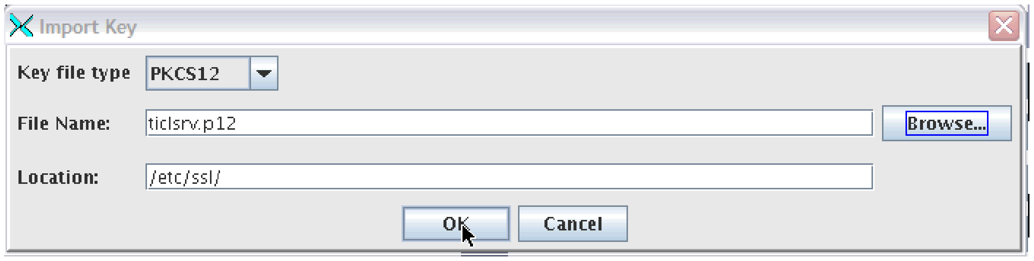
crypto43.jpg

i) With the Personal Certificates bar showing under the 'Key database content' heading, click Import:


Key database information
DB-Type: PKCS11 cryptographic token
File Name:
Token Label: share:
Key database content
Personal Certificates
share: websealcert
share: webseal-metlnx30
share: mike4tst
share: mike4tst2
share: mike4tst3
share: mike4tst4
share: mike4tst5
share: thawte-test

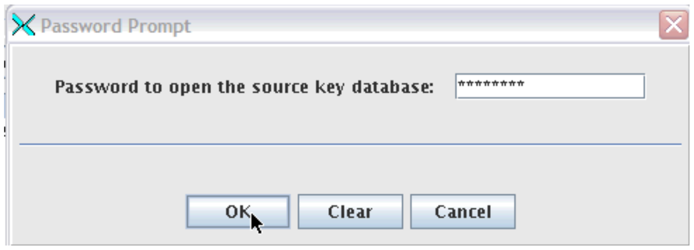
crypto44.jpg

j) Select Key File type of PKCS12
Fill in the Location (/etc/ssl/ in this example)
Fill in the File Name (ticlsrv.p12 in this example)
Click OK:


Import Key
Key file type: PKCS12
File Name: ticlsrv.p12
Location: /etc/ssl/

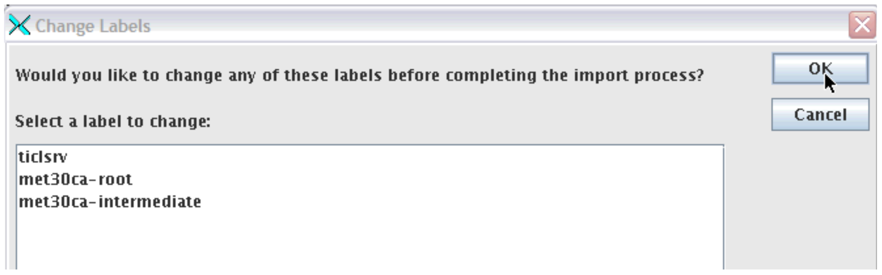
crypto45.jpg

k) Type the password used to secure the .p12 file and click OK:


Password Prompt
Password to open the source key database: *********

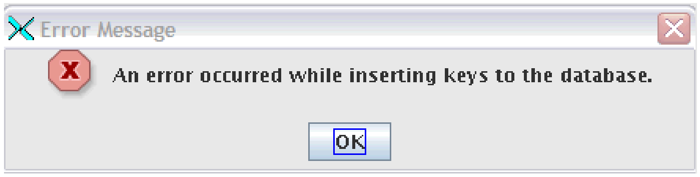
crypto46.jpg

l) The Server certificate, Intermediate Certificates, and Root Certificate are shown, with the option to change the labels, or click OK to import:


Change Labels
Would you like to change any of these labels before completing the import process?
Select a label to change:
ticlsrv
met30ca-root
met30ca-intermediate

crypto47.jpg

m) If you get the 'error occurred while inserting keys to the database' message:



An error occurred while inserting keys to the database.

crypto48.jpg

Then the Root and Intermediate certificates must be added separately to the Cryptographic token, before importing the .p12 file. The following procedure can be used:

1) Convert the PKCS12 file to PEM format using the 'openssl pkcs12' command.

For example:

```
/etc/ssl # openssl pkcs12 -in ticlsrv.p12 -out ticlsrv.pem
Enter Import Password:
MAC verified OK
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
metlnx30:/etc/ssl #
```

2) View the PEM file - note where each separate certificate and private key begins and ends.

In this example, the ticlsrv.pem file contains a private key and server certificate, followed by the root certificate, followed by an intermediate certificate:

```
/etc/ssl # cat ticlsrv.pem
Bag Attributes
    localKeyID: 31 32 33 33 31 37 34 30 30 38 39 33 30
    friendlyName: ticlsrv
Key Attributes: <No Attributes>
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,CC374216A5884F62

gHy/sXXpe+Wm3eS9Lwz9rpb/tHodLZgKjiw6JnGRegzHMEm9q3XY5UrcpvGD0nvY
KnvvF/dwc7UAi57n7Bokv2QPPHXEDHzOAtBFzJk/yEREo4Ptho3q+QyHz9sQ3WWi
nGS5QFpZv0HKeR7i7hUcnqZCuHi+Hx9s3Y0Mt2Z3g/eDRUxaJYGMOpwu8Do6RQJ+
+1WDJvVOyt8UpX5UAiS2z5gVJ7BMdsmosnsaUXgh439pW1dffQ2bxftVX9NLELVh
YK9mclrpjLLXSocLPCeFgQbyDmjApjrD1VBWtHSl32Ap1E+v6K9dA434wch/qZhU
LJ8R/kHgOSF/Od+6Wr/LycJvoZyt+qex5zCPUW/N8qA1daGDPJU4ucawgFbB9ucq
lBZxWC0mesrfi1FlqeqYK3wQcLGdsGh4YsPfkxgm3hmpqeWnCH/GqJJRrq88v6LF
uqPd25P6Ul4+TmFMbPygcCKq+UYjNCO6pSrGkJFcxaT6N/GNK/AoDhLxYRDNM4Sk
I06X3S6q1h37eMc7U49SJPq8VmJsMIvtmD7lSbDuzZm7dPt41ZvhBMW4gtf3tYBn
LjeAmqfcFrgfne4RmRb5XmbADx5GnNpeNomRyiFSyUJtfq5MD5oeW3l8u3a45CAF
36Fz1BIsOyCheTHeV3dFZ1ymXqh+gjsZXXX6fSgCuoZVlLBUkprZmRGvRuyzDHn2
A89sX3J2KP/h92f5Ooe73Doka97N6ZAuSySNZS4uLm0Swb3uBIz2hRRR6oeqmsGu
iFZEho6PrUaOnID5KlDIhtB7fKX+Jhg5fXpyHS0TlPBxvCA5TC/kPA==
-----END RSA PRIVATE KEY-----
Bag Attributes
    localKeyID: 31 32 33 33 31 37 34 30 30 38 39 33 30
    friendlyName: ticlsrv
```

```
subject=/C=US/O=IBM/CN=ticlsrv.pdl.pok.ibm.com
issuer=/C=US/O=IBM/CN=met30CA-Intermediate
-----BEGIN CERTIFICATE-----
MIICaDCCAdGgAwIBAgIBBjANBgkqhkiG9w0BAQUFADA6MQswCQYDVQQGEwJVUzEM
MAoGA1UEChMDSUJNMR0wGwYDVQQDExRtZXQzMENBLUludGVybWVkaWF0ZTAeFw0w
OTAxMjgyMDA3MDBaFw0xMDAxMjgyMDA3MDBaMD0xCzAJBgNVBAYTAlVTMQwwCgYD
VQQKEwNJQk0xIDAeBgNVBAMTF3RpY2xzcnYucGRsLnBvay5pYm0uY29tMIGfMA0G
CSqGSIb3DQEBAQUAA4GNADCBiQKBgQCSLPeonKzSpprZmFfkOhBXJGSgc1jXIe9k
goIPnk+g+hNDgDM34xd2Lyfez+zKHGUyQcuIOYMtgcuVUg9GEg23aKk5HIBmJfHW
LRYZtfrJ5f2Q+R9hjnZeVEKvQiwpwIzUS7F3VPQ5PQmJhTywt+S6Q/Ginq0oFlMG
It37SmJAEwIDAQABo3sweTAJBgNVHRMEAjAAMCwGCWCGSAGG+EIBDQQfFh1PcGVu
U1NMIEdlbmVyYXRlZCBDZXJ0aWZpY2F0ZTAdBgNVHQ4EFgQU5GWqCQvB233fMyVX
FNZAjExpFgAwHwYDVR0jBBgwFoAUolhALrG+kjLcAXW2WVsU3YebeCEwDQYJKoZI
hvcNAQEFBQADgYEAMGBkhqSboT+vTahA6LLSFCH9cxs4YnOLRTjBDRTDNRkUHz63
ZFKLy8eY9Wuf4BNtSZTQUBcMfDBytIiUlp+IBzWOlabjyoz7MYeXMjAGM6unRCuJ
FAC4sFz96oR43kmQEszgasbhHd+BGy0PowQnRYhtQHDPQ7TH1mzX3HmYuDo=
-----END CERTIFICATE-----
Bag Attributes
    localKeyID: 31 32 33 33 31 37 34 30 30 38 39 33 35
    friendlyName: met30ca-root
subject=/C=US/O=IBM/CN=met30CA-Root
issuer=/C=US/O=IBM/CN=met30CA-Root
-----BEGIN CERTIFICATE-----
MIIB2zCCAUQCCQD/Ia8vyJ7HtTANBgkqhkiG9w0BAQUFADAyMQswCQYDVQQGEwJV
UzEMMAoGA1UEChMDSUJNMRUwEwYDVQQDEwxtZXQzMENBLVJvb3QwHhcNMDkwMTI3
MTg0NDU4WhcNMTExMDIzMTg0NDU4WjAyMQswCQYDVQQGEwJVUzEMMAoGA1UEChMD
SUJNMRUwEwYDVQQDEwxtZXQzMENBLVJvb3QwgZ8wDQYJKoZIhvcNAQEBBQADgY0A
MIGJAoGBAKCcBiIGlcOasX3qE2CjByNuElT3brINjJKHqJ88AkuKcPM/DvkGk1Cc
gDp/phm4Faiq4Rmrx8CHoz3drICy03HPLSyR/kVFE9nKuBd+KAzV6AXmH+7CkIfT
m3W3dJBeEyCE+aFmW2ey7i5m8b3332peudLMQuBPBGf9+8kwZ+t1AgMBAAEwDQYJ
KoZIhvcNAQEFBQADgYEAn7rAs9tuL1pBXSd2wgtWWeSmZfvpIR0gwSkNjSzyH6hx
XHK5DHWJ/1FAEOgxg1KK0ir/l1lnTgzuCS8zXCrz33BJDUiX2xUvBwL8/v88FR8N
lx+CmyMp/BgQrK5xBjRUBqG+MuDvmyjoXHRfSJD1LZkBS7mQ9sDjFX8BzbgPYI8=
-----END CERTIFICATE-----
Bag Attributes
    localKeyID: 31 32 33 33 31 37 34 30 30 38 39 33 36
    friendlyName: met30ca-intermediate
subject=/C=US/O=IBM/CN=met30CA-Intermediate
issuer=/C=US/O=IBM/CN=met30CA-Root
-----BEGIN CERTIFICATE-----
MIICjzCCAfigAwIBAgIBBTANBgkqhkiG9w0BAQUFADAyMQswCQYDVQQGEwJVUzEM
MAoGA1UEChMDSUJNMRUwEwYDVQQDEwxtZXQzMENBLVJvb3QwHhcNMDkwMTI4MTk1
OTQxWhcNMTAwMTI4MTk1OTQxWjA6MQswCQYDVQQGEwJVUzEMMAoGA1UEChMDSUJN
MR0wGwYDVQQDExRtZXQzMENBLUludGVybWVkaWF0ZTCBnzANBgkqhkiG9w0BAQEF
AAOBjQAwgYkCgYEAqhJbkO2vWMAZpHf6sV6uJlOOO32QzQxtZpj0YzT3TMOFeo1K
441i9DV5vRGxQ5oVX5uNhQjxeGjGMpyePhwsRj8gd/gcw0aJdfSbsB8eYP2+tz6q
YzfCaOgtT2x9QoEUqnszLLy+l1dY2m5QRi790ISqOuxtgff9Y5buYK3NwIMCAwEA
AaOBrDCBqTAMBgNVHRMEBTADAQH/MCwGCWCGSAGG+EIBDQQfFh1PcGVuU1NMIEdl
bmVyYXRlZCBDZXJ0aWZpY2F0ZTAdBgNVHQ4EFgQUolhALrG+kjLcAXW2WVsU3Yeb
eCEwTAYDVR0jBEUwQ6E2pDQwMjELMAkGA1UEBhMCVVMxDDAKBgNVBAoTA0lCTTEV
MBMGA1UEAxMMbWV0MzBDQS1Sb290ggkA/yGvL8iex7UwDQYJKoZIhvcNAQEFBQAD
gYEARNOlgsQnG45FlZkrpK0ssV9MhdCVdpkXBXBnfzMvCP8d2M5A1eIF3+kV60JA
Y9oP1J5vuw2qh2AqQlSFyuOrBaxHm1/xdvZzXQXKLB3SvYvB1tHYDdaQu4Tk/01z
eGtoNB+nL+sFhWTH9quZoL1Fnly8ljWkqx1aluF/+GYoDUM=
-----END CERTIFICATE-----
metlnx30:/etc/ssl #
```

3) Break the PEM file up into three separate files:

```
/etc/ssl # cp ticlsrv.pem ticlsrv.justRoot.pem
/etc/ssl # cp ticlsrv.pem ticlsrv.justIntermediate.pem
/etc/ssl # cp ticlsrv.pem ticlsrv.justsrvcertandkey.pem
```

4) Edit one file so it just contains the Root Certificate:

```
Bag Attributes
    localKeyID: 31 32 33 33 31 37 34 30 30 38 39 33 35
    friendlyName: met30ca-root
subject=/C=US/O=IBM/CN=met30CA-Root
issuer=/C=US/O=IBM/CN=met30CA-Root
-----BEGIN CERTIFICATE-----
MIIB2zCCAUQCCQD/Ia8vyJ7HtTANBgkqhkiG9w0BAQUFADAyMQswCQYDVQQGEwJV
…......
lx+CmyMp/BgQrK5xBjRUBqG+MuDvmyjoXHRfSJD1LZkBS7mQ9sDjFX8BzbgPYI8=
-----END CERTIFICATE-----
```

5) Edit another file so it just contains the Intermediate Certificate:

```
Bag Attributes
    localKeyID: 31 32 33 33 31 37 34 30 30 38 39 33 36
    friendlyName: met30ca-intermediate
subject=/C=US/O=IBM/CN=met30CA-Intermediate
issuer=/C=US/O=IBM/CN=met30CA-Root
-----BEGIN CERTIFICATE-----
MIICjzCCAfigAwIBAgIBBTANBgkqhkiG9w0BAQUFADAyMQswCQYDVQQGEwJVUzEM
…......
eGtoNB+nL+sFhWTH9quZoL1Fnly8ljWkqx1aluF/+GYoDUM=
-----END CERTIFICATE-----
```

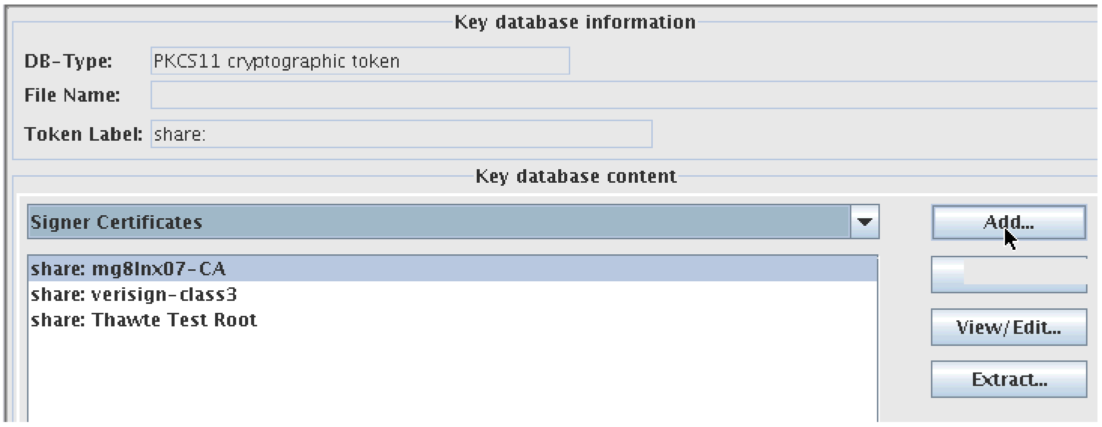
6) And edit the third file so it contains the server certificate and corresponding private key:

```
Bag Attributes
    localKeyID: 31 32 33 33 31 37 34 30 30 38 39 33 30
    friendlyName: ticlsrv
Key Attributes: <No Attributes>
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,CC374216A5884F62

gHy/sXXpe+Wm3eS9Lwz9rpb/tHodLZgKjiw6JnGRegzHMEm9q3XY5UrcpvGD0nvY
…......
iFZEho6PrUaOnID5KlDIhtB7fKX+Jhg5fXpyHS0TlPBxvCA5TC/kPA==
-----END RSA PRIVATE KEY-----
Bag Attributes
    localKeyID: 31 32 33 33 31 37 34 30 30 38 39 33 30
    friendlyName: ticlsrv
subject=/C=US/O=IBM/CN=ticlsrv.pdl.pok.ibm.com
issuer=/C=US/O=IBM/CN=met30CA-Intermediate
-----BEGIN CERTIFICATE-----
MIICaDCCAdGgAwIBAgIBBjANBgkqhkiG9w0BAQUFADA6MQswCQYDVQQGEwJVUzEM
…......
FAC4sFz96oR43kmQEszgasbhHd+BGy0PowQnRYhtQHDPQ7TH1mzX3HmYuDo=
-----END CERTIFICATE-----
```

7) Add the Root certificate to the Cryptographic token as a Trusted Signer:

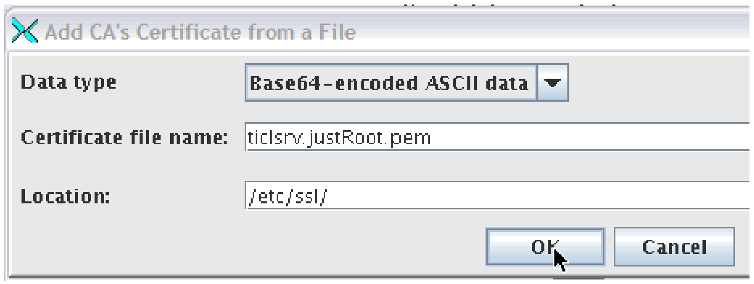
After starting the key management utility and opening the Hardware Cryptographic token, select 'Signer Certificates' from the drop-down bar under 'Key database content', and click Add:



Key database information
DB-Type: PKCS11 cryptographic token
File Name:
Token Label: share:
Key database content
Signer Certificates
share: mg8lnx07-CA
share: verisign-class3
share: Thawte Test Root

crypto49.jpg

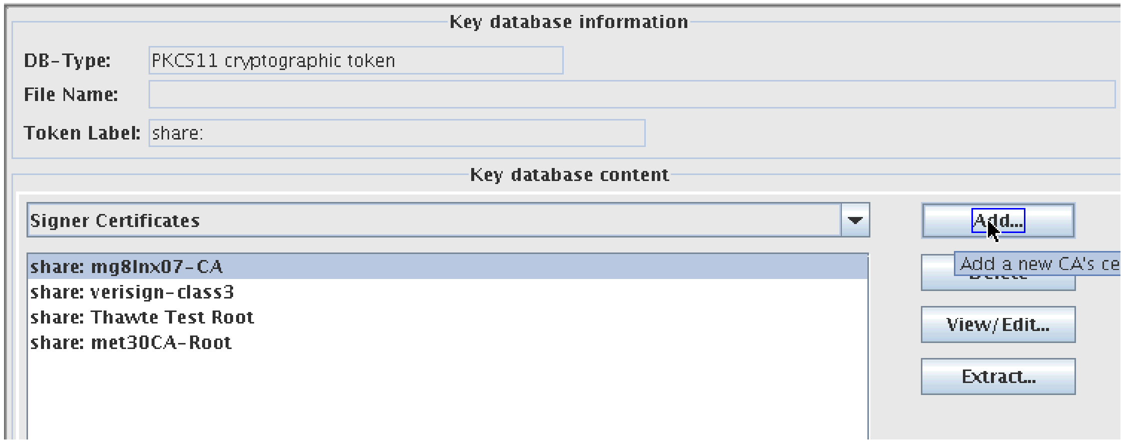
8) Select the file that has just the Root certificate in PEM format:



Add CA's Certificate from a File
Data type: Base64-encoded ASCII data
Certificate file name: ticlsrv.justRoot.pem
Location: /etc/ssl/

crypto50.jpg

9) Assign a label to the Certificate and click OK (this example assigned the label met30CA-Root).
The met30CA-Root certificate is displayed similar to this list of Signer Certificates:



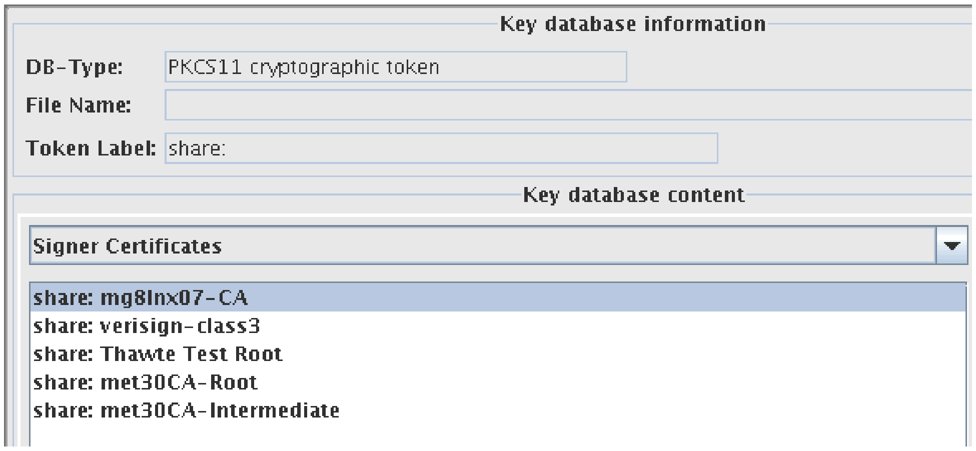
Key database information
DB-Type: PKCS11 cryptographic token
File Name:
Token Label: share:
Key database content
Signer Certificates
share: mg8lnx07-CA
share: verisign-class3
share: Thawte Test Root
share: met30CA-Root

crypto51.jpg

10) Add the Intermediate Certificate to the Cryptographic token as a trusted signer.

Again, with the Cryptographic token open, and the Signer Certificates displayed, click Add.
Select the file that has just the intermediate certificate, and click OK.
Assign a label to the certificate, and click OK (This example assigned the label met30CA-Intermediate):

The Intermediate Certificate is now displayed similar to this list of Signer Certificates:



DB-Type: PKCS11 cryptographic token
File Name:
Token Label: share:

Signer Certificates

share: mg8lnx07-CA
share: verisign-class3
share: Thawte Test Root
share: met30CA-Root
share: met30CA-Intermediate

crypto52.jpg

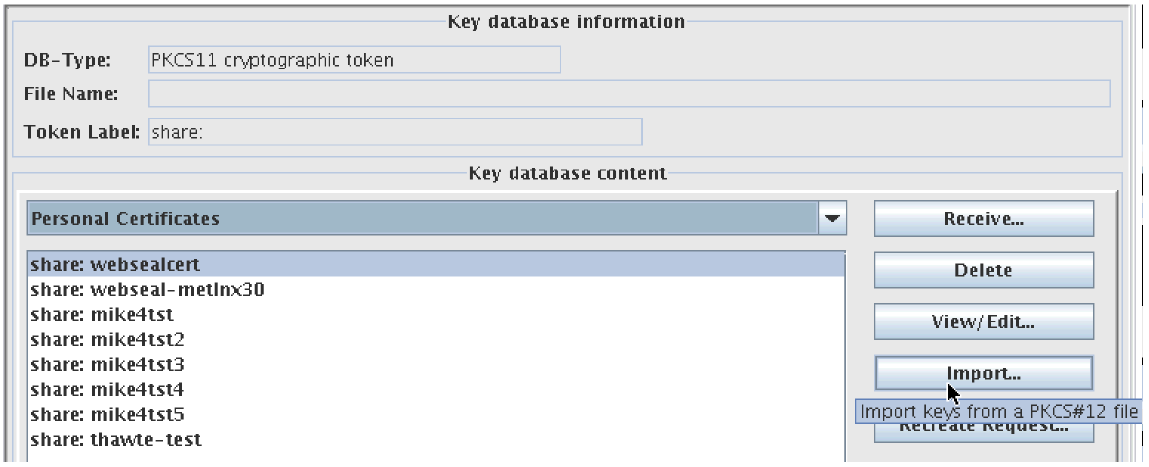
11) Convert the PEM file containing the server certificate and private key back to a PKCS12 file:

The 'openssl pkcs12' command can be used to convert the PEM file containing just the server certificate and private key to a PKCS12 file:

```
/etc/ssl # openssl pkcs12 -export -in ticlsrv.justsrvcertandkey.pem -out
ticlsrv.justsrvcertandkey.p12
Enter pass phrase for met30srv.justsrvcertandkey.pem:
Enter Export Password:
Verifying - Enter Export Password:
/etc/ssl #
```

12) Import the server certificate and private key into the PKCS#11 Cryptographic token.

In the key management utility, with the Cryptographic token open, Select 'Personal Certificates' from the drop-down bar under 'Key database content' and click Import:



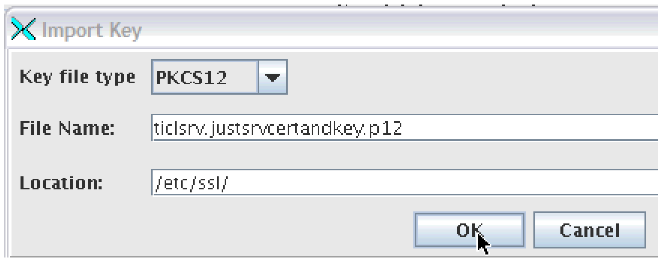
DB-Type: PKCS11 cryptographic token
File Name:
Token Label: share:

Personal Certificates

share: websealcert
share: webseal-metlnx30
share: mike4tst
share: mike4tst2
share: mike4tst3
share: mike4tst4
share: mike4tst5
share: thawte-test

crypto53.jpg

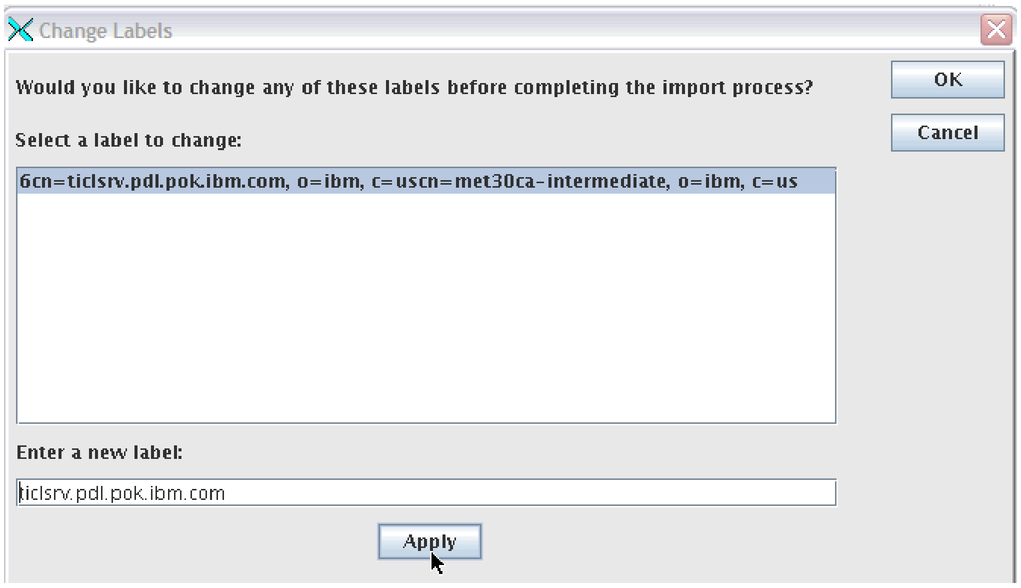
13) Select the key file type of PKCS12, and select the .p12 file containing just the server certificate and private key:



Key file type: PKCS12
File Name: ticlsrv.justsrvcertandkey.p12
Location: /etc/ssl/

crypto54.jpg

14) Type the password used to create the .p12 file.
Change the label of the certificate to be imported, or accept the label displayed - in this example, the label is changed to ticlsrv.pdl.pok.ibm.com:



Would you like to change any of these labels before completing the import process?

Select a label to change:

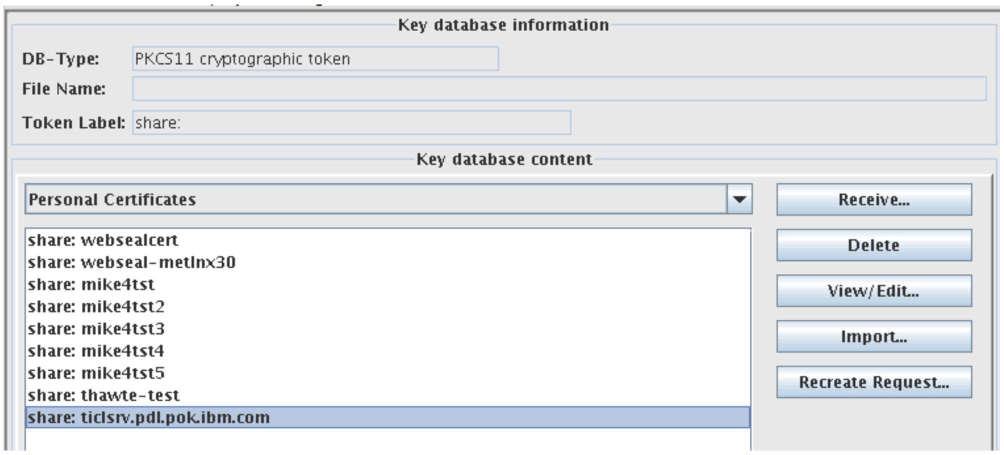
6cn=ticlsrv.pdl.pok.ibm.com, o=ibm, c=uscn=met30ca-intermediate, o=ibm, c=us

Enter a new label:

ticlsrv.pdl.pok.ibm.com

crypto55.jpg

15) Then with the new label, click OK to complete the import.
The server certificate is displayed similar to this among the Personal Certificates:



Key database information
DB-Type: PKCS11 cryptographic token
File Name:
Token Label: share:

Key database content
Personal Certificates

share: websealcert
share: webseal-metlnx30
share: mike4tst
share: mike4tst2
share: mike4tst3
share: mike4tst4
share: mike4tst5
share: thawte-test
share: ticlsrv.pdl.pok.ibm.com

crypto56.jpg

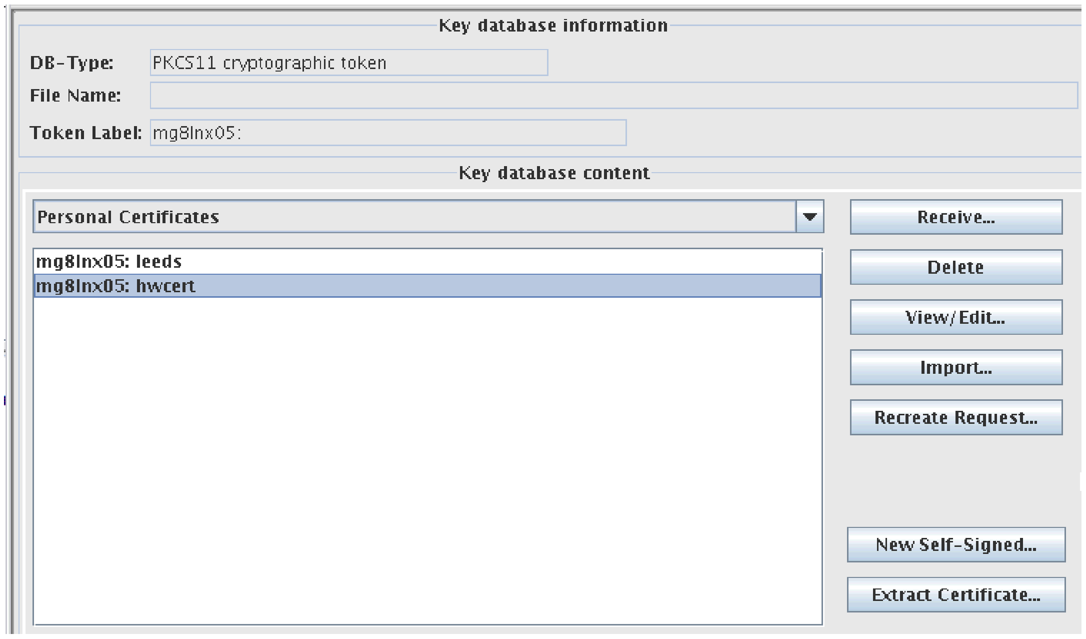
Use Case 6: Copying PKCS#11 token Information from one Linux on IBM System z Image to another

Background

Typically, an installation will use a cluster of Web servers to provide access to e-business applications. Multiple Web servers enable high-availability and load-balancing. The Web servers can be clones of each other, and it might be desired to have all of the clustered Web servers use the same server certificate.

If the Web servers are using a PKCS#11 cryptographic token, and if the request for the Web servers' server certificate was generated in one of the cryptographic tokens in the cluster, then there are some considerations with regards to copying the server certificate and keys among all of the Web servers.

When a certificate has been received into a PKCS#11 cryptographic token, the private key associated with that certificate cannot be exported using the ikeyman/gsk7ikm utility.
To illustrate: shown below is the gsk7ikm panel showing Personal certificates stored using the PKCS#11 cryptographic token. Note that, among the buttons down the right hand side of the panel, there is an 'Import' button, but no button used to 'Export' the personal certificates and private keys:


Key database information
DB-Type: PKCS11 cryptographic token
File Name:
Token Label: mg8lnx05:
Key database content
Personal Certificates
mg8lnx05: leeds
mg8lnx05: hwcert

crypto57.jpg

The reason an Export option is not available, is because the ikeyman/gsk7ikm utility treats the PKCS#11 cryptographic token as a secure device. Therefore, private keys can be added to the device, but they cannot be extracted.

A procedure, then, is needed to copy the certificates/keys (including private keys) from the PKCS#11 cryptographic token on one Linux image, to another image, so that the same certificates/keys can be used by the Web servers on each image.

Because we are dealing with clear-key cryptography, the keys are not actually stored in the cryptographic hardware, but rather, in the Linux file system, in an encrypted state. The procedure then, copies the directories containing the PKCS#11 cryptographic information, from one Linux system to the other.

When using openCryptoki 2.1, which is the level supplied with Novell SLES 9 and Red Hat Enterprise Linux 4 (RHEL 4), the PKCS#11 information to be copied is in the directory /etc/pkcs11/lite. When using openCryptoki 2.2, which is the level supplied with Novell SLES 10 and RHEL 5, the PKCS#11 information to be copied is in the directory /var/lib/opencryptoki/lite.

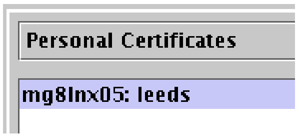
Two procedures are given below showing how to copy the PKCS#11 information between two Novell SLES 9 systems, and between two Novell SLES 10 systems. The Novell SLES 9 procedure could also be used on a RHEL4 system to copy to another RHEL 4 system. The Novell SLES 10 procedure could also be used on RHEL 5 system to copy to another RHEL 5 system. These procedures for copying the token were not tested to mix between SUSE and RHEL.

Use Case 6a: Copying PKCS#11 token Information - Novell SLES 9

The two Linux systems are on different machines:

- The source system, mg8lnx05, is a Novell SLES 9 (31-bit) system on an IBM System z9.
- The target system, metlnx29, is a Novell SLES 9 (31-bit) system on an IBM eServer zSeries 900 (z900).

Using the GSKit key management utility, named gsk7ikm, on the source system, open the CMS Cryptographic token, and you'll see that the source system has a Personal Certificate named: mg8lnx05:leeds


Personal Certificates
mg8lnx05: leeds

crypto58.jpg

Tivoli Access Manager WebSEAL on mg8lnx05 is configured to use this certificate.
Here are the relevant parameters in the webseal conf file -

```
unix-group = pkcs11
webseal-cert-keyfile-label = mg8lnx05:leeds
pkcs11-driver-path = /usr/lib/pkcs11/PKCS11_API.so
pkcs11-token-label = mg8lnx05
pkcs11-token-pwd = 01234567
pkcs11-symmetric-cipher-support = yes
```

To be able to run a similarly configured WebSEAL server, using the same certificate, on the target system, copy the PKCS#11 token information:

Step 1: Stop WebSEAL and the slot daemon on the source system:

```
mg8lnx05:/ # pdweb stop
mg8lnx05:/ # rcpkcsslotd stop
Shutting down pkcsslotd daemon:                                         done
```

Step 2: tar the /etc/pkcs11/lite directory on the source system:

```
mg8lnx05:/etc/pkcs11 # tar -cvf lite.mg8lnx05.tar lite
lite/
lite/TOK_OBJ/
lite/TOK_OBJ/00000000
lite/TOK_OBJ/OBJ.IDX
lite/TOK_OBJ/10000000
lite/TOK_OBJ/20000000
lite/MK_SO
lite/NVTOK.DAT
lite/MK_USER
```

The pkcsslotd daemon and WebSEAL server can be started again on the source system: mg8lnx05.

Step 3: Install the same levels of libica and openCryptoki on the target system as are on the source system.

These are the libica and openCryptoki levels on the source system, mg8lnx05:

```
libica-1.3.6rc2-0.10
openCryptoki-2.1.6_rc6-0.6
openCryptoki-32bit-2.1.6_rc6-0.6
```

On the target system, install the same levels:

```
metlnx29:/rpms # rpm -hiv libica-1.3.6rc2-0.10.s390.rpm
metlnx29:/rpms # rpm -hiv openCryptoki-2.1.6_rc6-0.6.s390.rpm
metlnx29:/rpms # rpm -hiv openCryptoki-32bit-2.1.6_rc6-0.6.s390.rpm
```

Step 4: Initialize the PKCS#11 environment on the target system.

a) Start the slot daemon on the target system (The z90crypt **cryptographic** device driver must already be loaded):

```
metlnx29:/ # rcpkcsslotd start
Starting pkcsslotd daemon:usermod: `root' is primary group name.
                                                                   done
```

b) Initialize the PKCS#11 token and PINs:

```
metlnx29:/usr/lib/pkcs11/methods # ./pkcsconf -c 0 -I
Enter the SO PIN: ********          --> 87654321
Enter a unique token label: metlnx29
metlnx29:/usr/lib/pkcs11/methods # ./pkcsconf -c 0 -P
Enter the SO PIN: ********          --> 87654321
Enter the new SO PIN: ********      --> 01234567
Re-enter the new SO PIN: ********
metlnx29:/usr/lib/pkcs11/methods # ./pkcsconf -c 0 -u
Enter the SO PIN: ********          --> 01234567
Enter the new user PIN: ********    --> 87654321
Re-enter the new user PIN: ******** --> 87654321
metlnx29:/usr/lib/pkcs11/methods # ./pkcsconf -c 0 -p
Enter user PIN: ********            --> 87654321
Enter the new user PIN: ********    --> 01234567
Re-enter the new user PIN: ******** --> 01234567
```

c) Note that the Security Officer PIN and User PIN will be overwritten with the values from the source system upon completion of the procedure.

Step 5: Stop the pkcsslotd daemon on the target system:

```
metlnx29:/ # rcpkcsslotd stop
Shutting down pkcsslotd daemon:                                 done
```

Step 6: Remove the /etc/pkcs11/lite directory on the target system:

```
metlnx29:/etc/pkcs11 # rm -R lite
```

Step 7: Transfer the tar file containing the source system /etc/pkcs11/lite directory to the target system:

```
metlnx29:/etc/pkcs11 # scp root@mg8lnx05:/etc/pkcs11/lite.mg8lnx05.tar .
Password:
lite.mg8lnx05.tar                            100%   10KB  10.0KB/s   00:00
```

Step 8: Un-tar the file on the target system:

```
metlnx29:/etc/pkcs11 # tar -xvf lite.mg8lnx05.tar
lite/
lite/TOK_OBJ/
lite/TOK_OBJ/00000000
lite/TOK_OBJ/OBJ.IDX
lite/TOK_OBJ/10000000
lite/TOK_OBJ/20000000
lite/MK_SO
lite/NVTOK.DAT
lite/MK_USER
```

Step 9: Chown the tar file on the target system:

```
metlnx29:/etc/pkcs11 # chown -R root:pkcs11 lite
```

Step 10: Reboot the target system.

Because a shared memory segment is used to access PKCS#11 token information, it is necessary to ensure that the shared memory segment is removed and recreated after copying the token information. This is accomplished by rebooting the target system.

Step 11: Load the z90crypt driver and start the pkcsslotd daemon on the target system:

```
metlnx29:/ # rcz90crypt start
Loading z90crypt module                                              done
metlnx29:/ # rcpkcsslotd start
Starting pkcsslotd daemon:usermod: `root' is primary group name.     done
```

The 'pkcsconf -t' command shows the token label from the source system –

```
metlnx29:/usr/lib/pkcs11/methods # ./pkcsconf -t
Token #0 Info:
       Label: mg8lnx05
```

Step 12: Update the PKCS#11 Application on the target system to use the certificate now on the target system.

For WebSEAL, update the webseal conf file to use the new label/certificate that has been transferred from the source system: mg8lnx05:

```
webseal-cert-keyfile-label = mg8lnx05:leeds
pkcs11-driver-path = /usr/lib/pkcs11/PKCS11_API.so
pkcs11-token-label = mg8lnx05
pkcs11-token-pwd = 01234567
pkcs11-symmetric-cipher-support = yes
```

Step 13: Start the PKCS#11 Application on the target system.

After restarting WebSEAL on the target system, metlnx29, you access WebSEAL with an https request, verify that the request is successful, and that the server certificate sent is the certificate that has been copied from the source system.

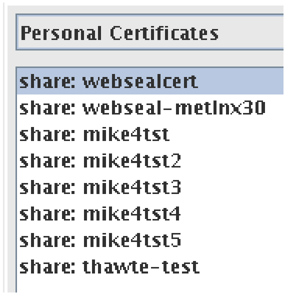
You can see from the z90crypt display - `metlnx29:/ # cat /proc/driver/z90crypt` the 'successfully completed request counts' for the **cryptographic** device is s incremented, as SSL handshakes are processed.

Use Case 6b: Copying PKCS#11 token Information - Novell SLES 10

The two Linux systems are on different machines:

- The source system, with host name 'metlnx30' is a Novell SLES 10 system on an IBM System z10, using a CEX2C device.
- The target system, with hostname 'GSKIT2', is a Novell SLES 10 system on an IBM System z9 Business Class machine, using a CEX2A device.

Using the GSKit key management utility, named gsk7ikm, on the source system, you open the CMS Cryptographic token, and you see the source system has several personal certificates defined in the PKCS#11 token:



```
Personal Certificates

share: websealcert
share: webseal-metlnx30
share: mike4tst
share: mike4tst2
share: mike4tst3
share: mike4tst4
share: mike4tst5
share: thawte-test
```

crypto59.jpg

The last one in the list, share:thawte-test, is used by the IBM HTTP Server on metlnx30. Here are the relevant parameters in the httpd.conf file for the HTTP Server defining the use of the PKCS#11 token:

```
<VirtualHost metlnx30.pdl.pok.ibm.com:443>

ServerName metlnx30.pdl.pok.ibm.com
SSLEnable
DocumentRoot /opt/IBM/HTTPServer/htdocs/en_US
KeyFile /opt/IBM/HTTPServer/Plugins/etc/plugin-key.kdb
SSLServerCert  share:thawte-test
SSLStashfile /opt/IBM/HTTPServer/ssl/pkcs11stash
SSLPKCSDriver /usr/lib/pkcs11/PKCS11_API.so
</VirtualHost>

SSLDisable
SSLV2Timeout 100
SSLV3Timeout 1000
```

To be able to run a similarly configured HTTP server, using the same certificate, on the target system, you copy the PKCS#11 token information:

Step 1: Stop the HTTP Server and slot daemon on the source system:

```
metlnx30:/opt/IBM/HTTPServer/bin # ./apachectl stop
metlnx30:/opt/IBM/HTTPServer/bin # rcpkcsslotd stop
Shutting down pkcsslotd daemon:                                  done
metlnx30:/opt/IBM/HTTPServer/bin #
```

Step 2: Tar the /var/lib/opencryptoki/lite directory on the source system:

```
metlnx30:/var/lib/opencryptoki # tar -cvf lite.metlnx30.tar lite
lite/
```

```
lite/TOK_OBJ/
lite/TOK_OBJ/00000000
lite/TOK_OBJ/OBJ.IDX
lite/TOK_OBJ/10000000
lite/TOK_OBJ/20000000
lite/TOK_OBJ/60000000
lite/TOK_OBJ/70000000
lite/TOK_OBJ/F0000000
lite/TOK_OBJ/IDX.TMP
lite/TOK_OBJ/90000000
lite/TOK_OBJ/A0000000
lite/TOK_OBJ/B0000000
lite/TOK_OBJ/C0000000
lite/TOK_OBJ/D0000000
lite/TOK_OBJ/E0000000
lite/TOK_OBJ/G0000000
lite/TOK_OBJ/H0000000
lite/TOK_OBJ/I0000000
lite/TOK_OBJ/J0000000
lite/TOK_OBJ/K0000000
lite/TOK_OBJ/L0000000
lite/TOK_OBJ/M0000000
lite/TOK_OBJ/N0000000
lite/TOK_OBJ/O0000000
lite/TOK_OBJ/P0000000
lite/TOK_OBJ/A1000000
lite/TOK_OBJ/B1000000
lite/TOK_OBJ/C1000000
lite/TOK_OBJ/D1000000
lite/TOK_OBJ/71000000
lite/MK_SO
lite/NVTOK.DAT
lite/MK_USER
metlnx30:/var/lib/opencryptoki #
```

The pkcsslotd daemon and IBM HTTP Server can be started again on the source system - metlnx30.

Step 3: Install the same levels of libica and openCryptoki on the target system as are on the source system.

These are the libica and openCryptoki levels on the source system – metlnx30:

```
libica-1.3.8-0.6
libica-32bit-1.3.8-0.6
openCryptoki-64bit-2.2.4-0.7
openCryptoki-2.2.4-0.7
openCryptoki-32bit-2.2.4-0.7
```

On the target system, install the same levels of libica and openCryptoki.

Step 4: Initialize the PKCS#11 environment on the target system.

a) Load the z90crypt driver, and start the slot daemon on the target system:

```
GSKIT2:/ # rcz90crypt start
Loading z90crypt module                                              done
GSKIT2:/ # rcpkcsslotd start
Starting pkcsslotd daemon:usermod: `root' is primary group name.
                                                                     done
GSKIT2:/ #
```

b) Initialize the PKCS#11 token and PINs:

```
GSKIT2:/ # pkcsconf -c 0 -I
Enter the SO PIN: ********           --> 87654321
Enter a unique token label: gskit2
GSKIT2:/ # pkcsconf -c 0 -P
Enter the SO PIN: ********           --> 87654321
Enter the new SO PIN: ********       --> 01234567
Re-enter the new SO PIN: ********    --> 01234567
GSKIT2:/ # pkcsconf -c 0 -u
Enter the SO PIN: ********           --> 01234567
Enter the new user PIN: ********     --> 87654321
Re-enter the new user PIN: ******** --> 87654321
GSKIT2:/ # pkcsconf -c 0 -p
Enter user PIN: ********             --> 87654321
Enter the new user PIN: ********     --> 01234567
Re-enter the new user PIN: ******** --> 01234567
```

c) Note that the Security Officer PIN and User PIN will be overwritten with the values from the source system upon completion of the procedure.

Step 5: Stop the pkcsslotd daemon on the target system:

```
GSKIT2:/ # rcpkcsslotd stop
Shutting down pkcsslotd daemon:                              done
GSKIT2:/ #
```

Step 6: Remove the /var/lib/opencryptoki/lite directory on the target system:

```
GSKIT2:/var/lib/opencryptoki # rm -R lite
```

Step 7: Copy the tar file from the source system:

```
GSKIT2:/var/lib/opencryptoki # scp
root@metlnx30:/var/lib/opencryptoki/lite.metlnx30.tar .
Password:
lite.metlnx30.tar                      100%   50KB  50.0KB/s   00:00
GSKIT2:/var/lib/opencryptoki #
```

Step 8: Untar the file on the target system:

```
GSKIT2:/var/lib/opencryptoki # tar -xvf lite.metlnx30.tar
```

Recursively chown the lite directory:

```
GSKIT2:/var/lib/opencryptoki # chown -R root:pkcs11 lite
```

Step 9: Reboot the target system.

Because a shared memory segment is used to access PKCS#11 token information, it is necessary to ensure that the shared memory segment is removed and recreated after copying the token information. This is accomplished by rebooting the target system.

Step 10: Load the z90crypt driver and start the PKCS#11 slot daemon on the target system:

```
GSKIT2:/ # rcz90crypt start
Loading z90crypt module                                              done
GSKIT2:/ # rcpkcsslotd start
Starting pkcsslotd daemon:usermod: `root' is primary group name.
                                                                     done
GSKIT2:/ #
```

The 'pkcsconf -t' command shows the token label from the source system -

```
GSKIT2:/ # pkcsconf -t
Token #0 Info:
        Label: share
        Manufacturer: IBM Corp.
        Model: IBM ICA
```

Step 11: Update the IBM HTTP Server httpd.conf file on the target system (GSKIT2) to use the token label and certificate from the source system (metlnx30):

```
<VirtualHost GSKIT2.pdl.pok.ibm.com:443>
ServerName metlnx30.pdl.pok.ibm.com
SSLEnable
DocumentRoot /opt/IBM/HTTPServer/htdocs/en_US
KeyFile /opt/IBM/HTTPServer/Plugins/etc/plugin-key.kdb
SSLServerCert  share:thawte-test
SSLStashfile /opt/IBM/HTTPServer/ssl/pkcs11stash
SSLPKCSDriver /usr/lib/pkcs11/PKCS11_API.so
</VirtualHost>
```

Step 12: Verify that the userid used to run the HTTP server is a member of the group pkcs11:

```
GSKIT2:/ # usermod -G nobody,pkcs11,nogroup nobody
usermod: `nobody' is primary group name.
```

Step 13: Start the HTTP Server on the target system, and make an HTTPS request from a browser. Verify the request is successful.

Display the z90crypt driver information:

```
# cat /proc/driver/z90crypt
```

Verify that each new HTTPS session between a browser and the HTTP Server results in the request counter being incremented:

```
Per-device successfully completed request counts
    00000001 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

ZSW03124-USEN-00