Jack Hoarau
Yann Kindelberger

Using Cryptographic Adapters for Web Servers with Linux on IBM System z9 and zSeries

Introduction

This redpaper describes how to configure Web servers (such as Apache2 and IBM® HTTP server) to use hardware cryptographic devices in Linux® running on IBM System z9™ and zSeries® platforms. Hardware cryptographic cards are used to improve SSL performance during SSL handshaking. RedHat Enterprise Linux Advanced Server and SUSE Linux Enterprise Server distributions include packages to use these crypto cards. This document studies the different configuration steps required with the Linux distributions.

The z90crypt device driver

The cryptographic *z90crypt* device driver for Linux for zSeries and System z9 is a generic character device that routes work to a supported cryptographic coprocessor or accelerator device installed on the system:

- PCI Cryptographic Coprocessor (PCICC)
- PCI Cryptographic Accelerator (PCICA)
- PCI-X Cryptographic Coprocessor (PCIXCC)
- Cryptographic Express2 Coprocessor (CEX2C)
- Cryptographic Express2 Accelerator (CEX2A)

The device driver uses hardware to enhance performance during SSL handshaking. It supports only clear-key functions (cryptographic operations used during the handshake) for:

- Public/private key encryption and decryption
- RSA exponentiation

Both SUSE Linux Enterprise Server 9 (SLES9) and RedHat Enterprise Linux 4 (RHEL4) distributions for IBM System z9 and zSeries provide the z90crypt device driver as a separate kernel module. Configuration options and command scripts are provided to control driver initialization.

Exploiting hardware encryption

Although an application can use the z90crypt application programming interface (API) directly, most applications access the encryption hardware through a standard set of high-level APIs. These include:

- OpenCryptoki

  An open source implementation of Public-Key Cryptography Standard #11 (PKCS#11), OpenCryptoki, uses the libica shared library to access IBM cryptographic adapters through the z90crpyt device driver.

- OpenSSL

  An open source implementation of Secure Sockets Layer, OpenSSL, can utilize the libica shared library for hardware encryption.

  Note: In SLES9, libica is provided as a separate RPM package. With RHEL4, libica is provided as part of the OpenSSL RPM package.

- Global Security Kit (GSKit)

  Provided as part of the IBM HTTP server, GSKit manages SSL certificates. It utilizes OpenCryptoki for hardware encryption.

The relationship between these components is shown in Figure 1 on page 3.

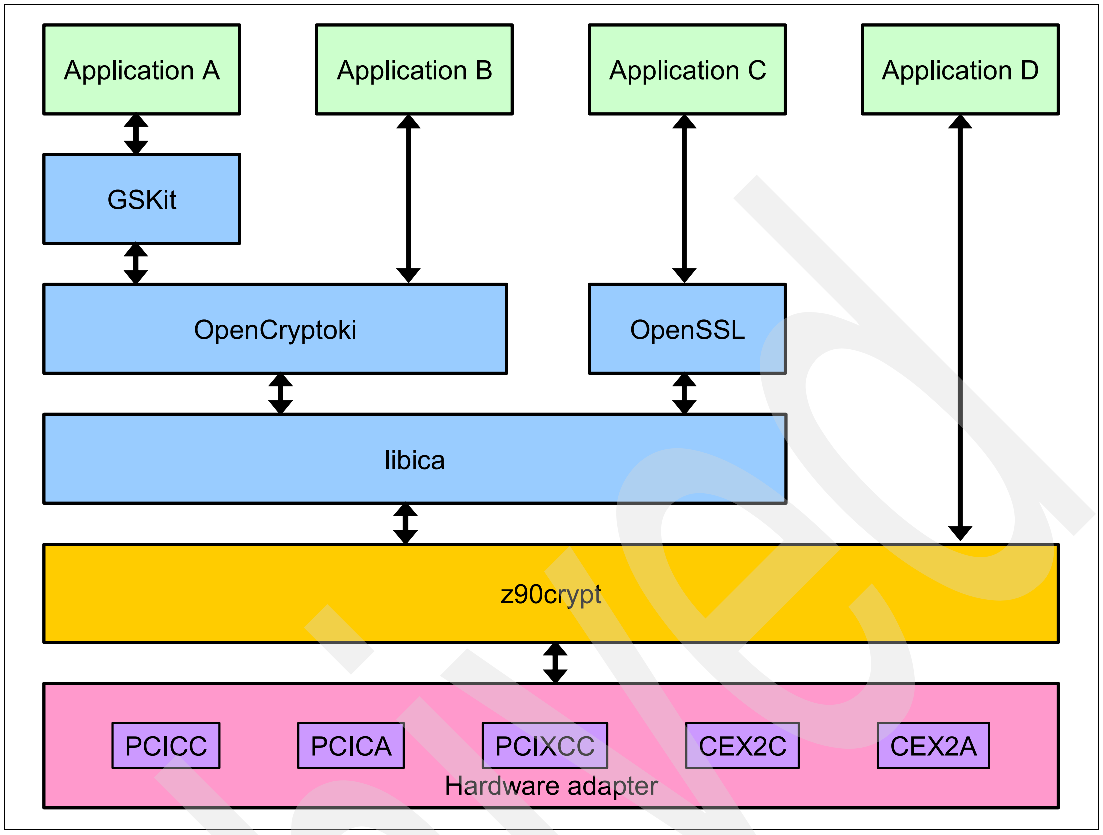


Figure 1 Exploiting hardware encryption

Software packaging for hardware cryptographic devices

Both SLES9 and RHEL4 distributions for Linux on IBM System z9 and zSeries provide RPM packages to support hardware cryptographic devices.

Required software for SLES9

Software packages provided in SLES9 for hardware cryptographic support include (the software versions are specific to SLES9 SP2):

- z90crypt device driver

  The z90crypt device driver is compiled as a loadable kernel module and installed in the kernel modules library directory. The full path name for the driver is /lib/modules/*kernel-level*/drivers/s390/crypto/z90crypt.ko (where *kernel-level* is the specific kernel version installed on the system).

- libica RPM package

  The libica shared library is provided as a separate RPM package. Use the **rpm** command to verify libica is installed:

```
# rpm -qa | grep libica
libica-32bit-9-200505240249
libica-1.3.5-3.10
```

Note: Both the 32-bit and 64-bit libica packages must be installed when running on a 64-bit Linux version (s390x).

Package contents are listed below:

```
# rpm -ql libica
/etc/init.d/z90crypt
/usr/include/ica_api.h
/usr/lib64/libica.so
/usr/sbin/rcz90crypt
/usr/share/doc/packages/libica
/usr/share/doc/packages/libica/README.SUSE
/var/adm/fillup-templates/sysconfig.z90crypt
# rpm -ql libica-32bit-9-200505240249
/usr/lib/libica.so
```

- openCryptoki RPM package

  The openCryptoki package is provided as a separate RPM:

```
# rpm -qa | grep openCryptoki
openCryptoki-32bit-2.1.5-2.6
openCryptoki-2.1.5-2.6
openCryptoki-64bit-2.1.5-2.6
```

> **Note:** Both the 32-bit and 64-bit openCryptoki packages must be installed when running on a 64-bit Linux version (s390x).

  Package contents are listed below:

```
# rpm -ql openCryptoki-32bit-2.1.5-2.6
/usr/lib/pkcs11
/usr/lib/pkcs11/PKCS11_API.so
/usr/lib/pkcs11/methods
/usr/lib/pkcs11/methods/pkcs11_startup
/usr/lib/pkcs11/methods/pkcs_slot
/usr/lib/pkcs11/methods/pkcsconf
/usr/lib/pkcs11/stdll
/usr/lib/pkcs11/stdll/PKCS11_ICA.so
/usr/sbin/pkcsslotd
# rpm -ql openCryptoki-2.1.5-2.6
/etc/init.d/pkcsslotd
/etc/pkcs11
/usr/sbin/rcpkcsslotd
```

Additional shared libraries may be required for compatibility with other applications. These can include:

- libncurses.so.1.9.7a
- libpng.so.1.0.89
- libstdc++.so.2.7.2
- libstdc++.so.2.8
- libstdc++.so.2.9
- libstdc++-3-libc6.1-2-2.10.0.so

These are available in *compat* and *compat-32bit* RPM packages that must also be installed.

Required software for RHEL4

Software packages provided in RHEL4 for hardware cryptographic support include (software versions are specific to RHEL4 Update 2):

- z90crypt device driver

The z90crypt device driver is compiled as a loadable kernel module and installed in the kernel modules library directory. The full path name for the driver is /lib/modules/*kernel-level*.EL/drivers/s390/crypto/z90crypt.ko (where *kernel-level* is the specific kernel version installed on the system).

- OpenSSL RPM package

  The libica shared library is provided as a part of the OpenSSL RPM package. Use the **`rpm`** command to verify OpenSSL is installed:

  ```
  # rpm -qa | grep openssl
  openssl-0.9.7a-43.2
  xmlsec1-openssl-1.2.6-3
  openssl-0.9.7a-43.2
  ```

  Note: The OpenSSL package appears twice. This is a 64-bit Linux distribution, and both the 32-bit and 64-bit versions of OpenSSL must be installed.

  Partial package contents are listed below:

  ```
  # rpm -ql openssl
  /lib64/libcrypto.so.0.9.7a
  /lib64/libssl.so.0.9.7a
  /usr/bin/openssl
  /usr/lib64/libica.so
  ....
  /lib/libcrypto.so.0.9.7a
  /lib/libssl.so.0.9.7a
  /usr/lib/libica.so
  ```

Other shared objects libraries may be required for full compatibility (these are part of specific *compat* packages). For 64-bit RHEL4, both the 32-bit and 64-bit libraries are required:

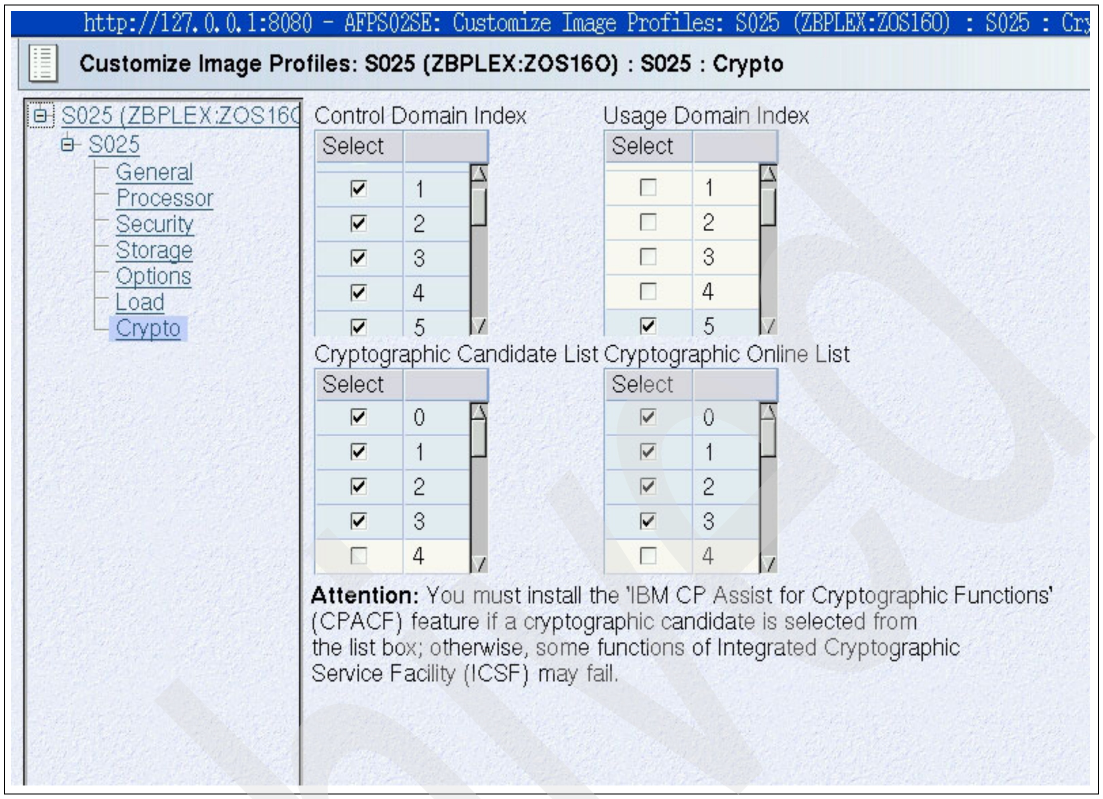
```
# rpm -qa | grep compat
compat-libstdc++-295-2.95.3-81
compat-libgcc-295-2.95.3-81
compat-libstdc++-33-3.2.3-47.3
compat-libstdc++-295-2.95.3-81
compat-libgcc-295-2.95.3-81
compat-libstdc++-33-3.2.3-47.3
# rpm -qa | grep libgcc
compat-libgcc-295-2.95.3-81
libgcc-3.4.4-2
libgcc-3.4.4-2
compat-libgcc-295-2.95.3-81
```

Defining cryptographic hardware to the Linux or z/VM LPAR

Cryptographic coprocessor or accelerator devices can be accessed by Linux running in either an LPAR or as a z/VM® guest. The LPAR running Linux or z/VM must be customized to enable the cryptographic operations on the devices. This is done from the Hardware Management Console and the Support Element. The following must be defined in the image profile of the partition:

- Usage Domain Index
- Control Domain Index
- PCI Cryptographic Coprocessor Candidate List
- PCI Cryptographic Coprocessor Online List

The Hardware Management Console dialog to define cryptographic devices is shown in Figure 2.


http://127.0.0.1:8080 - AFPS02SE: Customize Image Profiles: S025 (ZBPLEX:ZOS160) : S025 : Cry

Customize Image Profiles: S025 (ZBPLEX:ZOS16O) : S025 : Crypto

- S025 (ZBPLEX:ZOS16
  - S025
    - General
    - Processor
    - Security
    - Storage
    - Options
    - Load
    - Crypto

Control Domain Index

Select	
✓	1
✓	2
✓	3
✓	4
✓	5

Usage Domain Index

Select	
☐	1
☐	2
☐	3
☐	4
✓	5

Cryptographic Candidate List

Select	
✓	0
✓	1
✓	2
✓	3
☐	4

Cryptographic Online List

Select	
✓	0
✓	1
✓	2
✓	3
☐	4

Attention: You must install the 'IBM CP Assist for Cryptographic Functions' (CPACF) feature if a cryptographic candidate is selected from the list box; otherwise, some functions of Integrated Cryptographic Service Facility (ICSF) may fail.

Figure 2 Defining cryptographic hardware in the Hardware Management Console

Under z/VM, the cryptographic device is enabled for a guest usage by the CRYPTO directory control statement with one operand: APVIRT. This statement must appear in the user directory of the guest before any device statement. An example user directory entry is shown in Figure 3.

```
USER LINUX1  UNLOG   128M 768M      G
   INCLUDE PRFLINUX
   ACCOUNT  X25SP
   MACHINE ESA 4
   CRYPT APVIRT
   CPU 00 BASE
   CPU 01
   SPECIAL  814 QDIO 3 SYSTEM LANQ00
   LINK LINUX 191 191   RR
   MDISK 0100 3390 1   3000 LINX00 MR  READ WRITE MULT
   MDISK 0101 3390 3001 338 LINX01 MR
   MINIOPT NOMDC
   MDISK 0200 3390 1    1111 LINX10 MR
   MDISK 0201 3390 1112 1111 LINX12 MR
   MDISK 0202 3390 2223 1111 LINX14 MR
   LINK LINXTO 200 1200 RR
   LINK LINXTO 201 1201 RR
```

Figure 3 The CRYPTO APVIRT user directory statement

The CRYPTO statement with the APVIRT operand tells CP the Linux guest may use the clear-key functions on the Adjunct Processor (AP). z/VM hides PCICC and PCIXCC devices from a guest if a PCICA device is also available.

Loading the z90crypt device driver

The device node to the cryptographic hardware is /dev/z90crypt. This is automatically defined by udev (the dynamic /dev directory manager) at system initialization. The /dev/z90crypt major device number is set to misc device number; the minor number is dynamically assigned. Both values can be found in the /proc/misc interface. The z90crypt module can be automatically loaded as a system service, or manually using the **modprobe** command.

Loading the z90crypt device driver in SLES9

SLES9 provides the **rcz90crypt** command to load, unload, and query status of the z90crypt driver:

```
# rcz90crypt status
Checking for module z90crypt:                      running
# rcz90crypt stop
Unloading z90crypt module:                         done
# rcz90crypt status
Checking for module z90crypt:                      dead
# rcz90crypt start
Loading z90crypt module                            done
# rcz90crypt status
Checking for module z90crypt:                      running
```

To manually load the z90crypt device driver, use the **modprobe** command:

```
# modprobe z90crypt domain=-1
# lsmod | grep z90crypt
 z90crypt               74664  0
```

The domain=-1 option causes the device driver to automatically detect the domain to use. This is the default if no domain option is specified.

Note: The domain value can also be an integer in the range of 0 to 15 that identifies the cryptographic domain for the Linux instance. The option can be omitted if only one domain is defined.

The z90crypt driver can be automatically loaded at system boot. The z90crypt system service must be set to *on* for boot initialization:

```
# chkconfig z90crypt
z90crypt off
# chkconfig z90crypt on
```

Loading the z90crypt device driver in RHEL4

RHEL4 does not provide a script to load and unload the 90crypt device driver. Use the **modprobe** and **rmmod** commands to load and unload the device driver:

```
# cat /proc/misc
63 device-mapper
# modprobe z90crypt domain=-1
# cat /proc/misc
```

```
62 z90crypt
63 device-mapper
# lsmod | grep z90crypt
z90crypt              678800
```

To automate loading of the device driver at system initialization, append a **modprobe** command in the /etc/rc.d/rc.local startup script:

```
# cd /etc/rc.d
# cp -p rc.local rc.localOLD
# echo “modprobe z90crypt domain=-1” >> ./rc.local
```

Checking the z90crypt device status

Once the device driver is loaded, device status can be checked using the /proc/drivers/z90crypt interface, as shown in Figure 4.

```
# cat /proc/driver/z90crypt
z90crypt version: 1.3.1
Cryptographic domain: 8
Total device count: 1
PCICA count: 1
PCICC count: 0
PCIXCC count: 0
requestq count: 0
pendingq count: 0
Total open handles: 1                                                    1
Online devices: 1 means PCICA, 2 means PCICC, 3 means PCIXCC             2
    0000000000000000 0000000100000000 0000000000000000 0000000000000000
Waiting work element counts                                              3
    0000000000000000 0000000000000000 0000000000000000 0000000000000000
Per-device successfully completed request counts                         4
    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000 00000000 00000000 00000000 0000007C
    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
```

Figure 4 Status of hardware adapters as reported by the z90crypt device driver

In the report, output of interest includes:

1. *Total open handles* reports the number of current open connections (for instance, the number of active Web servers with a cryptographic device enabled for SSL).
2. *Online devices* reports the type and physical arrangement of detected cryptographic adapters.
3. *Waiting work element counts* reports the number of outstanding units of work for each detected adapter.
4. *Per-device successfully completed request counts* reports the number of successfully completed units of work for each detected adapter. Failed requests are not counted. Each time a Web server establishes a new SSL connection, this value is incremented.

The OpenSSL engine interface

OpenSSL implements Secure Sockets Layer V2/V3 (SSL) and Transport Layer Security V1 (TLS) protocols. It provides an interface to hardware engines (cryptographic devices) that perform cryptographic operations in hardware instead of software. In both SLES9 and RHEL4, OpenSSL includes an *engine interface* that links to the libica shared library. The engine interface can be queried to determine supported hardware and its associated capabilities:

```
# openssl engine -c
(dynamic) Dynamic engine loading support
(cswift) CryptoSwift hardware engine support
 [RSA, DSA, DH, RAND]
(chil) nCipher hardware engine support
 [RSA, DH, RAND]
(atalla) Atalla hardware engine support
 [RSA, DSA, DH]
(nuron) Nuron hardware engine support
 [RSA, DSA, DH]
(ubsec) UBSEC hardware engine support
 [RSA, DSA, DH]
(aep) Aep hardware engine support
 [RSA, DSA, DH]
(ibmca) Ibmca hardware engine support
 [RSA, DSA, DH, RAND, DES-ECB, DES-CBC, DES-EDE3, DES-EDE3-CBC, SHA1]
(sureware) SureWare hardware engine support
 [RSA, DSA, DH, RAND]
(4758cca) IBM 4758 CCA hardware engine support
 [RSA, RAND]
```

Note: OpenSSL supports IBM cryptographic adapters if the ”ibmca” engine interface is reported.

The OpenCryptoki PKCS#11 subsystem

The OpenCryptoki package provides a PKCS#11 interface to cryptographic hardware devices that can manage and store user keys on PKCS#11 devices. OpenCryptoki consists of a slot manager and an API for slot token dynamic link libraries (STDLLs). The slot manager runs as a daemon to control token slots provided to applications. Managed devices store tokens in the slot manager database. Components provided by OpenCryptoki include:

- Slot manager daemon (/usr/sbin/pkcsslotd)
- Slot manager daemon service control script (/etc/rc.d/init.d/pkcsslotd)
- APIs to the STDLLs:
  - /usr/lib/pkcs11/PKCS11_API.so
  - /usr/lib/pkcs11/PKCS11_API.so64
- Configuration utilities:
  - /usr/lib/pkcs11/methods/pkcs11_startup
  - /usr/lib/pkcs11/methods/pkcs_slot
  - /usr/lib/pkcs11/methods/pkcsconf
- STDLLs plugins to the cryptographic adapters:
  - /usr/lib/pkcs11/stdll/PKCS11_ICA.so
  - /usr/lib/pkcs11/stdll/PKCS11_ICA.so64

Configuring the PKCS subsystem

With both SLES9 and RHEL4, OpenCryptoki must first be configured using the **`pkcs11_startup`** script.

> **Note:** The z90crypt driver must be initialized as described in "Loading the z90crypt device driver" on page 7 before the PKCS subsystem is initialized.

When executed, the script:

- Creates Linux group "pkcs11"
- Scans for an installed device (/dev/z90crypt)
- Creates the slot configuration file (/etc/pkcs11/pk_config_data)

The slot manager daemon can then be started using the **`pkcsslotd`** command or the OpenCryptoki service control script. Any application that accesses the PKCS subsystem must run as root or under a Linux user that is a member of the pkcs11 group.

> **Note:** If z90crypt is not loaded before starting pkcsslotd, a message such as `z90crypt module is not installed – PKCS#11 will not be hardware accelerated` is generated.

Starting the PKCS subsystem with SLES9

To manually start the PKCS subsystem in SLES9, load the z90crypt device driver and start the pkcsslotd daemon using the **`rcpkcsslotd`** service control script:

```
# rcz90crypt start
Loading z90crypt module
done
# /usr/lib/pkcs11/methods/pkcs11_startup
usermod: `root' is primary group name.
# rcpkcsslotd start
Starting pkcsslotd daemon:usermod: `root' is primary group name.
done
```

Once OpenCryptoki has been configured using the **`pkcs11_startup`** command, the PKCS subsystem can be started automatically at system boot:

- Ensure the device driver is loaded at initialization:

  ```
  # chkconfig z90crypt
  z90crypt off
  # chkconfig z90crypt on
  ```

- Use the **`chkconfig`** command to start the pcksslotd daemon at initialization for runlevels 3 and 5:

  ```
  # chkconfig --list | grep pkcsslotd
  pkcsslotd          0:off  1:off  2:off  3:off  4:off  5:off  6:off
  # chkconfig pkcsslotd on
  # chkconfig --list | grep pkcsslotd
  pkcsslotd          0:off  1:off  2:off  3:on   4:off  5:on   6:off
  ```

Starting the PKCS subsystem with RHEL4

To manually start the PKCS subsystem in RHEL4, load the z90crypt device driver, configure the PKCS subsystem, and start the pkcsslotd daemon:

```
# modprobe z90crypt domain=-1
```

```
# /usr/lib/pkcs11/methods/pkcs11_startup
# service pkcsslotd status
pkcsslotd is stopped
# service pkcsslotd start
Starting pkcsslotd:
# service pkcsslotd status
pkcsslotd (pid 1090) is running...
```

With RHEL4, the pkcsslotd service control script (/etc/init.d/pkcsslotd) cannot be managed using the **chkconfig** command. To automatically start the PKCS subsystem at system boot, append an entry in the /etc/rc.d/rc.local startup script to start the pkcsslotd daemon after z90crypt has been loaded:

```
# cat /etc/rc.d/rc.local
#!/bin/sh
#
# This script will be executed *after* all the other init scripts.
# You can put your own initialization stuff in here if you don't
# want to do the full Sys V style init stuff.
touch /var/lock/subsys/local
modprobe z90crypt domain=-1

# echo "service pkcsslotd start" >> /etc/rc.d/rc.local

# cat /etc/rc.d/rc.local
#!/bin/sh
# This script will be executed *after* all the other init scripts.
# You can put your own initialization stuff in here if you don't
# want to do the full Sys V style init stuff.
touch /var/lock/subsys/local
modprobe z90crypt domain=-1
service pkcsslotd start
```

Managing the PKCS#11 device

Together, the PKCS#11 layer API and underlying cryptographic device is referred to as a *PKCS#11 device*. Tokens are associated with devices, and are used to manage and store users keys. OpenCryptoki provides the **pkcsconf** command to:

- Initialize tokens.
- Set Security Officer (SO) PINs.
- Initialize, set, and change user PINs.

The slot manager daemon (pkcsslotd) must by running when the **pkcsconf** command is executed. For help with the syntax of the **pkcsconf** command, use the -h option:

```
# /usr/lib/pkcs11/methods/pkcsconf -h
/usr/lib/pkcs11/methods/pkcsconf: invalid option -- h
usage:  /usr/lib/pkcs11/methods/pkcsconf [-itsmMIupP] [-c slotnumber -U userPIN -S SOPin -n
newpin]
        -i display PKCS11 info
        -t display token info
        -s display slot info
        -m display mechanism list
        -I initialize token
        -u initialize user PIN
        -p set the user PIN
        -P set the SO PIN
```

To display token information, use the -t option:

```
# /usr/lib/pkcs11/methods/pkcsconf -t
Token #0 Info:
        Label: IBM ICA  PKCS #11
        Manufacturer: IBM Corp.
        Model: IBM ICA
        Serial Number: 123
        Flags: 0x880045
        Sessions: -1/-1
        R/W Sessions: -1/-1
        PIN Length: 4-8
        Public Memory: 0xFFFFFFFF/0xFFFFFFFF
        Private Memory: 0xFFFFFFFF/0xFFFFFFFF
        Hardware Version: 1.0
        Firmware Version: 1.0
        Time: 06:31:47 PM
```

Note: “IBM ICA PKCS #11” is the default token label. This must be changed at token initialization.

To display PKCS#11 information, use the -i option:

```
# /usr/lib/pkcs11/methods/pkcsconf -i
PKCS#11 Info
        Version 2.11
        Manufacturer: IBM
        Flags: 0x0
        Library Description: Meta PKCS11 LIBRARY
        Library Version 1.4
```

To display slot information, use the -s option:

```
# /usr/lib/pkcs11/methods/pkcsconf -s
Slot #0 Info
        Description: Linux 2.6.5-7.191-s390x Linux (ICA)
        Manufacturer: Linux 2.6.5-7.191-s390x
        Flags: 0x5
        Hardware Version: 0.0
        Firmware Version: 1.1
```

Configuring a PKCS#11 device

To configure a PKCS#11 device:

1. Initialize the token.

   A token must be initialized before it can be used. To initialize the token label (replacing the default label), specify the slot number using the -c option and the -I option:

   ```
   # /usr/lib/pkcs11/methods/pkcsconf -c 0 -I
   Enter the SO PIN: ********
   Enter a unique token label: PCICA0
   ```

   When prompted, provide the default SO PIN (87654321). To check that the label has changed, use the -t option:

   ```
   # /usr/lib/pkcs11/methods/pkcsconf -t
   Token #0 Info:
           Label: PCICA0
           Manufacturer: IBM Corp.
           Model: IBM ICA
           Serial Number: 123
   ```

```
        Flags: 0x88044D
        Sessions: -1/-1
        R/W Sessions: -1/-1
        PIN Length: 4-8
        Public Memory: 0xFFFFFFFF/0xFFFFFFFF
        Private Memory: 0xFFFFFFFF/0xFFFFFFFF
        Hardware Version: 1.0
        Firmware Version: 1.0
        Time: 18:31:36
```

The token label ("PCICA0" in this example) identifies the cryptographic token for storing the HTTP SSL server certificate.

2. Set the SO PIN.

   It is a good security practice to set the SO PIN. The SO PIN secures access to the administrative functions for the PKCS#11 device. Use the -P option:

```
# /usr/lib/pkcs11/methods/pkcsconf -c 0 -P
Enter the SO PIN: ********
Enter the new SO PIN: ********
Re-enter the new SO PIN: ********
```

3. Set the user PIN.

   This is done by the security officer. The user PIN secures access to the token stored in the PKCS#11 device slot database. To access the token holding the SSL certificate, users (or applications such as IHS) must provide the user PIN. Specify the -u option:

```
 # /usr/lib/pkcs11/methods/pkcsconf -c 0 -u
Enter the SO PIN: ********
Enter the new user PIN: ********
Re-enter the new user PIN: ********
```

4. Change the user PIN.

   This is done by the user using the -p option:

```
# /usr/lib/pkcs11/methods/pkcsconf -c 0 -p
Enter user PIN: ********
Enter the new user PIN: ********
Re-enter the new user PIN: ********
```

   The user PIN number is required to access the token.

The PKCS#11 device is now configured to store and manage the keys for an application such as IBM HTTP Server.

Using cryptographic devices with SSL

Apache and IBM HTTP Server (IHS) Web servers are supported on Linux for zSeries and System z9. Both can utilize hardware cryptographic devices to improve performance during SSL handshake negotiation. Figure 5 on page 14 illustrates the software used to support hardware cryptographic devices.

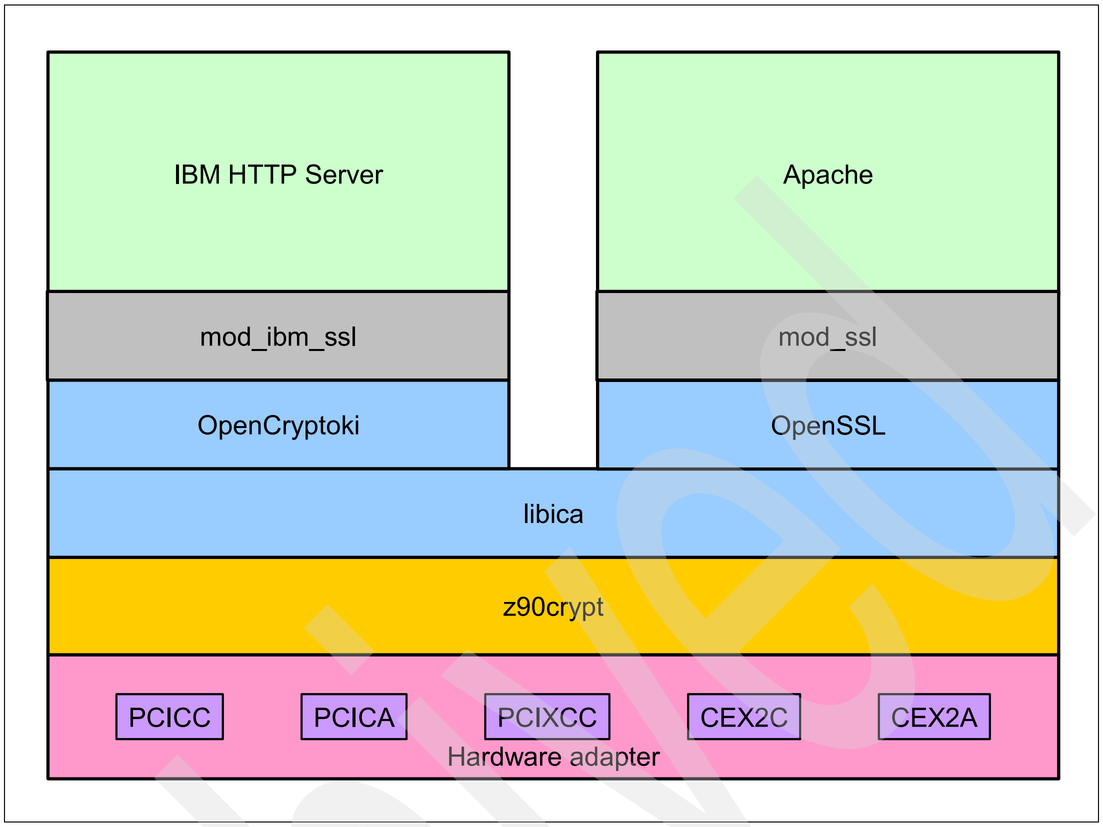


Figure 5 Using cryptographic adapters with Apache and IHS Web servers

To access the cryptographic adapters:

- Apache uses the mod_ssl module.

  To access the cryptographic adapter, mod_ssl links to OpenSSL.
- IHS uses the mod_ibm_ssl module.

  To access the cryptographic adapter, mod_ibm_ssl uses PKCS#11 services available with OpenCryptoki.

Next we show how to configure Apache 2.0 and IHS to use the hardware adapters with SLES9 and RHEL4.

Configuring cryptographic devices with Apache 2.0

Apache Version 2 is available with SLES9 SP2 and RHEL4. It uses OpenSSL to connect to cryptographic devices. The mod_ssl module provides the interface to the OpenSSL libraries.

Configuring Apache 2.0 on RHEL4

Apache 2.0 on RHEL4 provides a default configuration with SSL enabled. In RHEL4, mod_ssl is enabled and is SSL configured to use a fake self-signed certificate suitable for testing purpose only. Apache Version 2.0.52 is used in the examples that follow.

To enable SSL with Apache 2.0:

1. Load the z90crypt device driver.

Follow the steps outlined in “Loading the z90crypt device driver in RHEL4” on page 7.

2. Ensure that OpenSSL is installed and enabled for IBM cryptographic hardware.

   For details see “The OpenSSL engine interface” on page 9.

3. Add a certificate to the key database.

   The /etc/httpd/conf/httpd.conf file contains two directives to configure the key database:

   – The SSLCertificateFile directive specifies the SSL certificate.
   – The SSLCertificatekeyFile directive specifies the key file for the SSL certificate.

   By default, these point to a fake self-signed certificate and key suitable tor testing. OpenSSL provides an interface to create a new certificate and key. To use a new certificate and key:

   a. Create a new certificate and key files in the /etc/httpd/conf directory:

   ```
   # cd /etc/httpd/conf
   # openssl req -new > ./ssl.csr/ap2.cert.csr
   ```

   b. Remove the passphrase from the key (saving the key in a key file):

   ```
   # openssl rsa -in privkey.pem -out ./ssl.key/ap2.cert.key
   ```

   c. Convert the request to a signed certificate:

   ```
   # openssl x509 -in ./ssl.csr/ap2.cert.csr -out ./ssl.crt/ap2.cert.cert -req \
   -signkey ./ssl.key/ap2.cert.key -days 365
   ```

   d. Point to the new certificate and key in the /etc/httpd/conf/httpd.conf:

   ```
   SSLCertificateFile     /etc/httpd/conf/ssl.crt/ap2.cert.cert
   SSLCertificatekeyFile  /etc/httpd/conf/ssl.key/ap2.cert.key
   ```

4. Configure Apache 2.0 to use the ibmca OpenSSL engine interface.

   The /etc/httpd/conf/httpd.conf configuration file contains a directive to include other configuration files:

   ```
   # Load config files from the config directory "/etc/httpd/conf.d".
   Include conf.d/*.conf
   ```

   SSL configuration directives are contained in the /etc/httpd/conf.d/ssl.conf file. The “SSLCryptoDevice” directive defines which engine interface is enabled:

   ```
   # Use "SSLCryptoDevice" to enable any supported hardware
   # accelerators. Use "openssl engine -v" to list supported
   # engine names.  NOTE: If you enable an accelerator and the
   # server does not start, consult the error logs and ensure
   # your accelerator is functioning properly.
   #SSLCryptoDevice builtin
   #SSLCryptoDevice ubsec
   SSLCryptoDevice ibmca
   ```

   Add a SSLCryptoDevice directive for the “ibmca” device. Existing SSLCryptoDevice directives for other device types must be removed or commented out.

To start Apache, use the **apachectl start** command. Once started, look for a new open handle in the z90crypt status report, as described in “Checking the z90crypt device status” on page 8.

Configuring Apache 2.0 on SLES9

With SLES9, SSL is not enabled for Apache 2.0 by default. SSL configuration is controlled by several configuration files:

- The /etc/sysconfig/apache2 file defines general system configuration.

- The /etc/apache2 directory contains server-specific configuration files and directories:
  - The httpd.conf file is the main server configuration file
  - The ssl-global.conf file configures SSL for the main server and all virtual hosts.
  - The vhost.d directory contains virtual host configuration files.

SSL can be configured by either manually editing configuration files or using the YaST graphic interface. Apache Version 2.0.49 is used in the examples that follow.

Configuring Apache 2.0 with YaST

To configure Apache using YaST:

1. Start YaST using the `yast` command.
2. Navigate to **Network** → **HTTP Server** to the HTTP Server Configuration menu.
3. In the HTTP Server Configuration menu:
   a. Set the HTTP Service option to Enabled.
   b. Set the Listen on option to 80.
   c. Select the **Modules** option and click **Edit** to navigate to the Server Modules menu.
   d. In the Server Modules menu, select the **ssl** option and set Toggle Status to enabled.
4. Return to the HTTP Server Configuration menu, select **Default Host**, and click **Edit**.
5. In the Host default Configuration menu:
   a. Set the SSL option to SSL Allowed.
   b. In the SSL Configuration for default menu:
      i. Select the **SSL** option, click **Edit**, and select **SSL Allowed**.
      ii. Select the **Certificates** option to generate or import a server key and certificate. Selecting **Use common Server Certificate** from the opening list automatically adds the SSLCertificateFile and SSLCertificatekey File directives to the /etc/httpd/httpd.conf file.
      iii. Click **OK** to return.
6. In the HTTP Server Configuration menu, select **Finish** to complete the configuration.

YaST updates the system and server-specific configuration files.

Manually configuring Apache 2.0

To manually configure Apache 2.0:

1. Edit the /etc/sysconfig/apache2 file:
   - The APACHE_MODULES=”*module_list*” directive must include “ssl” in *module_list*.
   - The APACHE_SERVER_FLAGS=”SSL” directive must include “SSL”.

   Execute the `SuSEconfig` command to generate the Apache configuration files.
2. Generate a server certificate or import an existing server certificate.

   SLES9 provides the `/usr/share/packages/apache2/certificates.sh` command to generate a server certificate.
3. Create a virtual host configuration file

   Copy a template virtual host configuration file to the virtual host configuration directory:

   ```
   # cp /etc/apache2/vhost.d/vhost-ssl.template /etc/apache2/vhost.d/vhost-ssl.conf
   ```

   Edit the vhost-ssl.conf file and add the location of the server certificate and the server private key.

Starting the Apache 2.0 server

Before starting the Apache 2.0 server:

1. Load the z90crypt device driver.

   Follow the steps outlined in “Loading the z90crypt device driver in SLES9” on page 7.

2. Ensure OpenSSL is installed and enabled for IBM cryptographic hardware.

   For details see “The OpenSSL engine interface” on page 9.

3. Configure Apache 2.0 to enable SSL.

   This can be done using YaST (as described in “Configuring Apache 2.0 with YaST” on page 16) or manually using the steps described in “Manually configuring Apache 2.0” on page 16.

4. Configure Apache 2.0 to use the ibmca OpenSSL engine interface.

   Edit the /etc/apache2/ssl-global.conf file and set the SSLCryptoDevice directive to “ibmca”:

   ```
   # This global SSL configuration is ignored if
   # "SSL" is not defined, or if "NOSSL" is defined.
   <IfDefine SSL>
   <IfDefine !NOSSL>
   <IfModule mod_ssl.c>
   SSLCryptoDevice ibmca
   #
   # Some MIME-types for downloading Certificates and CRLs
   #
   AddType application/x-x509-ca-cert .crt
   AddType application/x-pkcs7-crl    .crl
   ```

The server can be started and stopped as a service using the **rcapache** command:

```
# rcapache start
Starting httpd2 (worker)                                          done
```

To automatically start apache2, register the service using the **insserv** command:

```
# insserv apache2
```

Once started, look for a new open handle in the z90crypt status report, as described in “Checking the z90crypt device status” on page 8.

Configuring cryptographic devices with IHS

IHS accesses cryptographic devices using the OpenCryptoki PKCS#11 subsystem. Configuration steps are the same for both RHEL4 and SLES9. Before configuring IHS:

- Load the z90crypt driver as described in “Loading the z90crypt device driver” on page 7.
- Configure and start the PKCS#11 subsystem as described in “The OpenCryptoki PKCS#11 subsystem” on page 9.
- Ensure GSKit is installed with IHS (check the installation log files in the IHS installation directory).

To enable SSL with IHS, we:

1. Create the server certificate.
2. Configure IHS for SSL.
3. Start the IHS server

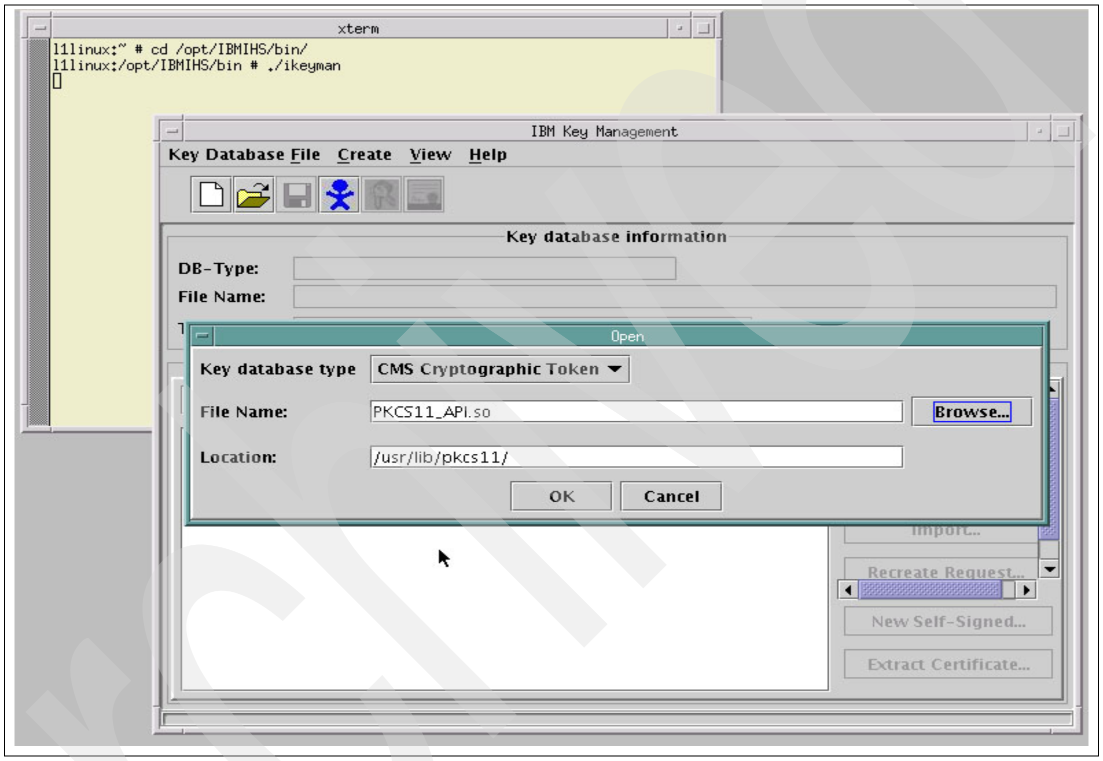
Examples in this section are specific to IHS Version 6.

Create the server certificate

Use GSKit tools to create the server certificate and store it in the PKCS#11 device. A graphic interface is provided to manage and store server certificates. To start the interface, use the **`ikeyman`** command:

```
# /opt/IBMIHS/bin/ikeyman
```

In the graphic interface, select the **Key Database File** → **Open** option to navigate to the Key database information dialog shown in Figure 6.



```
l11linux:~ # cd /opt/IBMIHS/bin/
l11linux:/opt/IBMIHS/bin # ./ikeyman
```

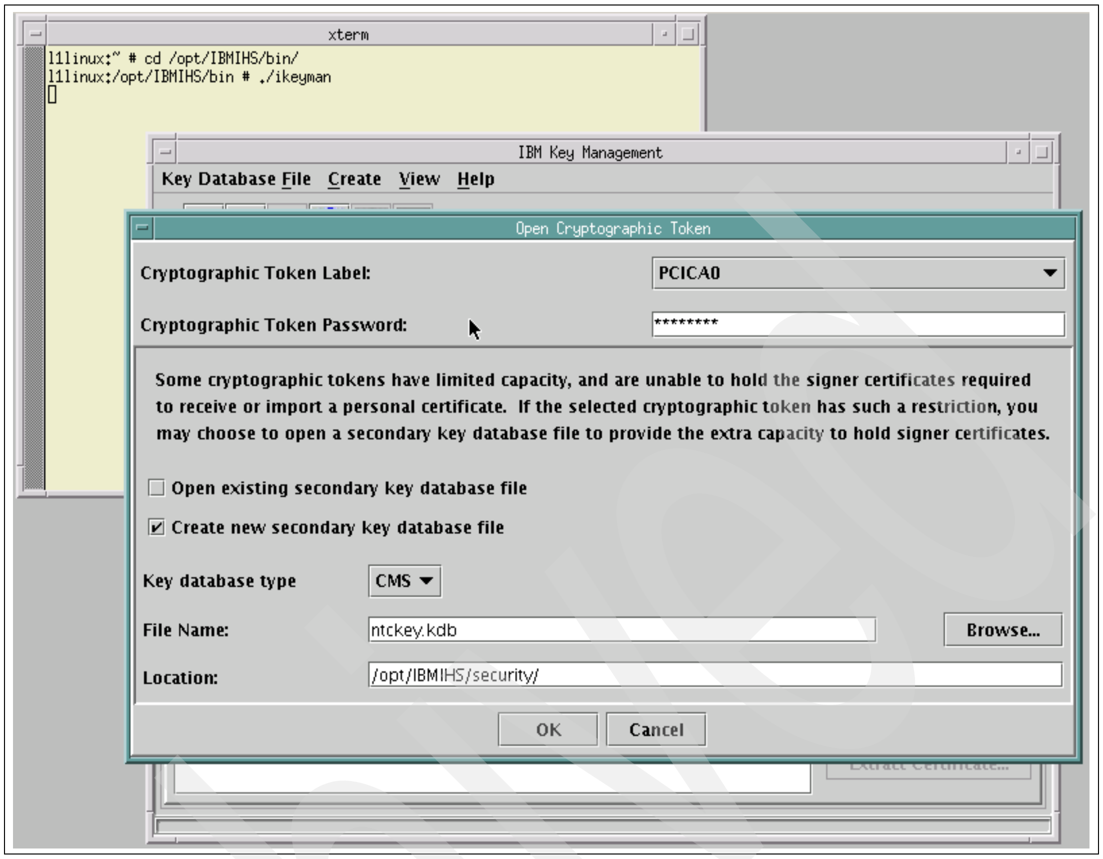
Figure 6 GSKit Key database information dialog

In the dialog:

- Choose **CMS Cryptographic Token** as the Key database type.
- Provide the full path name of the PKCS#11 API module:
  - Set File Name to "PKCS11_API.so".
  - Set Location to "/usr/lib/pkcs11/".

Tip: To search for PKCS#11 API module in the file system directories, click the **Browse** button.

Click **OK** to navigate to the Open Cryptographic Token dialog shown in Figure 7 on page 19.



```
l1linux:~ # cd /opt/IBMIHS/bin/
l1linux:/opt/IBMIHS/bin # ./ikeyman
```

Cryptographic Token Label: PCICA0

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: ntckey.kdb

Location: /opt/IBMIHS/security/

Figure 7 The GSKit Open Cryptographic Token dialog

In the dialog:

- Choose the **Cryptographic Token Label** of the PKCS#11 device.

  Use the token label assigned to the PCKS#11, as described in "Configuring a PKCS#11 device" on page 12. In this example, label "PCICA0" is selected.

- Provide the Cryptographic Token Password for the PKCS#11 device.

  Use the user PIN assigned to the PKCS#11 token (see "Configuring a PKCS#11 device" on page 12).

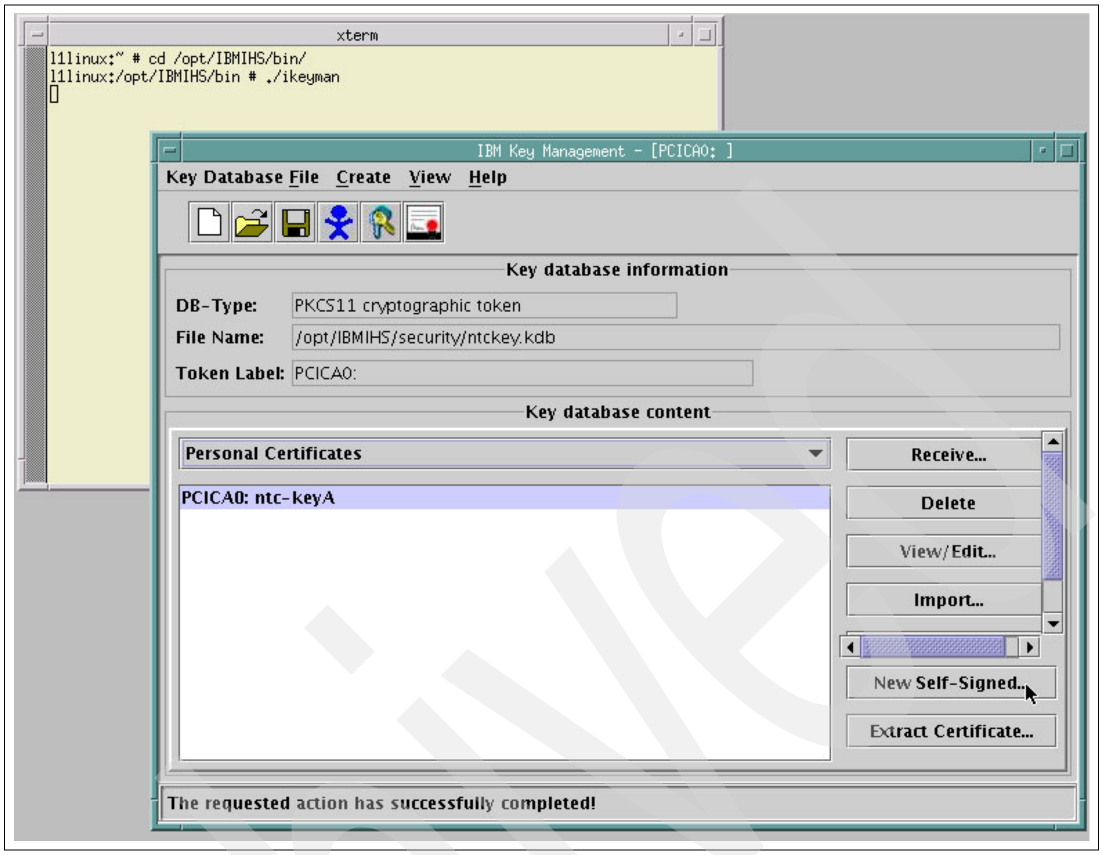
- Select the **Create new secondary key database file** option.

  Choose the **Key database type** of "CMS". In a separate window, create a directory for the key database file:

  ```
  # mkdir /opt/IBMIHS/security
  ```

  Provide the full path name of the key database file in the Location field (/opt/IBMIHS/security/ntckey.kdb in this example). The key database file name is used in the IHS httpd.conf file (provided as the Keyfile directive).

- Click **OK** to open the Cryptographic Token database and return to the Key database management dialog shown in Figure 8 on page 20.



```
l1linux:~ # cd /opt/IBMIHS/bin/
l1linux:/opt/IBMIHS/bin # ./ikeyman
```

IBM Key Management - [PCICA0:]

Key database information

DB-Type: PKCS11 cryptographic token

File Name: /opt/IBMIHS/security/ntckey.kdb

Token Label: PCICA0:

Key database content

Personal Certificates

PCICA0: ntc-keyA

The requested action has successfully completed!

Figure 8 The certificate is saved

- A certificate can now be created or registered in the Cryptographic token database.

  To create a self-signed certificate:

  a. Select **Personal Certificates**.
  b. Click **New Self-Signed**.
  c. Provide the required fields in the next dialog.
  d. Click **Create** to save the certificate.

  To register an officially delivered certificate:

  a. Select **Signer Certificates**.
  b. Click **Receive** and **Import** the certificate, as appropriate.

  The certificate label appears in the dialog ("PCICA0:ntc-keyA" in this example). To close the interface, select **Key Database File** → **Exit**.

Configure IHS for SSL

To configure IHS to use SSL, edit the /opt/IBMIHS/conf/httpd.conf server configuration file:

1. Define the server name.

   Set the ServerName directive to the server host name:

   ```
   ServerName hostname
   ```

2. Enable loading of the mod_ibm_ssl module.

   Add a LoadModule directive to conditionally load the mod_ibm_ssl module:

   ```
   #*** ADDED FOR SSL ***
   ```

```
<IfDefine SSL>
LoadModule ibm_ssl_module modules/mod_ibm_ssl.so
</IfDefine>
```

3. Set the user and group used to run the server.

   By default, IHS runs under user and group *nobody*:

```
##User nobody
##Group nobody
User wwwrun
Group www
```

> **Important:** Because IHS uses the PKCS subsystem, the selected user (in this case, user “wwwrun”) must be a member of group “pkcs11”. To add the user to the pkcs11 group, use the **usermod** command:
>
> ```
> # usermod -G pkcs11 wwwrun
> ```

4. Create a stash file to hold the user PIN of the PKCS#11 token.

   To create a stash file to hold the user PIN for the token, execute the **sslstash** command:

```
# /opt/IBMIHS/bin/sslstash -h
Usage: sslstash [-c] <file> <function>  <password>
-c       = Create a new stash file. If not specified, sslstash will
           attempt to update an existing file.
file     = Fully qualified name of the file to be created or updated.
function = Function for which the password will be used. Valid
           values are "crl" or "crypto".
password = The password to be stashed
```

   To create a stash file for the token, execute the **sslstash** command:

```
# /opt/IBMIHS/bin/sslstash -c /opt/IBMIHS/security/pkcs11.passwd crypto 1234567
```

   In the example, 1234567 is the user PIN for the token (the value assigned when the PKCS#11 is configured as described in “Configuring a PKCS#11 device” on page 12).

5. Define a virtual host for the SSL connections.

   VirtualHost directives can be inserted at the end of section 3 in configuration file:

```
#</VirtualHost>
<IfDefine SSL>
  Listen 443
  <VirtualHost 0.0.0.0:443>
    SSLEnable
    SSLClientAuth none
    SSLServerCert PCICA0:ntc-keyA
    SSLStashfile /opt/IBMIHS/security/pkcs11.passwd
    SSLPKCSDriver /usr/lib/pkcs11/PKCS11_API.so
  </VirtualHost>
  SSLDisable
  Keyfile /opt/IBMIHS/security/ntckey.kdb
</IfDefine>
<IfDefine SSL>
  Addtype application/x-x509-ca-cert .crt
  Addtype application/x-pkcs7-crl    .crl
</IfDefine>
```

   Important configuration directives values include:

   – SSLServerCert

     Use the certificate label defined in “Create the server certificate” on page 18.

– SSLStashfile

  Use the previously created stash file name.

– SSLPKCSDriver

  Specify the PKCS#11 API module (/usr/lib/pkcs11/PKCS11_API.so).

– Keyfile

  Use the secondary key database file name defined when creating the server certificate.

Start the IHS server

The server can be started in SSL mode. Use the **apachectl** command with the startssl option:

```
# /opt/IBMIHS/bin/apachectl startssl
```

The team that wrote this Redpaper

This Redpaper was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Jack Hoarau is an IT Specialist at the New Technology Center of the PSSC in Montpellier. He is responsible for Linux on IBM System z™ for presale technical support, workshops, wildfire "accelerate and grow" sessions on Linux on IBM System z in Europe, benchmark and POC support, and advance support. He joined IBM in Montpellier in 1970 in manufacturing, then moved to application development in networking and CIM, and participated in field support and international assignments as a systems specialist.

Yann Kindelberger is an IT Architect at the ATS PSSC IBM System z New Technology Center, Montpellier, France. He holds a degree in IT engineering and has five years of experience in the IBM System z environment. He has worked on several Linux on IBM System z projects.

Notices

This document created or updated on February 21, 2006.

Send us your comments in one of the following ways:

- Use the online **Contact us** review redbook form found at:
  ibm.com/redbooks
- Send your comments in an email to:
  redbook@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYJ Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks