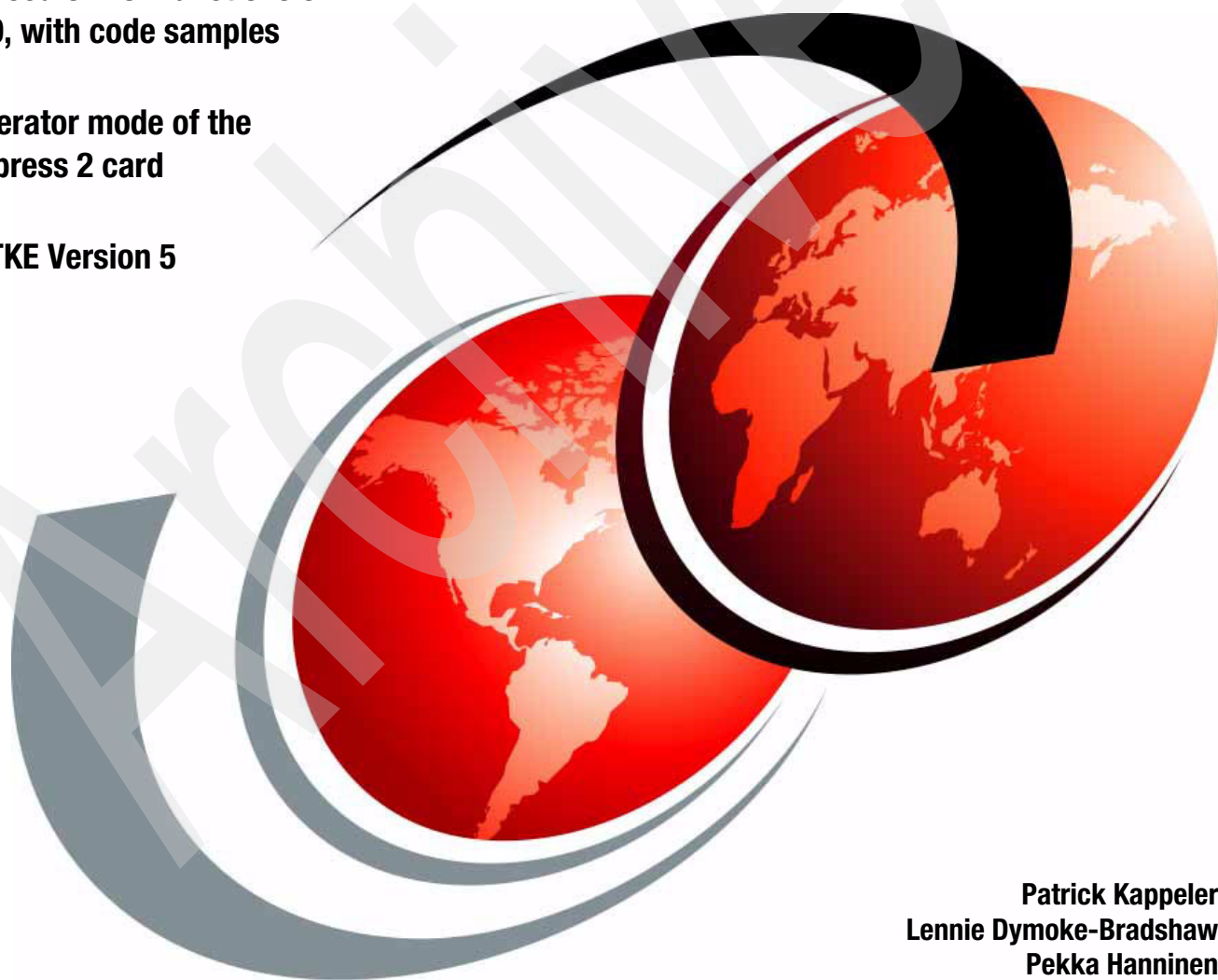


# z9-109 Crypto and TKE V5 Update

The enhanced CPACF functions of  
System z9, with code samples

The Accelerator mode of the  
Crypto Express 2 card

The new TKE Version 5



Patrick Kappeler  
Lennie Dymoke-Bradshaw  
Pekka Hanninen

# Redbooks





International Technical Support Organization

**z9-109 Crypto and TKE V5 Update**

December 2005

Archived

**Note:** Before using this information and the product it supports, read the information in “Notices” on page vii.

## **First Edition (December 2005)**

This edition applies to Version 5, Release 0 of the TKE Workstation and to FMID HCR7730 of the z/OS Integrated Cryptographic Service Facility (ICSF).

© Copyright International Business Machines Corporation 2005. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Notices</b> .....	vii
Trademarks .....	viii
<b>Preface</b> .....	ix
The team that wrote this redbook. ....	ix
Become a published author .....	x
Comments welcome. ....	x
<b>Chapter 1. Introduction and overview</b> .....	1
1.1 Cryptographic function support in System z9 .....	2
1.2 Overview of the cryptographic processors .....	2
1.2.1 CP Assist for Cryptographic Function (CPACF) .....	2
1.2.2 Crypto Express 2 Coprocessor (CEX2C) .....	3
1.2.3 Crypto Express 2 Accelerator (CEX2A) reconfiguration .....	4
1.2.4 Configuration data. ....	5
1.2.5 z990 cryptographic feature codes. ....	6
1.2.6 TKE workstation feature .....	6
1.3 Cryptographic features comparison .....	7
1.4 Software requirements .....	8
<b>Chapter 2. CPACF enhancements in System z9</b> .....	13
2.1 CPACF hardware implementation. ....	14
2.1.1 Confirmation that CPACF is available in the system. ....	14
2.2 Invocation of the new CPACF functions .....	15
2.2.1 What the Message-Security Assist instructions do .....	17
2.3 Calling the CPACF via ICSF services .....	18
2.4 The implications of using clear keys .....	19
2.4.1 What about RACF protection? .....	20
2.5 Some facts about AES .....	20
2.5.1 Who developed AES? .....	20
2.5.2 Why do we need AES? .....	20
2.6 What is SHA-256? .....	21
2.7 Logical partitioning considerations .....	21
2.8 Performance reporting .....	21
2.9 Testing the new CPACF functions .....	22
2.9.1 Encryption and Decryption using KMC-AES-128 .....	22
2.9.2 Encryption and Decryption using ICSF .....	22
2.9.3 Generation of an SHA-256 hash value using KLMD. ....	22
2.9.4 Generation of an SHA-256 hash value using ICSF. ....	22
<b>Chapter 3. The Crypto Express 2 Coprocessor</b> .....	25
3.1 Overview of the Crypto Express 2 Coprocessor .....	26
3.1.1 The coprocessor hardware implementation .....	26
3.1.2 Crypto Express 2 cryptographic functions and coprocessor software layers ....	29
3.1.3 Physical status of the Crypto Express 2 feature .....	30
3.2 Reconfiguration of the coprocessor to accelerator .....	32
3.3 Logical partitioning considerations for System z9 .....	39
3.4 Crypto Express 2 performance .....	42

<b>Chapter 4. ICSF overview, support for CEX2A and sysplex</b>	45
4.1 ICSF releases	46
4.2 Highlights of ICSF HCR7730	48
4.3 System z9 and Crypto hardware support	48
4.3.1 CEX2A support	49
4.4 Enhanced key management for clear DES and AES keys	49
4.5 Sysplex support	52
4.5.1 Sysplex support prior to ICSF HCR7730	52
4.5.2 Sysplex support with ICSF HCR7730	54
4.5.3 How the new CKDS sysplex sharing works	55
4.5.4 How we tested the HCR7730 new sysplex support	57
4.5.5 Updates to the CKDS using KGUP	58
4.5.6 Messages during ICSF startup and shutdown	59
4.5.7 Multiple CKDS data sets in the sysplex	60
4.5.8 Options for sharing the CKDS data set	61
4.5.9 Changing the Master Key in a sysplex	62
4.5.10 Managing the PKDS data set	62
4.5.11 Other sysplex support changes	63
<b>Chapter 5. User Defined Extensions (UDX)</b>	67
5.1 Refresher on the UDX implementation	68
5.1.1 The UDX callable service and the stub	69
5.2 The UDX on System z9	71
5.3 Initial load and activation of the UDX	71
5.3.1 Installation of the UDX	71
5.3.2 UDX activation	72
5.4 UDX microcode update process	73
<b>Chapter 6. TKE V5.0 overview and setup</b>	75
6.1 About the TKE workstation	76
6.2 TKE V5.0 overview	77
6.2.1 TKE V5.0 hardware	77
6.2.2 TKE software levels	78
6.2.3 TKE V5.0 installation	79
6.2.4 TKE V5.0 use	79
6.2.5 Migrating from previous TKE versions	79
6.3 TKE V5.0 functions compared to TKE V4.2	82
6.3.1 Navigation	82
6.4 TKE V5.0 installation and setup	87
6.4.1 Setting TKE workstation time	88
6.4.2 Cryptographic Adapter initialization	89
6.4.3 Cryptographic Node Management and Smart Card Utility Program	91
6.4.4 TCP/IP setup	91
6.4.5 3270 emulator configuration	95
6.5 TKE V5.0 management	96
6.5.1 TKE V5.0 application	96
6.5.2 TKE Media management	98
6.5.3 Backing up critical console data and customizing scheduled operations	101
6.5.4 Shutdown or Restart	102
<b>Appendix A. CPACF programs</b>	103
CPACF010 program	104
REXCP010 program	107
REXCP011 program	108

CPACF020 program .....	110
REXCP020 program .....	114
REXBOWH program .....	115
REXBSYE program .....	116
REXBSYED program .....	118
<b>Appendix B. Programs used in sysplex testing</b> .....	123
REXBKRC program .....	124
REXBKRD program .....	128
REXBKRR program .....	130
<b>Related publications</b> .....	133
IBM Redbooks .....	133
Other publications .....	133
Online resources .....	133
How to get IBM Redbooks .....	134
Help from IBM .....	134
<b>Index</b> .....	135

Archived



# Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.*

*The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:* INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

## **COPYRIGHT LICENSE:**


This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

## Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

@server®  
@server®  
eServer™  
xSeries®  
z/Architecture™  
z/OS®  
z/VM®  
z/VSE™

zSeries®  
z9™  
CICS®  
IBM®  
OS/2®  
OS/390®  
PowerPC®  
Redbooks™

Redbooks (logo) ™  
RACF®  
RMF™  
S/390®  
System z9™  
VSE/ESA™

The following terms are trademarks of other companies:

Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

# Preface

This IBM® Redbook provides detailed information on the implementation of hardware cryptography in the new System z9™, along with the new version of the Trusted Key Entry (TKE) workstation that is required when a TKE is to manage System z9 cryptographic coprocessors. It also addresses the CKDS sysplex support delivered in ICSF HCR7730, which is not dependent on the use of a System z9.

It is expected that the reader is familiar with zSeries® hardware cryptography implementation and the purpose and usage of the TKE workstation.

Other Redbooks™ that may help provide necessary background information are:

- ▶ *Exploiting S/390 Hardware Cryptography with Trusted Key Entry*, SG24-5455
- ▶ *S/390 Crypto PCI Implementation Guide*, SG24-5942
- ▶ *zSeries Crypto Guide Update*, SG24-6870
- ▶ *IBM @server zSeries 990 (z990) Cryptography Implementation*, SG24-7070

## The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, in the Montpellier Products and Solutions Support Center (PSSC).

**Patrick Kappeler** led this redbook project. He joined IBM in 1970 as a diagnostic program designer. He has held several specialist and management positions as well as international assignments, all dealing with S/390® and zSeries Technical Support. He has been part of the EMEA Products and Solutions Support Center in Montpellier (France) since 1996, where his area of expertise is e-business Security on zSeries. He extensively writes and presents on this topic.

**Lennie Dymoke-Bradshaw** has been an IT Specialist at IBM for nine years. He has had a 30- year career in mainframe computing after starting as a PL/1 applications programmer in 1975. He currently works in Integrated Technology Services delivering zSeries services and consultancy. During his career he has maintained an interest in programming, but he also has expertise in RACF® and JES3. In recent years he has become involved in cryptography, as the demand for cryptography services is rising.

**Pekka Hanninen** is an IT specialist working with the Integrated Technology Services team in Finland. He has 30 years of experience in IBM Large Systems software. He has worked at IBM for nine years, and his areas of expertise include RACF, cryptography, and security administration. He holds certificates for CISSP, CISA, and CISM.

Thanks to the following people for their contributions to this project:

Bill White  
Chris Rayns  
International Technical Support Organization, Poughkeepsie Center

Jean-Jacques Noguera  
Alain Dalmier  
Philippe Cochy  
IBM European Products and Solutions Support Center (PSSC) in Montpellier

Jessica Bonner  
Peggy Enichen  
Steven Hart  
Kenneth Kerr  
Donyelle Mahler  
IBM Systems & Technology Group, Development Crypto Test

## Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

[ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

[ibm.com/redbooks](http://ibm.com/redbooks)

- Send your comments in an email to:

[redbook@us.ibm.com](mailto: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

# Introduction and overview

System z9 implements enhanced functions of the cryptographic facilities already available in z990 and z890, that is, the Crypto Express 2 (CEX2C) feature and the CP Assist Cryptographic Facility (CPACF). This chapter introduces these facilities and their exploitation environment on System z9, and provides miscellaneous general information on the hardware cryptographic services available on System z9.

Note that the z990 CPACF and the PCIXCC card are described in *IBM @server zSeries 990 (z990) Cryptography Implementation*, SG24-7070.

As for the previous system implementations, the Integrated Cryptographic Service Facility (ICSF) component of z/OS® provides the high-level API for invoking the hardware cryptographic services.

## 1.1 Cryptographic function support in System z9

System z9 includes both standard cryptographic hardware and optional cryptographic features for flexibility and growth capability. IBM has a long history of providing hardware cryptographic solutions, from the development of Data Encryption Standard (DES) in the 1970s to delivering integrated cryptographic hardware in a server to achieve the US Government's highest FIPS 140-2 Level 4 rating for secure cryptographic hardware. Information on the FIPS 140-2 standard can be found at:

<http://csrc.nist.gov/cryptval/140-2.htm>

The System z9 cryptographic functions include the full range of cryptographic operations needed for e-business, e-commerce, and financial institution applications. In addition, custom cryptographic functions can be added to the set of functions that System z9 offers.

## 1.2 Overview of the cryptographic processors

Two types of cryptographic hardware features are available on System z9. These features are usable only when explicitly enabled through Feature Code 3863, except for the CPACF SHA-1 and SHA-256 functions, which are always enabled.

### 1.2.1 CP Assist for Cryptographic Function (CPACF)

Each system CP (Central Processor) has an assist processor on the chip in support of cryptography. The CP Assist for Cryptographic Function offers a set of symmetric cryptographic functions that enhance the encryption and decryption performance of clear key operations for SSL, VPN, and data storing applications that do not require FIPS 140-2 level 4 security. The cryptographic architecture includes DES, T-DES, AES data encryption and decryption, MAC message authentication, and SHA-1 and SHA-256 hashing. These functions are directly available to application programs, because they are provided as problem state z/Architecture™ instructions, reducing programming overhead. Alternatively, these functions can also be called through the Integrated Cryptographic Service Facility (ICSF) component of z/OS by an ICSF-aware application.

The following five problem-state instructions were introduced with the cryptographic assist function of the z/990 and z/890:

- ▶ **KMAC: Compute Message Authentic Code**
- ▶ **KM: Cipher Message**  
The KM instruction has been extended in System z9 to support the AES-128 encryption/decryption function.
- ▶ **KMC: Cipher message with chaining**  
The KMC instruction has been extended in System z9 to support the AES-128 and PRNG functions.
- ▶ **KIMD: Compute Intermediate Message Digest**  
The KIMD instruction has been extended in System z9 to support the SHA-256 function.
- ▶ **KLMD: Compute Last Message Digest**  
The KLMD instruction has been extended in System z9 to support the SHA-256 function. The CP Assist for Cryptographic Function runs at System z9 processor speed, and since the facility is available on every CP in the system, there are no affinity issues, as in earlier CMOS processors.

Further information on the CPACF is provided in Chapter 2, "CPACF enhancements in System z9" on page 13.

## 1.2.2 Crypto Express 2 Coprocessor (CEX2C)

The optional Crypto Express 2 Coprocessor (CEX2C) comes as a PCI-X (Peripheral Component Interconnect eXtended) pluggable feature that provides a high performance and secure cryptographic environment. The CEX2C Cryptographic Coprocessor consolidates the functions previously offered on the z900 by the Cryptographic Coprocessor feature (CCF), the PCI Cryptographic Coprocessor (PCICC), and the PCI Cryptographic Accelerator (PCICA) feature. These features are not available on System z9. The CEX2C feature performs the following functions:

- ▶ Data encryption/decryption algorithms
  - Data Encryption Standard (DES)
  - Double length-key DES
  - Triple length- key DES
- ▶ DES key generation and distribution
- ▶ PIN generation, verification, and translation functions
- ▶ Pseudo Random Number (PRN) Generator
- ▶ Public Key Algorithm (PKA) Facility

These commands are intended for application programs using public key algorithms, including:

- Importing RSA public-private key pairs in clear and encrypted forms
- Rivest-Shamir-Adelman (RSA)
  - Key generation, up to 2048-bit
  - Signature Verification, up to 2048-bit
  - Import and export of DES keys under an RSA key, up to 2048-bit
- Public Key Encrypt (CSNDPKE)

Public Key Encrypt service is provided for assisting the SSL/TLS handshake, and when used with the Mod\_Raised\_to Power (MRP) function it can be used to offload compute intensive portions of the Diffie-Hellman protocol onto the CEX2C features of System z9.

- Public Key Decrypt (CSNDPKD)

Public Key Decrypt supports a zero-pad option for clear RSA private keys. PKD is used as an accelerator for raw RSA private operations such as required by the SSL/TLS handshake and digital signature generation. The zero-pad option is exploited on Linux® to allow use of the CEX2C features of System z9 for improved performance of the SSL/TLS handshake and digital signature generation.

- Derived Unique Key Per Transaction (DUKPT)

This service is provided to write applications that implement the DUKPT algorithms as defined by the ANSI X9.24 standard. DUKPT provides additional security for point-of-sale transactions that are standard in the retail industry. DUKPT algorithms are supported on the CEX2C feature for triple-DES with double-length keys.

- Europay Mastercard VISA (EMV) 2000 standard

Applications may be written to comply with the EMV 2000 standard for financial transactions between heterogeneous hardware and software. Support for EMV 2000 applies only to the CEX2C feature of System z9.

Other key functions of CEX2C serve to enhance the security of public/private key encryption processing:

- ▶ Retained key support (RSA private keys generated and kept stored within the secure hardware boundary)
- ▶ Support for 4753 Network Security Processor migration
- ▶ User-Defined Extensions (UDX)

User-Defined Extensions to the Common Cryptographic Architecture (CCA) support custom algorithms that execute within the CEX2C Cryptographic Coprocessor. The UDX customized algorithm is added as specific coprocessor code built by IBM or by an approved third party. Building a UDX is an IBM service offering performed under contract. More information on UDX is given in Chapter 5, “User Defined Extensions (UDX)” on page 67.

### The coprocessors in the CEX2C

The CEX2C feature contains two coprocessors, or “cards”. These are PCIXCC coprocessors, and as such the CEX2C provides equivalent functions to the PCIXCC with double the throughput.

**Attention:** The IBM terminology might be somehow confusing here: a “card” is a coprocessor; what is actually being plugged into the system is a “feature”.

The CEX2C feature is designed for FIPS 140-2 Level 4 compliance rating for secure cryptographic hardware modules. Among the many protective functions required by the standard is that an unauthorized removal of the card or feature “zeroizes” its content to preserve secrecy.

The CEX2C Cryptographic Coprocessor features on System z9 enable the user to do the following, using application keys protected by a Master Key (“secure keys”):

- ▶ Encrypt and decrypt data utilizing secret-key algorithms. Triple-length key DES and double-length key DES algorithms are supported.
- ▶ Generate, install, and distribute cryptographic keys securely using both public and secret-key cryptographic methods.
- ▶ Generate, verify, and translate personal identification numbers (PINs).
- ▶ Ensure the integrity of data by using message authentication codes (MACs), hashing algorithms, and Rivest-Shamir-Adelman (RSA) public key algorithm (PKA) digital signatures.

The security-relevant portion of the cryptographic functions is performed inside the secure physical boundary of a tamper-resistant card. Master keys and other security-relevant information is also maintained inside this secure boundary.

### 1.2.3 Crypto Express 2 Accelerator (CEX2A) reconfiguration

The CEX2A is actually a CEX2C that has been reconfigured by the user to only provide a subset of the CEX2C functions at enhanced speed. This reconfiguration is a manual process performed at the System z9 Support Element and is described in 3.2, “Reconfiguration of the coprocessor to accelerator” on page 32

Note that:

- ▶ The reconfiguration is done at the coprocessor level, that is, a CEX2C feature can host a CEX2C coprocessor and a CEX2A accelerator, or two CEX2C coprocessors or two CEX2A accelerators.



- ▶ The reconfiguration is working both ways, that is, from CEX2C to CEX2A, and from CEX2A to CEX2C. Master keys in the CEX2C domains can be optionally preserved when reconfiguring from CEX2C to CEX2A.
- ▶ The reconfiguration process is disruptive to the involved coprocessor/accelerator operations. The coprocessor/accelerator must be deactivated at ICSF before engaging the manual reconfiguration process.
- ▶ The FIPS 140-2 certification is not relevant to CEX2A because it is operating with clear keys only.
- ▶ The function extension capability via UDX is not available to CEX2A.

Actually, the only functions that remain available when reconfigured into a CEX2A are the former PCICA functions. These functions are used for the acceleration of modular arithmetic operations, that is, the RSA cryptographic operations used with the SSL/TLS protocol:

- ▶ PKA Decrypt (CSNDPKD), with PKCS-1.2 formatting
- ▶ PKA Encrypt (CSNDPKE), with ZERO-PAD formatting
- ▶ Digital Signature Verify

The Encrypt and Decrypt RSA functions support key lengths of 512 to 2048-bit, in the Modulus Exponent (ME) and Chinese Remainder Theorem (CRT) formats.

The maximum number of SSL transactions per second that can be supported on a System z9 by any combination of CPACF and CEX2A coprocessors is limited by the amount of cycles available to perform the software portion of the SSL/TLS transactions. When both PCI-X coprocessors on a Crypto Express2 feature are configured as accelerators, the Crypto Express2 feature is designed to perform up to 6000 SSL handshakes per second. This represents, approximately, a 3X performance improvement compared to z990 when using either a PCI Cryptographic Accelerator (PCICA) feature or the current CEX2C feature.

**Note:** These figures indicate a throughput, that is, it is necessary to initiate several threads of parallel requests to the CEX2A to achieve this performance.

In System z9, there can be a maximum of eight CEX2C features reconfigured as Crypto Express 2 Accelerator (CEX2A).

More information on the Crypto Express 2 feature is given in Chapter 3, “The Crypto Express 2 Coprocessor” on page 25.

## 1.2.4 Configuration data

Table 1-1 summarizes support of partitions on System z9 for CEX2C and CEX2A.

Table 1-1 PCI Cryptography features

	Maximum number of features per System z9 server	Number of cryptographic coprocessors or accelerators per feature	Maximum number of cryptographic coprocessors or accelerators per System z9 server	Number of cryptographic domains per accelerator <sup>a</sup> or coprocessor	Number of logical partitions per System z9 server (Defined/Active)
CEX2C	8	2	16	16	60/60

a. Although an accelerator is not using a cryptographic domain to protect a Master Key, the notion of domain still exists and refers to a request /response queue that the device maintains with a specific logical partition.

## 1.2.5 z990 cryptographic feature codes

What follows is a list of the cryptographic features available with System z9.

Feature code	Description
<b>3863</b>	Crypto enablement CD Prerequisite to use the CPACF (except for SHA-1 and SHA-256) and of the CEX2C/CEX2A hardware features. The Feature is installed once in the system.
<b>0863</b>	Crypto Express 2 Coprocessor (CEX2C) feature - Also re-configurable as a Crypto Express 2 Accelerator (CEX2A)
<b>0859</b>	TKE V5 hardware with Ethernet connection - Up to three features per System z9. TKE workstation hardware with Ethernet connection, DVD drive, and 17-inch flat panel monitor.
<b>0855</b>	TKE 5.0 LIC CD
<b>0887</b>	TKE Smart Card Reader
<b>0888</b>	TKE additional smart cards

**Important:** You must use the TKE 5.0 workstations to control System z9. These may also be used to control z990, z890, z900 and z800 servers. Previous TKE versions cannot be upgraded to TKE V5 hardware.

## 1.2.6 TKE workstation feature

A TKE workstation is part of a customized solution for using the Integrated Cryptographic Service Facility for z/OS program product to manage cryptographic keys of a System z9 that has CEX2C features installed **and intended** for the use of Data Encryption Standard (DES) and Public Key Algorithm (PKA) with secure cryptographic keys.

The TKE workstation provides secure control of the CEX2C features, including loading of master keys.

If one or more logical partitions are customized for using CEX2C cards, the TKE workstation can be used to manage DES master keys and PKA master keys for all cryptographic domains of each CEX2C coprocessor defined to the TKE workstation.

Each logical partition in the same physical system using a domain managed through a TKE workstation connection is either a TKE host or a TKE target. A logical partition with TCP/IP connection to the TKE is referred to as TKE host; all other partitions are TKE targets.

The cryptographic controls set for a logical partition, through the System z9 Support Element, determine whether it can be a TKE host or TKE target.

More information on TKE V5.0 is given in Chapter 6, “TKE V5.0 overview and setup” on page 75.

## 1.3 Cryptographic features comparison

Table 1-2 summarizes the functions and attributes of the cryptographic hardware features on System z9.

Table 1-2 System z9 cryptographic features comparison

Functions or attributes	CPACF	CEX2C	CEX2A
Supports z/OS applications using ICSF	X	X	X
Encryption and decryption using a secret-key algorithm		X	
Provides highest SSL handshake performance			X <sup>(1)</sup>
Provides highest symmetric (clear key) encryption performance	X		
Provides highest asymmetric (clear key) encryption performance			X
Provides highest asymmetric (encrypted key) encryption performance		X	
Disruptive process to enable		(2)	(2)
Requires IOCDS definition			
Uses CHPID numbers			
Is assigned PCHIDs		X <sup>(4)</sup>	X <sup>(4)</sup>
Physically embedded on each Central Processor (CP)	X		
Requires CPACF Enablement FC 3863	X <sup>(3)</sup>	X <sup>(3)</sup>	X <sup>(3)</sup>
Requires ICSF to be active		X	X
Offers user programming function support (UDX)		X	
Usable for data privacy - encryption and decryption processing	X	X	
Usable for data integrity - hashing and message authentication	X	X	
Usable for financial processes and key management operations		X	
Crypto performance RMF™ monitoring		X	X
Requires system master keys to be loaded		X	
System (master) key storage		X	
Retained key storage		X	
Tamper-resistant hardware packaging		X	

Functions or attributes	CPACF	CEX2C	CEX2A
Designed for FIPS 140-2 Level 4 certification		X	
Supports SSL functions	X	X	X
Supports Linux applications doing SSL handshakes			X
RSA functions		X	X
High performance SHA-1, Hash function	X		
Clear key DES/T-DES	X		
AES 128-bit key	X		
Pseudo Random Number generator	X	X	
Clear key RSA		X	X
Double length DUKPT support		X	
Europay Mastercard VISA (EMV) support		X	
Public Key Decrypt (PKD) support for Zero-Pad option for clear RSA private keys)		X	X
Public Key Encrypt (PKE) support for MRP function		X	X

### Notes

1. Requires CPACF enablement. FC 3863
2. In order to make addition of CEX2C/CEX2A features non-disruptive, the logical partition must be predefined with the appropriate PCI-X cryptographic processor number selected in its candidate list in the partition image profile.
3. The CPACF enablement is not required for Linux if only the RSA clear key operations, provided by CEX2C, are being used. Using the CPACF for DES or T-DES encryption requires the CPACF to be enabled, even when invoked from Linux.
4. CEX2C/CEX2A is assigned two PCHIDs per feature (one per coprocessor or accelerator).

## 1.4 Software requirements

The CP Assist for Cryptographic Function (CPACF), Crypto Express 2 Coprocessor (CEX2C) and Crypto Express 2 Accelerator (CEX2A) have specific software requirements.

The Integrated Cryptographic Service Facility (ICSF) is the support program for the cryptographic features CPACF, CEX2C, and CEX2A. ICSF is integrated into z/OS.

A specific chapter is dedicated to the new functions of ICSF; see Chapter 4, “ICSF overview, support for CEX2A and sysplex” on page 45.

### Reminder on ICSF

Figure 1-1 describes the overall hardware and software layout of the hardware cryptography in System z9 and z/OS.

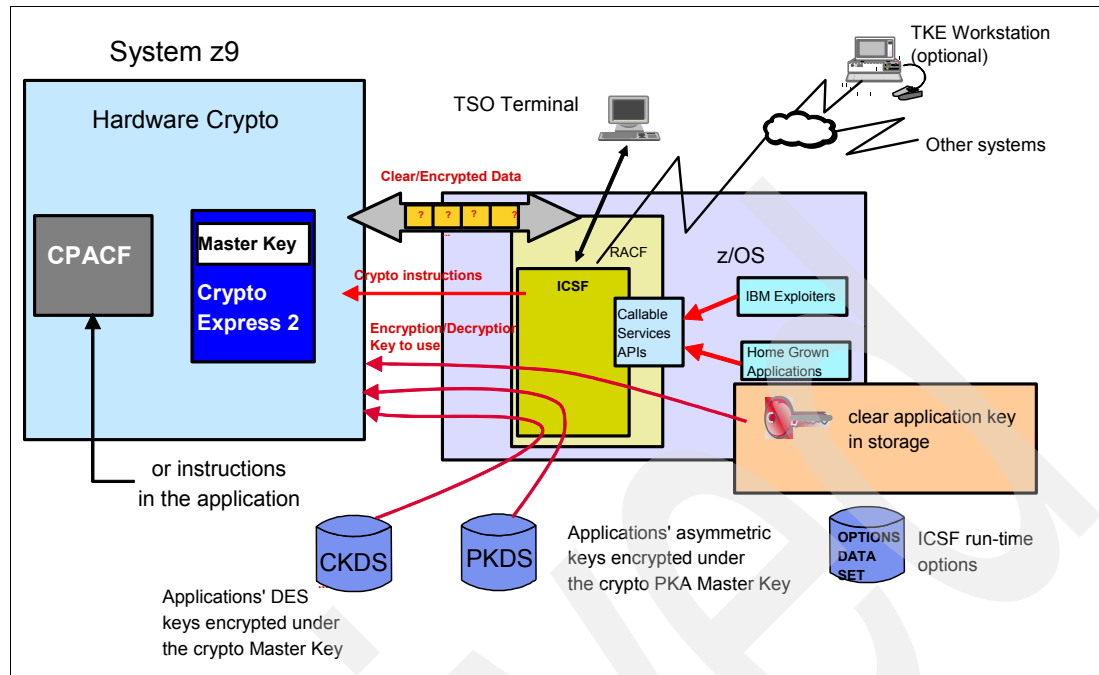


Figure 1-1 Overall hardware and software layout

- ▶ The exploiters of the cryptographic services call the ICSF API. Some functions are performed by the ICSF software without invoking the cryptographic coprocessor; other functions result in ICSF going into routines containing the crypto instructions. The crypto instructions to drive CEX2C are IBM proprietary and are not disclosed; the crypto instructions to interface with CPACF are published in *z/Architecture Principles of Operation*.
- ▶ These instructions are executed by a CPU engine and, if not addressing the CPACF functions, result in a work request being generated for a cryptographic coprocessor.
- ▶ The crypto coprocessor is provided with the following:
  - Data to encrypt or decrypt from the system memory.
  - The key used to encrypt or decrypt provided by ICSF as per the exploiter's request. Note that these keys are represented as sealed envelopes here, the intent being to stress the fact that these encryption/decryption keys are themselves encrypted and, therefore, unusable when residing outside of the crypto coprocessor.
  - Physically, these keys can be stored in ICSF-managed VSAM data sets and pointed to by the application using the label they are stored under. The Cryptographic Key Data Set (CKDS) is used to store the symmetric keys in their encrypted form, and the Public Key Data Set (PKDS) is used to store the asymmetric keys. The application also has the capability of providing an encrypted encryption key or a clear encryption key directly in memory (that is, to use *as is*) to the coprocessor.

For high-speed access to symmetric cryptographic keys, the keys in the CKDS are duplicated into an ICSF-owned data space.

## ICSF releases

Cryptographic support for System z9 for z/OS V1.4 and later is made available as a Web deliverable found at:

<http://www.ibm.com/eserver/zseries/zos/downloads>

The z990 Cryptographic Support was the initial download available to upgrade ICSF to support the z990 CPACF, PCIXCC, and PCICA cryptographic hardware. This download is

not available any more but, if installed, can be upgraded via PTF to support the System z9 CEX2C (excluding CEX2A support). On May 28, 2004, this support was replaced by z990 and z890 Enhancements to Cryptographic Support.

z990 and z890 Enhancements to Cryptographic Support, if already installed, can be upgraded via PTF to support the System z9 CEX2C (excluding CEX2A support). The following ICSF functions were also provided by this Web deliverable:

- ▶ Double-length Derived Unique Key Per Translation (DUKPT) on PCIXCC
- ▶ EMV 2000 Standard on PCIXCC
- ▶ Public Key Decrypt (PKD) enhancements on PCICA and PCIXCC
- ▶ Public Key Encrypt (PKE) enhancements on PCICA and PCIXCC

A new Web deliverable, *Cryptographic Support for z/OS V1R6/R7 and z/OS.e V1R6/R7*, is available since September 2005. It provides full support of the System z9 CEX2A and new CPACF functions, and also supports the sharing of the CKDS in a sysplex configuration. More information on the latest release of ICSF (ICSF HCR7730) is given in Chapter 4, “ICSF overview, support for CEX2A and sysplex” on page 45.

## Minimum software levels

The minimum cryptographic software requirements are:

- ▶ CP Assist for Cryptographic Function (CPACF):
  - Excluding support for PRNG, SHA-256 and AES 128-bit
    - z/OS V1.4 and later with z990 Cryptographic Support, or z990 and z890 Enhancements to Cryptographic Support
    - z/VM® 4.4 and later
    - Linux distributions with the most recent cryptographic libraries, found at:  
<http://www-124.ibm.com/developerworks/projects/libica>
  - With support for PRNG, SHA-256 and AES 128-bit
    - z/OS V1.6 with *Cryptographic Support for z/OS V1R6/R7 and z/OS.e V1R6/R7*.
    - z/VM 4.4 and later.
    - As of the writing of this book, IBM is still working with the Linux Distributions Partners (LDPs) to make these functions available in future Linux distributions. Linux distributions with the most recent cryptographic libraries can be found at:  
<http://www-124.ibm.com/developerworks/projects/libica>
- ▶ Crypto Express 2 Coprocessor (CEX2C)
  - z/OS V1.4 and later with z990 *Cryptographic Support*, or z990 and z890 *Enhancements to Cryptographic Support* with APAR OA09157.  
**Note:** CEX2A is ignored but requires APAR OA11946 to avoid having ICSF abend when a CEX2C has been reconfigured as a CEX2A.
  - z/VM V5.1 for z/OS and Linux guests
    - Including dedicated queue support for secure-key and clear-key cryptographic functions for z/OS guests
    - Including shared queue and dedicated queue support for clear-key cryptographic functions for Linux guests

- Linux for zSeries with SUSE SLES9 (as of the writing of this book). The crypto support is delivered as an Open Source contribution. See:

<http://www10.software.ibm.com/developerworks/opensource/linux390/index.shtml>

- Crypto Express 2 Coprocessor User-Defined Extensions (UDX)
  - z/OS V1.4 and later with *z990 Cryptographic Support*, or *z990 and z890 Enhancements to Cryptographic Support* with APAR OA09157.  
**Note:** CEX2A is ignored but requires APAR OA11946 to avoid an ICSFabend.
- Crypto Express 2 Accelerator (CEX2A)
  - z/OS V1.6 with *Cryptographic Support for z/OS V1R6/R7 and z/OS.e V1R6/R7*
  - z/VM V5.1 for z/OS and Linux guests
  - Linux for zSeries support of the CEX2A is still being worked on as of the writing of this book.

Table 1-3 summarizes the software support requirements by operating system.

Table 1-3 Software requirements to support cryptographic features

Operating system	CPACF	CEX2C	CEX2A
z/OS V1.4 and later with z990 Cryptographic Support or z990 and z890 Enhancements to Cryptographic Support	Y(1)	Y	N (2)
z/OS V1.6 and z/OS V1.6 with Cryptographic Support for z/OS V1R6/R7 and z/OS.e V1R6/R7	Y	Y	Y
z/VM V3.1 and V4.3 and later	Y		
z/VM V4.3 and later for Linux guests	Y		Y
z/VM V5.1 for z/OS and Linux guests	Y	Y	Y
Linux on zSeries	Y(4)	Y(3)	
z/VSE™ V3.1		Y(3)	
VSE/ESA™ V2.7 and later		Y(3)	

1. PRNG, SHA-256 and AES 128-bit not supported by ICSF
2. Toleration APAR OA11946 required if one or more CEX2A.
3. When used only for clear-key RSA modular exponentiation.
4. No support yet for the PRNG, SHA-256 and AES functions as of the writing of this book.

Archived



## CPACF enhancements in System z9

In this chapter we describe the new CPACF functions supplied with the IBM System z9 109 processor and how they can be used.

We discuss the following:

- ▶ Confirmation that CPACF facilities are available
- ▶ Description of the CPACF functions, highlighting the new functions for IBM System z9 109
- ▶ Discussion of capabilities of the CPACF instruction set
- ▶ Description of clear-key processing
- ▶ Description of the AES algorithm
- ▶ Description of SHA-256
- ▶ Considerations using CPACF with regard to LPAR

We also include details of programs we used to test some of the new CPACF functions. The source code is provided in Appendix A, “CPACF programs” on page 103.

## 2.1 CPACF hardware implementation

CP Assist for Cryptographic Functions (CPACF) was first introduced on the z990 and z890. CPACF provides for hash functions and clear key encryption and decryption functions. The concept of clear key encryption is explained in 2.4, “The implications of using clear keys” on page 19.

CPACF operates with a specific set of machine instructions, the Message-Security Assist (MSA) instructions, which are problem state instructions and therefore available to all applications. The MSA instructions are described in *z/Architecture Principles of Operation*, SA22-7832.

The MSA instructions are all executed synchronously with respect to the CP instruction stream, contrary to the operations executed on the Crypto Express 2 cards, which execute asynchronously. The CPACF operations are therefore quite fast and can be used to support a high volume of cryptographic requests. Because the CPACF instructions are available on every PU within System z9, as they are for the zSeries z990 or z890, and since the CPACF operates with clear keys only, there is no notion of logical partition sharing or cryptographic domains with CPACF.

CPACF has been enhanced in the IBM System z9 109 processor to provide these new functions:

- ▶ SHA-256 hashing
- ▶ Clear key AES encryption and decryption hardware support (128-bit clear key only)
- ▶ Pseudo random number generation

### 2.1.1 Confirmation that CPACF is available in the system

The installation of the CPACF enablement, Feature Code 3863, is required to use CPACF in the PUs, except for SHA-1 and SHA-256. The SHA-1 and SHA-256 algorithms are always enabled to applications even if Feature Code 3863 is not installed.

Using an HMC or the IBM System z9 Support Element, you can verify that the processor has the feature installed, as follows:

1. From the Views area, open Groups and CPC.
2. Select the CPC icon in the CPC Work Area view and double-click to open the CPC details window as shown in Figure 2-1.
3. In the window, verify that the CPACF enablement feature Feature Code 3863 is installed.

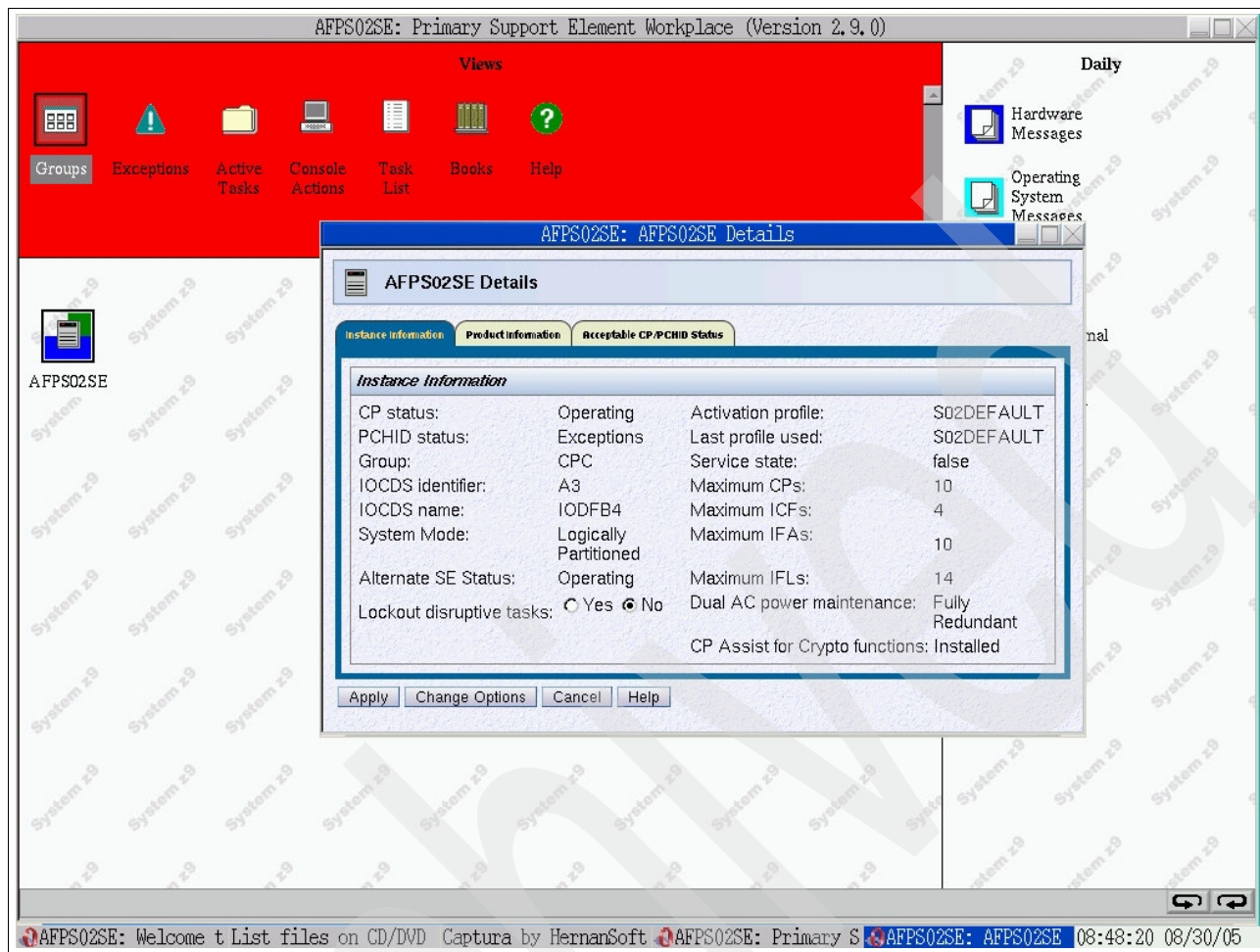


Figure 2-1 Processor status with Feature Code 3863 installed

## 2.2 Invocation of the new CPACF functions

The new CPACF functions are not invoked with new instructions but by new options for existing instructions. There are five instructions in the MSA set, shown in Table 2-1, highlighting the new facilities available in the IBM System z9 109. Each line in the table shows a different CPACF function.

Table 2-1 Message-Security Assist and CPACF functions

Instruction	Description	Features	New for z9
KM - Query	Cipher Message	Query available functions.	<b>Updated results</b>
KM - DEA	Cipher Message	Provide Electronic Code Book (ECB) encryption and decryption of data using a DEA algorithm with single length keys.	
KM - TDEA-128	Cipher Message	Provide ECB encryption and decryption of data using a DEA algorithm with double length keys.	
KM - TDEA-192	Cipher Message	Provide ECB encryption and decryption of data using a DEA algorithm with triple length keys.	

<b>Instruction</b>	<b>Description</b>	<b>Features</b>	<b>New for z9</b>
KM - AES-128	Cipher Message	Provide ECB encryption and decryption of data using an AES algorithm with 16-byte keys.	<b>Yes</b>
KMC - Query	Cipher Message with Chaining	Query available functions.	<b>Updated results</b>
KMC - DEA	Cipher Message with Chaining	Provide Cipher Block Chaining (CBC) encryption and decryption of data using a DEA algorithm with single length keys.	
KMC - TDEA-128	Cipher Message with Chaining	Provide CBC encryption and decryption of data using a DEA algorithm with double length keys.	
KMC - TDEA-192	Cipher Message with Chaining	Provide CBC encryption and decryption of data using a DEA algorithm with triple length keys.	
KMC - AES-128	Cipher Message with Chaining	Provide CBC encryption and decryption of data using an AES algorithm with 16-byte keys.	<b>Yes</b>
KMC - PRNG	Cipher Message with Chaining	Pseudo Random Number Generation.	<b>Yes</b>
KIMD - Query	Compute Intermediate Message Digest	Query available functions.	<b>Updated results</b>
KIMD - SHA-1	Compute Intermediate Message Digest	Provide SHA-1 generation for blocks of data in multiples of 64 bytes.	
KIMD - SHA-256	Compute Intermediate Message Digest	Provide SHA-256 generation for blocks of data in multiples of 64 bytes.	<b>Yes</b>
KLMD - Query	Compute Last Message Digest	Query available functions.	<b>Updated results</b>
KLMD - SHA-1	Compute Last Message Digest	Provide SHA-1 generation for blocks of data that are not multiples of 64 bytes.	
KLMD - SHA-256	Compute Last Message Digest	Provide SHA-256 generation for blocks of data that are not multiples of 64 bytes.	<b>Yes</b>
KMAC - Query	Compute Message Authentication Code	Query available functions.	
KMAC - DEA	Compute Message Authentication Code	Provide Message Authentication Code (MAC) generation using a DEA algorithm with single length keys.	
KMAC - TDEA-128	Compute Message Authentication Code	Provide MAC generation using a DEA algorithm with double length keys.	
KMAC - TDEA-192	Compute Message Authentication Code	Provide MAC generation using a DEA algorithm with double length keys.	

## 2.2.1 What the Message-Security Assist instructions do

The MSA instructions provide for four types of cryptographic operations. These are,

- ▶ Symmetric Encryption and Decryption (clear key only)
- ▶ Generation of hash values
- ▶ Generation of Message Authentication Codes (MAC)
- ▶ Pseudo Random Number Generation

In addition, each function supplies a query option so that the programmer can determine whether a given function is available on a given processor. Attempted use of a function that is not available will yield a program interruption with interruption code 6 (specification exception). In z/OS this is normally presented as a 0C6 abend.

Let us now examine each of the functions in turn.

### Symmetric Encryption and Decryption

This is one of the most basic functions and involves the use of a key that is used to encrypt a string of bytes using an algorithm. Once encrypted, the string of bytes (or message) can be restored to its former state only by decrypting using the original key and reversing the algorithm. Thus a message could be transmitted between two individuals and as long as the encryption key is known only to those two, the message can be considered secure.

This encryption mechanism is “symmetric” because the same key is used to encrypt and to decrypt.

The byte string may be very long, especially in the case of the 64-bit version of the instruction. The key can be of several different types:

- ▶ DES keys. These can be single length, double length, or triple length.
- ▶ AES keys. This is a new hardware function available with CPACF of the System z9 109 and provides support for keys of length 16 bytes (128 bits) only<sup>1</sup>.

Note that DES is an abbreviation for Data Encryption Standard (DES) and the instructions provide processing equivalence to the Data Encryption Algorithm (DEA), which is used at the heart of DES. Broadly speaking, you may consider these to be equivalent. The same equivalence applies to Triple DES (TDES) and TDEA.

### Generation of hash values

Hash values are generated using a specified and repeatable algorithm. No key is ever involved in the generation process. Hash values are used for various functions, including:

- ▶ Mapping a large number of items to a smaller naming space.
- ▶ Providing a form of check pattern to detect changes to a piece of data. In this case it is sometimes known as a Message Digest.

Due to the method used, and the size of the digest produced, it is theoretically possible that several different data configurations will yield the same Message Digest value. Such a possibility is called a “collision.” It is expected that hash algorithms will keep the risk of yielding collisions as low as possible, and will keep voluntary discovery of data configurations that would result in collisions as difficult as possible.

A commonly used hash algorithm today is Secure Hash Algorithm-1 (SHA-1), which generates a 20-byte hash value.

---

<sup>1</sup> AES defines support for keys of length 16, 24, and 32 bytes.

SHA-256 is an improved algorithm and generates a 32-byte hash value. SHA-256 is considered to generate message digest values that are less likely to yield collisions.

Hashing techniques are frequently used for message integrity checking. A hash value can be generated for a given message and it can then be encrypted using an asymmetric encryption technique. This can then be used to confirm that a message has not been altered en route and also to prove proper origin of the message. This technique is called “digital signature.”

On System z9 an application can request an SHA-1 or SHA-256 hash to be computed by CPACF and then forward the hash to a Crypto Express 2 coprocessor, via ICSF services, so that the hash can be encrypted using the RSA asymmetric algorithm.

### Generation of Message Authentication Codes

Messages can be authenticated using MAC values. Message Authentication Codes or MAC values are generated using as input a block of data and a secret symmetric key. Thus MAC codes can be used to confirm that messages have not been altered during transmission and are from the claimed origin. To do this it is necessary that the sender and the receiver both know the secret key used to produce the MAC. This same key can then be used to verify the MAC. If the message has been changed in any way, then the MAC value will no longer show a match between the one received and the one generated at the reception.

The objective of the MAC is similar to the digital signature. However, it relies on symmetric encryption, which is more efficient than asymmetric encryption, but involves more complex key distribution schemes because the key to be used by both parties is a secret key.

### Pseudo Random Number Generation

Random numbers are very important for cryptographic applications, and it is quite important that the randomness be of good “quality”. The CPACF pseudo random number generator (PRNG) uses a variation of TDEA; the algorithm is described in *z/Architecture Principles of Operation*.

## 2.3 Calling the CPACF via ICSF services

Some CPACF functions can be called via ICSF services as opposed to calling them with MSA machine instructions in the application.

The following ICSF services invoke the new CPACF hardware functions on System z9, provided that ICSF is at level HCR7730 or above:

- Symmetric Key Encipher (CSNBSYE) and Symmetric Key Decipher (CSNBSYD), with a keyword of AES in the rule array

**Important:** AES is performed by the CPACF hardware only when the key length is 128 bits. Other specified key lengths, 192 or 256 bits, result in the AES algorithm to be performed by the ICSF software.

Running with an ICSF level below HCR7730 results in all AES services being provided by ICSF software only.

Note that clear keys used by the CPACF can reside in the CKDS beginning with ICSF HCR770A with SPE. The clear keys are really kept as clear values in the CKDS, that is, they are not encrypted with the Master Key as for the other key types stored in the CKDS. There is further discussion on clear keys in the section 2.4, “The implications of using clear keys” on page 19.

- One-Way Hash Generate (CSNBOWH), with the following keywords in the rule array: SHA-256.

**Important:** the SHA-256 function is only available with System z9. An attempt to invoke this function on another system yields Return code 12 and Reason code 8 from ICSF.

There is no ICSF service for the following CPACF functions:

- MAC - ICSF is providing the MAC services (MAC Generate, CSNBMGN, or MAC Verify, CSNBMVR) only if at least one Crypto Express 2 Coprocessor is available in the system.
- PRNG - ICSF does not use the CPACF Pseudo Random Number Generator. ICSF is providing the Random Number Generate service (CSNBRNG) only if at least one Crypto Express 2 coprocessor is available in the stem.

**Note:** Invoking the CPACF functions via ICSF obviously adds instruction path length to the execution of the involved cryptographic services. However, application developers can benefit from some additional functions provided to ICSF, like RMF reporting for SHA activities and the capability of using clear keys stored in the CKDS, along with RACF protection of the keys, as discussed below.

## 2.4 The implications of using clear keys

The CPACF instructions are described as being “clear key” encryption instructions. Let us examine what this means.

With the cryptographic coprocessors available on zSeries and System z9, the secret keys used are all encrypted themselves under a Master Key. The Master Key is held in non-volatile memory within the cryptographic module, that is, the CCF, PCICC, PCIXCC, or Crypto Express 2 coprocessor. Such encrypted keys are called “secure keys.”

Operations requiring the use of these secret keys are performed within the crypto coprocessor where the encrypted secret key is decrypted. This is applicable to keys held in the cryptographic data sets (CKDS and PKDS), or kept in *in-storage* key tokens.

Using this mechanism, the clear value of any key is never exposed in the zSeries or System z9 memory or any storage media associated with it.

Clear key operations are different, in that the instructions that invoke them are deliberately pointing at a memory field where the application secret key resides in clear value. This is self-evidently less secure; however, it yields a more efficient implementation of the cryptographic engines circuitry and provides inherently better performance than running the same operations using secure keys.

There is, however, a way to decrease this security exposure where clear keys can be compromised, and this is by invoking the CPACF encryption and decryption functions via ICSF services.

The clear key can be stored in the CKDS (still remaining in clear) and CSNBSYE and CSNBSYD can be invoked with specifying the CKDS key token label. The clear key value is fetched from the CKDS by ICSF and eventually forwarded to the CPACF from the ICSF address space. That is, no application address space gets knowledge of the clear key value.

This approach is further explained in 4.4, “Enhanced key management for clear DES and AES keys” on page 49.

Or the application, prior to invoking the CSNBSYE or CSNBSYD service, can read into its own storage the key token with the Key Record Read (CSNBKRR) service. However, CSNBKRR requires the requesting application to be in supervisor state to read a clear key token, thus providing an additional level of protection. Invoking CSNBKRR without being in supervisor state yields Return code 4 and Reason code 2078 (X'81E').

There is another advantage to using the ICSF facilities for clear keys. IBM System z9 CPACF only provides hardware support for AES keys of length 16 bytes (128 bits). If it is needed to use AES keys of a longer length, such as 24 or 32 byte keys, then ICSF is to transparently provide the function by software.

### 2.4.1 What about RACF protection?

If the CPACF clear key functions are invoked by the application using the MSA instructions, then there is no access control provided to the services and keys, because the CPACF instructions operate in problem program mode and there is no task switching involved.

However, if the clear keys are held in the CKDS and ICSF services are used to invoke CPACF, the keys can then be protected using RACF profiles in the CSFKEYS general resources class.

However, access to the ICSF services CSNBSYE and CSNYSYD is not protected by profiles in the CSFSERV class.

**Important:** Remember that any user who is able to read the CKDS data set will have access to these clear keys, so it is even more important to restrict READ access to the CKDS to properly protect these keys.

## 2.5 Some facts about AES

The Advanced Encryption Standard (AES) is sometimes known as Rijndael. It is a block cipher adopted as an encryption standard by the US government. It is considered the successor to DES and TDES and is expected to be used worldwide. AES was adopted by the National Institute of Standards and Technology (NIST) as US FIPS PUB 197 in November 2001 after a 5-year standardization process.

### 2.5.1 Who developed AES?

AES was developed as successor to DES by two Belgian cryptographers, Joan Daemen and Vincent Rijmen. It was submitted to the AES selection process under the name "Rijndael", which is formed from the names of the inventors. (Rijndael is pronounced "Rhine dahl", a long "i" and a silent "e").

### 2.5.2 Why do we need AES?

Until the development of AES, the Data Encryption Standard was thought to be adequate for the protection of data using symmetric keys. However, the widespread adoption of DES and Triple DES has lead to a great deal of mathematical examination of the DES standard. It has been discovered that once the message size increases, the chance of the code being broken increases (TDES is considered safe for the transmission of messages up to about 32 gigabytes, which has been adequate until recently).

Another consideration is that, with today's technology, one can gather extremely large computing power, which makes the finding of a DES or TDES secret key an achievable



objective within a reasonable amount of time

AES uses a larger “block size” than DES and TDES. While DES uses a block size of 8 bytes (64 bits), AES uses a block size of 16 bytes (128 bits) along with the capability of using longer keys than DES or TDES. This should be acceptable for messages of up to 256 exabytes of data, and the bigger length of the keys delays for quite a few years the possibility of finding the key value using brute force.

## 2.6 What is SHA-256?

The Secure Hash Algorithm (SHA) family is a set of related cryptographic hash functions.

The most commonly used of these functions is SHA-1, which is employed in a large variety of popular security applications and protocols, including TLS, SSL, PGP, SSH, S/MIME, and IPSec.

SHA-1 is usually considered to be superior, collision- wise, to MD5, which is another widely-used hash function. All the SHA algorithms were designed by the National Security Agency (NSA) and have been published as U.S. government standards.

SHA-1 was not the first version of SHA, but is certainly the most widely used. Since its introduction more variants have been proposed, each with slight variations in design. Four of these variants are: SHA-224, SHA-256, SHA-384, and SHA-512. These are sometimes collectively referred to as SHA-2.

Attacks have been found for both the original SHA and the SHA-1 methods, while no attacks have been reported on the SHA-2 variants.

The new CPACF functions include the ability to generate SHA-256 hash codes. The hash code produced by SHA-1 is 20 bytes long, whereas the hash code produced by SHA-256 is 32 bytes long.

## 2.7 Logical partitioning considerations

There are no LPAR sharing considerations with regard to CPACF.

The CPACF functions are available to all logical partitions dispatched on any PU in the system (provided that Feature Code 3863 has been installed if other functions than SHA-1 and HSA-256 are required). Because there is no secret key used by CPACF, the notion of cryptographic domain does not exist with CPACF. Actually, there is no logical partition parameter to be set to allow use of CPACF by the applications running in a logical partition.

## 2.8 Performance reporting

There are no RMF records generated for the use of CPACF instructions, except for SHA-1 and SHA-256 when they are invoked via the CSNBOWH service of ICSF.

Details on the performance of hardware cryptography in zSeries or System z9 can be found at:

<http://www-03.ibm.com/servers/eserver/zseries/security/cryptography.html>

## 2.9 Testing the new CPACF functions

We performed several tests on the CPACF facilities of the new IBM z9 processor. Some of these tests were done using assembler modules that use the new options of the instructions, while others used the ICSF API to invoke the CPACF functions. In some cases we were able to test the results of the two methods to show that they are equal.

We performed the following tests:

1. Encryption and decryption of a data string using a CPACF KMC - AES-128 instruction
2. Encryption and decryption of the same data string using the ICSF interface using key lengths of 16, 24 and 32
3. Generation of an SHA-256 hash value using KLMD
4. Generation of an SHA-256 hash value using the ICSF interface to CPACF

Several assembler and REXX programs were used during this testing; see Appendix A, "CPACF programs" on page 103.

These programs can also be used to invoke some of the older CPACF functions.

### 2.9.1 Encryption and Decryption using KMC-AES-128

AES encryption is new on the IBM System z9 109 processor.

To test AES encryption and decryption we developed an assembler program, CPACF010, which can be invoked using the REXX programs REXCP010 and REXCP011. (The second program performs both encryption and decryption.)

As can be seen in these programs, we used an initial vector of zeros and a 48-byte input data block of zeros. We used a key of X'000102030405060708090A0B0C0D0E0F'.

### 2.9.2 Encryption and Decryption using ICSF

To test this we used the REXX programs REXBSYE and REXBSYED to invoke the ICSF services CSNBSYE and CSNBSYD. The first REXX program performed only encryption, while the second performed decryption and confirmed that we obtained the original data.

REXBSYED was also modified to use key lengths of 24 and 32 bytes to test the ICSF software facilities for AES keys.

The results we obtained from encryption using ICSF AES were consistent with those obtained using the CPACF010 program in the previous test.

### 2.9.3 Generation of an SHA-256 hash value using KLMD

New on the IBM System z9 109 processor is the capability to use the KLMD instruction to generate SHA-256 hash values.

We tested this by developing an assembler program called CPACF020. This program can be invoked from the REXX program REXCP020.

### 2.9.4 Generation of an SHA-256 hash value using ICSF

We also developed a REXX program to call the ICSF service CSNBOWH to perform the same function.

In each case we used an input field of zeros and of length 48 bytes.

The results we obtained from the two methods (programs CPACF020 and REXBOWH) were consistent.

Archived

Archived



## The Crypto Express 2 Coprocessor

This chapter describes the Crypto Express 2 hardware cryptographic coprocessor implementation and how to prepare System z9 for its exploitation.

## 3.1 Overview of the Crypto Express 2 Coprocessor

We now discuss the Crypto Express 2 Coprocessor.

### 3.1.1 The coprocessor hardware implementation

The Crypto Express 2 Coprocessor is replacing the z990/z890 PCIX Cryptographic Coprocessor, which itself was providing a replacement for both the PCICC and the Cryptographic Coprocessor Facility (CCF) of the previous zSeries and 9672 systems. The Crypto Express 2 Coprocessor is also replacing the PCI Cryptographic Accelerator (the PCICA), which is dedicated to accelerating the SSL/TLS protocol handshake.

Each Crypto Express 2 Coprocessor feature contains two PCIXCC cryptographic coprocessor cards. System z9 allows for up to eight Crypto Express 2 Coprocessor features (or sixteen cards) to be installed.

The Crypto Express 2 Coprocessor provides the same functions as the PCIXCC, with, in addition, the capability of being reconfigured as an accelerator, as described in 3.2, “Reconfiguration of the coprocessor to accelerator” on page 32.

As for the PCIXCC, the Crypto Express 2 feature to be enabled in the system requires Feature Code 3863 to be installed.

**Note:** The zSeries PCIXCC cryptographic coprocessor, available first to be plugged into the z990 and z890 systems, became an IBM product by itself, with versions that can be plugged into other eServer™ machines than zSeries. Although we keep referring to it as the “PCIXCC” in this book, the official name is now the IBM 4764-PCIXCC.

There is an excellent description of the PCIXCC technology in an IBM Systems Journal article at:

<http://www.research.ibm.com/journal/sj/>

Once in the IBM Systems Journal site, search the archives for PCIXCC.

#### Cryptographic accelerator

The Crypto Express 2 Coprocessor (CEX2C) can also be configured as a Crypto Express 2 Accelerator (CEX2A), providing only PCICA-equivalent functions with an expected throughput of approximately three times the PCICA throughput. Switching from Crypto Express 2 Coprocessor, the default mode of a coprocessor, to CEX2A mode is a manual process and an exclusive feature of System z9. Note that the Crypto Express 2 Coprocessor features that were running on z990 or z890 can be moved to System z9 and then be manually reconfigured as CEX2A.

#### Crypto Express 2 card sharing by logical partitions

Each CEX2C card has 16 domains that can be allocated to logical partitions. Therefore, a CEX2C feature can provide cryptographic services to up to 32 logical partitions.

See 3.3, “Logical partitioning considerations for System z9” on page 39 for further information on the logical partition cryptographic setup on System z9.

#### A look inside the hardware

The card, once plugged into an I/O cage, is attached to the system by a Self Timed Interface (STI) high-speed interface on the I/O cage board and has no other external interfaces. This is

shown in Figure 3-1. As for the previous zSeries PCI cards, all communications with the coprocessors are routed via message queues in the HSA over the STI cables.

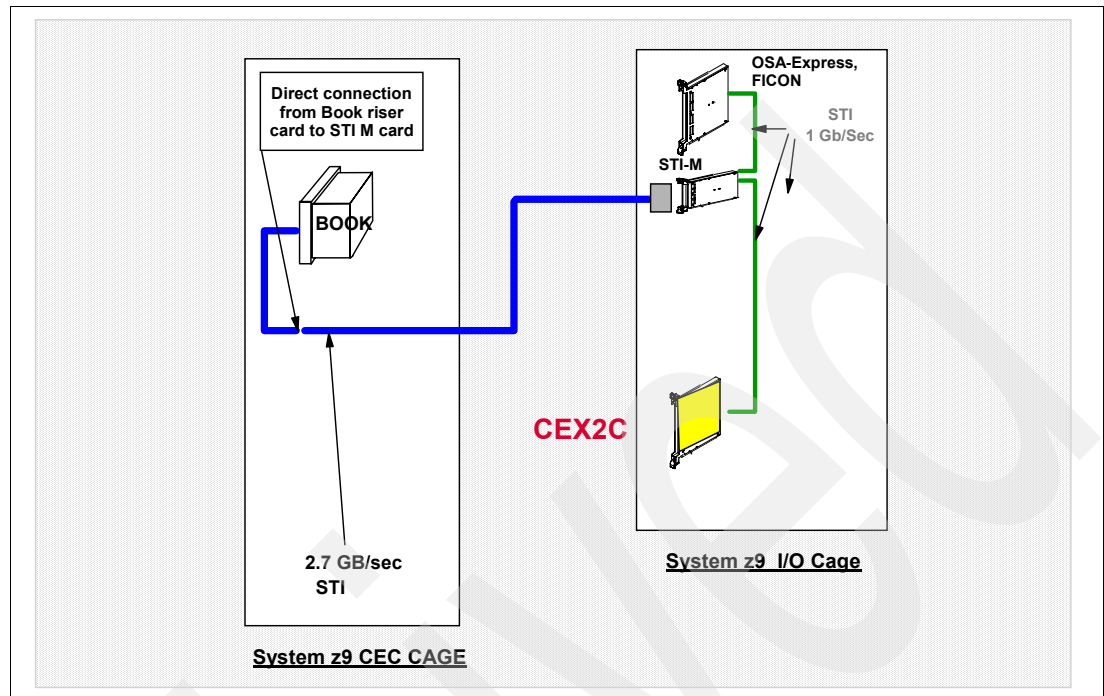


Figure 3-1 CEX2C feature in the System z9 I/O cage

The internal of the Crypto Express 2 feature is shown in Figure 3-2 on page 28.

**Note:** In order to provide maximum availability of the cryptographic services offered by the Crypto Express 2 feature, the IBM configuration and ordering system automatically makes the first Crypto Express 2 order for any given system an order for two features.

# Battery

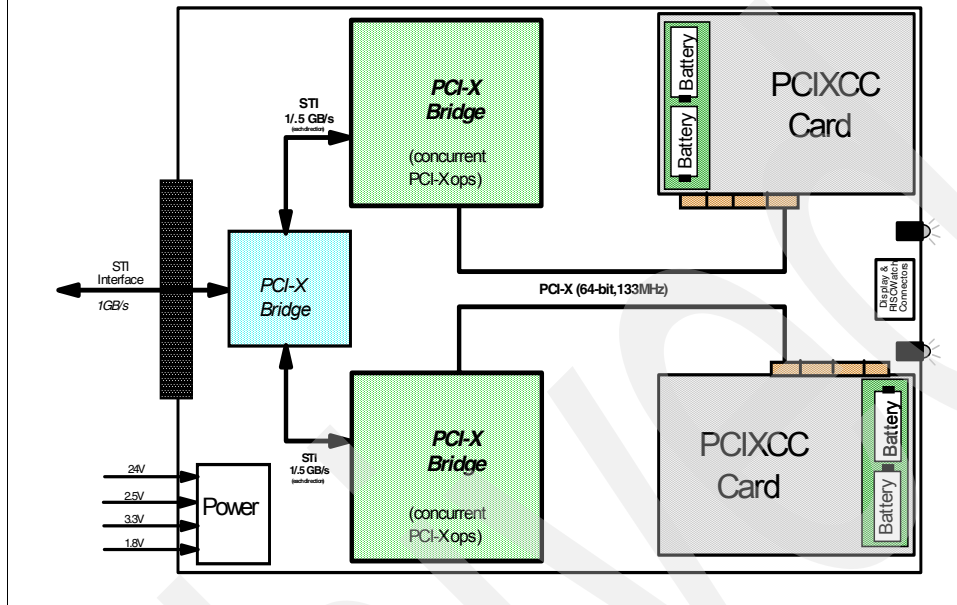


Figure 3-2 Crypto Express 2 feature layout

**Important:** A coprocessor can be reached from any PU in the system, because all communications are routed through message queues in the HSA. However, this implementation dictates that all services provided by the Crypto Express 2 appear asynchronous to the instruction stream in ICSF. That is, ICSF execution resumes after initiating the function at the coprocessor and a check will be made later whether the operation completed at the coprocessor. This obviously has a performance impact, which is further discussed in 3.4, “Crypto Express 2 performance” on page 42.

The following system-specific data is used for coprocessor management:

- ▶ The PCIXCC has an eight-character serial number, for reference in a variety of panels and to keep track of the retained keys. This serial number is visible when the PCIXCC status is online/operating to the logical partition.
- ▶ Each PCIXCC has a two-digit Adjunct Processor (AP) number or ID. This number is an index used by the system LIC and ICSF. The number of APs is limited to 16 on System z9, and a CEX2C is therefore given an AP number between 0 and 15.
- ▶ Because of the way the crypto feature is connected in the system, both crypto coprocessor and accelerator have a PCHID number. This PCHID number assignment is automatic; the coprocessor does not have to be declared in the IOCDS. The PCHID number is not known nor relevant to ICSF, but is used only for hardware management of the coprocessor.



### 3.1.2 Crypto Express 2 cryptographic functions and coprocessor software layers

The PCIXCC coprocessor has a full set of specialized hardware circuits to provide high-speed and secure cryptographic functions. The circuits exploited for the zSeries z990/z890 and System z9 Crypto Express 2 feature are as follows:

- ▶ DES and TDES engine
- ▶ High performance RSA modular arithmetic
- ▶ Random Number Generation
- ▶ Secure storage for:
  - The symmetric and asymmetric Master keys
  - The “retained” RSA private keys
  - The roles and profiles defined to the coprocessor

With the higher level of API implemented via the software layers in the coprocessor itself and ICSF, the hardware-assisted functions available to the applications are as follows:

- ▶ DES and TDES encryption
- ▶ MAC functions that support ANSI standards X9.9 and X9.19
- ▶ TDES-based key management using control vectors
- ▶ RSA key generation
- ▶ RSA-based digital signatures
- ▶ RSA-based key management of DES keys
- ▶ SET for electronic commerce functions
- ▶ PIN processing functions
- ▶ Processing Europay-MasterCard-Visa (EMV) smartcard-based transactions

Other hardware-assisted functions are also available in the IBM 4764 PCIXCC because they are exploited on other platforms besides zSeries and system z9.

#### **Coprocessor software layers**

The structure of the microcode executed in the PCIXCC remains similar to the implementation already seen with the other IBM PCI coprocessors, that is, a layered approach with code “segments.” The segments are digitally signed by IBM and are verified for integrity when they are loaded into the coprocessor. Generally speaking, the code segments are loaded whenever the coprocessor switches from the offline state to being online for the first time to a logical partition.

Note that with the PCIXCC the coprocessor operating system is an open source Linux operating system, which is executed by an embedded PowerPC® 405GPr microprocessor.

The software layer structure is shown in Figure 3-3.

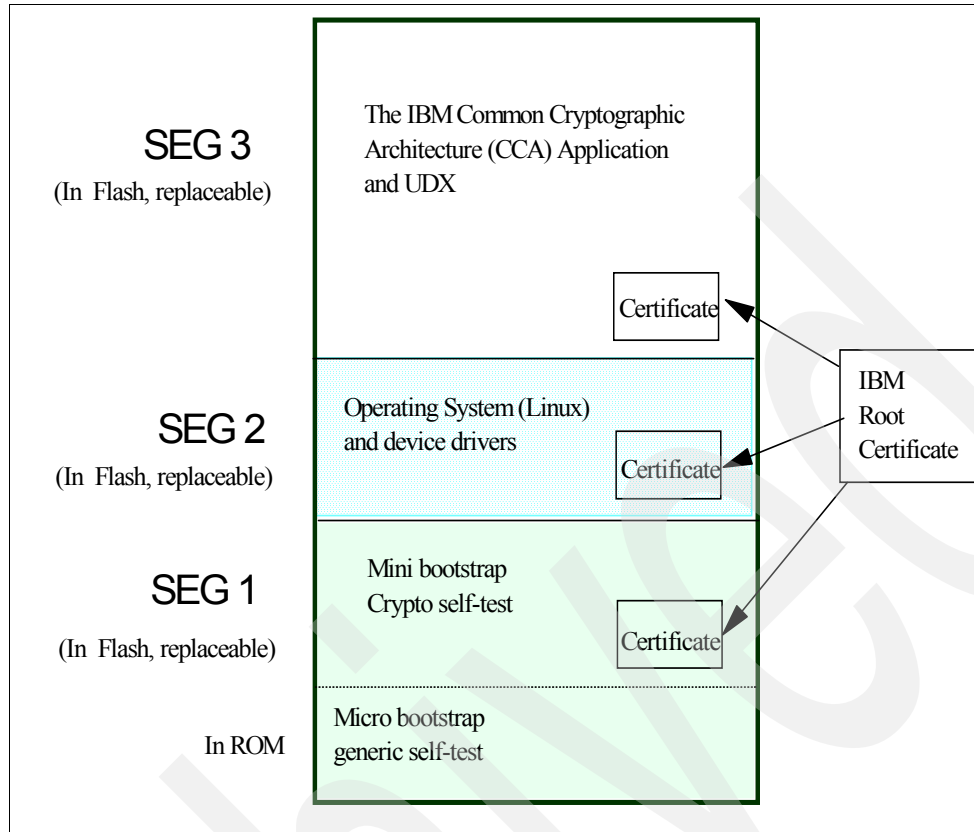


Figure 3-3 Software layers of the PCIXCC

An explanation of the segments shown in Figure 3-3 follows.

- ▶ Segment 0 contains the generic self-test run at the coprocessor power-on and a micro bootstrap, which are stored in read-only memory that is unalterable once the card leaves the factory. The micro bootstrap is the lowest-level software for control of loading software into Segments 1, 2, and 3.
- ▶ Segment 1 contains extensions to the self-test and to the micro bootstrap of Segment 0, but Segment 1 can be securely reloaded after the card has been manufactured. This is a provision for updating the self-test and micro bootstrap functions in the field, if needed.
- ▶ Segment 2 contains the open-source embedded Linux operating system. Special device drivers have been written to allow the operating system and application programs to use the unique hardware in the card.
- ▶ Segment 3 contains the coprocessor program that provides the CCA API functions seen by ICSF. Segment 3 is also hosting the User Defined Extensions (UDXs) that customers may want to implement. UDXs are further discussed in 5.2, “The UDX on System z9” on page 71.

### 3.1.3 Physical status of the Crypto Express 2 feature

The physical status of the Crypto Express 2 feature is indicated by an “A” indicator on the feature, as shown in Figure 3-4. The meaning of the indicator is explained in Table 3-1.

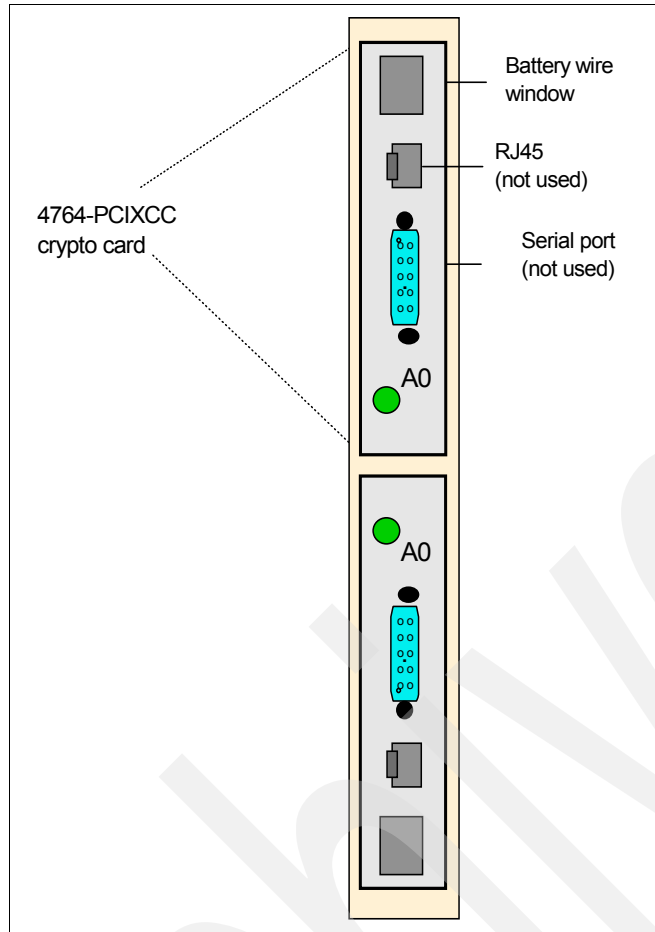


Figure 3-4 Crypto Express 2 physical status

Table 3-1 lists the Crypto Express 2 status and indicator states.

Table 3-1 Crypto Express 2 indicator

Indicator "A" state	Card status
On	Power On Offline Reset Pending Processor Frozen
Off	Non-functional No Power Processor Frozen
Flashing	Ready Online

### Tamper detection and FIPS certification

The PCIXCC card is designed with industry-leading tamper-detection features. The security-related electronic components are wrapped in a flexible mesh with overlapping conductive lines that detect any attempt of physical intrusion by drilling, mechanical abrasion, chemical etching, or other means.

If the conductive lines are damaged, it is sensed by circuits inside the module, and all sensitive data is immediately destroyed by zeroizing the battery-backed memory it is held in.

Other special circuits sense attacks that can cause imprinting in the CMOS memory. Imprinting is a process that can permanently burn data into CMOS circuits, so that the same data appears each time the CMOS chip is powered on and makes the finding of secret values by physical analysis of the chip quite easier. Imprinting can be caused by exposing the chip to either very low temperatures or X-rays. The tamper circuitry detects these conditions and zeroizes the memory before imprinting can occur.

Finally, there are attacks that are driven by manipulating the power-supply voltages to the card. These conditions are also detected to prevent the attacks from succeeding by zeroizing the sensitive data.

The security architecture of the hardware complements the secure code loading design, and the combination of the two provides the features that support FIPS 140-2 Level 4 security.

As of the writing of this book, the IBM 4764-001 has been certified at FIPS 140-2 Level 4.

## 3.2 Reconfiguration of the coprocessor to accelerator

As already mentioned, the CEX2A is actually a Crypto Express 2 Coprocessor card reconfigured as an accelerator. The reconfiguration is fully supported in Licensed Internal Code, and therefore current Crypto Express2 features carried forward from z990 to System z9 may take advantage of the reconfiguration capability.

A Crypto Express 2 Coprocessor feature can therefore be configured as the following:

- ▶ Two Crypto Express 2 Coprocessors, that is, CEX2C only
- ▶ One Crypto Express 2 Coprocessor and one accelerator (one CEX2C and one CEX2A)
- ▶ Two CEX2A Accelerators

As for the Crypto Express 2 Coprocessor card, a CEX2A coprocessor can be shared between 16 logical partitions.

The only cryptographic services that are available when reconfigured into a CEX2A are the former PCICA services. These functions are used for the acceleration of modular arithmetic operations, that is, the RSA cryptographic operations used with the SSL/TLS protocol:

- ▶ PKA Decrypt (CSNDPKD), with PKCS-1.2 formatting
- ▶ PKA Encrypt (CSNDPKE), with ZERO-PAD formatting
- ▶ Digital Signature Verify

The Encrypt and Decrypt RSA functions support key lengths from 512 to 2048-bit, in the Modulus Exponent (ME) and Chinese Remainder Theorem (CRT) formats.

### The reconfiguration process

The CEX2C reconfiguration process is disruptive to the involved coprocessor operations. A reconfiguration is performed as follows:

- ▶ Deactivate the target coprocessors in all logical partitions that are using the coprocessors, using the Coprocessor Management panel of each ICSF instance.
- ▶ Configure the target coprocessors offline at the Support Element or the HMC as follows:
  - Double-click Groups - CPC, then right-click the CPC icon and select **Cryptos**.

- a. Select the target Crypto coprocessors.
  - b. In the Crypto Services Operations view on the right, select the **Configure On/Off** function.
  - c. Select **Toggle All Off** on the Configure Channel Path On/Off panel and select **Apply changes**.
- Configure the target coprocessors to be accelerators or coprocessors in the Cryptographic Configuration panel as follows:
    - a. Double-click Groups- CPC. Select the CPC icon. In the CPC configuration view on the right select **Cryptographic Configuration**.
    - b. Double-click the target Crypto coprocessor and click **Crypto Type Configuration**.
    - c. On the Crypto Type Configuration panel select the desired type of the target Crypto Express 2 card. If you select **Accelerator**, you must also choose whether the target coprocessor will be zeroized when reconfigured as an Accelerator. A confirmation window is displayed if the zeroizing option is selected.

**Caution:** When reconfiguring the coprocessor to an accelerator, zeroizing is selected by default.

- Configure the target coprocessors back to online as follows:
  - Double-click the Groups icon - CPC, right-click the CPC icon and select **Cryptos**.
  - From the Crypto Work Area view, select the target Crypto processors.
  - In the Crypto Services Operations view on the right, select the **Configure On/Off** function.
  - Select **Toggle All On** on the Configure Channel Path On/Off panel and select **Apply changes**. The online configuration process takes a few minutes. The status of the coprocessors can be inspected in the Cryptographic Configuration panel.
- When the coprocessor is back to the online status, it can then be activated in the ICSF Coprocessor Management panel.

In our residency we had two CEX2C features, that is, four coprocessor cards, installed in System z9. They were initialized with Master Keys and enabled for TKE commands. To test the reconfiguration process we first reconfigured three coprocessors to accelerators and then reconfigured two of them back to coprocessors.

First, we deactivated the three coprocessors to be reconfigured in the ICSF Coprocessor Management panel, as shown in Figure 3-5.

```

----- ICSF Coprocessor Management ----- Row 1 to 4 of 4

Select the coprocessors to be processed and press ENTER.
Action characters are: A, D, E, K, R and S. See the help panel for details.

      COPROCESSOR      SERIAL NUMBER      STATUS
      -----
.   E00                95000224          ACTIVE
.   E01                95000225          DEACTIVATED
.   E02                95000182          DEACTIVATED
.   E03                95000180          DEACTIVATED
***** Bottom of data *****

```

Figure 3-5 Deactivate coprocessors to be reconfigured

Then we configured the three processors offline using the Configure Channel Path Off/On panel at the Support Element (or HMC). You can see the configuration status of the coprocessors in the Cryptographic Configuration panel shown in Figure 3-6.

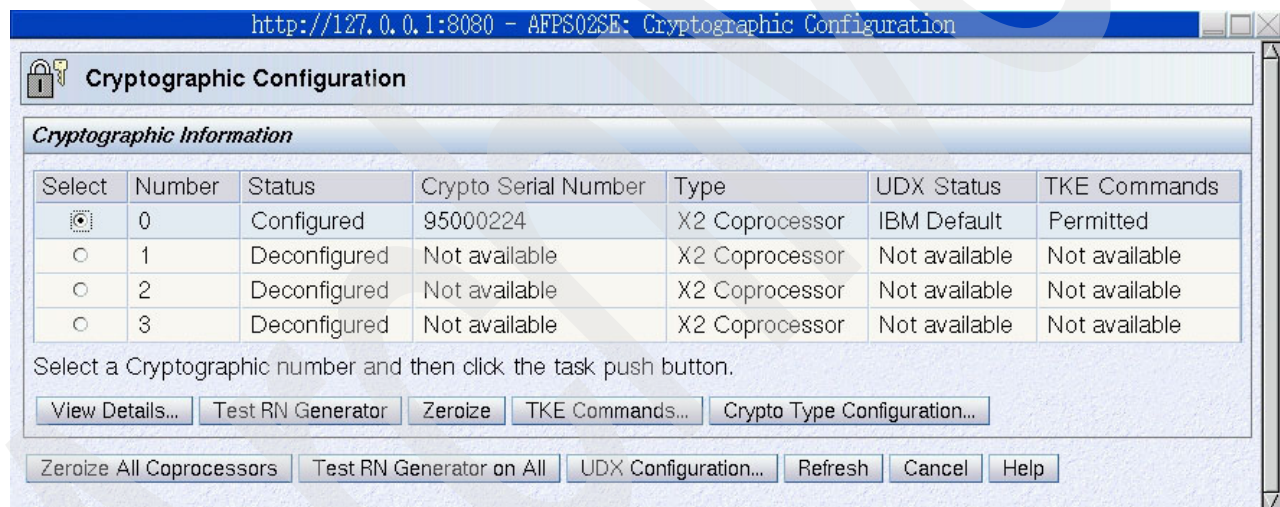


Figure 3-6 The target coprocessors are configured offline

In the Cryptographic Configuration panel we selected the processor card Number 1 and clicked **Crypto Type Configuration**. For this processor we selected the configuration type as **Accelerator** and unmarked the “Zeroize the Coprocessor” option, as shown in Figure 3-7.

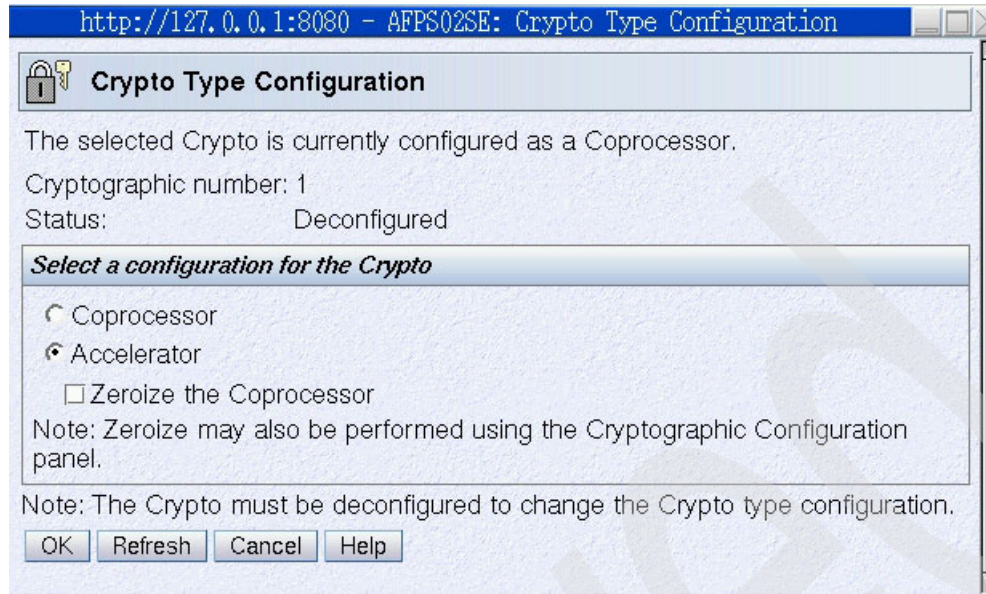


Figure 3-7 Crypto Express 2 Coprocessor re-configured to CEX2A without zeroizing

For the two remaining cards 2 and 3, we chose to zeroize the coprocessor. The zeroize option has to be confirmed in the panel shown in Figure 3-8.

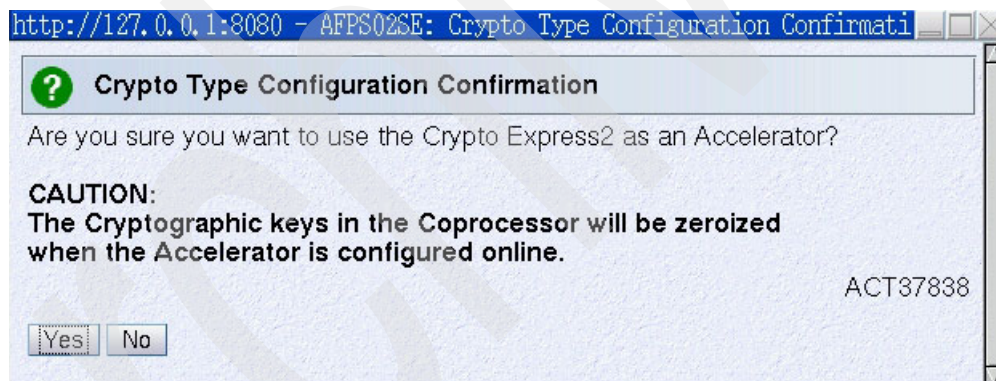


Figure 3-8 Confirming the zeroize option

**Note:** Reconfiguring a Crypto Express 2 Coprocessor to an accelerator with the zeroize option selected suppresses all Master keys in all domains. Also all TKE definitions, such as roles and profiles, are deleted and the TKE Commands enablement is set to be “denied”.

Then we configured the three coprocessors back online using the Configure Channel Path Off/On process. When configured back online, the Cryptographic Configuration panel displayed the status shown in Figure 3-9.



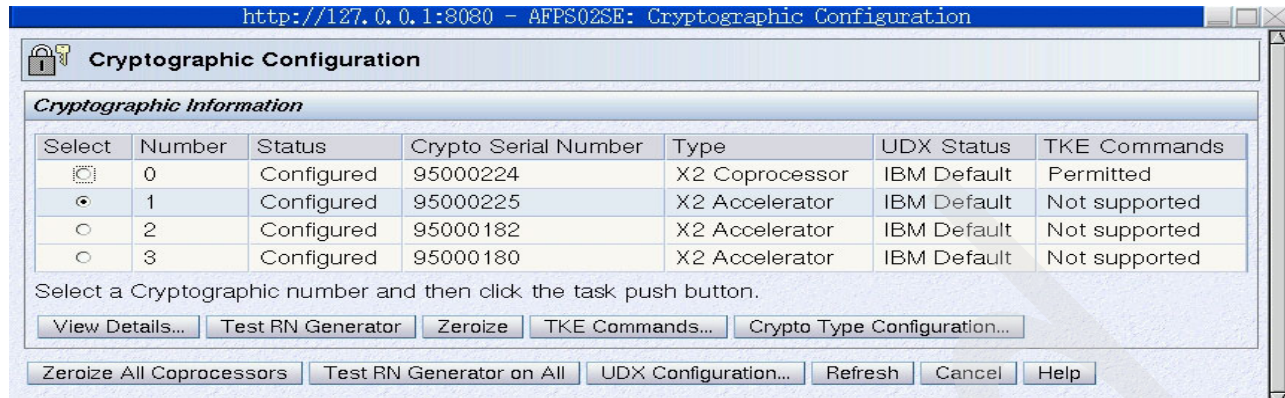


Figure 3-9 Three coprocessors reconfigured to accelerators and configured online

Next we activated the three processors in the ICSF Coprocessor Management panel. In the panel the coprocessors are identified with the letter "E" in the coprocessor column and the accelerators are identified with the letter "F".

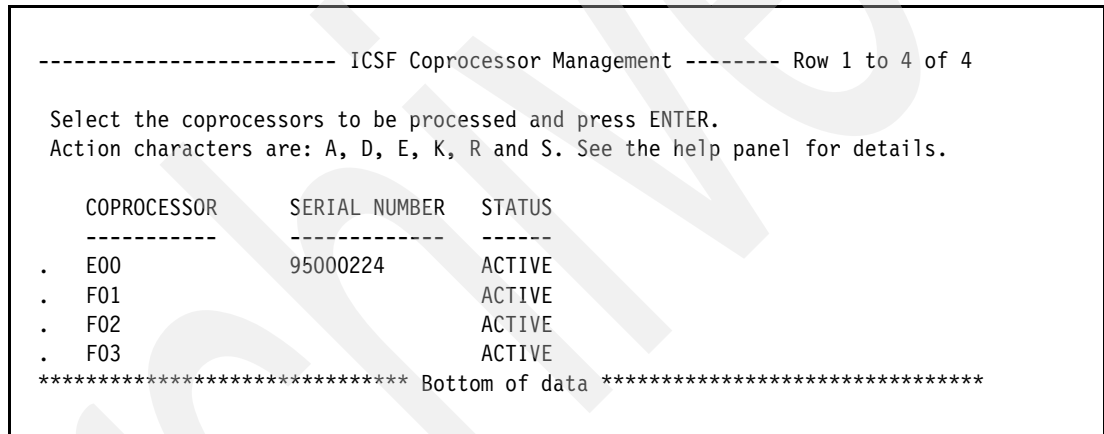


Figure 3-10 Activated accelerators in the ICSF Coprocessor Management panel

**Important:** If the coprocessors that are reconfigured to be accelerators are defined in a TKE group of coprocessors, the group should be changed by removing these accelerators from the TKE group at the TKE workstation.

To further test the reconfiguration process we reconfigured the two accelerators, F01 and F02, back to coprocessors.

First we had to deactivate these accelerators in the ICSF Coprocessor Management panel as shown in Figure 3-11.



```

----- ICSF Coprocessor Management ----- Row 1 to 4 of 4

Select the coprocessors to be processed and press ENTER.
Action characters are: A, D, E, K, R and S. See the help panel for details.

      COPROCESSOR      SERIAL NUMBER      STATUS
      -----
.   E00                95000224          ACTIVE
.   F01                                DEACTIVATED
.   F02                                DEACTIVATED
.   F03                                ACTIVE
***** Bottom of data *****

```

Figure 3-11 Two accelerators to be reconfigured as coprocessors are being deactivated

Then these two accelerators are configured offline to the system with the Configure Channel Path Off/On panel on the Support Element, (or HMC). The Crypto Type configuration panel was again selected for these accelerators where we specified that they had to be reconfigured as coprocessors.

After configuring them back online to the system, we ended up with the Cryptographic Configuration shown in Figure 3-12.

**Important:** When we initially reconfigured Coprocessor 2 to an accelerator, we selected the zeroize option. Coprocessor 2 does not have any more Master keys set and the TKE Commands enablement is set back to the default value, that is, “denied.”

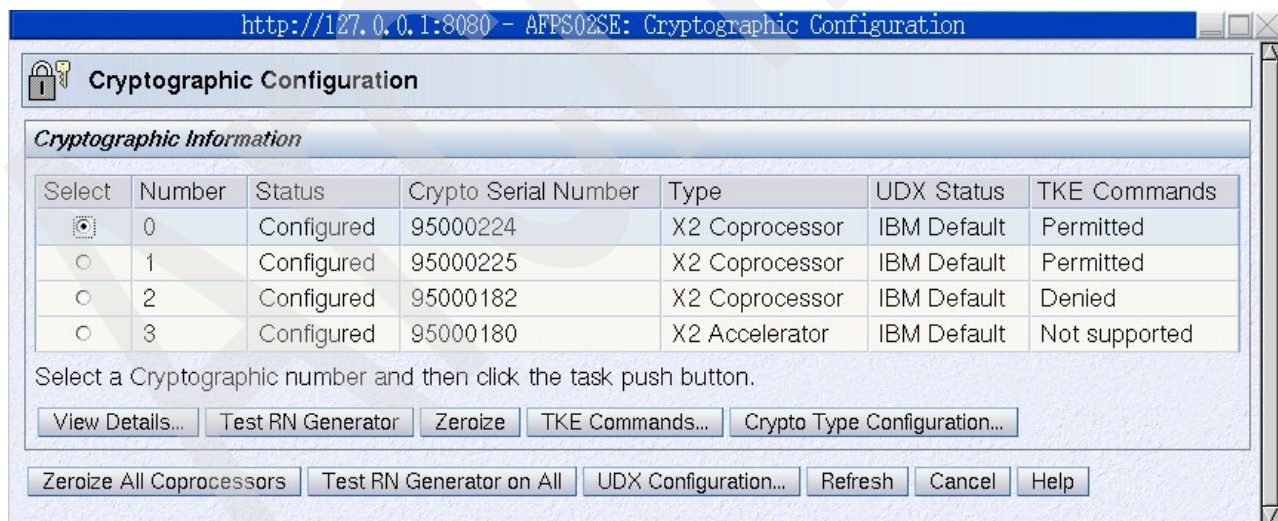


Figure 3-12 Two accelerators reconfigured to coprocessors

To set TKE Commands enablement to the Permit value, we selected Coprocessor number 2 and clicked **TKE Commands**. We then selected **Permit** in the TKE Commands Configuration panel as shown in Figure 3-13.

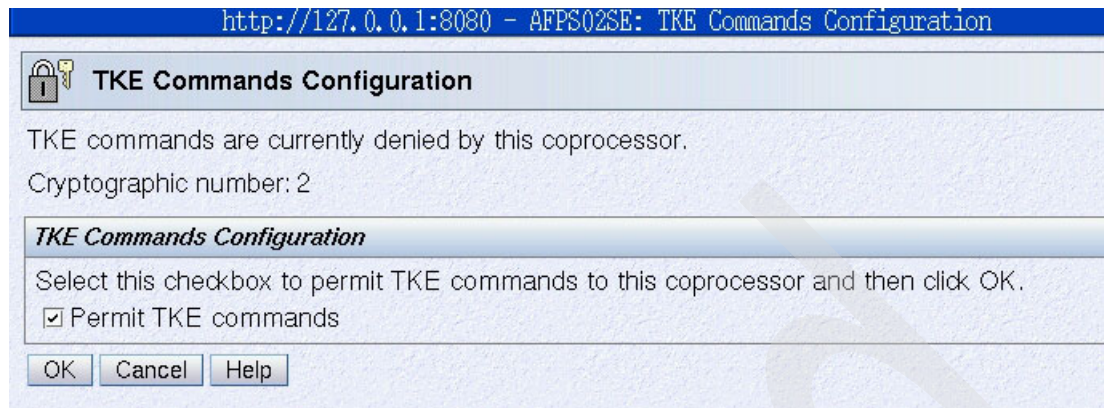


Figure 3-13 Permitting TKE commands to be used with the Coprocessor

After clicking **OK**, the panel shown was displayed and warned about the exposure of having not yet set up proper access control to the TKE workstation.

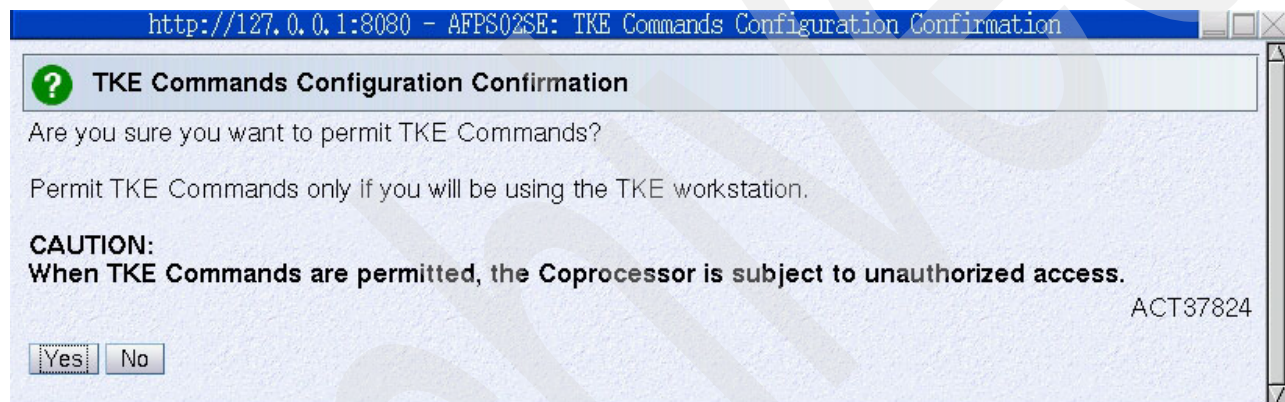


Figure 3-14 Verification of the TKE command permission change

The last step to perform is to activate the two processors in the ICSF Coprocessor Management panel as shown in Figure 3-15. We can also see in this panel that the Coprocessor E02 was zeroized and is for the moment only “online”, waiting for Master keys to be set up.

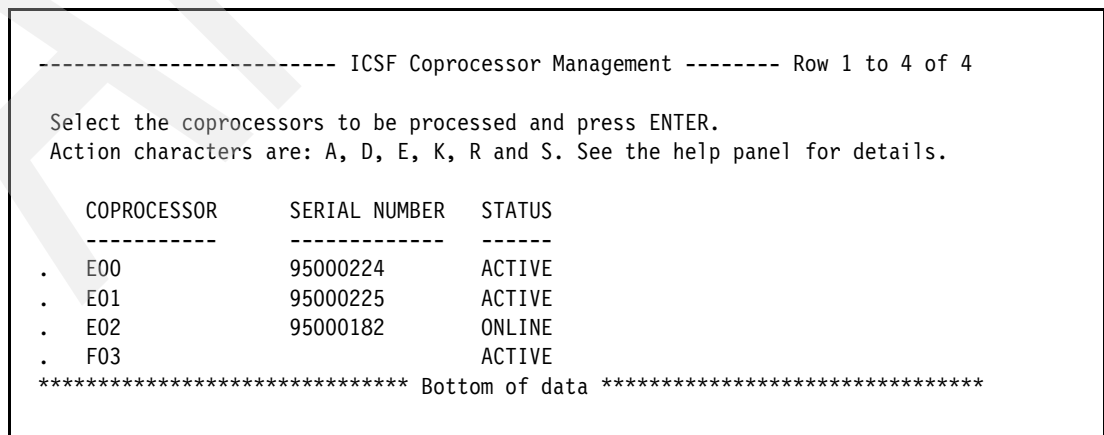


Figure 3-15 Activate Coprocessors after the reconfigure process

### 3.3 Logical partitioning considerations for System z9

Although the cryptographic definitions to be made at the Support Element (or HMC) have not been changed from a functional standpoint, the panel layout is new. We show in this section the panels related to LPAR Cryptographic configuration.

To configure a logical partition for cryptographic functions, do the following:

- ▶ Double-click **Groups-Images**, then select the desired partition.
- ▶ Drag and drop the selected image icon to the right over the “Customize/Delete Activation profile” icon in the CPC Operational customization view.
- ▶ Click **Customize** to get access to the image profile.

The Customize Image Profiles General view is shown in Figure 3-16.

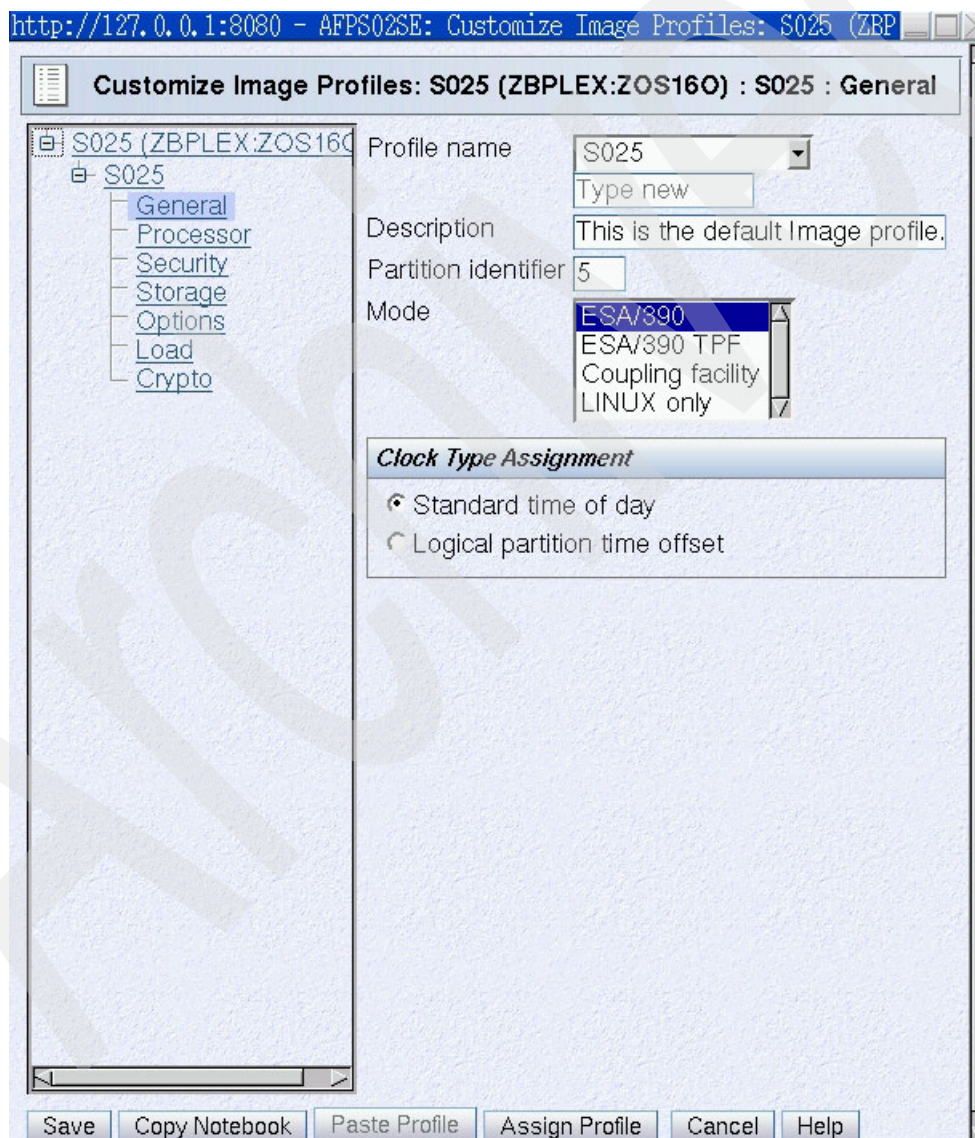


Figure 3-16 Image profile customization main view



To establish the cryptographic definitions for the logical partition, select the Crypto tab, which gives access to the panel in Figure 3-17. The definition parameters on the Crypto panel view did not change. Here is a brief reminder of these parameters:

- ▶ The Usage Domain index

This must point to the domain number or numbers for this logical partition and it should match the domain number that has been set in the ICSF Options data set.

- ▶ The Control Domain index

This identifies the crypto coprocessor domains that can be administered from this logical partition when it is acting as a TKE host.

- ▶ The Cryptographic Candidate list

This identifies the cryptographic coprocessors, or accelerators, that are eligible to be accessed by this logical partition. When a cryptographic coprocessor, or accelerator, is installed and its number has been selected in the partition Cryptographic Candidate list, it stays in the *configured off* state after the partition activation and can then be brought dynamically to the online state via the Support Element (or HMC) to the partition. It can be dynamically configured on to the partition.

- ▶ The Cryptographic Online list

This identifies the cryptographic coprocessor, or accelerator, numbers that are automatically brought online during logical partition activation. The coprocessor numbers that are selected in the Online list must also be part of the Candidate list.

**Notes:**

1. These parameters cannot be modified dynamically. Any change to them makes it necessary to deactivate, then reactivate, the logical partition to be taken into account.
2. Coprocessors specified in the cryptographic Candidate list are not required to be physically installed in the system. The Candidate list can refer to coprocessors that will be installed later, but can then be dynamically varied online when present in the system.

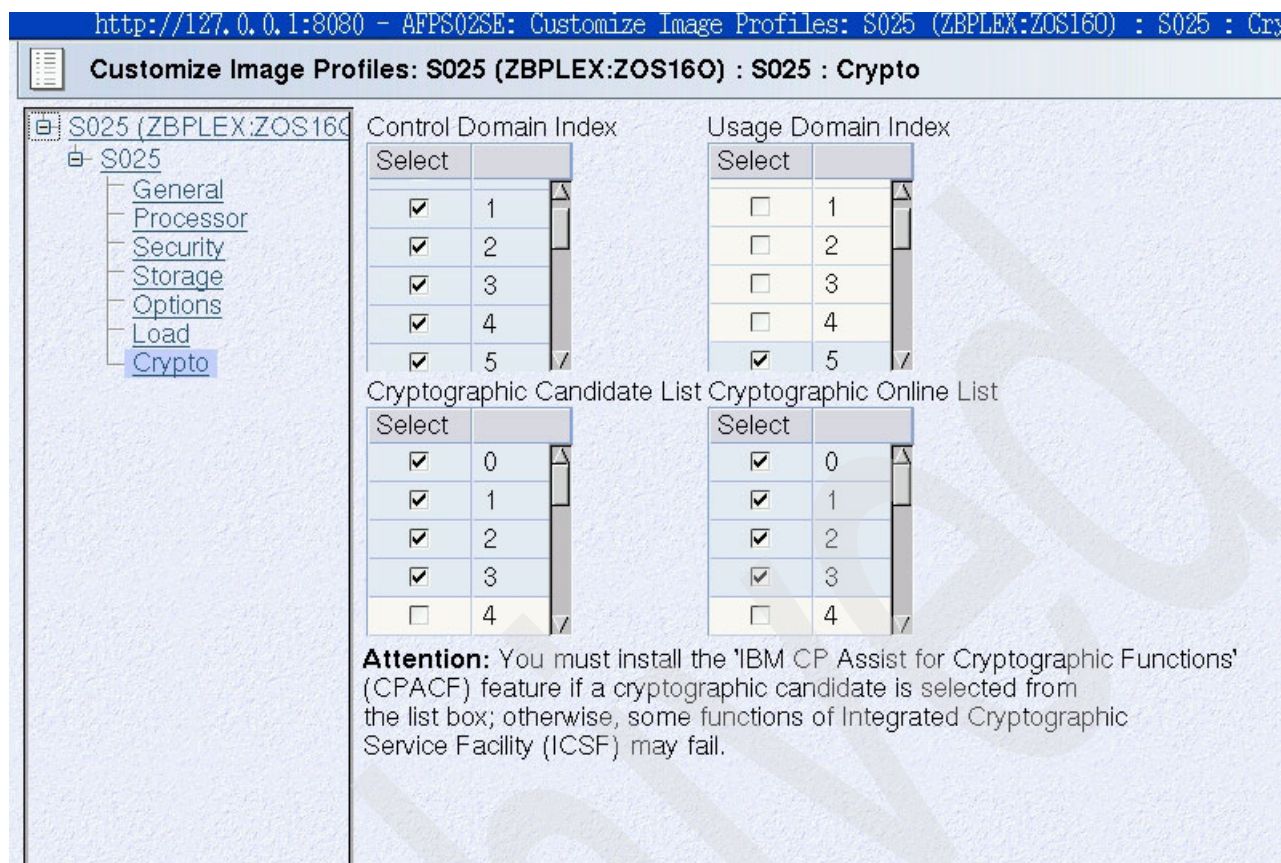


Figure 3-17 Customization of the LPAR Cryptographic definitions

### Viewing the LPAR Cryptographic Controls

The LPAR Cryptographic Control definitions that have been set in the image profile can be viewed without getting to the profile by selecting the LPAR Cryptographic Controls view:

- ▶ Double-click **Groups** and select the CPC icon.
- ▶ Drag and drop the CPC icon to the right over the View LPAR Cryptographic Controls icon.

The LPAR Cryptographic Controls panel view is displayed, as shown in Figure 3-18.

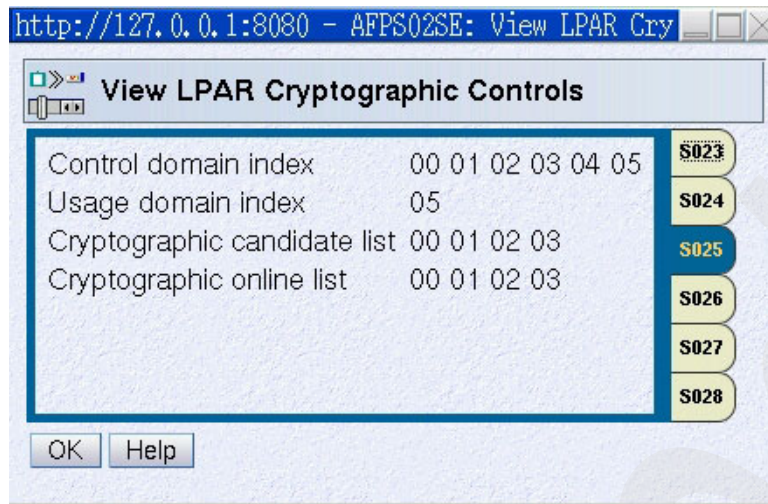


Figure 3-18 Viewing the LPAR Cryptographic Controls

On this window you can see what Control and Usage Domain values are assigned and what Cryptographic Candidate and Online selections were chosen for the selected LPAR. This window cannot be used to change any of these values.

### 3.4 Crypto Express 2 performance

**Note:** Official IBM Crypto performance measurements are available at:

<http://www.ibm.com/servers/eserver/zseries/security/cryptography.html>

The content of this Web site is updated to address the current hardware.

Most cryptographic hardware in zSeries and System z9 processors can only be used through ICSF, which shields the complexity of the hardware communication from the user.

The ICSF API call has a system path length that has to be added, from an application's view, to the execution time of the cryptographic hardware. For example, for symmetric key operations, it is advisable to have the application initiate a single API call for the total block of data, in order to limit the number of ICSF API calls.

Applications in the Internet environment will not use the cryptographic operations themselves; instead, they will follow protocol standards as, for example, Secure Socket Layer (SSL). Executing the SSL protocol for a server (or client) on a System z9 system will result in a series of cryptographic operations which, under z/OS and ICSF, will either exploit available cryptographic hardware or be executed in software.

The maximum number of SSL transactions per second that can be supported on System z9 by any combination of CPACF and Crypto Express 2 Coprocessors is limited by the amount of cycles available to perform the software portion of the SSL/TLS transactions. When both coprocessors on a Crypto Express2 feature are configured as accelerators, IBM has measured a throughput of approximately 6000 SSL handshakes per second per Crypto Express 2 feature. This represents, approximately, a 3X performance improvement compared to z990 when using either a PCI Cryptographic Accelerator (PCICA) feature, or the current Crypto Express 2 Coprocessor feature.

**Note:** These numbers indicate a throughput, that is, it is necessary to initiate several threads of parallel requests to the CEX2A to achieve this performance.

In the z9 there can be a maximum of eight Crypto Express 2 Coprocessor features reconfigured as Crypto Express 2 Accelerator (CEX2A), that is, a total of 16 accelerators.

Archived



## ICSF overview, support for CEX2A and sysplex

This chapter describes the new features within the HCR7730 level of ICSF with particular notes on the new hardware support, including the ability to configure Crypto Express 2 as an accelerator, and also the improved facilities for managing a CKDS in a sysplex environment.

This last item includes the following:

- ▶ Configuration of sysplex updating facilities
- ▶ Description of the serialization mechanisms
- ▶ Explanation of dataspace updating mechanisms
- ▶ Running multiple CKDS data sets in a sysplex
- ▶ New messages
- ▶ New abend and reason codes

## 4.1 ICSF releases

The decision was made circa 2003 to provide downloadable releases of ICSF from the Web at:

<http://www.ibm.com/servers/eserver/zseries/zos/downloads>

When the content of a z/OS release is closed it comprises the then current level of ICSF. Once installed at customers, the z/OS release can be updated with respect to ICSF by downloading a more recent ICSF level from the Web site.

Table 4-1 shows what levels of ICSF have been available, what their main features are, and in which z/OS releases they are distributed.

Table 4-1 ICSF levels.

ICSF Level	Delivered with	Highlights	Requirements	Server Hardware Supported	Crypto Hardware Supported	Minimum z/OS level supported
<b>HCR7706</b>	z/OS 1.3, z/OS 1.4	New ISPF Panels		G5, G6, z800, z900	CCF, PCICC, PCICA	z/OS 1.3
<b>HCR7708</b>	z/OS 1.5, Web Deliverable #1			G5, G6, z800, z900, z890, z990	CCF, PCICC, PCICA	z/OS 1.3, z/OS 1.4
<b>HCR770A</b>	z/OS 1.6, Web Deliverable #2	Support for PCIXCC & z/990, New TSO panels		G5, G6, z800, z900, z890, z990	CCF, PCICC, PCICA, PCIXCC	OS/390® 2.10
<b>HCR770B</b>	Web Deliverable #3	Operational Key Entry, CSFIQF facility		G5, G6, z800, z900, z890, z990	CCF, PCICC, PCICA, PCIXCC	OS/390 2.10 z/OS 1.6
<b>HCR7720</b>	z/OS 1.7, Web Deliverable #4	64-bit interface support, Crypto Express 2 Support, Clear key support	ESAME hardware required	z800, z900, z890, z990	CCF, PCICC, PCICA, PCIXCC, Crypto Express 2	z/OS 1.6
<b>HCR7730</b>	OS after z/OS 1.7, Web Deliverable #5	System z9 support, Sysplex support, System z9 support	ESAME hardware required	z800, z900, z890, z990, System z9	CCF, PCICC, PCICA, PCIXCC, Crypto Express 2	z/OS 1.6

Table 4-2 shows the various types of server and the crypto hardware they are actually hosting. Table 4-1 should be used along with Table 4-2 to precisely depict the status of a particular system.

*Table 4-2 Crypto hardware per server type.*

Server	CCF	PCICC	PCICA	PCIXCC	Crypto Express 2 as CEX2C	Crypto Express 2 as CEX2A
G5, G6	YES	YES	no	no	no	no
z800, z900	YES	YES	YES	no	no	no
z890, z990	no	no	YES	YES	YES <sup>a</sup>	no
System z9	no	no	no	no	YES	YES

a. PCICA and PCIXCC have been withdrawn from marketing with the arrival of Crypto Express 2, which provides a replacement for both.

## Licensed Internal Code (LIC) levels

Be aware that new coprocessors were introduced by the two following releases of System LIC previous to the availability of System z9.

### *The May 2004 LIC release*

This LIC release brought the following enhancements to the PCIXCC and PCICA coprocessors. All these changes were integrated into the Crypto Express 2 coprocessor.

- ▶ PCIX Cryptographic Coprocessor (PCIXCC)
  - Derived Unique Key Per Transaction (DUKPT)
    - The triple DES support (double length keys) was added.
  - Europay Mastercard and VISA (EMV) 2000 standard support
    - Enhancements to the Diversified Key Generate service
      - Session keys for secure messaging for PINs
      - Session keys for secure messaging for keys using SESS-XOR scheme
      - Session keys for all applicable EMV key types using the EMV 2000 Annex A1.3.1 derivation scheme
    - New PIN\_Change\_Unblock service to handle VISA 1.4 PIN
      - Change/Unblock command
- ▶ PCIXCC and PCI Cryptographic Accelerator (PCICA)
  - Public Key Decrypt/Public Key Encrypt (PKD/PKE) enhancements
    - PKE Mod Raised to Power (MRP) support
    - PKD zero pad support

### *The January 2005 LIC release*

This LIC release brought the following enhancements to the PCICC, PCIXCC and Crypto Express 2 coprocessors. The Crypto Express 2 enhancements were picked up in the System z9 LIC.

- ▶ 19-digit Personal Account Numbers (PANs) for PCIXCC and Crypto Express2
  - The previously supported 13-digit and 16-digit PANs have been extended to 19 digits.
  - Designed to meet the industry standard for Card Validation Value (CVV) to increase antifraud security.
- ▶ Less than 512-bit clear key RSA operations - PCIXCC, Crypto Express2
  - Designed to allow clear key RSA operations using keys less than 512 bits - Digital Signature Verify (CSNDDSV), Public Key Encrypt (CSNDPKE), and Public Key Decrypt (CSNDPKD).

- This was dictated by the need to migrate existing applications so that they can exploit the hardware coprocessors.
- ▶ 2048-bit key (clear and secure) RSA operations for PCICC
  - This is a new parameter for the PCICC on z800 and z900, as it was previously supporting a key length of 1024 bits at a maximum.
  - Four ICSF services can use the 2048-bit key: Public Key Decrypt, Symmetric Key Import, Export, and Generate.

## 4.2 Highlights of ICSF HCR7730

HCR7730 is the latest level of ICSF available as of the writing of this book. It is referred to in the download Web site as “Cryptographic Support for z/OS V1R6/R7 and z/OS.e V1R6/R7” and includes the following changes and components:

- System z9 and Crypto hardware support
- Enhanced key management for clear DES and AES keys
- Sysplex support for CKDS updating

We now examine these items in detail.

## 4.3 System z9 and Crypto hardware support

We describe here the functions that the System z9 cryptographic hardware supports that are new with respect to the z990/z890 implementation.

IBM System z9 processors provide these new functions with the CPACF Message-Security Assist instructions:

- ▶ Hardware execution of the AES algorithm with a 128-bit clear key
- ▶ Hardware execution of the SHA-256 algorithm
- ▶ Hardware Pseudo Random Number Generator (PRNG)

These functions can also be invoked, as described below, via ICSF services, except for the PRNG function, which is available through the CPACF KMC instruction only.

- ▶ Hardware AES with 128-bit clear key support - The function is invoked through the Symmetric Key Encipher (CSNBSYE) or Symmetric Key Decipher (CSNBSYD) services with the keyword “AES” in the rule array. Note that for a key length of 192 or 256 bits ICSF still provides the service, but via software only.
- ▶ SHA-256 - The function is provided by the One Way Hash Generate (CSNBOWH) service with the keyword “SHA-256” in the rule array.

**Note:** A Crypto Express 2 card is required for ICSF to provide a hardware-generated pseudo random number via the Random Number Generate (CSNBRNG) service.

Examples of CPACF invocation via the Message-Security Assist instructions and ICSF services are given in Appendix A, “CPACF programs” on page 103.

### Performance reporting with CPACF

ICSF reports performance data for the CSNBOWH service with SHA-1 or SHA-256 as the selected algorithm.

### 4.3.1 CEX2A support

The Crypto Express 2 feature is the only Crypto coprocessor that can be used on the IBM System z9 109 processor. While it is possible to use the same Crypto Express 2 feature on the z890 and z990 processors, it could only be configured as two Crypto coprocessors also known as CEX2C cards. When installed on the IBM System Z9 109 processor, the Crypto Express 2 cards can be configured either as Crypto accelerators (CEX2A) or Crypto coprocessors (CEX2C). As was explained earlier, the following configurations are possible for one CEX2C feature:

- ▶ 2 x Crypto coprocessors (2 x CEX2C)
- ▶ 2 x Crypto accelerators (2 x CEX2A)
- ▶ 1 x Crypto accelerators and 1 x Crypto processor (1 x CEX2A, 1 x CEX2C)

ICSF recognizes the CEX2A type of coprocessor beginning with HCR7730. Previous levels of ICSF will just ignore the CEX2A cards. The coprocessor management panel for ICSF HCR7730 shows the CEX2A card with a type prefix of "F" as shown in Figure 4-1.

----- ICSF Coprocessor Management ----- Row 1 to 4 of 4		
Select the coprocessors to be processed and press ENTER. Action characters are: A, D, E, K, R and S. See the help panel for details.		
COPROCESSOR	SERIAL NUMBER	STATUS
-----	-----	-----
. E00	95000224	ACTIVE
. F01		ACTIVE
. F02		ACTIVE
. F03		ACTIVE

Figure 4-1 - Displaying the CEX2A cards with the "F" prefix

The CEX2A cards provide only the following services:

- ▶ Digital Signature Verify (CSNDDSV)
- ▶ PKA Decrypt (CSNDPKD)
- ▶ PKA Encrypt (CSNDPKE) with ZERO-PAD and MRP only

Note that ICSF routes these requests to CEX2A cards only if available in the system, otherwise they are routed to CEX2Cs.

**Important:** Although running previous levels of ICSF on a System z9 is still expected to provide the CEX2C cryptographic services available previously to z9, we strongly recommend to install HCR7730 as soon as possible even if there is no current need for the new functions.

## 4.4 Enhanced key management for clear DES and AES keys

The HCR7730 level of the KGUP utility has been enhanced to allow some new parameters in support of clear AES keys. Clear DES keys were introduced with an SPE for the previous ICSF levels HCR770A and HCR7720, that was also modifying the services.

- ▶ CSNBKTB (Key Token Build)  
Which therefore is also a means to generate a clear key token and write it to the CKDS.





parts shown in the KGUP control CKDS is shown in Figure 4-2.

Byte	Description						
0	x'01' (internal token only)						
1-3	Implementation-dependent bytes (x'000000' for ICSF)						
4	x'04' Key token version number						
5	x'00' Reserved						
6	Flag byte <table border="0"> <tr> <td>Bit</td><td>(both bits are '0')</td></tr> <tr> <td>0</td><td>Encrypted key and master key verification pattern (MKVP) are present</td></tr> <tr> <td>1</td><td>Control vector (CV) value in this token has been applied to the key</td></tr> </table>	Bit	(both bits are '0')	0	Encrypted key and master key verification pattern (MKVP) are present	1	Control vector (CV) value in this token has been applied to the key
Bit	(both bits are '0')						
0	Encrypted key and master key verification pattern (MKVP) are present						
1	Control vector (CV) value in this token has been applied to the key						
7	1-byte Longitudinal Redundancy Check (LRC) checksum of clear key value						
8-15	Master key verification pattern (For a clear key AES token this value will be hex zero)						
16-47	128-bit, 192-bit, or 256-bit key value, left-justified and padded on the right with hex zeroes						
48-55	8-byte control vector. (For a clear key AES token this value will be hex zero)						
56-57	2-byte integer specifying the length in bits of the clear key value						
58-59	2-byte integer specifying the length in bytes of the encrypted key value (For a clear key AES token this value will be hex zero)						
60-63	Token validation value (TVV)						

Figure 4-2 AES clear key internal token

## 4.5 Sysplex support

ICSF was not, so far, providing full sysplex support, because while the CKDS data set could be shared between different ICSF instances, but the in-storage copies of the data set were not automatically synchronized when one ICSF instance was adding, modifying or deleting a key in the shared data set. Synchronization had to rely on manual processes.

New functions have been added to ICSF HCR7730 to maintain the coherency of the multiple in-storage copies of the CKDS in a sysplex configuration. Before we proceed and describe these new functions, we give with more details about how CKDS sharing used to be achieved before the availability of ICSF HCR7730.

#### 4.5.1 Sysplex support prior to ICSF HCR7730

ICSF uses two key data sets: the Cryptographic Key Data Set (CKDS) and the Public Key Data Set (PKDS). The CKDS is used to hold DES keys used in symmetric encryption and MAC algorithms, while the PKDS is used to hold asymmetric keys (that is, RSA keys with the Crypto Express 2 coprocessor).



The content of CKDS is updated and accessed much more frequently, due to the relatively ephemeral nature of the symmetric keys. On the contrary, asymmetric keys are mostly long-living keys, and the decision was made by the ICSF laboratory to focus on developing full sysplex support for the CKDS only in HCR7730.

The CKDS can be defined using the following IDCAMS statements:

```

DEFINE CLUSTER (NAME(LENNIE.CSFCKDS)      -
                RECORDS(100 50)           -
                RECORDSIZE(252,252)       -
                KEYS(72 0)                 -
                FREESPACE(10,10)           -
                SHAREOPTIONS(2)) -
DATA (NAME(LENNIE.CSFCKDS.DATA) -
      BUFFERSPACE(100000) -
      ERASE -
      WRITECHECK) -
INDEX (NAME(LENNIE.CSFCKDS.INDEX))

```

The SHAREOPTIONS above are those used for data sets that are shared and that are managed by the application. ICSF makes use of system scope ENQ macros to serialize access to the CKDS so that read integrity is maintained when the CKDS is updated by the ICSF address space, as long as only one system is accessing the data set for updates. However, read integrity is not maintained if KGUP updates the CKDS.

The entire CKDS is read into a dataspace named CSFDS001 when ICSF starts. This in-storage copy of the CKDS is managed by ICSF as follows:

- Updates that are made to keys using services such as CSNBKRC (Key Record Create) and CSNBKRW (Key Record Write), CSNBKRD (Key Record Delete) and CSNBKPI (Key Part Import), or any service that invokes these implicitly, are made to the CKDS dataspace and are also written to the CKDS.
- However, READ activity using the CSNBKRR (Key Record read) or any service that invokes it implicitly, is always directed only to the dataspace.

The KGUP utility can be used to add, modify, and delete keys in the CKDS, but this is done *without* updating the CKDS in-storage copy. That is, the dataspace and the CKDS can become desynchronized because the DASD version of the CKDS is being updated by the local KGUP.

This is also true for updates done to the shared CKDS by remote instances of ICSF or KGUP, because none of them will be reflected in the local in-storage copy of the CKDS.

The solution to putting the CKDS data set and its in-storage copy back in synchronization again is then to use the CSFEUTIL program or the REFRESH facility from the ICSF ISPF panels. Either of these methods will refresh the local in-storage copy of the CKDS from the CKDS data set.

The mode of operation when not in full sysplex support, that is, with ICSF levels previous to HCR7730, is illustrated by Figure 4-3 and Figure 4-4.

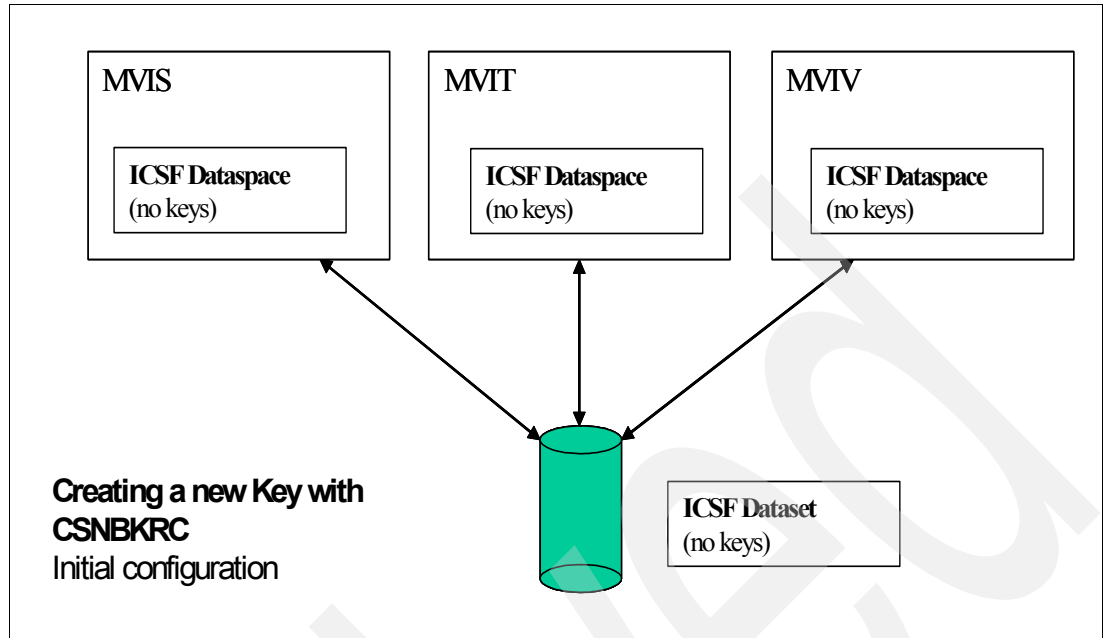


Figure 4-3 Pre-HCR7730 sysplex configuration - Key Record Create part 1 of 2

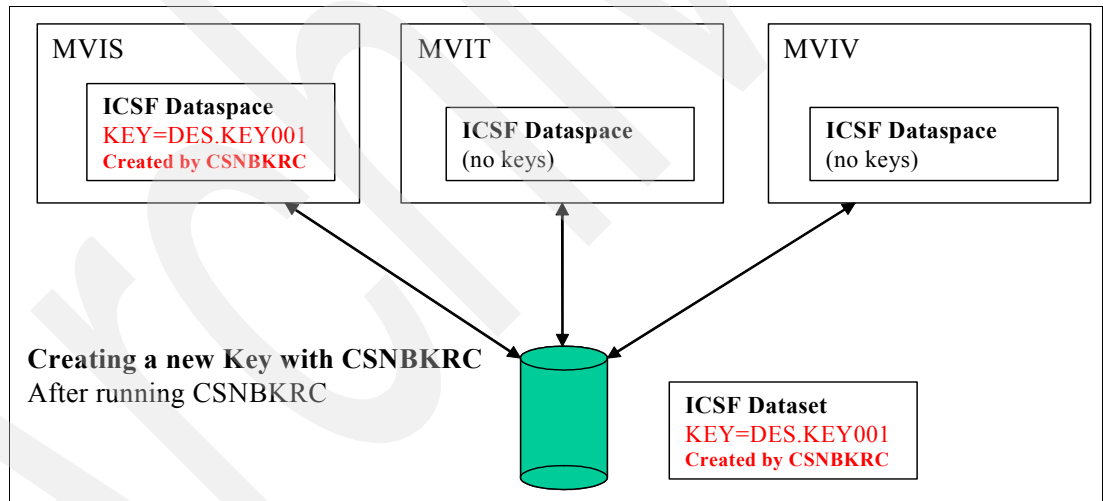


Figure 4-4 Pre-HCR7730 sysplex configuration - Key Record Create part 2 of 2

It is the lack of dataspace synchronization and data set cross-system serialization that is addressed in the HCR7730 version of ICSF.

## 4.5.2 Sysplex support with ICSF HCR7730

ICSF HCR7730 provides a mechanism for ensuring synchronization between the CKDS data set and in-storage copies, in a sysplex environment.

### The Options data set SYSPLEXCKDS keyword

The SYSPLEXCKDS keyword is new with ICSF level HCR7730. Its syntax is:

```
SYSPLEXCKDS (YES|NO, FAIL (YES|NO))
```

Note that the FAIL parameter is only applicable when the first parameter is YES.

SYSPLEXCKDS(NO,FAIL(NO)) is the default setting. The first NO indicates that no sysplex sharing protocol is to be used. This is equivalent to the serialization mechanism used prior to ICSF HCR7730.

SYSPLEXCKDS(NO,FAIL(YES)) is the same as SYSPLEXCKDS(NO,FAIL(NO)).

SYSPLEXCKDS(YES,FAIL(NO)) and SYSPLEXCKDS(NO,FAIL(YES)) will cause the new CKDS sharing protocol to be used when accessing the CKDS within a sysplex.

The two options FAIL(YES) and FAIL(NO) indicate what action should be taken when an attempt to join the sysplex group by ICSF fails. If FAIL(YES) is specified, the ICSF initialization will fail if the attempt to join the sysplex group fails. If FAIL(NO) is specified, then ICSF initialization will continue but the sysplex sharing protocol will not be used. Updates to the CKDS will not be received from other systems and updates performed will not be notified to other systems, just as if SYSPLEXCKDS(NO,FAIL(YES/NO)) had been coded.

Once we had established a CKDS shared across a sysplex, we wanted it to continue to be maintained correctly, so we generally specified SYSPLEXCKDS(YES,FAIL(YES)) in our testing.

### 4.5.3 How the new CKDS sysplex sharing works

All systems sharing the CKDS must be using HCR7730. Systems using lower levels of ICSF will not be able to take part in the CKDS sharing process.

Sysplex XCF messages are sent to the sysplex members in the ICSF group. At any given time the messages may originate from any one system, reflecting an update done to the CKDS data set from that system. The systems receiving messages take appropriate actions to update their CKDS in-storage copy. The XCF group name is fixed to SYSICSF by design.

The messages are sent when one system updates or creates a CKDS record using one of the following ICSF services:

- ▶ Key Record Create (CSNBKRC)
- ▶ Key Record Delete (CSNBKRD)
- ▶ Key Record Write (CSNBKRW)
- ▶ Key Part Import (CSNBKPI)

If the CKDS is updated as a result of using the KGUP utility or by performing a CKDS re-encipher, then the updates will *not* be propagated to the other systems, except if the CKDS update is the completion step of a Load Operational Key process started at the TKE. In this latter case the CKDS update driven either by KGUP or from the ICSF ISPF panel is propagated to the in-storage copy of all the ICSF instances sharing the CKDS.

Access to the entire CKDS data set is serialized using ENQ macros with a Qname of SYSZCKT and an Rname of ckdsname. This ENQ is used to ensure serialization between the ICSF address spaces and block writers of the CKDS that write using CSNBKRC, CSNBKRD, or CSNBKRW. But KGUP is not prevented from writing to the CKDS.

In addition to the above level of serialization, all ICSF address space I/O to the CKDS data set and updates of in-storage copies of the CKDS are serialized using an ENQ with a Qname of SYSZCKDS and an Rname of ckdsname. However, this ENQ is subject to a time-out to allow for situations where one system is holding the ENQ for an excessive time. Note also that XCF messages are broadcast requesting other ICSF address spaces to release the

ENQ. The time-out is designed to handle software or hardware failures that may cause WAITs.

Each of the ENQs above is issued with a scope of SYSTEMS and so will be propagated to the whole sysplex (and potentially beyond if the GRSplex includes systems outside the sysplex).

Internally to the ICSF address space there are 512 latches that are used to serialize access to particular key records within the CKDS. This ensures that the same key record cannot be updated simultaneously by requests from two independent processes making requests of ICSF. A hashing method is used to select a particular latch to correspond with a given key record. A latch is obtained whenever a key record is being read or written.

### Creating a record

When a new record is created, via CSNBKRC, the local in-storage copy of the CKDS is updated along with the CKDS data set itself. In the same timeframe a signal is broadcast via the XCF links to instruct the other systems that a new record is available to be read into the CKDS in-storage copy. The other systems will then read the record from the CKDS data set into their local in-storage copy.

### Updating a record

When a CKDS record is updated, via CSNBKRW or CSNBKPI, the local in-storage copy of the CKDS is updated, along with the CKDS data set itself, with the new record contents. In the same time frame a signal is sent via the sysplex XCF links to instruct the other systems to invalidate the record in the local CKDS in-storage copy and to read the updated version from the CKDS data set.

### Deleting a record

When a CKDS record is deleted, via CSNBKRD, the local CKDS in-storage copy along with the CKDS data set itself are updated. In the same time frame a signal is broadcast via the XCF links to the other systems to invalidate the record in their local in-storage copy. The other ICSF instances then delete that record from their in-storage copy.

The operation of ICSF in this new mode is shown in Figure 4-5 and Figure 4-6.

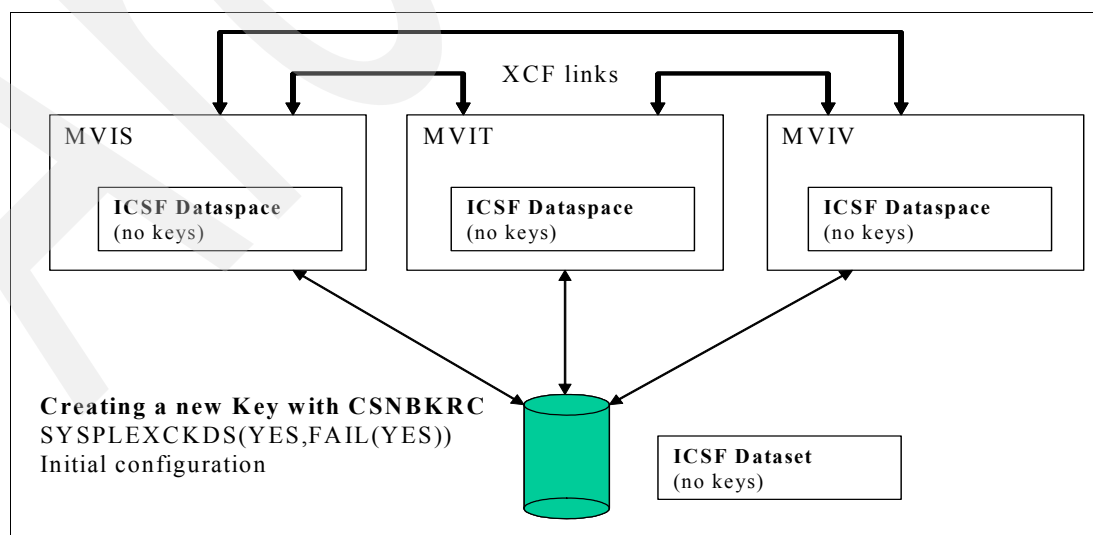


Figure 4-5 HCR7730 sysplex configuration - Key Record Create part 1 of 2

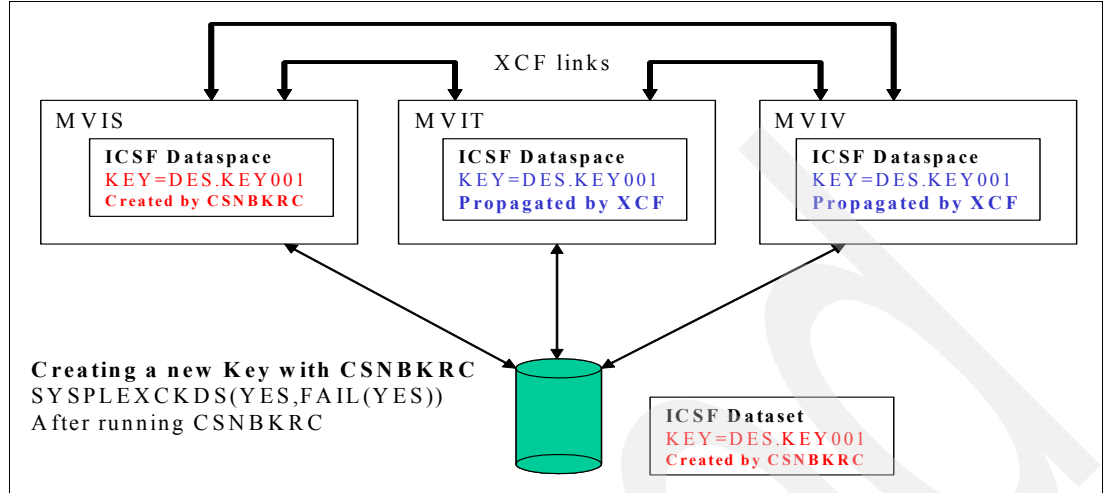


Figure 4-6 HCR7730 sysplex configuration - Key Record Create part 2 of 2

#### 4.5.4 How we tested the HCR7730 new sysplex support

We dedicated part of the residency to exercise the CKDS sysplex support functions. In this section we describe our setup and the test cases we used.

##### The environment

We started with a 3-system sysplex running z/OS 1.7 on a z990 processor. The HCR7730 level of ICSF was installed. The systems were MVIT, MVIS and MVIV. (MVIV was not needed for these tests, but is shown in the diagrams.)

Each system has access to four crypto coprocessor CE2XC cards.

New CKDS and PKDS data sets were allocated and initialized, specifying the master key using the PPINIT function on system MVIT. The master key was then also set on system MVIS.

##### Testing methods

The testing methods involved running REXX routines that invoked various ICSF interfaces. Three REXX routines were written and in each case the key name is hardcoded into the routines for simplicity.

##### REXBKRC

This routine was used to create a new key. It uses the following routines:

- ▶ CSNBKRD to conditionally delete the key
- ▶ CSNBKGN to generate a key
- ▶ CSNBKRC to create the key name
- ▶ CSNBKRW to write the key using the key name

##### REXBKRD

This routine is used solely to delete the key. It calls CSNBKRD to delete the key.

## REXBKRR

This routine is used to read the keyed record and uses CSNBKRR to read the hardcoded key name.

### Testing sequence

The sequence of events is shown in Table 4-3.

Table 4-3 Our testing sequence of sysplex support

Item	Action	MVIS	MVIT
1	Ensure ICSF is not active.	Yes	Yes
2	List the CKDS data set using IDCAMS to show our test key does not exist.	Yes	N/A
3	Start ICSF specifying SYSPLEXCKDS(NO,FAIL(NO)) in the Options data set on both systems.	Yes	Yes
4	Run REXBKRC to create a new key.	Yes	N/A
5	List the CKDS data set to show the created key (using IDCAMS).	Yes	N/A
6	Run REXBKRR to read the key. This is expected to succeed. Result: It does succeed.	Yes	N/A
7	Run REXBKRR to read the key. This is expected to fail because the key will not be available on this system. Result: It does indeed fail with Return code 8 and Reason code 271C.	N/A	Yes
8	Stop and then restart ICSF on both systems changing the setting to SYSPLEXCKDS(YES,FAIL(YES)).	Yes	Yes
9	Run REXBKRD to delete the key.	Yes	N/A
10	Run REXBKRR to read the key. This is expected to fail as the sysplex propagation code should have logically deleted the record on MVIT. Result: It does indeed fail with Return code 8 and Reason code 271C.	N/A	Yes
11	Run REXBKRC to create a new key.	Yes	N/A
12	Run REXBKRR to read the key. This is expected to succeed as the sysplex propagation code should have logically propagated the updated record to MVIT. Result: It does indeed succeed.	N/A	Yes
13	List the CKDS data set to show the test key does indeed exist.	Yes	N/A

Each of the REXX routines above was run as a batch job. They are provided in Appendix B, "Programs used in sysplex testing" on page 123.

### 4.5.5 Updates to the CKDS using KGUP

If the KGUP utility is used to create keys, these updates will *not* be placed in any dataspace. It is necessary to use CSFEUTIL or the ICSF ISPF panels to refresh the dataspace on each system. This situation is shown in Figure 4-7 and Figure 4-8.

The Job step that one might use for CSFEUTIL is:

```
//STEP EXEC PGM=CSFEUTIL,PARM='NEW.CKDS,REFRESH'
```

where 'NEW.CKDS' is the CKDS the in-storage copy of which is to be refreshed.

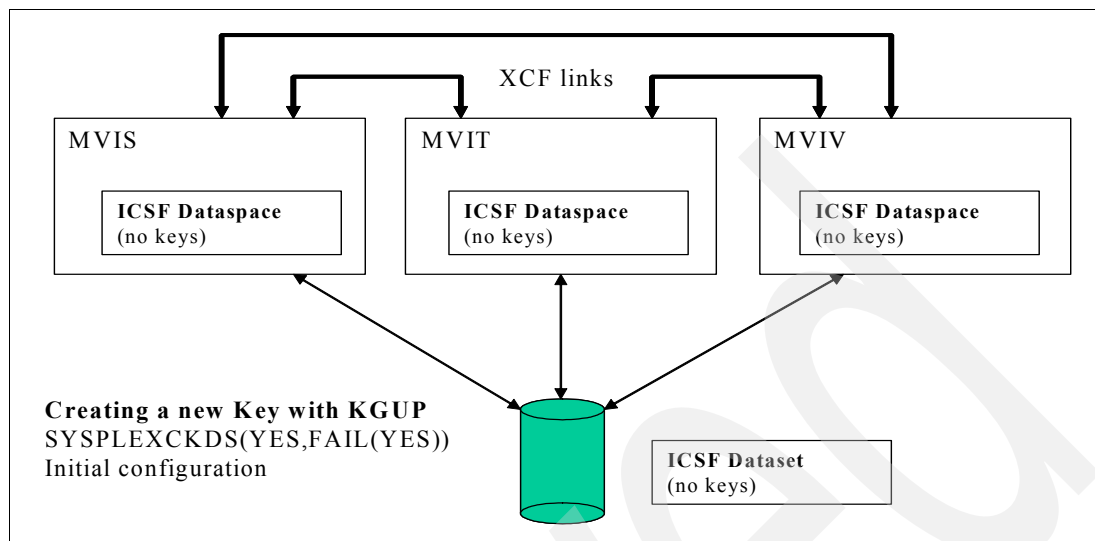


Figure 4-7 HCR7730 sysplex configuration with KGUP update - Part 1 of 2

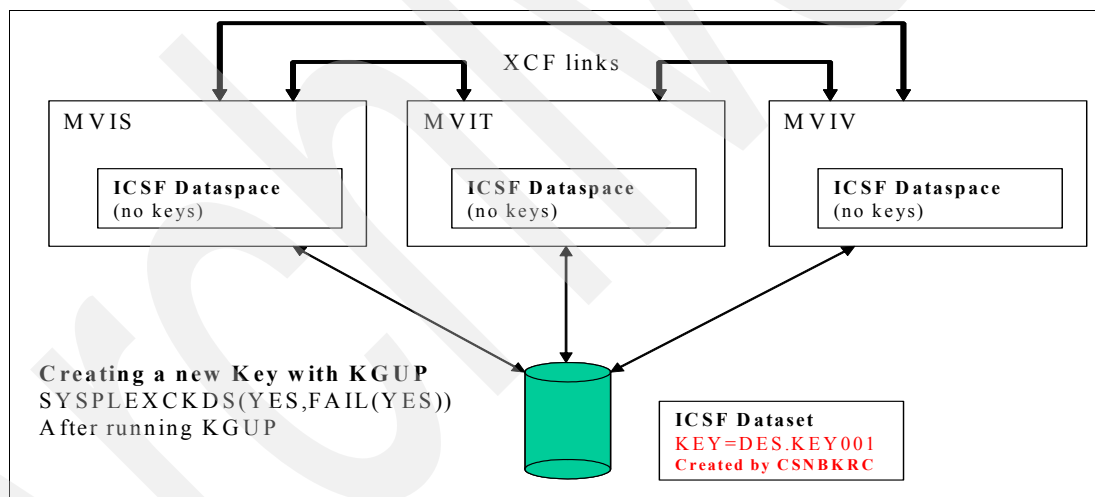


Figure 4-8 HCR7730 sysplex configuration with KGUP update - Part 2 of 2

## 4.5.6 Messages during ICSF startup and shutdown

### ICSF startup

Following are the messages produced when starting ICSF on one member (MVIT) of our sysplex. The START command is shown first. Note that there are four crypto engines available on this processor.

```
S ICSF
$HASP100 ICSF      ON STCINRDR
IEF695I START ICSF      WITH JOBNAME ICSF      IS ASSIGNED TO USER ICSF
, GROUP OMVSGRP
$HASP373 ICSF      STARTED
CSFM600I CONNECTION ESTABLISHED TO ICSF SYSPLEX GROUP SYSICSF, MEMBER
MVIT.
```

```

CSFM441I CRYPTO EXPRESS2 COPROCESSOR E00, SERIAL NUMBER 95000051,
ACTIVE.
CSFM441I CRYPTO EXPRESS2 COPROCESSOR E01, SERIAL NUMBER 95000060,
ACTIVE.
CSFM441I CRYPTO EXPRESS2 COPROCESSOR E02, SERIAL NUMBER 95000383,
ACTIVE.
CSFM441I CRYPTO EXPRESS2 COPROCESSOR E03, SERIAL NUMBER 95000387,
ACTIVE.
CSFM431I BOTH MASTER KEYS CORRECT ON CRYPTO EXPRESS2 COPROCESSOR E00,
SERIAL NUMBER 95000051.
CSFM431I BOTH MASTER KEYS CORRECT ON CRYPTO EXPRESS2 COPROCESSOR E01,
SERIAL NUMBER 95000060.
CSFM431I BOTH MASTER KEYS CORRECT ON CRYPTO EXPRESS2 COPROCESSOR E02,
SERIAL NUMBER 95000383.
CSFM431I BOTH MASTER KEYS CORRECT ON CRYPTO EXPRESS2 COPROCESSOR E03,
SERIAL NUMBER 95000387.
CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.
CSFM001I ICSF INITIALIZATION COMPLETE
CSFM400I CRYPTOGRAPHY - SERVICES ARE NOW AVAILABLE.

```

## ICSF shutdown

Following are the messages produced when shutting down ICSF on one member (MVIT) of our sysplex.

```

P ICSF
IEA989I SLIP TRAP ID=X13E MATCHED.  JOBNAME=ICSF      , ASID=0053.
IEA989I SLIP TRAP ID=X13E MATCHED.  JOBNAME=ICSF      , ASID=0053.
IEA989I SLIP TRAP ID=X13E MATCHED.  JOBNAME=ICSF      , ASID=0053.
CSFM601I CONNECTION DISABLED TO ICSF SYSPLEX GROUP SYSICSF, MEMBER MVIT.
IEA989I SLIP TRAP ID=X13E MATCHED.  JOBNAME=ICSF      , ASID=0053.
IEA989I SLIP TRAP ID=X13E MATCHED.  JOBNAME=ICSF      , ASID=0053.
CSFM401I CRYPTOGRAPHY - SERVICES ARE NO LONGER AVAILABLE.
IEF352I ADDRESS SPACE UNAVAILABLE
$HASP395 ICSF      ENDED
IEA989I SLIP TRAP ID=X33E MATCHED.  JOBNAME=*UNAVAIL, ASID=0053.

```

## 4.5.7 Multiple CKDS data sets in the sysplex

The sysplex sharing protocol allows for multiple CKDS data sets to be used within a sysplex. Updates will be propagated to the appropriate CKDS. However, all updates will be propagated using the same sysplex XCF group name, SYSICSF.

Figure 4-9 shows a dual CKDS environment.



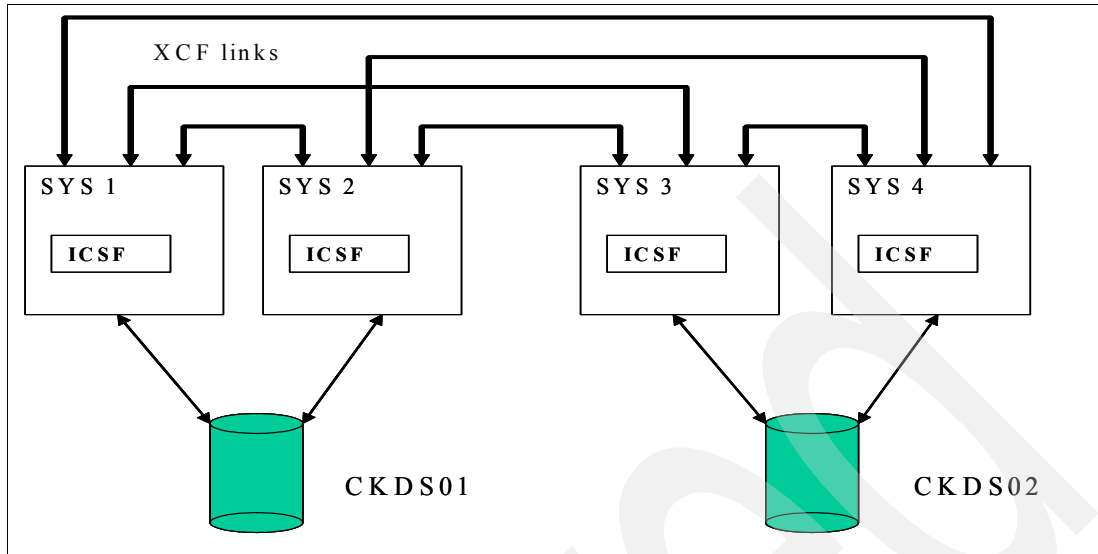


Figure 4-9 Sysplex configuration with two CKDS

The CKDS data set name is included as part of the Rname referenced in each of the ENQ macros used to serialize access. This ensures that if more than one CKDS data set is used, they may be used concurrently.

In addition, the name of the CKDS data set is used in the process of passing XCF messages between members of the sysplex. Each sysplex will only take actions to update its CKDS in-storage copy if the data set name matches its own CKDS data set name.

**Important:** Users should ensure that if multiple CKDS data sets are used, they have different data set names. Failure to do this may have implications both for CKDS serialization and for the integrity of the CKDS in-storage copies.

### 4.5.8 Options for sharing the CKDS data set

Thus, there are three main ways of sharing the CKDS data set:

1. Independent systems accessing the CKDS with a managed update process. Systems will specify `SYSPLEXCKDS(NO,FAIL(YESINO))`.
2. Systems in the same GRS complex (but not necessarily in the same sysplex) who access the CKDS for read and update, but with a manually managed refresh process that is not using sysplex CKDS sharing.  
Systems will have `SYSPLEXCKDS(NO,FAIL(YESINO))` specified. In addition, customers can adjust their GRS SYSTEM inclusion RNL to include the Qname `SYSZCSF`. This will convert the SYSTEM ENQ to have scope of SYSTEMS.
3. Systems within the same sysplex using `SYSPLEXCKDS(YES,FAIL(YESINO))`.

Let us discuss the implications of each method.

#### Method 1

In this method the systems sharing the CKDS might be within sysplexes or may be other independent systems. It would be the administrator's responsibility to ensure that updates are coordinated, and that they are performed from one system at a time. It is also the administrator's responsibility to ensure that in-storage copies are refreshed on each system using CSFEUTIL or the ICSF ISPF panels.

Access to the CKDS will not be serialized across systems. If two or more systems attempt to update the CKDS simultaneously, then it is possible that corruptions could occur.

## Method 2

In this method, all systems sharing the CKDS are part of the same GRS complex, which could be a GRS Star within a sysplex or a GRS Ring comprising systems both within and outside of sysplexes. As explained above, the systems would need to modify the GRS SYSTEM inclusion RNL so that SYSZCSF Qnames are converted to have a scope of SYSTEMS.

Because of this, the serialization mechanisms will ensure that multiple systems should be able to update the CKDS and maintain its integrity. The CKDS in-storage copies will not always be synchronized with the CKDS data set, but this can be corrected by using CSFEUTIL or the ICSF ISPL panels to refresh the in-storage copies.

However, this method is a theoretical sharing mechanism, and has not been tested formally.

## Method 3

This is the new sysplex sharing method. All systems sharing the CKDS need to be part of the same sysplex. All systems will maintain accurate, timely, coherent dataspace containing the CKDS records.

All systems will need to be using the HCR7730 level of ICSF or later.

No manual refreshing of the dataspace will be needed during normal operation, unless KGUP has been used to add, update, or delete keys. Remember, however, that as part of the Load Operational Key TKE process, the CKDS update by KGUP is to be propagated to the in-storage copies of the ICSF instances sharing the CKDS in the sysplex.

To use this type of sharing, specify SYSPLEXCKDS(YES,FAIL(YES|NO)) on all sharing systems.

**Note:** It is IBM's recommendation to share the CKDS using Method 3.

### 4.5.9 Changing the Master Key in a sysplex

The recommended procedure is as follows:

1. Disable CKDS dynamic access via panels for all systems sharing the CKDS.
2. Load the new symmetric-keys master key for all systems.
3. Reencipher the CKDS on one system.
4. Change master keys on all systems.
5. Enable CKDS dynamic access for all systems.
6. Update the CSFPRMnn member to point to the correct CKDS.

### 4.5.10 Managing the PKDS data set

The PKDS data set does not have a dataspace into which keys are loaded. Instead, it has a cache of the most recently used keys. By default this is set at 64 keys but can be set as high as 256 keys, or can be eliminated entirely by specifying PKDSCACHE(0) in the Options data set.

There is no mechanism to ensure synchronization of the multiple PKDS caches within a sysplex. However, the life of RSA keys is normally far longer than DES keys' life and it is expected that the number of keys to maintain will be relatively small. Consequently, it is not envisioned that this lack of automatic synchronization will be an issue to sysplex users.

If the number of keys maintained is 256 or less, they can all be kept in the cache.

If the cache is used, any changes to the keys will need to be handled manually using the CSFPUTIL utility or the ICSF ISPF panels to refresh the cache on all systems sharing the PKDS.

If it is essential to always have the correct version of all PKDS records immediately on all systems accessing the PKDS, then you should specify PKDSCACHE(0).

#### 4.5.11 Other sysplex support changes

##### SMF Record Type 82 (Subtype 21)

This SMF record is written when

- ▶ A system joins the ICSF sysplex group.
- ▶ A system leaves the ICSF sysplex group.
- ▶ If a system left the ICSF group, the record also indicates whether the reason was normal ICSF termination processing or error recovery processing, that is, not a termination of ICSF.

The record contains the name of the sysplex group, the name of the sysplex member, the time of join/leave, and the name of the active CKDS.

##### Component Trace

The following are new ICSF Component Trace Entries:

- ▶ Entry type 13 'Send of XCF message' Trace sending of an XCF message
- ▶ Entry type 14 'Receipt of XCF message' Trace receipt of an XCF message
- ▶ Entry type 15 'Exclusive CKDS ENQ' Trace return of control to CKDS I/O subtask following a request for exclusive ENQ on the SYSZCKDS.ckdsdsn resource

##### New console messages - 1

CSFM301A FAILURE UPDATING CKT AFTER CKDS UPDATE, RET=return\_code, RSN=reason\_code.  
MANUAL REFRESH OF CKDS REQUIRED, MEMBER member\_name.  
CSFM303E CKT UPDATE FAILED, LABEL label.

These messages are issued together when a DASD copy of a CKDS has been updated successfully, but a member of the ICSF sysplex group has not been able to update its in-storage copy. The CKDS dataspace is now out of sync with the CKDS data set.

##### New console messages - 2

CSFM302A TIMED OUT WAITING FOR RESOURCE SYSZCKDS.ckdsdsn. CKDS UPDATE FAILED.

This message is issued when the CKDS I/O subtask timed out waiting for an exclusive ENQ on SYSZCKDS.ckdsdsn. At least one member of the sysplex group has not relinquished its ENQ on the resource.

ICSF processing continues. The CKDS update operation is failed with C/BBD

The operator should issue D GRS,RES=(qname,rname) to determine who is holding the ENQ and why it has not been released.

### **New console messages - 3**

CSFM600I CONNECTION ESTABLISHED TO ICSF SYSPLEX GROUP group\_name, MEMBER member\_name.

This message is issued during ICSF initialization processing when ICSF has successfully joined the XCF group. This system will now participate in sysplex-wide coherency for the CKDS data set.

### **New console messages - 4**

CSFM601I CONNECTION DISABLED TO ICSF SYSPLEX GROUP group\_name, MEMBER member\_name.

This message is issued during ICSF termination processing when ICSF has successfully left the XCF group. This system will no longer participate in sysplex-wide coherency for the CKDS data set.

### **New console messages - 5**

CSFM602E CONNECTION BROKEN TO ICSF SYSPLEX GROUP group\_name, MEMBER member\_name.

This message is issued during ICSF recovery processing when the Cross-System Services environment is terminated. This system will no longer participate in sysplex-wide coherency for the CKDS data set.

### **New console messages - 5**

CSFM603E FAILURE IN XCF SERVICE xcf\_service FOR MEMBER member\_name, GROUP group\_name. RETURN CODE = return\_code, REASON CODE = reason\_code.

This message is issued when a failure is encountered while attempting to join or leave the XCF group or while processing an inter-system message.

The system action for an IXCJOIN failure depends upon the specification of the FAIL keyword in the SYSPLEXCKDS installation option. For an IXCMSGI failure, the ICSF Cross-System Services environment is terminated.

### **New console messages - 6**

CSFM604E FAILURE INITIALIZING ICSF CROSS-SYSTEM SERVICES ENVIRONMENT, FUNCTION = code, RETURN CODE = return\_code, REASON CODE = reason\_code.

This message is issued when a failure is encountered while setting up the ICSF Cross-System Services environment.

- ▶ Code = 1: Error occurred in IXCJOIN processing
- ▶ Code = 2: Error occurred when creating CKDS latch set

The system action depends upon the specification of the FAIL keyword in the SYSPLEXCKDS installation option.

### **New abend codes**

Table 4-1 shows the new abend codes that can be produced by ICSF, and the circumstances under which they are produced.

Table 4-4 ICSF HCR7730 new abend codes

Abend - Reason Code	Description
18F - 54	SYSplex(YES,FAIL(YES)) specified and failure occurred during the IXCJOIN request
18F - 55	SYSplex(YES,FAIL(YES)) specified and failure occurred while creating the CKDS latch set
18F - 56	Program error issuing ISGENQ
18F - 57	Error establishing ESTAE for ICSF Cross-System Services task
18F - 58	Program error invoking IXCQUERY
18F - BD	Bad entry code passed to CSFMXCFC
18F - BE	Error in ATTACH processing in CSFMXCFC
18F - BF	Error in DETACH processing in CSFMXCFC

### New reason codes for return code x'C'

Table 4-5 shows new reason codes for return code x'C' from the various ICSF calls.

Table 4-5 ICSF HCR7730 new reason codes

Reason Code	Meaning
3005 (x'BBD')	The CKDS I/O Subtask timed out waiting for an exclusive ENQ on SYSZCKDS.ckdsdsn. The operator should issue D GRS,RES=(qname,rname). (See message CSFM302A)
3006 (x'BBE')	XCMMSGO issued from the CKDS I/O Subtask failed after exhausting the maximum retry attempts.
3007 (x'BBF')	CKDS service failed due to unexpected termination of the ICSF Cross-System Services environment.

Archived

## User Defined Extensions (UDX)

The UDX is the facility offered by the S/390, zSeries and System z9 PCI coprocessors, to have users designing and implementing their own cryptographic services to be executed in the PCI or PCI-X card itself. This gives a high level of flexibility to the coprocessor user along with the performance and the protection that the card can provide for the customized algorithm. More detailed information on the UDX development and implementation can be found in *zSeries Crypto Guide Update*, SG24-6870.

This chapter provides information on how to install the UDX code and apply changes to it, in the Crypto Express 2 coprocessor of System z9. It is not intended to give a complete information on UDX management. For the latter topic, refer to *z/OS Cryptographic Services Integrated Cryptographic Service Facility Administrator's Guide*, SA22-7521.

## 5.1 Refresher on the UDX implementation

UDX has been an option since the availability of the PCICC Cryptographic Coprocessor on S/390. The UDX code developed for the PCICC does not operate in the PCIXCC or Crypto Express 2 card. The UDX code developed for the PCIXCC can be used with the Crypto Express 2 card without any changes. If a UDX developed for the PCICC is still needed to execute in the Crypto Express 2 card, then the UDX must be recoded to run in PCIXCC or Crypto Express 2.

**Note:** The UDX customized algorithm is added as specific coprocessor code built by IBM or an approved third party. Building a UDX is an IBM service offering performed under contract. The resulting product of the UDX development process is a customized CD to be loaded into the customer system.

As of the writing of this book there is no plan to provide a means for a customer to build a Crypto Express 2 UDX by himself/herself.

A UDX implementation on zSeries and System z9 requires three parts:

- ▶ An ICSF callable service, which when called by a user application will invoke the UDX code being executed in Crypto Express 2. This is a user-defined host service that matches the specific UDX code running in the card. Applications call the service using a designated number.
- ▶ The Crypto Express 2 UDX code. This is the coprocessor piece installed by the user into the Crypto Express 2 code itself, and intended to be invoked by the ICSF service above. A UDX is designated by a specific two-character identity.
- ▶ A service “stub”, to be link-edited with the application calling the user-defined service for proper dispatching of the service by ICSF.

Additionally, the specific UDX service can be enabled or disabled using access control points managed from the TKE workstation. This makes it necessary to design and install an exit to the CSFPCI (TKE communication) ICSF callable service.

For the Crypto Express 2 card, the set of functions and API at the card level have to be the IBM CCA services and are implemented as a set of “CCA command processors” in the Crypto Express 2 code.

The UDX code to be installed in a Crypto Express 2 card consists of a user-designed command processor that is link-edited into the original coprocessor code during the UDX generation process. When loaded into the coprocessor, a UDX is checked for origin and integrity using a digital signature scheme. That is, the new coprocessor code version, with the UDX integrated, should have been signed by an IBM-approved key.

Note that a customer may elect to have more than one UDX in a single coprocessor. Each UDX will be uniquely referred to using its function identifier.

### UDX Function code identifier

The UDX is in essence another command processor in the coprocessor, like the already existing CCA command processors, which has its own function identifier (the following range of identifiers is reserved for UDX: XA-XZ, X0-X9, YA-YZ, Y0-Y9, WA-WZ, W0-W9).

The function identifier corresponding to an installed UDX is known to the coprocessor after a coprocessor reset. For ICSF, an existing UDX is indicated in the Options Data Set, once the UDX has been installed in Crypto Express 2, by a UDX statement as follows:



```
UDX(UDX-id,service-number,load-module-name,'comment_text',FAIL(failoption))
```

Where UDX-id is the function code that identifies the UDX command processor according to the function code specified during the UDX generation process, service-number is to be used internally to refer to this callable service (see the “stub” in 5.1.1, “The UDX callable service and the stub” on page 69), and load-module-name is the name of the user-defined callable service that is eventually calling the UDX in the Crypto Express 2 card (refer to the *ICSF System Programmer's Guide* for a complete description of the statement).

### 5.1.1 The UDX callable service and the stub

A UDX, not being a base CCA “verb”, requires a specific additional routine in ICSF to provide the service at the application API level. This ICSF callable service in turn manages to call the UDX in Crypto Express 2.

The capability of creating customized services in ICSF (the “installation-defined services”) has been available since the early releases of ICSF. These services are customer-written modules link-edited with ICSF, and identified in the Options Data Set (refer to the *ICSF System Programmer's Guide*) by the SERVICE statement where a unique number (from 1 to 32767, inclusive) is associated to the service load module.

After you write the callable service, you need to link-edit it into a load module, and install the load module into an APF-authorized library. ICSF uses the following normal search order to locate the service:

- ▶ Job pack area
- ▶ Steplib (if one exists)
- ▶ Link pack area (LPA)
- ▶ Link list (SYS1.LINKLIB concatenation)

During ICSF startup, ICSF loads the load module that contains the service into the ICSF address space with the ICSF callable services. ICSF binds the service with the service number that you specified in the installation Options Data Set. To call such an installation-defined service, the application has to actually call an intermediary piece of code called the “stub”. This is depicted in Figure 5-1.

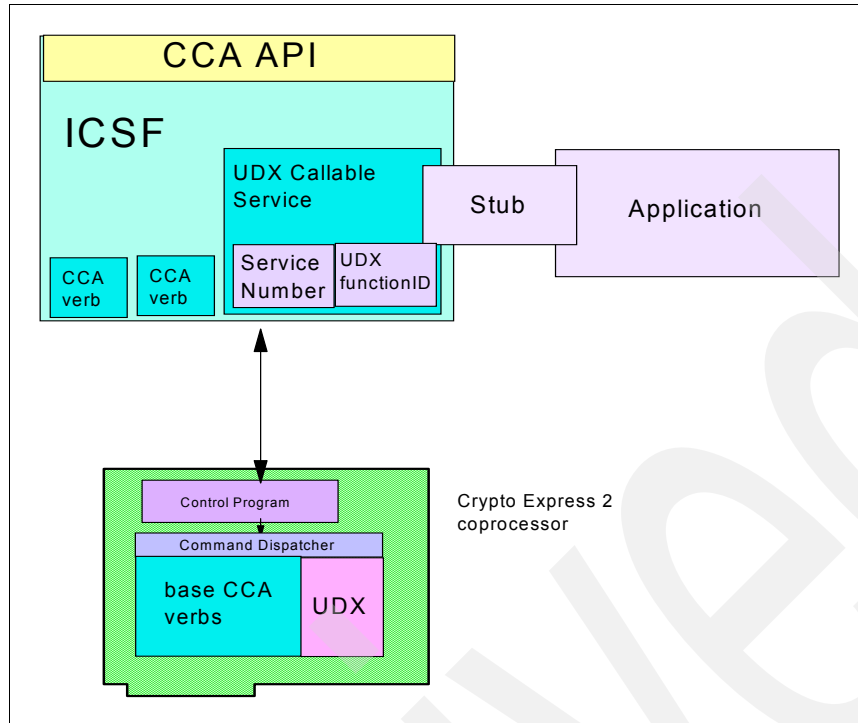


Figure 5-1 UDX implementation in the Crypto Express 2 coprocessor and ICSF

The service stub is also designed and installed by the user and must do the following:

- ▶ Check that ICSF is active.
- ▶ Place the service number for the installation-defined callable service into register 0.
- ▶ Call the IBM-supplied processing routine, CSFAPRPC, which is internally used by ICSF to access the callable services. The stub has to first retrieve the location of CSFAPRPC from the ICSF cryptographic communication vector table (CCVT).

A stub example is given in the *ICSF System Programmer's Guide*.

Any application program that calls a service stub must be link-edited with the service stub. To call an installation-defined service from an application program, use the following statement:

```
CALL <service-stub-name><service-parameters>
```

The service-stub-name is the name of the service stub for the installation-defined callable service. The service-parameters are the parameters you want to pass to the installation-defined service. You supply the parameters according to the syntax of the programming language that you use to write the application program.

This capability is extended to allow interaction with the UDX in Crypto Express 2 by installing a user-defined ICSF module that will request Crypto Express 2 to run the UDX integrated in its code. This "UDX Callable Service" must also have a unique number that is not shared with another UDX or non-UDX installation-defined service, the service number being specified in the UDX statement of the Options Data Set as indicated above. As for a regular installation-defined service, the service invocation is performed via the stub.

Note again that with a UDX callable service the actual service is performed by the UDX code running in the card, whereas with a regular user-defined service the service is provided by the code running in the ICSF address space.

More information can be found in the publication *IBM @server zSeries CCA User Defined Extensions Reference and Guide*, available on the cryptocards Web site at:

<http://www.ibm.com/security/cryptocards>

The Web site will direct your request to an IBM Global Services (IGS) location appropriate for your geographic location. A special contract will be negotiated between IGS and you, covering development of the UDX by IGS per your specifications, as well as an agreed-upon level of the UDX.

## 5.2 The UDX on System z9

We now discuss the installation and activation of UDX.

## 5.3 Initial load and activation of the UDX

This procedure describes how to install and activate UDX functions in the System z9 environment. The installation process is concurrent to the system operations and is divided into two parts:

- ▶ The UDX installation in the Crypto Express 2 coprocessors and the Master Keys (SYMMK and ASYMMK) initialization.

**Important:** The initial UDX installation deletes all Master Keys in each domain of the coprocessor. All TKE settings, roles, authorities, etc. stored in the target coprocessor are also cleared. The TKE commands enablement setting is also set back to “deny”.

- ▶ The UDX activation, which enables applications to access the UDX code.

### 5.3.1 Installation of the UDX

We assume here that there is at least one Crypto Express 2 feature installed in System z9 and the coprocessors have already been initialized, that is, the Master Keys are loaded and the TKE settings properly done.

The installation is done as follows:

1. Deactivate the target Crypto Express 2 coprocessor on *all* logical partitions in the system where the target coprocessor is active:
  - In the ICSF ISPF main panel, select option 1 Coprocessor Management.
  - In the ICSF Coprocessor Management panel, select the target coprocessor using the action character D (Deactivate). The status of the coprocessor should become DEACTIVATED.
2. From the HMC Console or the Support Element (SE) configure the target coprocessor offline (that is, in “standby” status).
3. From the HMC/SE import the UDX code CD via the Cryptographic Configuration panel.
4. From the HMC/SE configure the target coprocessor back on line. It will take approximately between five and ten minutes for the coprocessor to unitize.
5. From the HMC/SE enable the TKE Commands for the target coprocessor.
6. Activate the target Crypto Express 2 coprocessor on *all* logical partitions that require it:

- In the ICSF ISPF main panel, select option 1 Coprocessor Management.
  - In the ICSF Coprocessor Management panel, select the target coprocessor using the action character A (Activate). The status of the coprocessor should switch to ONLINE.
7. Recreate the TKE authorities settings:
- a. On TKE, log on as TKEUSER.
  - b. Open the host where the UDX was just installed. You will be prompted to authenticate the Crypto module.
  - c. Open the target Crypto module (not the group of Cryptos).
  - d. Use the authority 0 default signature key to recreate roles and authorities for the target Crypto module. Usually this should be the same authorities and keys as for the other coprocessors in the system.
  - e. If you are reloading the coprocessor Master Keys from the TKE, make sure that you have an authority enabled for loading Master Keys parts.
  - f. Load the Master Keys parts.
- Note:** If you have more than one target Crypto module, load the key parts in all Crypto modules before setting the master keys.
- g. Set the asymmetric Master Key from the TKE.
  - h. Set the symmetric Master Key in the Crypto Express 2 coprocessor from the ICSF panel. The status of the coprocessor in the Coprocessor Management panel should become ACTIVE.
  - i. If needed, add the target Crypto module to a group by doing a TKE group change.
8. Repeat Step 7, substeps e-h, for each domain being used in the coprocessor.
9. Repeat the same process for the other coprocessors in the system that were receiving a new UDX.

### 5.3.2 UDX activation

There are several steps in the UDX activation process. No specific sequence is required.

- ▶ Enable the UDX access point from TKE - On the TKE, open the target Crypto module, select **domains, controls** and mark the UDX enabled.
- ▶ The UDX host modules, the ICSF stub, and the service module must be installed and the ICSF Installation Options Data Set must be customized in order to specify the available UDX. The following steps have to be performed:
  - a. Link-edit the OBJ file for the UDX callable service into a load module and install it into an APF-authorized library (for example, CSF.SCSFMOD0).
  - b. Link-edit the OBJ file for the service stub into a load module and install it into an APF-authorized library (for example, CSF.SCSFMOD0). Any application program that calls the UDX must be link-edited with the service stub.

**Important:** If a new ICSF level is installed and a new CSF.SCSFMOD0 is created, you should remember to reinstall the UDX callable service and the service stub load modules.

- c. Identify the UDX service in the ICSF Installation Options Data Set by using the UDX keyword. For information about the specification of the UDX keyword, refer to the z/OS

*z/OS ICSF System Programmer's Guide*, SA22-7520. You must specify information including the UDX subfunction code, a service number, and a load module name. For example, the following line could specify the new UDX service:

```
UDX(XA,50,UDXSVC,'ITSO TEST UDX',FAIL(NO))
```

- If the UDX will be invoked by a CICS® transaction, you must update the ICSF CICS Wait List used by the CICS-ICSF Attachment Facility.

For information on how to install the CICS-ICSF Attachment Facility and how to add your UDX to ICSF's CICS Wait List, refer to the *z/OS ICSF System Programmer's Guide*, SA22-7520.

After all above steps are completed, stop and restart ICSF in order for all changes to take effect.

## 5.4 UDX microcode update process

This procedure describes how to load a new version of a UDX and microcode package into the Crypto Express 2 coprocessor. We explain here how to proceed to make the loading concurrent with other coprocessors, if any, still providing services to the applications.

**Important:** The loading process that we are describing here requires that all logical partitions in the system that need cryptographic services have access to at least two coprocessors.

These microcode updates of the UDX package will *not* clear any Master Keys or TKE authorities settings and there is no need to deactivate and reactivate the logical partition.

The installation of the new microcode level together with a UDX into coprocessors is usually done by an IBM Customer Engineer. The update process usually takes about 30 minutes.

Here are the steps we recommend so that there are always cryptographic services available to the applications:

1. Deactivate one target coprocessor through the ICSF ISPF COPROCESSOR MANAGEMENT. If there are several logical partitions using this coprocessor, it must be deactivated in every logical partition.
2. The target coprocessor is configured offline (that is, put in the “standby” status from the SE or the HMC). The microcode combined with a UDX will replace the current code in the coprocessor. This is done through the Cryptographic Configuration panel on the SE. When the UDX status becomes active, the target coprocessor can be configured back to online. This triggers the loading process of the new microcode, which takes about five to ten minutes. The coprocessor actually switches to the online status at the completion of this loading.
3. When the loading process above is completed and the coprocessor is online, then the next step is to activate the target coprocessor in the ICSF Coprocessor Management panel. The status of the target coprocessor should become ACTIVE in the ICSF panel. This is also a proof that all Master Keys are valid and the coprocessor operates correctly.

**Note:** If the Master keys have been cleared, the microcode update process should be stopped and an investigation should be carried out to find the error and to correct the situation.

4. When the first target coprocessor with the new microcode containing a UDX is available, a verification test should be run against the UDX to verify that the UDX still works as expected.
5. When the previous steps have been completed successfully for the target coprocessor, the same process can be executed against the next coprocessor to receive the change.

**Important:** Special considerations should be given to the following:

- ▶ The scheduling of the microcode update, so that the cryptographic capacity available while updating each coprocessor fits the running applications' requirements.
- ▶ The time allowed to the execution of the UDX verification test so that only a verified UDX can be switched to production.

## TKE V5.0 overview and setup

In this chapter we provide an overview of the Trusted Key Entry (TKE) workstation version 5.0. V5.0 is the follow-on to the TKE V4.2 level.

V5.0 is the level required to support System z9 coprocessors. TKE V5.0 also differentiates itself from previous versions by implementing a new IBM proprietary operating system in place of the IBM OS/2® Operating System, and by using a new cryptographic adapter.

Several other IBM Redbooks address, from a broader perspective, cryptographic services implementation in the S/390 and zSeries systems, as well as the use of the Trusted Key Entry (TKE) workstation in this context. We recommend that readers who are unfamiliar with the TKE concept and implementation refer to *zSeries TKE V4.2 Update*, SG24-6499, for further background information.

For further information about TKE V5.0, refer to *z/OS Cryptographic Services ICSF Trusted Key Entry PCIX Workstation User's Guide*, SA23-2211, and *Maintenance Information for Desktop Consoles*, GC28-6847.

## 6.1 About the TKE workstation

The TKE workstation is a priced optional feature used for management of zSeries and System z9 secure coprocessors in an installation. Secure coprocessors operate with a Master Key that resides inside the coprocessor itself.

These secure coprocessors use application keys that are protected by being encrypted with the Master Key. The application keys are only decrypted inside the coprocessor secure enclosure.

TKE provides secure remote key management capabilities for the coprocessors' Master Keys and operational keys by communicating directly with the coprocessors, using encrypted and digitally signed communications. The coprocessors execute commands entered at the workstation by security officers who have been authenticated using RSA public cryptography, and who are authorized to issue TKE commands.

The TKE workstation communicates, via TCP/IP, with one or several ICSF instances known as the "TKE hosts". These ICSF instances act just like a TCP/IP listener, and they relay the communication to the coprocessors themselves. A single TKE can be used to manage secure coprocessors in the systems it has TCP/IP connectivity to. Secure coprocessors can also be logically grouped at the TKE level so that the TKE application will automatically propagate a single command to all coprocessor members of the group.

Figure 6-1 shows a schematic view of TKE workstation implementation.

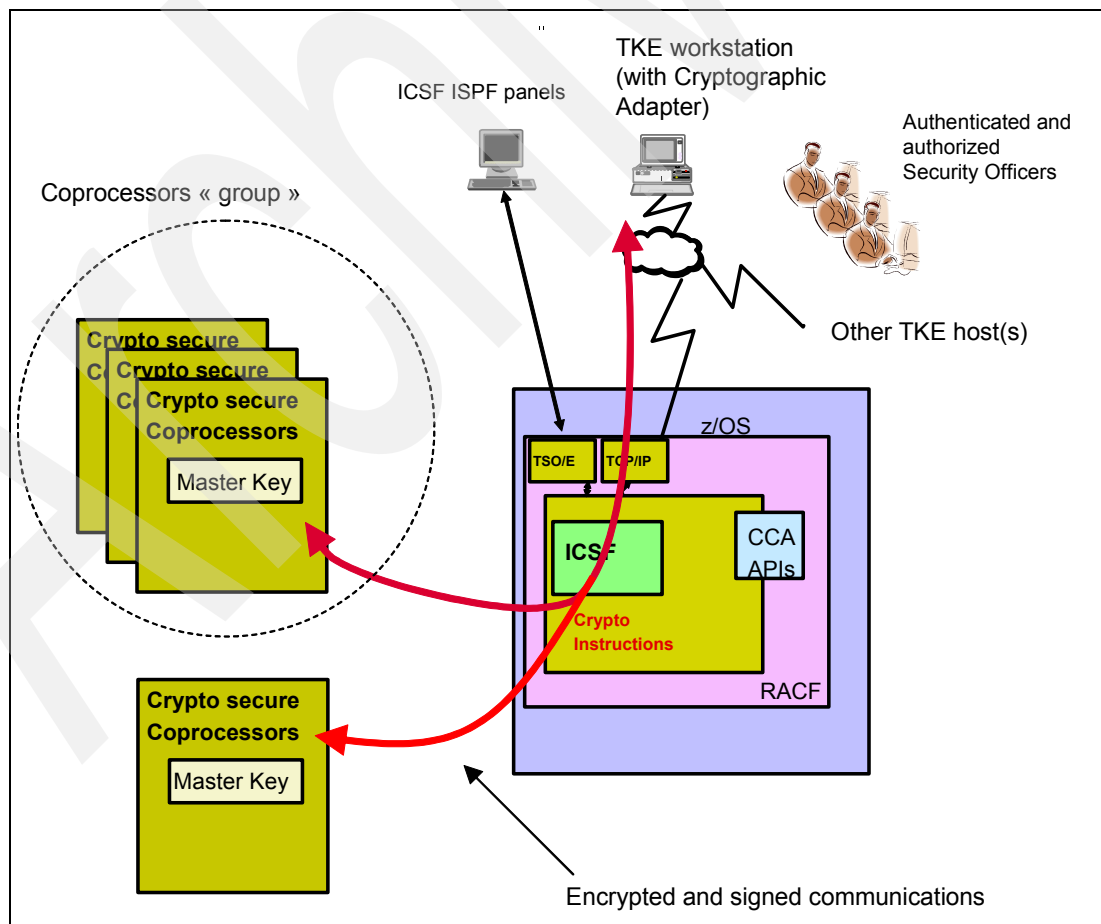


Figure 6-1 TKE workstation implementation



The following coprocessors are used in zSeries and System z9:

- ▶ The CP Assist for Cryptographic Functions (CPACF) - z990, z890 and System z9 only  
Note that this is not an actual coprocessor, but rather a hardware facility imbedded in the system's processing units (PUs). The CPACF does not use a Master Key.
- ▶ The PCI Cryptographic Accelerator (PCICA) - all zSeries models (FC 0862)  
One PCICA feature contains two coprocessors. The PCICA does not use a Master Key.
- ▶ The PCI Cryptographic Coprocessor (PCICC) - secure coprocessor, on z900 and z800 and 9672 G5/G6 (FC 0860/0861)  
One PCICC feature on z900 and z800 contains two coprocessors at 16 domains each.
- ▶ The Cryptographic Coprocessor Facility (CCF) - secure coprocessor, on z900 and z800 and 9672 (FC 0800)  
There are two CCFs available (except for uniprocessor models). One CCF has 16 domains.
- ▶ The PCI Cryptographic Coprocessor (PCIXCC) - secure coprocessor, on z990 and z890 only (FC 0868)  
One PCIXCC feature contains one coprocessor with 16 domains.
- ▶ The Crypto Express2 coprocessor (CEX2C) - secure coprocessor, z990, z890 and System z9 only (FC 0863)  
The Crypto Express2 feature contains two coprocessors at 16 domains each.  
With System z9, the Crypto Express 2 coprocessor can be manually configured into an accelerator (CEX2A). A CEX2A does not use a Master key.

The term "crypto modules" is also used to designate the coprocessors, in TKE terminology.

**Important:** The TKE workstation is not supported on systems with CPACF only, or with CPACF and PCICA only, or with CPACF and CEX2A only, because these configurations do not include at least one secure coprocessor.

## 6.2 TKE V5.0 overview

The Trusted Key Entry (TKE) V5.0 workstation is the follow-on to the TKE V4.2 workstation. It is functionally equivalent to TKE V4.2. However, its implementation differs in two ways:

- ▶ TKE V5.0 uses a 4764 Cryptographic adapter—replacing the 4758-2 cryptographic adapter of TKE V4.2.
- ▶ The OS/2 operating system has been replaced by an IBM proprietary operating system.

As with TKE V4.2, V5.0 supports two smart card readers.

### 6.2.1 TKE V5.0 hardware

The basic workstation hardware consists of the following elements:

- ▶ The TKE workstation itself, with FC 0859.  
The workstation we used for this residency was an IBM eserver xSeries® 206, model 8482.
- ▶ The IBM 4764 Cryptographic Adapter, which is delivered separately from the workstation itself but is included in the workstation order.

The IBM 4764 Cryptographic Adapter supports a broad range of DES and public-key cryptographic processes. It is the TKE workstation engine and has key storage for DES and PKA keys. The IBM 4764 implements the same concepts as its IBM 4758 predecessor regarding user roles and profiles and TKE access control points.

- ▶ The TKE workstation supports connectivity to an Ethernet Local Area Network (LAN) operating at 10, 100, or 1000 Mbps.

**Notes:**

- ▶ There is no hardware upgrade path from TKE versions previous to 5.0. Upgrading to V5.0 implies that you need to obtain a new TKE workstation.
- ▶ A TKE V5.x order for a System z9 can comprise up to three workstations (in contrast to one TKE only for the other systems).

**Optional features**

The TKE 5.0 workstation allows for the attachment and use of two smart card readers, with smart cards that contain an embedded microprocessor and associated memory for data storage that can hold secret keys values. Access to and the use of confidential data on the smart cards is protected by a user-defined Personal Identification Number (PIN).

- ▶ TKE Smart Card Reader (FC 0887) - contains two card readers and 20 smart cards.
- ▶ TKE additional smart cards (FC 0888) - contains 10 additional smart cards.
- ▶ The Smart Card Reader, which can be attached to a TKE workstation with the 5.0 level of LIC, is available on System z9, z990, z890, z900, and z800.

## 6.2.2 TKE software levels

**ICSF host software**

- ▶ z/OS V1R3 and OS/390 Release 10 require APAR OW44816 and APAR OW46381 to be installed.
- ▶ z/OS V1R3 and higher and OS/390 Release 10 without the z990 Cryptographic Support Web deliverable require APAR OW53666 to be installed.
- ▶ The z990 Cryptographic CP Assist support requires FMID HCR7708 or later.
- ▶ The z990 PCI X Cryptographic Coprocessor support requires FMID HCR770A or later.
- ▶ The z990 and z890 Crypto Express2 Coprocessor support requires FMID HCR7720 or toleration APAR OA09157 on FMID HCR770A and HCR770B.
- ▶ Systems with FMID HCR770A and below require APAR OA07393.
- ▶ To use TKE 4.1 or higher to load operational keys, you must be running HCR770B or higher.

**Note:** If the TKE host is at z/OS 1.7 or above, then the TKE user can log on to the ICSF host using mixed-case passwords. (This assumes that RACF, or an equivalent product, has been set up in the host to not fold passwords to upper case.)

**Host data sets required by TKE**

When TKE connects for the first time, ICSF allocates (if it does not already exist) a hlq.TKECM data set that the TKE application uses to keep the crypto module descriptions, domain descriptions, and authority information for a host.

If the TKECM data set becomes unusable (for example, by becoming deleted or corrupted), it will be necessary to redefine the crypto module and authority information to continue using TKE. Additionally, certain upgrades in the TKE code or the host system may require a new TKECM data set to be generated. Under these conditions, refer to the Migration section instructions in the *z/OS Cryptographic Services ICSF Trusted Key Entry PCIX Workstation User's Guide*, SA23-2211 for detailed information.

### **TKE workstation software**

The following software is preinstalled on the TKE workstation:

- ▶ IBM Cryptographic Coprocessor Support Program Release 3.10SC, is used to drive and manage the IBM 4764 Cryptographic Adapter.
- ▶ Trusted Key Entry version LIC V5.0, which is the TKE “application”, running above the operating system.
- ▶ The new TKE V5.0 operating system.

**Note:** TKE software should not be changed without instructions from IBM service.

## **6.2.3 TKE V5.0 installation**

For the information you need to install and maintain the TKE workstation and the 4764 Cryptographic Adapter, refer to *Maintenance Information for Desktop Consoles*, GC28-6847.

## **6.2.4 TKE V5.0 use**

For detailed information about using TKE V5.0, as compared to the previous version, refer to 6.3, “TKE V5.0 functions compared to TKE V4.2” on page 82.

### **TKE enablement at the Support Element for z990, z890 and System z9**

If you have a z890 or z990 system with May 2004 or a later version of Licensed Internal Code (LIC) installed—or a System z9 with MCL 029 Stream J12220 or later installed—TKE commands must be permitted on the Support Element before any commands issued by the TKE workstation can be executed.

This is a requirement beginning with the May 2004 Licensed Internal Code. The default setting for TKE commands is Denied.

### **Systems supported by TKE V5.0**

The TKE V5.0 is available on systems z800, z900, z890, z990, and System z9.

## **6.2.5 Migrating from previous TKE versions**

The steps needed to migrate from previous TKE versions are detailed in *z/OS Cryptographic Services ICSF Trusted Key Entry PCIX Workstation User's Guide*, SA23-2211. In the following section we list the main considerations to keep in mind when migrating to TKE V5.0.

### **Migration from TKE V2 to TKE V5.0**

- ▶ TKE V2 Personal Security Cards (PSC) are not compatible with the TKE V5.0 smart card readers.
- ▶ TKE V2 uses APPC connections, which must be replaced by TCP/IP connectivity.

- ▶ The TKECM data set at TKE V2 is not compatible with TKE V5.0. TKE V2 uses a TKEFLAGS data set, which is not needed by later versions.
- ▶ TKE V2 authority signature keys are handled in different ways, as described here.
  - If they are in the PSCs, and you do *not* intend to use smart card with TKE V5.0:  
On TKE V2, change the signature keys, load the new public keys to the host and save the new private keys to diskettes for use with the TKE V5.0.
  - If they are in the PSCs, and you *do* intend to use smart cards with TKE V5.0:  
On TKE V2, change the signature requirements in the host so that authorities with signatures in the PSCs are not used anymore. When on TKE V5.0, generate new signature keys and save them on smart cards.
  - If they are in the workstation PKA Key Storage:  
No migration is possible from the TKE V2 IBM 4755 Cryptographic Adapter. Proceed as described, generating new signature keys.
- ▶ The Master and Operational Key Parts are also handled in different ways, as described here.
  - In binary files on the TKE V2 hard disk:  
Transfer the key parts to the TKE V5.0 via diskettes.
  - Entered via the keyboard:  
Reenter on the TKE V5.0 keyboard.
  - Saved on PSCs:  
On TKE V2, the data block in the PSCs has to be copied to a binary file by using the HIKM utility. Transfer the binary files to TKE V5.0.

### **Migrating from TKE V3 or TKE V4 to TKE V5.0**

Follow these steps to migrate from TKE V3 or TKE V4 to TKE V5.0.

- ▶ On the TKE V3, back up the TKE configuration data.  
This includes the user-defined 4758 roles and profiles, Authority Signature Keys, Master key parts (host and/or 4758) and Operational Key parts (if saved onto the TKE hard drive), TKE Host and Group Definitions, DES and PKA Key storages, 3270 emulator data and TCP/IP information.
- ▶ On the TKE V5.0, the “Migrate previous TKE version to TKE 5.0 Task” is used to restore this information.

#### ***TKECM data set***

The TKECM data set at V3.x or V4.x is no longer compatible with TKE V5.0 if the system is also being migrated from z900 to z990 or System z9, or from z800 to z890.

### **Other migration considerations**

There are additional considerations to keep in mind when migrating to TKE V5.0, as described here.

In order to be able to use migrated DES and PKA Key Storages, you must load the cryptographic adapter master key parts from your previous TKE workstation to the TKE 5.0 workstation.

If you do not know the Master Key parts that were used for the previous TKE, then the migrated DES and PKS storages will not be usable. You will need to reinitialize both key

storages. Any keys in DES Key Storage and the Authority Signature Key in PKA Key Storage will need to be recreated, as appropriate, using TKE 5.0.

## **TKE access control points**

Note the following version-specific information regarding TKE access control points.

### ***TKE services access control points to be added to roles***

For TKE V4.1 customers only:

- ▶ If you have any user-defined profiles that you want to continue to use to log on to the cryptographic adapter to use TKE, you must add several new access control points to the roles that the profiles are mapped to. The new access control points are:
  - X'8002' - TKE Logon
  - X'0250' - Load Diffie-Hellman key mod/gen
  - X'0251' - Combine Diffie-Hellman key parts
  - X'0252' - Clear Diffie-Hellman key values
  - X'027A' - Unrestricted Combine key parts
- ▶ If you have defined roles for TKE administrator functions, you must add new access control points:
  - X'030B' - Reset battery low indicator
  - X'0107' - One-Way Hash SHA-1

For TKE V4.2 customers only:

- ▶ If you have defined roles for TKE administrator functions, you must add new access control points:
  - X'0107' - One-Way Hash SHA-1

### ***ICSF callable services access control points***

For TKE V3.0 customers only:

- ▶ This section is only applicable to customers whose workstation was at TKE V3.0 prior to the upgrade to TKE V5.0, and whose configuration includes PCI cryptographic coprocessors.

An authorized TKE Authority must enable all applicable access control points for ICSF callable services. APAR OW46381 must be installed on z/OS V1R1 and OS/390 V2R10 systems; otherwise, access control failures could occur.

For z890 or z990 TKE V4.X customers only:

- ▶ This section is only applicable to z890 or z990 customers whose workstation was at TKEV4.x prior to the upgrade to TKE V5.0. If you are upgrading your ICSF level, an authorized TKE Authority must enable all applicable new access control points.

For IBM System z9 109 customers only:

- ▶ An authorized TKE authority must enable or disable access control points, as appropriate.

### ***Smart card readers and smart cards***

Smart card readers attached to TKE V4.2 can be moved to TKE V5.0. A CA card built on TKE V4.2 can be used on TKE V5.0 to enroll the cryptographic adapter in the same zone as the cryptographic adapter in TKE V4.2. This will allow the TKE smart card created on the TKE V4.2 to be usable on the TKE V5.0. TKE smart cards can contain authority keys, master key parts, operational key parts, and so on.

## 6.3 TKE V5.0 functions compared to TKE V4.2

There are no functional differences between TKE V5.0 and TKE V4.2 from the perspective of TKE application and host coprocessor management. However, the user operating framework has been changed and all processes are panel-driven. The main changes, as perceived by the user, pertain to the use interface and are related to the following areas:

- ▶ Framework presentation
- ▶ Invocation of TKE-related tasks
- ▶ Backing up TKE-related data
- ▶ File Chooser
- ▶ No command line support
- ▶ Editing TKE files
- ▶ Handling media devices

In the following section we provide an overview of the principles of operation of TKE V5.0. Refer to *z/OS Cryptographic Services ICSF Trusted Key Entry PCIX Workstation User's Guide*, SA23-2211, for more detailed information.

### 6.3.1 Navigation

On startup, the TKE Console is automatically started with the Welcome panel shown in Figure 6-2.

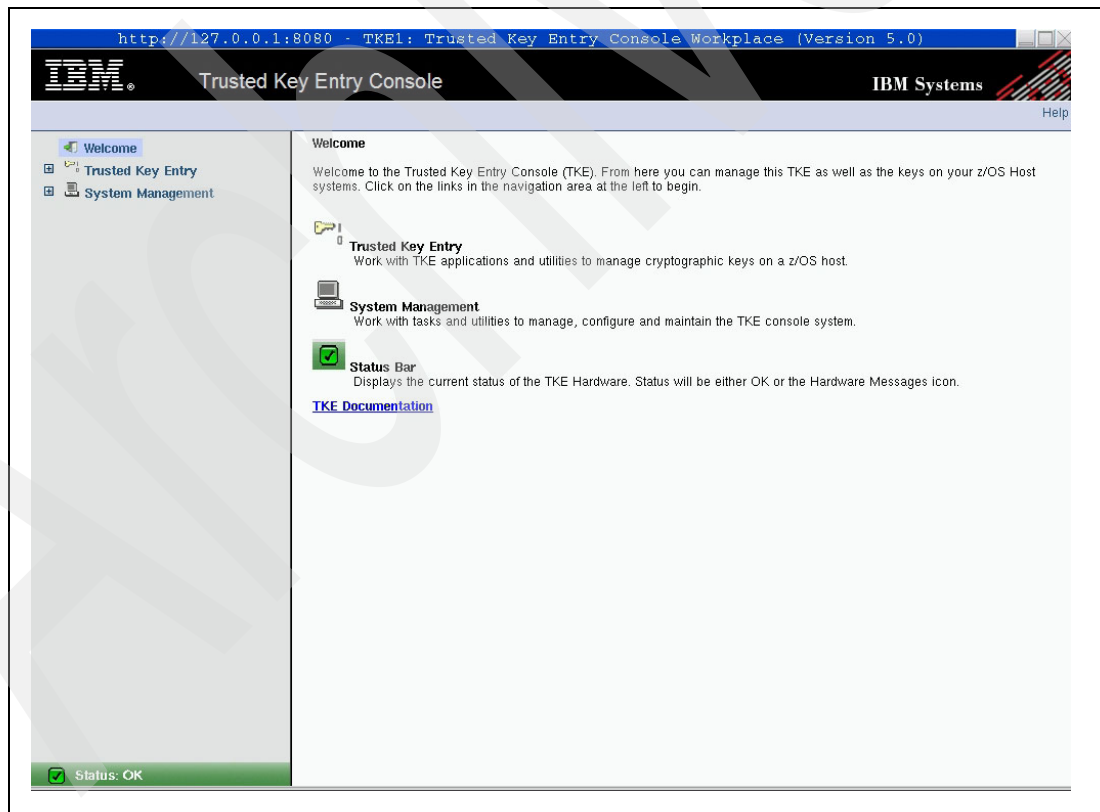


Figure 6-2 Welcome page

The TKE Console displays a tree view on the left side of the Welcome page, which is used for navigation between tasks (Trusted Key Entry, System Management). The right side of the page displays a brief description of the Framework in the tree view, as well as a link to TKE

Documentation. Expanding the Trusted Key Entry and System Management tasks displays subtasks where you can access the various TKE Applications, Utilities, and support tasks.

## Trusted Key Entry tasks

The Trusted Key Entry task is subdivided into the Applications and the Utilities subtasks.

### *The Applications subtask in the Trusted Key Entry task*

The Applications subtask contains the primary TKE applications, as shown in Figure 6-3.

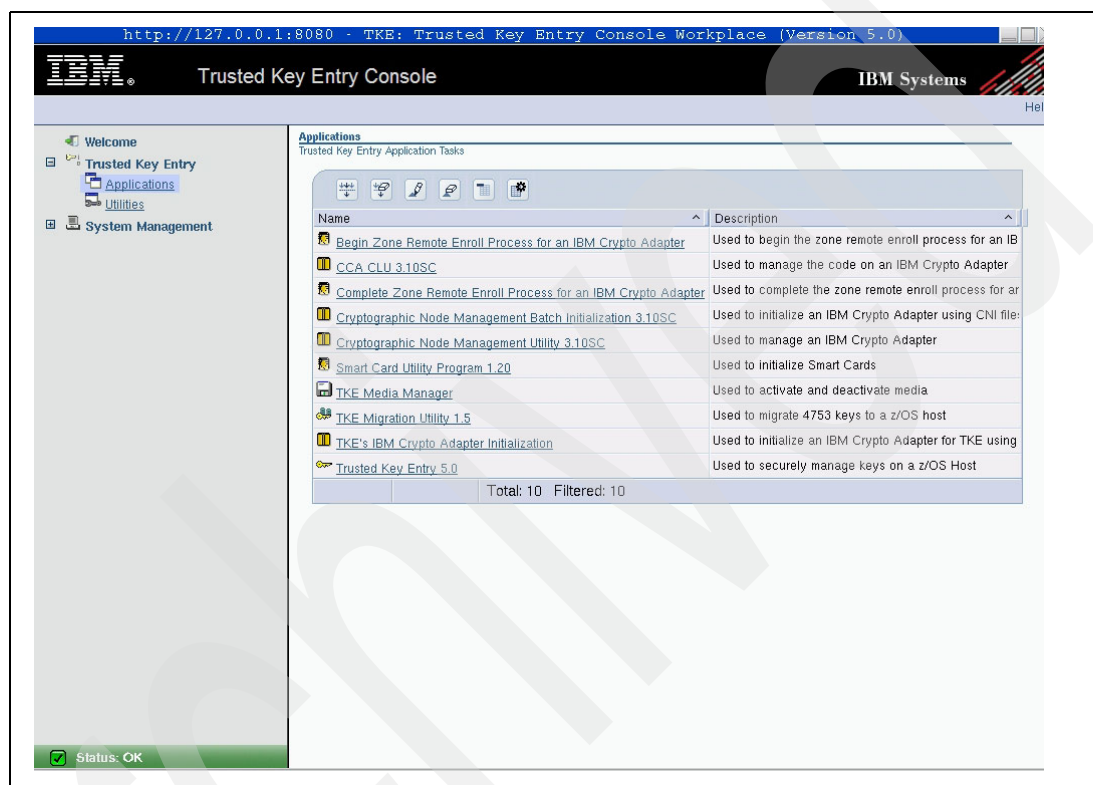


Figure 6-3 Applications Tasks view

The TKE Applications Tasks view displays several options which, on previous TKE versions, were executed by issuing commands in the OS/2 command window.

Following are commonly used options:

- ▶ Begin Zone Remote Process and Complete Zone Remote Enroll Process are related to the enrolling of remote or offsite TKE workstations when using the Smart Card option. These options were achieved on TKE4.2 by using ENROLL\_REQ.CMD and ENROLL\_INST.CMD from the OS/2 command prompt.
- ▶ Cryptographic Node Management Utility is used to manage Crypto adapter issues such as TKE workstation profiles and roles. This process is the same as the task which was started in TKE V4.2 by clicking the CNM icon in the Trusted Key Entry folder.
- ▶ Smart Card Utility Program is used for maintenance operations, such as the creation, initialization, and personalization of CA and TKE smart cards. This process is the same as the task which was started in TKE V4.2 by clicking the SCUP icon in the Trusted Key Entry folder.

- ▶ TKE Media Manager is a new feature which is used to activate and deactivate media drives (floppy or CD/DVD) to be used by TKE Applications and Utilities.
- ▶ TKE's IBM Crypto Adapter Initialization is used to initialize the IBM 4764 Crypto adapter in the TKE workstation.

This function was performed for the 4758 of TKE V4.2 by issuing the CSUECNI command in an OS/2 window; refer to "Cryptographic Adapter initialization" on page 89 for more information.

- ▶ Trusted Key Entry 5.0 is used to start the actual TKE application.

This function remains the same as for TKE4.2

### ***The TKE Utilities subtask in the Trusted Key Entry task***

The TKE Utilities subtask contains common utilities to support TKE, as shown in Figure 6-4.

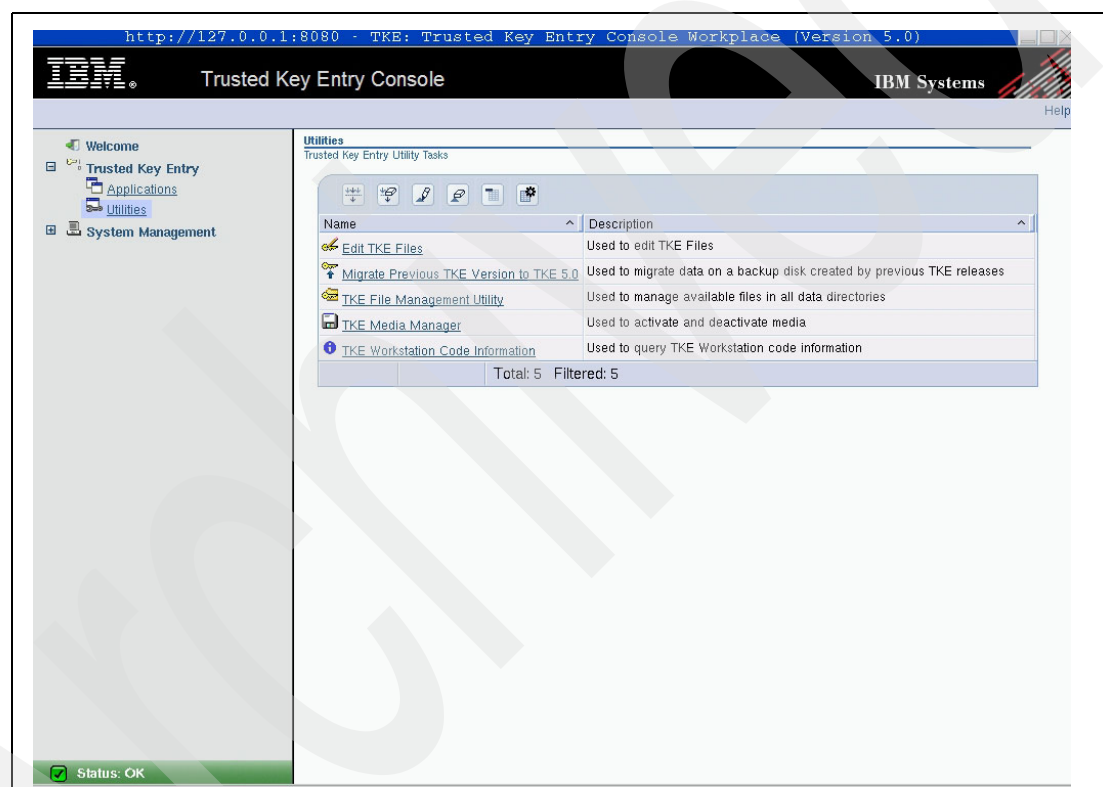


Figure 6-4 Utilities tasks view

Following are commonly used options:

- ▶ Edit TKE files utilities provide the only way to create and view files on TKE V5.0.
- ▶ TKE File Management Utility is a new function to copy and manage files stored into local hard drive, floppy drive and CD/DVD drive.
- ▶ TKE Media Manager is a new feature which is used to activate and deactivate media drives (floppy and CD/DVD) to be used by TKE Applications and Utilities.

### **System Managements task**

The System Management task is subdivided into the Service Applications, Configuration, and Maintenance subtasks.



### ***The Service Applications subtask in the System Management task***

The Service Applications tasks are selected in the view shown in Figure 6-5.

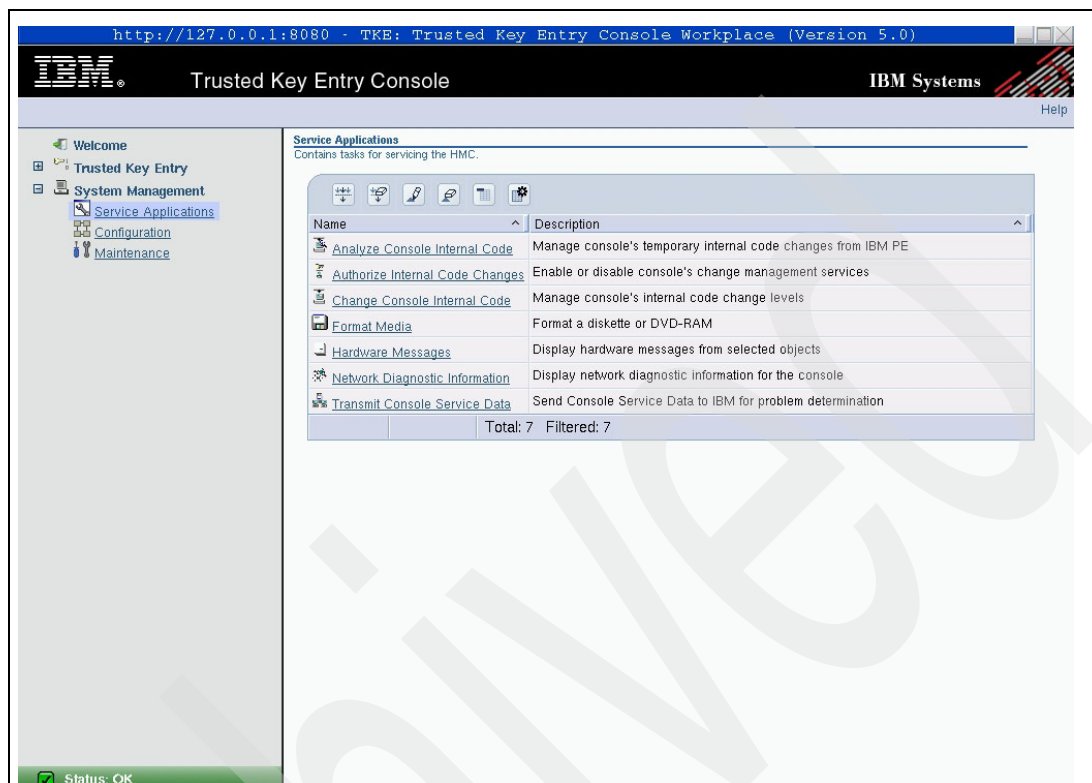


Figure 6-5 Service Applications tasks view

Following are commonly used options:

- **Format Media** is a new function used to format either floppy disk or DVD-RAM, which can then be used copy files to and from the TKE workstation.  
This function was performed in TKE V4.2 by issuing commands in the OS/2 command prompt window.
- **Network Diagnostic Information** is a new function used to diagnose connection problems. Refer to “Diagnosing network problems” on page 94 for further information.

### ***The Configuration subtask in the System Management task***

The configuration tasks can be selected from the view shown in Figure 6-6.

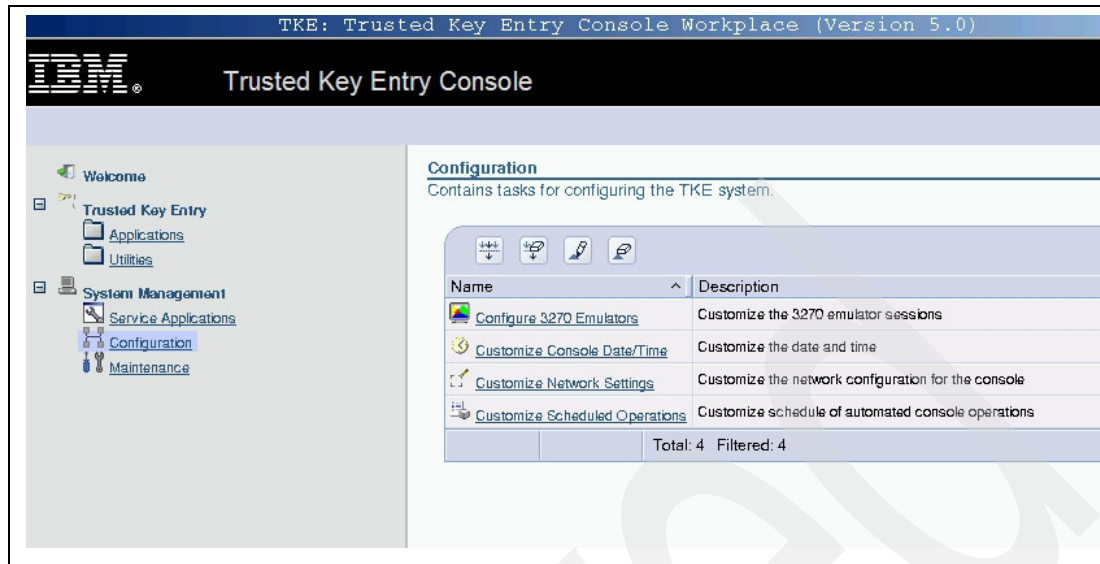


Figure 6-6 Configuration tasks view

Following are the options presented in this view:

- ▶ Configure 3270 Emulators is used to define the 3270 emulator definitions.  
This function was performed on TKE V4.2 by double-clicking the TKE 3270 Emulator icon on the OS/2 desktop.
- ▶ Customize Console Date/Time is a new function used to set up date and time.  
This function was performed on TKE V4.2 by issuing the time command in the OS/2 command prompt window.
- ▶ Customize Network Settings is a new function used to define TCP/IP settings for the TKE workstation.  
This function was started in the TKE V4.2 by double-clicking the TCP/IP Configuration icon on the OS/2 desktop.
- ▶ Customize Scheduled Operations allows you to schedule automatic backups of critical console data, including patches applied and changes to TKE-related information.

### ***The Maintenance subtask in the System Management task***

The Maintenance tasks can be selected from the view shown in Figure 6-7.

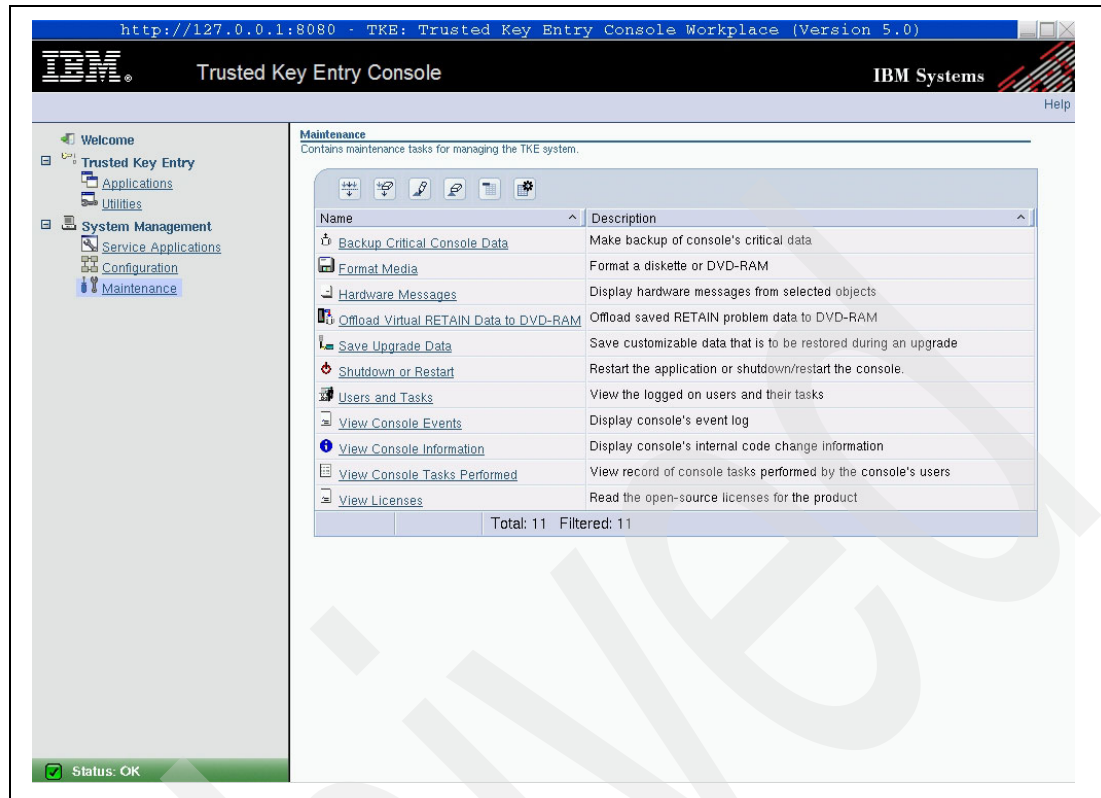


Figure 6-7 Maintenance tasks view

Following are commonly used options:

- Format Media is the same option as on the Service Applications view.
- Save Upgrade Data allows you to save information, including TKE-related data, TCP/IP data, and emulator session data, to the hard drive or DVD for use during an upgrade to a new level of TKE code.

**Important:** Although both options (save to the hard drive, and save to DVD) are available for saving the upgrade data, data can only be *restored* from the hard drive. Therefore, save to DVD should not be used.

- Shutdown or Restart allows you to shut down the TKE workstation or restart the TKE Console.

## 6.4 TKE V5.0 installation and setup

During TKE installation, the IBM CE installs the TKE cryptographic adapter into the TKE workstation and then powers it up. Refer to *Maintenance Information for Desktop Consoles*, GC28-6847, for more information about this topic.

Note the following constraints regarding the operation and storage environment:

- For reliable TKE operation, the installation area ambient temperature must be in the range of 10 degrees Celsius to 40 degrees Celsius, plus or minus 5 degrees Celsius.

- For TKE storage, the storage area ambient temperature must be in the range of 1 degree Celsius to 60 degrees Celsius, plus or minus 5 degrees Celsius. In addition, the ambient relative humidity must not exceed 80 percent.

The TKE administrator is responsible for setting up other TKE definitions (such as setting up the time, TCP/IP definitions, and Cryptographic Adapter initialization) before TKE workstation customization can begin.

### 6.4.1 Setting TKE workstation time

To set the system clock on your TKE workstation, open the Customize Console Date/Time task under System Management, Configuration.

Setting the clock to Local or UTC:

- **Local** sets the time to the current time of the time zone that you selected. Choose a city from the list that has the same time as the one you need. Although the TKE workstation works on the local time zone, the Cryptographic adapter uses UTC internally.
- **UTC** sets the time to the Greenwich Mean Time (GMT) regardless of which time zone you have chosen.

A time is required for your local system operation. Enter either the local time or the UTC time. Specify the new time using the same format as shown in the Time field (for example, 11:40:45 AM).

Specify the new date using the same format as shown in the Date field (for example, Aug 28, 2005). Press Customize when finished.

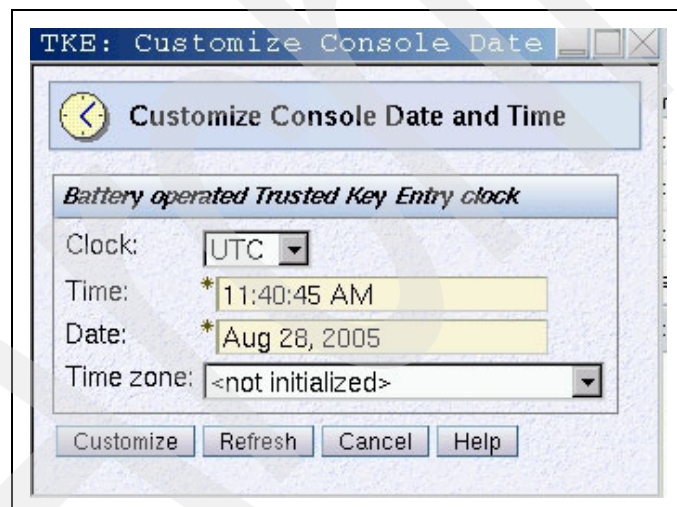


Figure 6-8 Setting the time for TKE workstation hardware

#### Notes:

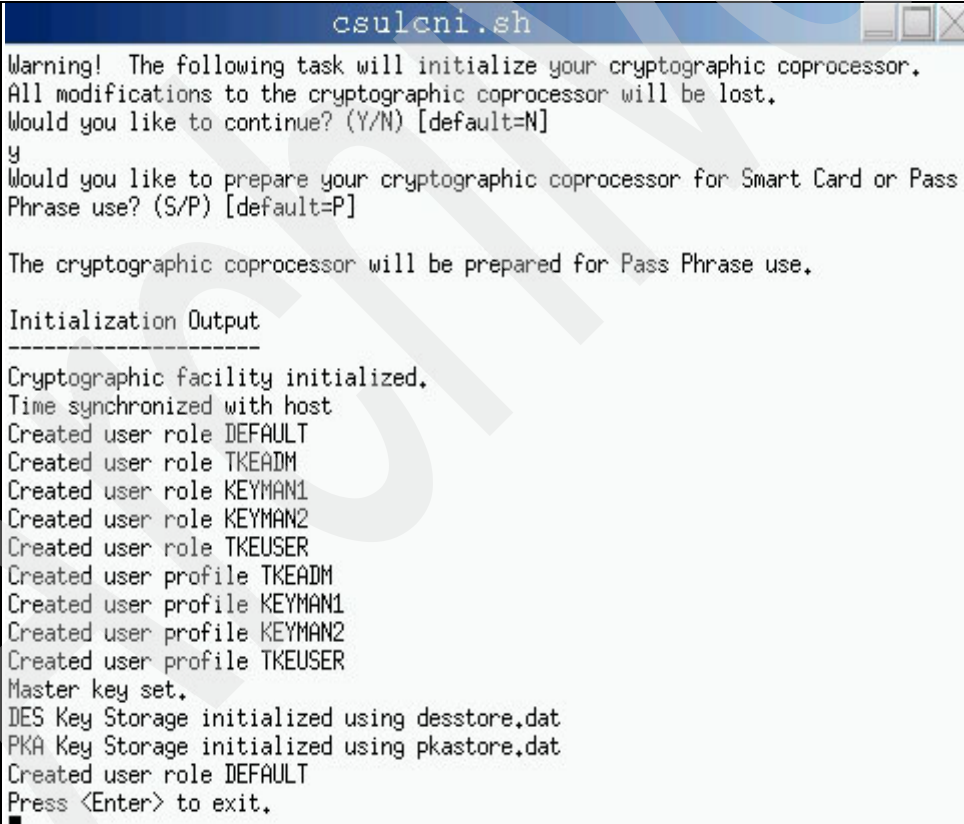
- ▶ If the time on the workstation and the time on the Cryptographic Adapter card are different by more than 5 minutes, an error will be seen when you attempt to log on to the Cryptographic Adapter.

You can correct this situation by using CNM to read the time on the Cryptographic Adapter (it is not necessary to log on to the adapter to read the time), and then set the clock on the workstation within the 5-minute time difference, allowing for GMT if applicable.

- ▶ Another option is to set the clock on the workstation and then reinitialize the Cryptographic Adapter. However, this option will return all settings to the default and any customer changes that had been performed will be lost.

## 6.4.2 Cryptographic Adapter initialization

To initialize the Cryptographic Adapter, open the task under **Trusted Key Entry - Applications- TKE's IBM Crypto Adapter initialization**. You must select whether the initialization is done for Passphrase or Smart Card usage. The resulting display for Passphrase initialization is shown in Figure 6-9.



```
csulcni.sh
Warning! The following task will initialize your cryptographic coprocessor.
All modifications to the cryptographic coprocessor will be lost.
Would you like to continue? (Y/N) [default=N]
y
Would you like to prepare your cryptographic coprocessor for Smart Card or Pass
Phrase use? (S/P) [default=P]

The cryptographic coprocessor will be prepared for Pass Phrase use.

Initialization Output
-----
Cryptographic facility initialized.
Time synchronized with host
Created user role DEFAULT
Created user role TKEADM
Created user role KEYMAN1
Created user role KEYMAN2
Created user role TKEUSER
Created user profile TKEADM
Created user profile KEYMAN1
Created user profile KEYMAN2
Created user profile TKEUSER
Master key set.
DES Key Storage initialized using desstore.dat
PKA Key Storage initialized using pkastore.dat
Created user role DEFAULT
Press <Enter> to exit.
```

Figure 6-9 Initializing Cryptographic Adapter using default CNI with Passphrase option

The Cryptographic Adapter can be initialized by using the Cryptographic Node Management batch interface. Open the task under **Trusted Key Entry- Applications- Cryptographic Node Management batch interface**. Then select which script is to be executed, as shown in Figure 6-10.

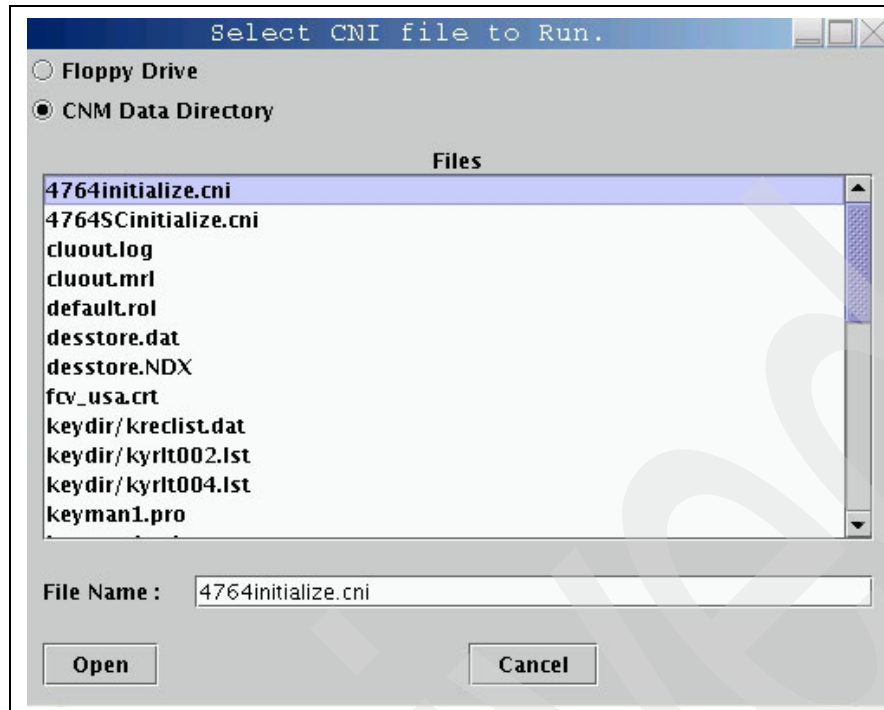


Figure 6-10 Initializing Cryptographic card with Passphrase option

We selected 4764initialize.cni to initialize the Cryptographic Adapter using Passphrase option, and then clicked Open. The result is shown in Figure 6-11.

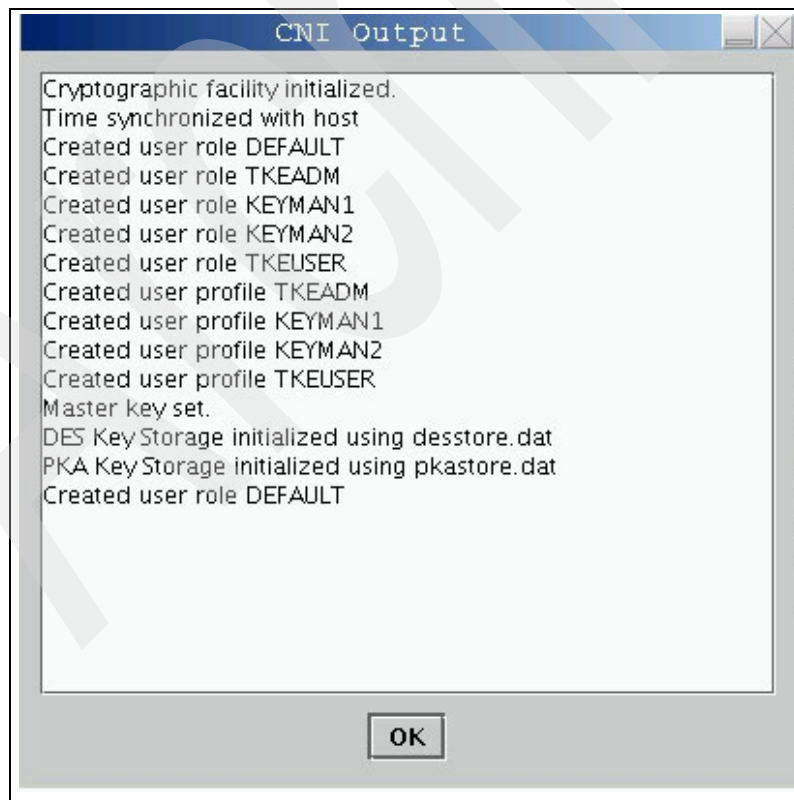


Figure 6-11 TKE workstation Cryptographic Adapter is initialized

### 6.4.3 Cryptographic Node Management and Smart Card Utility Program

The Cryptographic Node Management (CNM) utility is a Java™ application that provides a graphical user interface to initialize and manage the TKE cryptographic adapter. It is part of the IBM Cryptographic Coprocessor CCA Support Program.

**Note:** Smart Card and Smart Card Group options in the CNM panels will only be available if the smart card option has been previously chosen when invoking the Cryptographic Adapter initialization process.

Smart Card support is enabled in CNM under the File option: Enable Smart Card Readers. In order for the change to become effective, you must close CNM and then reopen it.

The CNM utility works functionally on TKE V5.0 the same way as on TKE V4.2. CNM is used to define—among other options—roles and profiles to access CNM and the TKE application using Passphrase or Smart Cards, with the possibility of using the group logon feature.

The TKE Smart Card Utility Program (SCUP) support to the smart card system works functionally on TKE V5.0 the same way as on TKE V4.2. The SCUP supports following functions:

- ▶ Initialize and personalize the CA smart card
- ▶ Backup the CA smart card
- ▶ Initialize and enroll TKE smart cards
- ▶ Personalize TKE smart cards
- ▶ Display smart card information
- ▶ Enroll the TKE cryptographic adapter
- ▶ Unblock a TKE smart card
- ▶ Change PIN number

### 6.4.4 TCP/IP setup

To configure TCP/IP for the TKE workstation, open the task under **System Management** → **Configuration**. In the right frame of the Trusted Key Entry Console, click **Customize Network Settings**; a window will open, as shown in Figure 6-12.

The default host name is TKE.



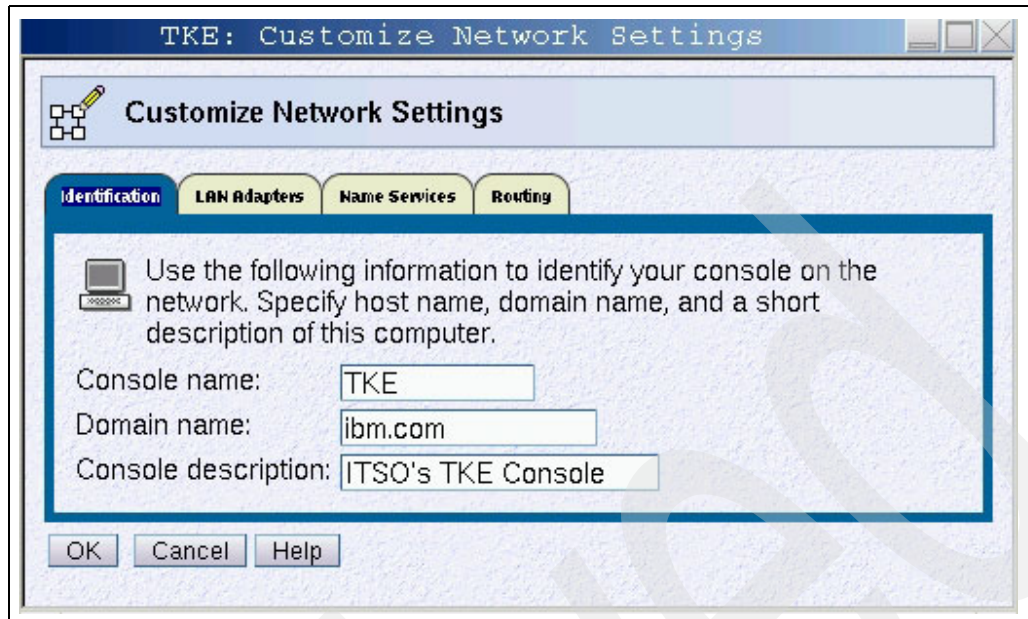


Figure 6-12 TKE workstation TCP/IP settings

To define the Local Area Network Information, select the tab **LAN Adapters**, as shown in Figure 6-13.

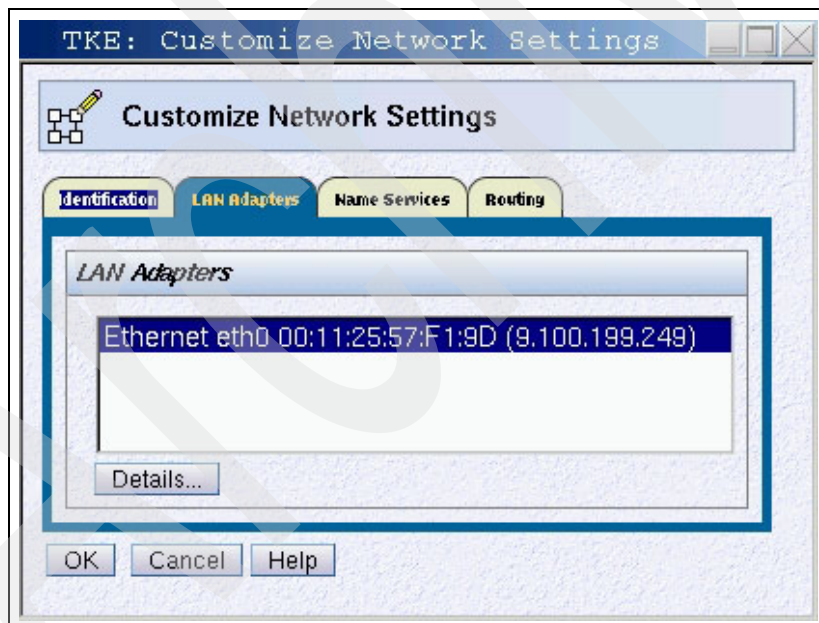


Figure 6-13 LAN adapter on the TCP/IP

Click **Details** to specify the DHCP Client/IP address information for your network, as shown in Figure 6-14. Click **OK** to store the specified values.



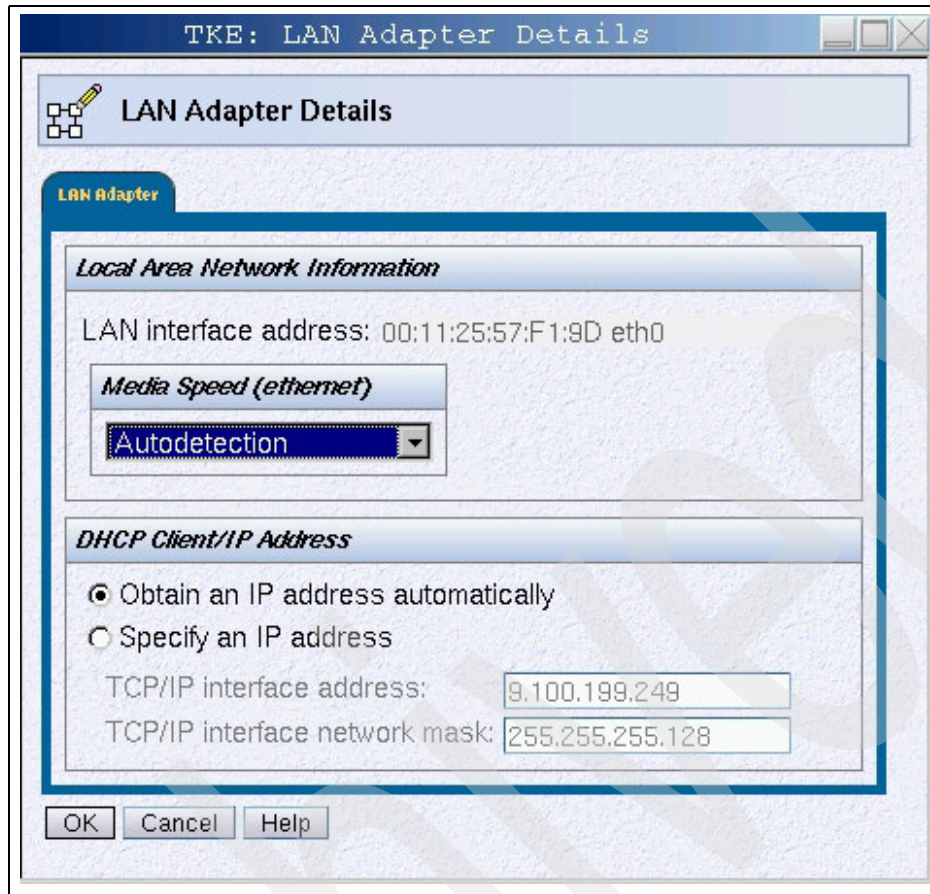


Figure 6-14 Defining TCP/IP address of the TKE workstation

To select whether DNS is enabled or disabled, select the Name Services tab for the configuration and define the DNS Server Search Order and the Domain Suffix Search Order for your network. In our case we only had DNS enabled, as shown in Figure 6-15.

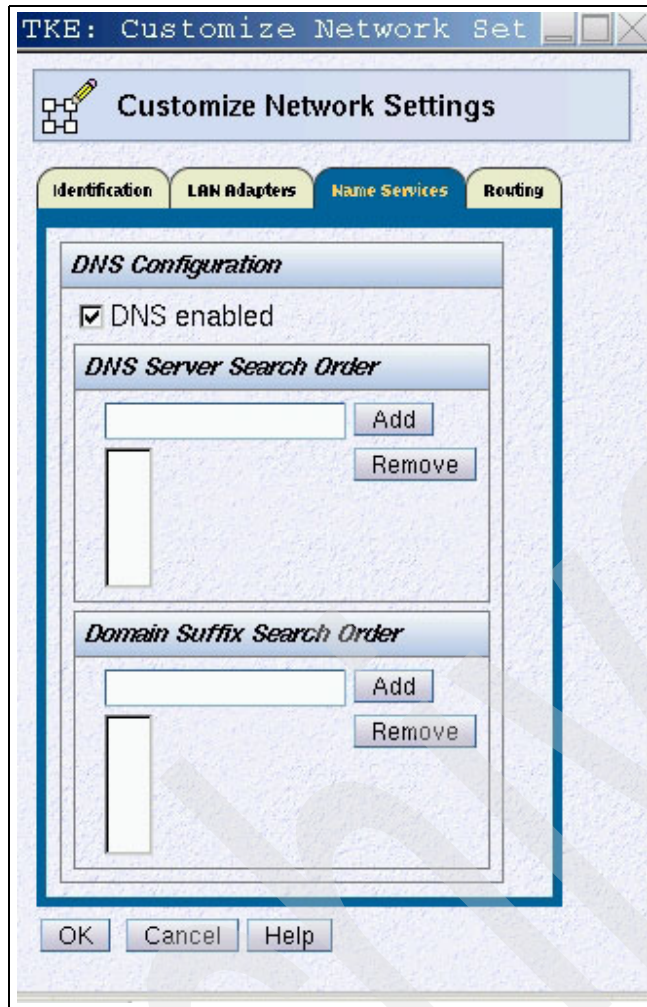


Figure 6-15 Defining name services under the TCP/IP configuration

When done, click **OK** to update the network settings. The Console application will also be restarted, to complete the changes to the TCP/IP settings.

## Diagnosing network problems

Problems associated with networking can be diagnosed with the Network Diagnostic Information task. To open this task, select **System Management** → **Service Applications** → **Network Diagnostic Information**.

You can enter a target TCP/IP address and click **Ping** to perform a ping function, as shown in Figure 6-16. There are also several other tabs that you can use to analyze network problems.

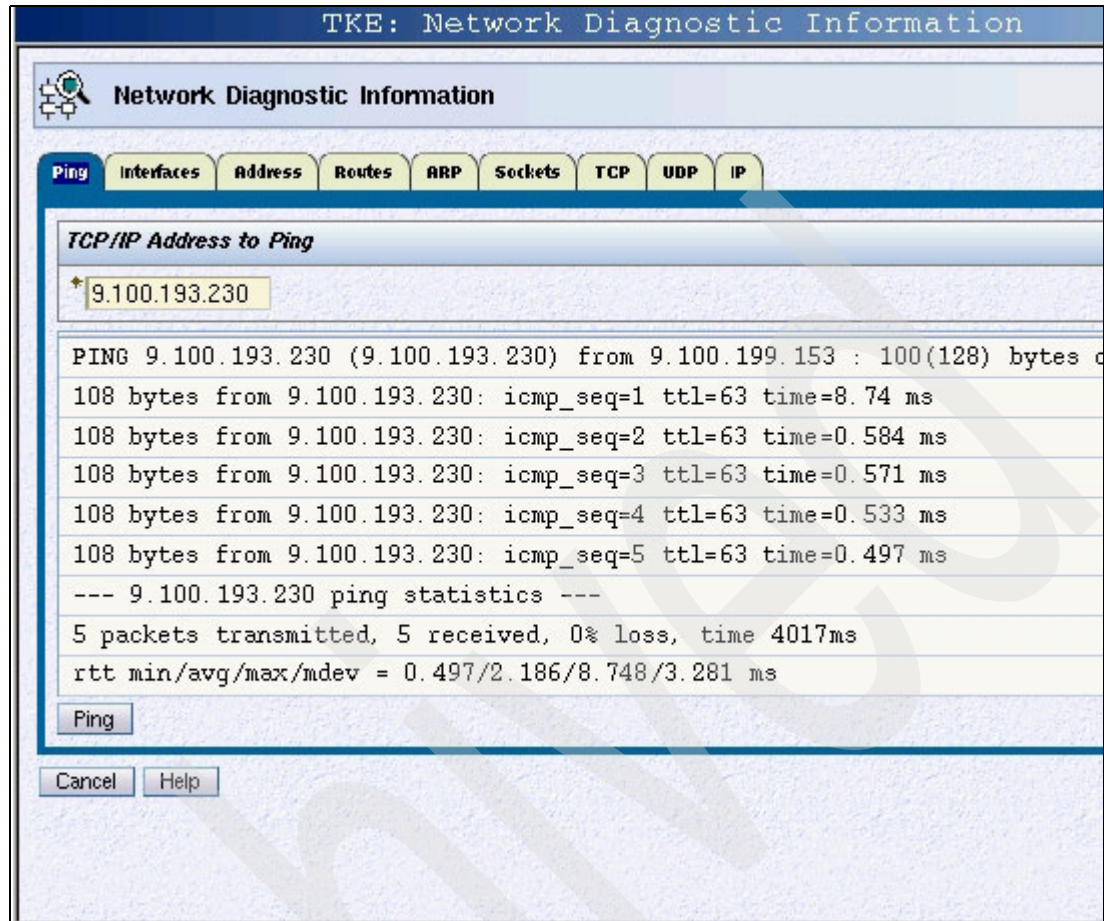


Figure 6-16 Pinging the network address

### 6.4.5 3270 emulator configuration

To set up the 3270 emulator session, open the configuration task by selecting **System Management** → **Configuration** → **Configure 3270 Emulators**.

On the window, click **New**, enter the host TCP/IP address in the Host address field, and click **OK**. The 3270 emulator configuration uses port 23 as a default, but this can be overwritten by specifying a port number after the Host address separated by a colon (:), as shown in Figure 6-17.

If the Enabled option is selected under the Start at Console startup, then the two defined 3270 sessions shown in Figure 6-17 will be started automatically every time the TKE workstation is started or the Console is restarted. With the Disabled option, select the desired 3270 session and click **Start**.

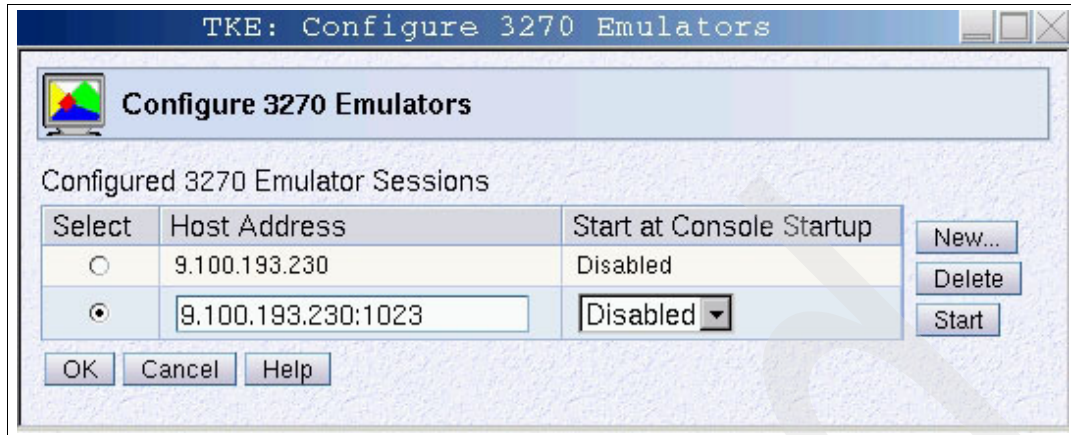


Figure 6-17 Defining 3270 emulators

## 6.5 TKE V5.0 management

Trusted Key Entry V5.0 offers new daily management options. TKE customization options controlled by the TKE.INI file in TKE V4.2 are now controlled by a new Preference menu in the TKE Application. In some cases, new Preferences have been added for TKE V5.0. In the following section, we discuss all Preferences options in more detail.

### 6.5.1 TKE V5.0 application

To start the TKE V5.0 application, open the task by selecting **Trusted Key Entry → Applications → Trusted Key Entry 5.0**.

For TKE V4.2, when the TKE application is started, a logon prompt is displayed. Select the profile ID to be used for the logon and enter the password for this profile (if using Passphrase) or insert the appropriate TKE Smart Card and enter the PIN (if using smart cards). The TKE main window will be displayed.

The main purpose of this window is to allow you to select a crypto module or a group of crypto modules. From the main window, you also create host definitions and group definitions.

To customize the TKE workstation, update TKE Preferences using the Preferences menu in TKE. Click **Preferences** on the toolbar, as shown in Figure 6-18. Preference options are enabled or disabled by clicking the check box. A check indicates that the preference option is enabled. By default, only the Blind Key Entry option is enabled.

- ▶ **Blind Key Entry** controls whether key values entered at the TKE keyboard are displayed or hidden.

With hidden entry, an asterisk (\*) is displayed for each entered hexadecimal character. Ensure the menu item is checked if you want hidden entry; otherwise uncheck the menu item.

On TKE V4.2, this information is defined in the TKE.INI file.

- ▶ **Enable Tracing** activates the trace facility in TKE. The output can be used to help debug problems with TKE.

**Note:** Do not check this menu item unless an IBM service representative instructs you to do so. When checked, TKE produces a trace file named trace.txt in the TKE Data



Directory. Every time TKE is restarted, the trace.txt file is overwritten and a new file is created. This is a new feature on TKE.

- ▶ Enable Smart Card Readers enables the smart card option for TKE. If the option is unchecked, TKE will hide all smart card options from the user.

**Note:** The TKE application must be closed and reopened in order for a change to the Enable Smart Card Readers Preference option to become effective.

Compliance to the ZKA standard requires specific settings in TKE preferences, as described here:

- ▶ Floppy Drive Only specifies where files can be retrieved and stored. Ensure this option is checked if you want to restrict access to the floppy drive only; otherwise, uncheck the menu item. For ZKA compliance, this check box *must* be checked.

On TKE V4.2, this information is defined in the TKE.INI file.

- ▶ Show ZKA ECM Bits controls whether or not to display two additional CCF Environment Control Mask (ECM) bits when working with the CCF Domain Controls task:
  - Reset Domain
  - Load Clear Master Key

For ZKA compliance, this check box must be selected. Furthermore, for the CCF Domain Controls, the only ECM bit that should remain enabled is Cryptographic functions. All other ECM selections must be disabled.

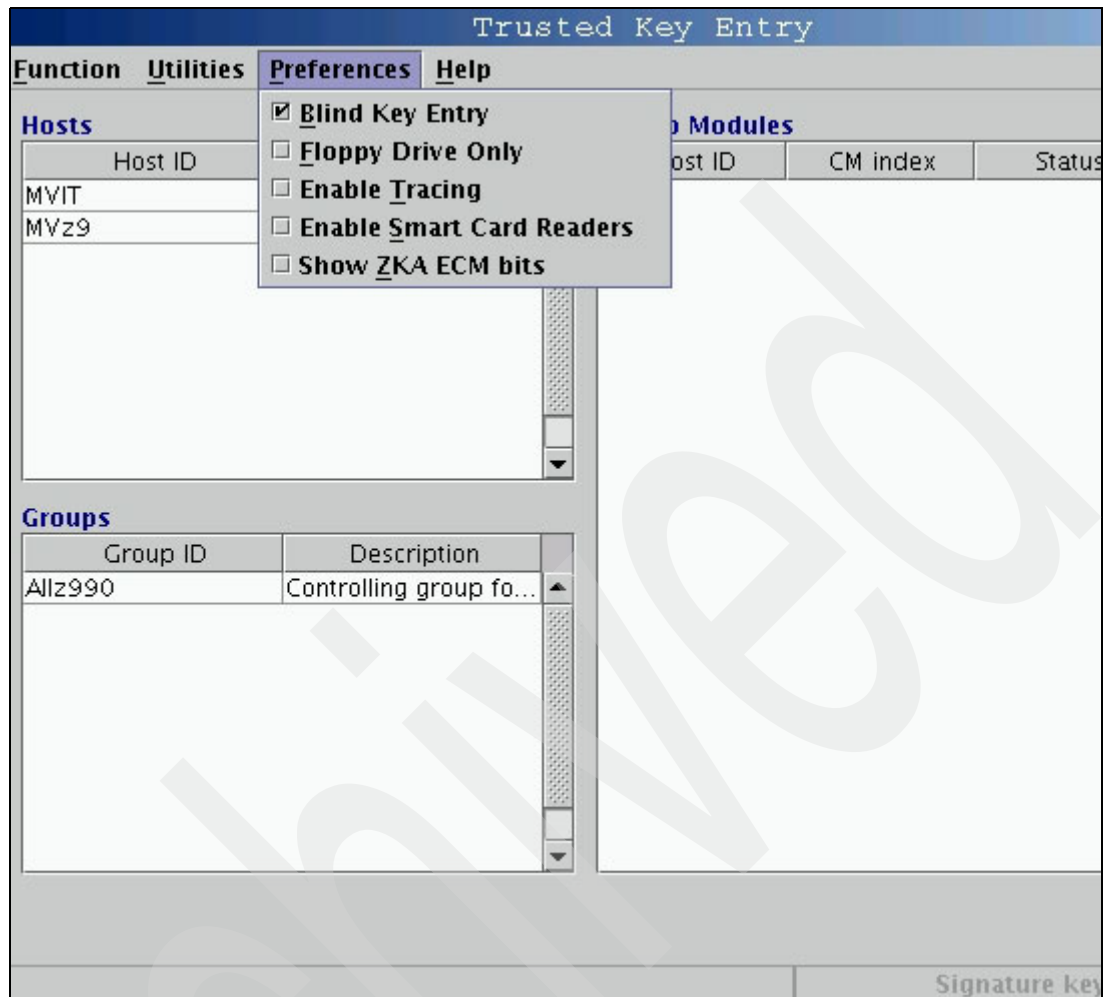


Figure 6-18 TKE V5.0 application preferences

## 6.5.2 TKE Media management

The TKE V5.0 Workstation provides a new way to operate and manage media, as compared to the OS/2 commands used with TKE V4.2.

The TKE Media management has three tasks:

- ▶ TKE Media Manager
- ▶ Format Media
- ▶ TKE File Management utility

### TKE Media Manager

Before accessing or updating an inserted floppy or DVD-RAM/CD-ROM media from any of the Trusted Key Entry (Applications and Utilities) tasks, the media must first be activated using TKE Media Manager.

To activate media, open **Trusted Key Entry** → **Applications**, or **Utilities**, and **TKE Media manager**. From the drop-down menu, you can activate the media that is currently deactivated, or deactivate the media that is currently active. After the operation is finished, the TKE Media Manager will update the status of the corresponding drive. Select Cancel to exit TKE Media Manager.

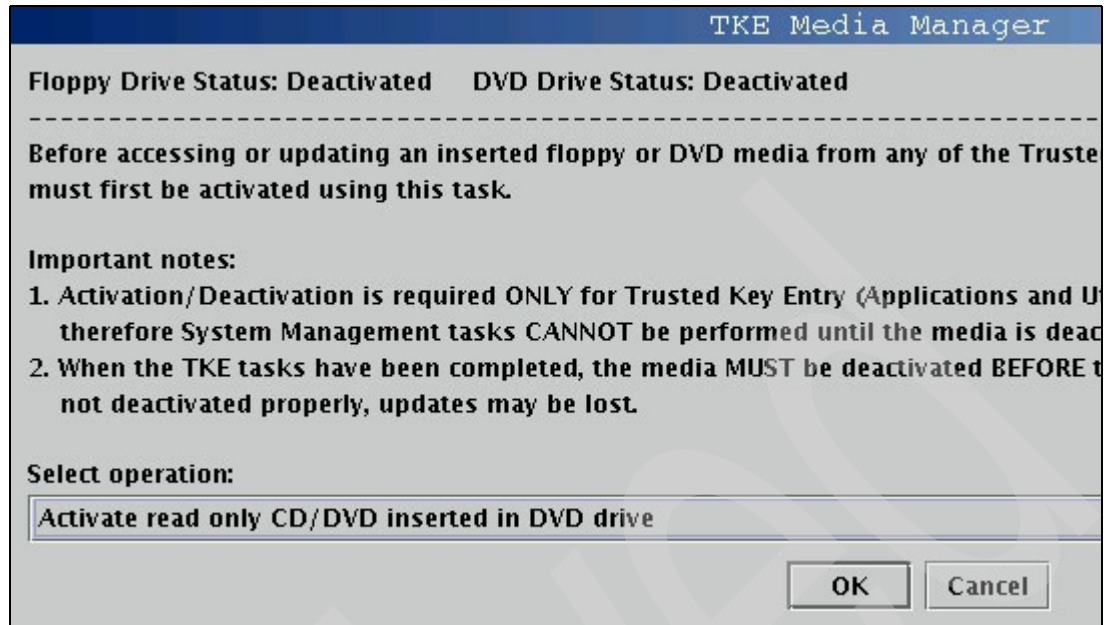


Figure 6-19 Activation and deactivation options of external media used in the TKE workstation

### **Important information when using TKE Media Manager**

- ▶ Only DVD-RAM and CD-ROM disks are supported. The DVD-RAM is R/W, while the CD-ROM is R/O. DVD-RAM format is single-sided Type II (4.7 GB).
- ▶ Activation/Deactivation is required only for Trusted Key Entry (Applications and Utilities) tasks. This action locks the media for TKE tasks; therefore, System Management tasks cannot be performed until the media is deactivated.
- ▶ When the TKE tasks have been completed, the media must be deactivated before the media is removed from the drive. If the media is not deactivated properly, updates may be lost.
- ▶ If a media device is inserted but not activated and the device is selected to use with a TKE application, the application will attempt to activate the device. Even though the media was not activated directly with the TKE Media Manager, the media must still be *deactivated* using the TKE Media Manager before it is removed.
- ▶ Any media activated in the DVD-RAM/CD-ROM drive will not eject until the drive is deactivated. TKE Media Manager must be used to deactivate a drive.
- ▶ If a floppy disk is ejected without deactivating the drive, your disk may not retain output written to it and may become corrupted. You must deactivate the floppy drive before ejecting your floppy disk.

**Important:** Even if you are using the media for input only, the media *must* be deactivated before it is removed.

If the media is not deactivated before it is removed, new media inserted may not be handled correctly.

## Format media

**Attention:** Prior to formatting any media, ensure that the applicable Floppy or DVD-RAM drive is deactivated using TKE Media Manager.

If the media is not deactivated, the format will fail.

To format the media, open **System Management** → **Service Applications** → **Format media**. The view shown in Figure 6-20 will be displayed.

**Note:** The Format Media task is used to format DVD-RAMs and diskettes only.

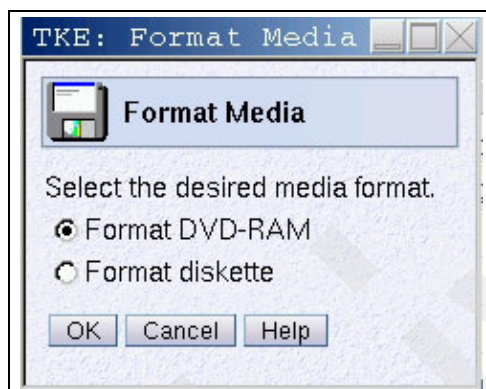


Figure 6-20 Formatting DVD-RAM or a diskette

If Format DVD-RAM is selected, the DVD-RAM label is automatically written to the DVD depending on the intended use of the DVD-RAM, as shown in Figure 6-21.

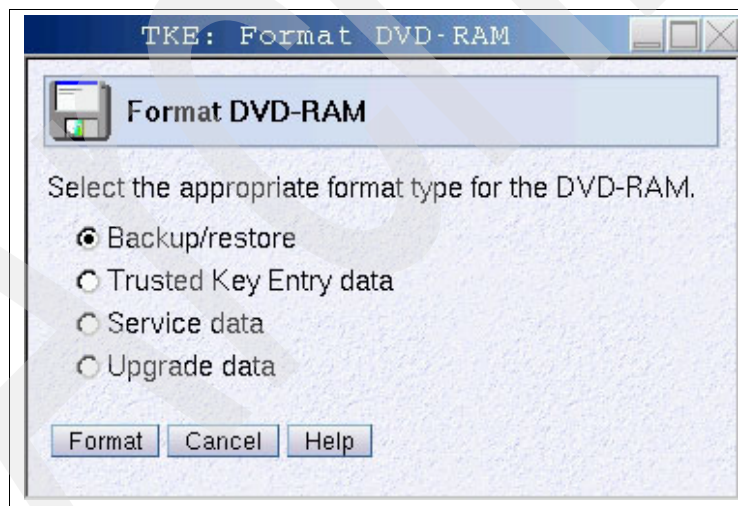


Figure 6-21 Select DVD-RAM usage

## TKE File Management utility

The TKE File Management utility task allows you to manage files on diskette, CD/DVD, or within TKE Data directories. It allows you to Delete, Rename, and Copy files.

To activate the task, open **Trusted Key Entry** → **Utilities** → **TKE File management utility**. The window shown in Figure 6-22 will be presented.



Selecting the hard drive for either Source or Target will allow you to select one out of four data directories:

- ▶ TKE Data Directory
- ▶ Migration Utility Data Directory
- ▶ CNM Data Directory
- ▶ SCUP Data Directory

From the displayed list you can select a single file, numerous files, blocks of files, or the entire display for copy, delete, or rename processes.

**Important:** If updates are done to the floppy drive or DVD-RAM, the media *must* be deactivated before it is removed. Otherwise, the updates may be lost.

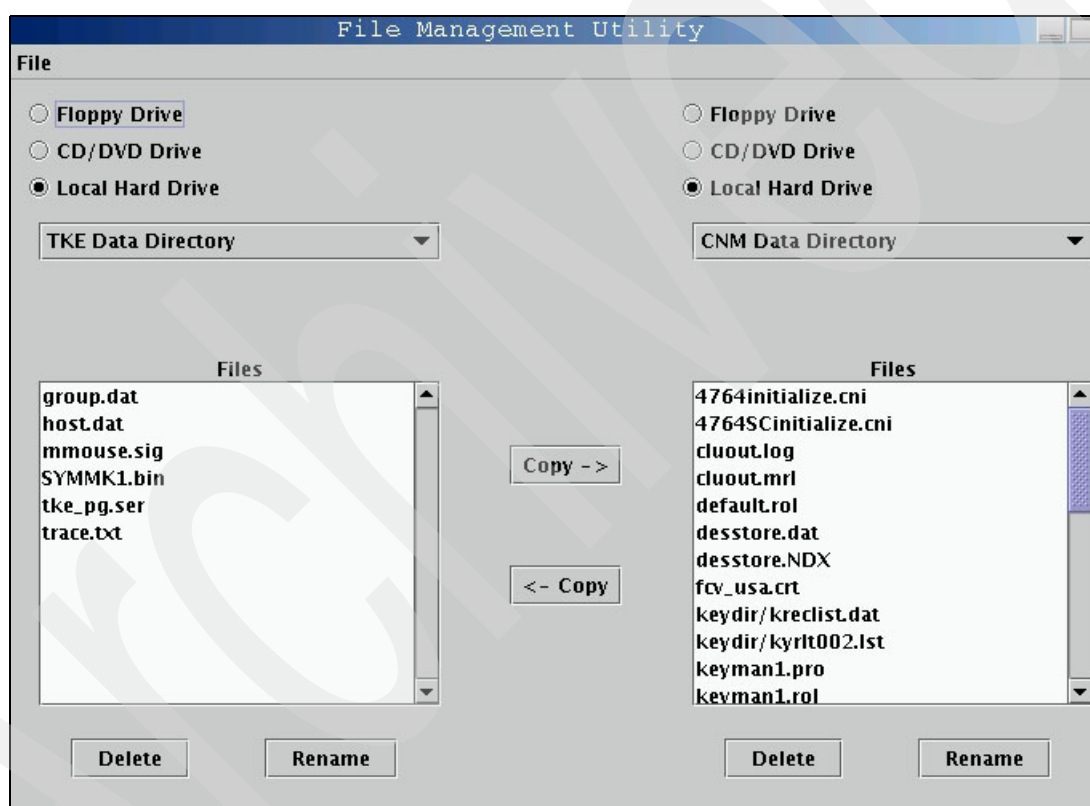


Figure 6-22 TKE file management options

### 6.5.3 Backing up critical console data and customizing scheduled operations

Because of time constraints, we could not cover this important topic when developing this redbook. (For complete information on this subject, refer to *z/OS Cryptographic Services ICSF Trusted Key Entry PCIX Workstation User's Guide*, SA23-2211.)

However, within this context it is vital to stress that data which is backed up for TKE V5.0 is very different from TKE V4.2 backup. Since MCLs can now be installed to fix the Operating System, Framework and TKE code problem, it is imperative that a backup be performed either every time an MCL is installed, or at least on a scheduled basis.

In addition to backing up MCL changes, all TCP/IP definitions, 3270 emulator sessions, default roles and profiles, and any data changed in the TKE-related data directory is also saved.

### 6.5.4 Shutdown or Restart

If you need to restart the application or console, or to power off, open **System Management-Maintenance-Shutdown or Restart**. The panel shown in Figure 6-23 will be displayed.

From here you can perform the following tasks:

- ▶ Close the Trusted Key Entry workstation and restart the application.
- ▶ Close the Trusted Key Entry workstation, perform a system Power-on Reset, and restart the console.
- ▶ Close the Trusted Key Entry workstation, shut down the operating system, and power off the hardware.

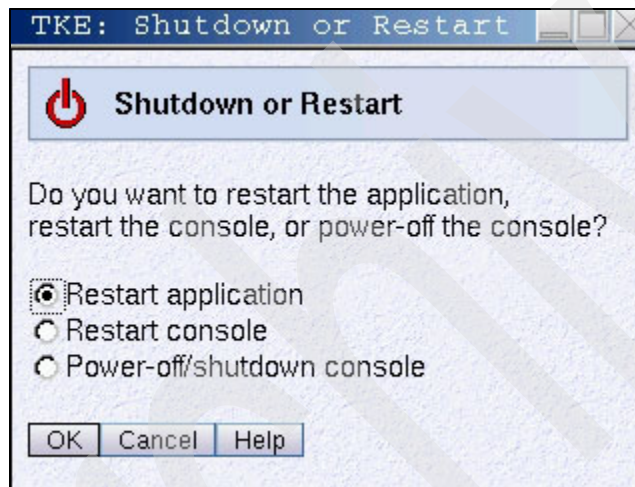


Figure 6-23 Shutting down or restarting the TKE workstation

Selecting any of these options will present you with a confirmation window, where you can enter **yes** or **no** to accept or reject the action.

## CPACF programs

This appendix provides the source of the assembler and REXX programs used to test the new CPACF functions.

**Note:** The code supplied here has not been subjected to any formal IBM test and is distributed on an “AS IS” basis without any warranty either express or implied. The implementation of any of the techniques described or used herein is a customer responsibility and depends on the customer’s operational environment. While each item may have been reviewed for accuracy in a specific situation and may run in a specific environment, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

## CPACF010 program

Example A-1 is an assembler program that can be used to invoke the KMC instruction to perform clear key encryption and decryption.

*Example: A-1 Program to invoke KMC instructions*

```

CPACF010 CSECT                                00000100
CPACF010 AMODE 31                             00000200
CPACF010 RMODE ANY                           00000300
* ----- * 00000400
* * 00000500
* C P A C F 0 1 0 - Encrypt test block using KMC * 00000600
* ----- * 00000700
* * 00000800
* Last Updated 29/08/2005 * 00000900
* * 00001000
* Module : CPACF010 * 00001100
* Date written : 29th August 2005 * 00001200
* Function : Encrypt a text block * 00001300
* * 00001400
* HISTORY : 29/08/2005 - Initial Version * 00001500
* * 00001600
* Module attributes: RENT,REUS,AMODE(31),RMODE(31) * 00001700
* * 00001800
* ----- * 00001900
* * 00002000
* ----- * 00002100
* * 00002200
* Parameters: 1 - Return Code * 00002300
* 2 - Option Block * 00002400
* 3 - Data to Encrypt or Decrypt (INPUT) * 00002500
* 4 - Encrypted or Decrypted Data (OUTPUT) * 00002600
* 5 - Key * 00002700
* * 00002800
* Option Block format * 00002900
* Byte 1-1 Function Code * 00003000
* Byte 2-2 E = Encrypt, D = Decrypt * 00003100
* Byte 3-4 Length of Data * 00003200
* * 00003300
* * 00003400
* ----- * 00003500
* SAVE (14,12),,CPACF010-&SYSDATE-&SYSTIME 00003600
* * 00003700
* LR R12,R15 00003800
* * 00003900
* USING CPACF010,R12 Module addressability 00004000
* * 00004100
* LR R10,R1 Save param address 00004200
* * 00004300
* LA R0,WkEnd-WkStart Length of work area 00004400
* XR R15,R15 Subpool 0 00004500
* * 00004600
* STORAGE OBTAIN,SP=(15),ADDR=(1),LENGTH=(0) 00004700
* * 00004800
* USING #Work,R1 Workarea addressability 00004900
* * 00005000
* ST R0,WkLen Save length 00005100
* ST R15,WkSp1 Save subpool 00005200
* * 00005300

```

Main000	DS	OH		00005400
	ST	R1,8(,R13)	Link save areas	00005500
	ST	R13,4(,R1)	for os	00005600
*				00005700
	LR	R13,R1	Set work/save area address	00005800
	LR	R1,R10	Reinstate parameter register	00005900
*				00006000
	DROP	R1	#Work	00006100
	USING	#Work,R13	Workarea addressability	00006200
*				00006300
	BAS	R14,Encr000	Do Encrypt processing	00006400
*				00006500
Main900	DS	OH		00006600
*				00006700
	L	R10,WkRetcde	Save return code	00006800
	L	R0,WkLen	Length work area	00006900
	L	R15,WkSp1	Subpool	00007000
	LR	R1,R13	Point to my save area	00007100
	L	R13,4(R1)	Rescue pointer to high save area	00007200
*				00007300
		STORAGE RELEASE,SP=(15),ADDR=(1),LENGTH=(0)		00007400
*				00007500
	LR	R15,R10	Reinstate return code	00007600
	RETURN	(14,12),RC=(15)	Return	00007700
*				00007800
Encr	DC	CL4'Encr'		00007900
Encr000	DS	OH		00008000
	STM	R0,R15,WkSaveA	Store registers	00008100
	MVC	WkPhase,Encr	Set phase name	00008200
	XC	WkRetcde,WkRetcde	Clear return code	00008300
*				00008400
	LM	R7,R11,0(R1)	R7 to R11 point to my parameters	00008500
	STM	R7,R11,WkParms	Store parms for debugging etc.	00008600
*				00008700
	XR	R5,R5	Clear Length	00008800
	ICM	R5,B'0011',2(R8)	Insert length from Option Block	00008900
	BZ	Encr720	Too short, error	00009000
*				00009100
	C	R5,=F'32767'	Test Data Length	00009200
	BH	Encr720	Too Long, error	00009300
*				00009400
	LR	R2,R10	Set target address	00009500
	LR	R4,R9	Set source address	00009600
*				00009700
	XC	WkReg0,WkReg0	Clear Reg0 field	00009800
	CLI	1(R8),C'E'	Is this Encrypt?	00009900
	BE	Encr020	Yes, continue	00010000
*				00010100
	OC	WkReg0,=X'00000080'	Turn on Decipher bit	00010200
	CLI	1(R8),C'D'	Is this Decrypt	00010300
	BNE	Encr700	No, invalid	00010400
*				00010500
Encr020	DS	OH		00010600
	LA	R3,8	Assume DEA(DES) operation	00010700
	LA	R6,8	Assume Single length Key	00010800
	CLI	0(R8),X'01'	Is this Single DES	00010900
	BE	Encr050	Yes, proceed.	00011000
*				00011100
	LA	R6,16	Assume Double length Key	00011200
	CLI	0(R8),X'02'	Is this Triple DES, Double Len Key	00011300

	BE	Encr050	Yes, proceed.	00011400
*				00011500
	LA	R6,24	Assume Triple length Key	00011600
	CLI	0(R8),X'03'	Is this Triple DES, Triple Len Key	00011700
	BE	Encr050	Yes, proceed.	00011800
*				00011900
	LA	R3,16	Assume AES operation	00012000
	LA	R6,16	Assume AES-128	00012100
	CLI	0(R8),X'12'	Is this AES-128 ?	00012200
	BE	Encr050	Yes, proceed.	00012300
*				00012400
*	LA	R6,24	Assume AES-192	00012500
*	CLI	0(R8),X'13'	Is this AES-192 ?	00012600
*	BE	Encr050	Yes, proceed.	00012700
*				00012800
*	LA	R6,32	Assume AES-256	00012900
*	CLI	0(R8),X'14'	Is this AES-256 ?	00013000
	BNE	Encr710	No, bad retcode	00013100
*				00013200
Encr050	DS	0H		00013300
	OC	WkReg0+3(1),0(R8)	Insert 7-bit Function code	00013400
	L	R0,WkReg0	Load Function Code	00013500
*				00013600
	XC	WkKMCPrm,WkKMCPrm	Clear parameter block	00013700
	LA	R1,WkKMCPrm(R3)	Point to Key Area	00013800
*				00013900
	BCTR	R6,R0	Reduce length by 1	00014000
	MVC	0(1,R1),0(R11)	Move Key	00014100
	EX	R6,*-6	Move Key	00014200
*				00014300
Encr100	DS	0H		00014400
	LA	R1,WkKMCPrm	Set Parm Block Address	00014500
	KMC	R2,R4	Perform operation	00014600
*				00014700
	BNZ	Encr100	Loop back if partial completion	00014800
	BZ	Encr900		00014900
*				00015000
Encr700	DS	0H		00015100
	MVC	WkRetcde+1(1),1(R8)	Save failing byte	00015200
	MVI	WkRetcde+2,1	Set reason	00015300
	B	Encr800		00015400
*				00015500
Encr710	DS	0H		00015600
	MVC	WkRetcde+1(1),0(R8)	Save failing option code	00015700
	MVI	WkRetcde+2,2	Set reason	00015800
	B	Encr800		00015900
*				00016000
Encr720	DS	0H		00016100
	STCM	R5,B'0011',WkRetcde	Save failing length	00016200
	MVI	WkRetcde+2,3	Set reason	00016300
	B	Encr800		00016400
*				00016500
Encr800	DS	0H		00016600
	MVI	WkRetcde+3,12	Set retcode 12	00016700
*				00016800
Encr900	DS	0H		00016900
	MVC	0(4,R7),WkRetcde	Store RC into callers parmlist	00017000
	LM	R0,R15,WkSaveA	Reload registers	00017100
	BR	R14	Return to caller	00017200
*				00017300

	DROP	R12	CPACF010	00017400
	DROP	R13	#Work	00017500
	LTORG			00017600
*				00017700
	TITLE	***** CPACF010 : EQUATES *****		00017800
*				00017900
R0	EQU	0		00018000
R1	EQU	1		00018100
R2	EQU	2		00018200
R3	EQU	3		00018300
R4	EQU	4		00018400
R5	EQU	5		00018500
R6	EQU	6		00018600
R7	EQU	7		00018700
R8	EQU	8		00018800
R9	EQU	9		00018900
R10	EQU	10		00019000
R11	EQU	11		00019100
R12	EQU	12		00019200
R13	EQU	13		00019300
R14	EQU	14		00019400
R15	EQU	15		00019500
	TITLE	***** CPACF010 : WORK AREA *****		00019600
#Work	DSECT			00019700
WkStart	EQU	*		00019800
WkOsSave	DS	18F	OS save area	00019900
WkLen	DS	F	Length	00020000
WkSpl	DS	F	Subpool	00020100
*				00020200
WkPhase	DS	CL4	Phase name	00020300
WkSaveA	DS	16F	Subroutine save area 1	00020400
*				00020500
WkParms	DS	5F	Parm addresses	00020600
*				00020700
WkRetcde	DS	F	Return Code	00020800
*				00020900
WkReg0	DS	F	Work area	00021000
*				00021100
WkKMCPrm	DS	XL48	Max length of KMC parameter block	00021200
*				00021300
WkEnd	EQU	*		00021400
*				00021500
	END			00021600

## REXCP010 program

Example A-2 is a REXX program that can be used to invoke the program CPACF010 in Example A-1.

*Example: A-2* REXX program to invoke CPACF010

/* Rextx */	00000100
/*-----*/	00000200
/*	00000300
/* Invoke CPACF010	00000400
/*	00000500
/*-----*/	00000600
	00000700

```

/* trace i                                     */
say ' ';
say ' ';
say ' ';
say ' ';
say '-----';
say ' ';
say 'Running CPACF010 to perform AES encryption'
say ' ';

parm1 = '00000000'X
parm2 = '12'X||'E'X||'0030'X
parm3 = COPIES('00'X,50)
parm4 = COPIES('00'X,50)
parm5 = '000102030405060708090A0B0C0D0E0F'X
parm5 = parm5||'1011121314151617181910111A1B1C1D1E1F'X

address linkpgm 'CPACF010',
              'parm1',
              'parm2',
              'parm3',
              'parm4',
              'parm5',

/* check the return and reason codes */

if (parm1 <> '00000000'X) then do
  say 'CPACF010 FAILED : RC =' c2x(parm1);
  say ' ';
  say c2X(parm1)
  say c2X(parm2)
  say c2X(parm5)
end ;
else do ;
  say 'CPACF010 OK '
  say 'Clear_Text (part1)...'||c2x(SUBSTR(parm3,01,16))
  say 'Cipher_Text (part1)...'||c2x(SUBSTR(parm4,01,16))
  say ' ';
  say 'Clear_Text (part2)...'||c2x(SUBSTR(parm3,17,16))
  say 'Cipher_Text (part2)...'||c2x(SUBSTR(parm4,17,16))
  say ' ';
  say 'Clear_Text (part3)...'||c2x(SUBSTR(parm3,33,16))
  say 'Cipher_Text (part3)...'||c2x(SUBSTR(parm4,33,16))
  say ' ';
  say 'CPACF010 completed'
end ;

exit

```

## REXCP011 program

Example A-3 is a REXX program that can be used to encrypt and decrypt data using CPACF010.

*Example: A-3 Program to use CPACF010*

```

/* REXX */
/*-----*/

```



```

/*                                                     */ 00000300
/* Invoke CPACF010                                     */ 00000400
/*                                                     */ 00000500
/*-----*/ 00000600
/* trace i                                           */ 00000700
say ' ';                                           00000800
say ' ';                                           00000900
say ' ';                                           00001000
say ' ';                                           00001100
say ' ';                                           00001200
say '-----';                                     00001300
say ' ';                                           00001400
say ' Running CPACF010 to perform AES Encryption' 00001500
say ' ';                                           00001600
                                                    00001700
parm1 = '00000000'X                               00001800
parm2 = '12'X||'E'X||'0030'X                     00001900
parm3 = COPIES('00'X,48)                          00002000
parm4 = COPIES('00'X,48)                          00002100
parm5 = '000102030405060708090A0B0C0D0E0F'X      00002200
                                                    00002300
address linkpgm 'CPACF010',                       00002400
        'parm1',                                   00002500
        'parm2',                                   00002600
        'parm3',                                   00002700
        'parm4',                                   00002800
        'parm5',                                   00002900
                                                    00003000
/* check the return and reason codes */             00003100
                                                    00003200
if (parm1 <> '00000000'X) then do                  00003300
    say 'CPACF010 FAILED : RC =' c2x(parm1);       00003400
    say ' ';                                         00003500
    say c2X(parm1)                                   00003600
    say c2X(parm2)                                   00003700
    say c2X(parm5)                                   00003800
    end ;                                           00003900
else do ;                                           00004000
    say 'CPACF010 OK '                              00004100
    say 'Clear_Text (part1)...'||c2x(SUBSTR(parm3,01,16)) 00004200
    say 'Cipher_Text (part1)...'||c2x(SUBSTR(parm4,01,16)) 00004300
    say ' ';                                         00004400
    say 'Clear_Text (part2)...'||c2x(SUBSTR(parm3,17,16)) 00004500
    say 'Cipher_Text (part2)...'||c2x(SUBSTR(parm4,17,16)) 00004600
    say ' ';                                         00004700
    say 'Clear_Text (part3)...'||c2x(SUBSTR(parm3,33,16)) 00004800
    say 'Cipher_Text (part3)...'||c2x(SUBSTR(parm4,33,16)) 00004900
    say ' ';                                         00005000
    say 'CPACF010 completed'                       00005100
    end ;                                           00005200
                                                    00005300
say ' ';                                           00005400
say ' ';                                           00005500
say ' ';                                           00005600
say ' ';                                           00005700
say '-----';                                     00005800
say ' ';                                           00005900
say ' Running CPACF010 to perform AES Decryption' 00006000
say ' ';                                           00006100
                                                    00006200

```

parm1 = '00000000'X	00006300
parm2 = '12'X  'D'  '0030'X	00006400
	00006500
address linkpgm 'CPACF010',	00006600
'parm1',	00006700
'parm2',	00006800
'parm4', /* Reversed */	00006900
'parm3', /* Reversed */	00007000
'parm5',	00007100
	00007200
/* check the return and reason codes */	00007300
	00007400
if (parm1 <> '00000000'X) then do	00007500
say 'CPACF010 FAILED : RC =' c2x(parm1);	00007600
say ' ' ;	00007700
say c2X(parm1)	00007800
say c2X(parm2)	00007900
say c2X(parm5)	00008000
end ;	00008100
else do ;	00008200
say 'CPACF010 OK '	00008300
say 'Cipher_Text (part1)...'  c2x(SUBSTR(parm4,01,16))	00008400
say 'Clear_text (part1)...'  c2x(SUBSTR(parm3,01,16))	00008500
say ' ' ;	00008600
say 'Cipher_Text (part2)...'  c2x(SUBSTR(parm4,17,16))	00008700
say 'Clear_text (part2)...'  c2x(SUBSTR(parm3,17,16))	00008800
say ' ' ;	00008900
say 'Cipher_Text (part3)...'  c2x(SUBSTR(parm4,33,16))	00009000
say 'Clear_text (part3)...'  c2x(SUBSTR(parm3,33,16))	00009100
say ' ' ;	00009200
say 'CPACF010 completed'	00009300
end ;	00009400
	00009500
exit	00009600

---

## CPACF020 program

Example A-4 is an assembler program that can be used to invoke the KLMD instruction to produce SHA-256 hash values.

*Example: A-4 Program to invoke KLMD instructions*

CPACF020 CSECT	00010000
CPACF020 AMODE 31	00020000
CPACF020 RMODE ANY	00030000
* -----	* 00040000
*	* 00050000
*     C P A C F 0 2 0   -   Generate SHA-256 or SHA-1 Hash	* 00060000
*     -----	* 00070000
*	* 00080000
*     Last Updated 01/09/2005	* 00090000
*	* 00100000
*     Module : CPACF020	* 00110000
*     Date written : 31st August 2005	* 00120000
*     Function : Generate SHA Hash	* 00130000
*	* 00140000
*     HISTORY : 31/08/2005 - Initial Version	* 00150000
*	* 00160000

```

*   Module attributes: RENT,REUS,AMODE(31),RMODE(31)                                * 00170000
*                                                                                       * 00180000
* ----- * 00190000
*                                                                                       * 00200000
* ----- * 00210000
*                                                                                       * 00220000
*   Parameters: 1 - Return Code                                                         * 00230000
*               2 - Option Block                                                         * 00240000
*               3 - Data to Hash (INPUT)                                                 * 00250000
*               4 - Hash Code   (OUTPUT)                                                 * 00260000
*                                                                                       * 00270000
*   Option Block format                                                                * 00280000
*       Byte 1-1 Function Code, 1=SHA-1, 2=SHA-256                                    * 00290000
*       Byte 2-2 Not used                                                             * 00300000
*       Byte 3-4 Length of Data (Max 32767)                                           * 00310000
*                                                                                       * 00320000
*                                                                                       * 00330000
* ----- * 00340000
*   SAVE   (14,12),,CPACF020-&SYSDATE-&SYSTIME                                         00350000
*                                                                                       00360000
*   LR     R12,R15                                                                    00370000
*                                                                                       00380000
*   USING  CPACF020,R12      Module addressability                                   00390000
*                                                                                       00400000
*   LR     R10,R1              Save param address                                   00410000
*                                                                                       00420000
*   LA     R0,WkEnd-WkStart   Length of work area                                   00430000
*   XR     R15,R15            Subpool 0                                             00440000
*                                                                                       00450000
*   STORAGE OBTAIN,SP=(15),ADDR=(1),LENGTH=(0)                                       00460000
*                                                                                       00470000
*   USING  #Work,R1              Workarea addressability                           00480000
*                                                                                       00490000
*   ST     R0,WkLen              Save length                                       00500000
*   ST     R15,WkSp1             Save subpool                                       00510000
*                                                                                       00520000
* Main000 DS     0H                                                         00530000
*          ST     R1,8(,R13)      Link save areas                                   00540000
*          ST     R13,4(,R1)      for os                                           00550000
*                                                                                       00560000
*          LR     R13,R1          Set work/save area address                       00570000
*          LR     R1,R10          Reinstate parameter register                     00580000
*                                                                                       00590000
*          DROP   R1              #Work                                             00600000
*          USING  #Work,R13       Workarea addressability                         00610000
*                                                                                       00620000
*          BAS    R14,Sha000      Do SHA processing                               00630000
*                                                                                       00640000
* Main900 DS     0H                                                         00650000
*                                                                                       00660000
*          L      R10,WkRetcde     Save return code                               00670000
*          L      R0,WkLen         Length work area                               00680000
*          L      R15,WkSp1        Subpool                                         00690000
*          LR     R1,R13           Point to my save area                           00700000
*          L      R13,4(R1)        Rescue pointer to high save area                 00710000
*                                                                                       00720000
*          STORAGE RELEASE,SP=(15),ADDR=(1),LENGTH=(0)                             00730000
*                                                                                       00740000
*          LR     R15,R10          Reinstate return code                           00750000
*          RETURN (14,12),RC=(15) Return                                           00760000

```

*				00770000
Sha	DC	CL4'Sha'		00780000
Sha000	DS	OH		00790000
	STM	R0,R15,WkSaveA	Store registers	00800000
	MVC	WkPhase,Sha	Set phase name	00810000
	XC	WkRetcde,WkRetcde	Clear return code	00820000
*				00830000
	LM	R8,R11,0(R1)	R8 to R11 point to my parameters	00840000
	STM	R8,R11,WkParms	Store parms for debugging etc.	00850000
*				00860000
	XR	R5,R5	Clear Length	00870000
	ICM	R5,B'0011',2(R9)	Insert length from Option Block	00880000
	BZ	Sha720	Too short, error	00890000
*				00900000
	C	R5,=F'32767'	Test Data Length	00910000
	BH	Sha720	Too Long, error	00920000
*				00930000
Sha020	DS	OH	*** Which SHA ***	00940000
	LA	R2,20	Assume SHA-1 length	00950000
	LA	R3,WkKLMDPm+24	Point to SHA-1 bit len (last half)	00960000
	MVC	WkKLMDPm,SHA1	Assume SHA-1	00970000
	CLI	0(R9),X'01'	Is this SHA-1 ?	00980000
	BE	Sha050	Yes, proceed.	00990000
*				01000000
	LA	R2,32	Assume SHA-256 length	01010000
	LA	R3,WkKLMDPm+36	Point to SHA-256 bit len (last half)	01020000
	MVC	WkKLMDPm,SHA256	Assume SHA-256	01030000
	CLI	0(R9),X'02'	Is this SHA-256?	01040000
	BE	Sha050	Yes, proceed.	01050000
*				01060000
	B	Sha710	No, bad retcode	01070000
*				01080000
Sha050	DS	OH		01090000
	XR	R0,R0	Clear Reg 0	01100000
	IC	R0,0(R9)	Insert 7-bit Function code	01110000
*				01120000
Sha100	DS	OH		01130000
	SLL	R5,3	Convert length to bits	01140000
	ST	R5,0(R3)	Store length in parm block	01150000
	SRL	R5,3	Length back to bytes	01160000
	LR	R4,R10	Point to Input data	01170000
	LA	R1,WkKLMDPm	Set Parm Block Address	01180000
*				01190000
Sha120	DS	OH		01200000
	KLMD	R2,R4	Perform operation	01210000
	BNZ	Sha120	Loop back for partial completion	01220000
*				01230000
	BCTR	R2,R0	reduce length for move	01240000
	MVC	0(1,R11),WkKLMDPm	Move hash code to callers parm	01250000
	EX	R3,*-6	Move hash code to callers parm	01260000
	BZ	Sha900	Finished	01270000
*				01280000
Sha710	DS	OH		01290000
	MVC	WkRetcde+1(1),0(R9)	Save failing option code	01300000
	MVI	WkRetcde+2,2	Set reason	01310000
	B	Sha800		01320000
*				01330000
Sha720	DS	OH		01340000
	STCM	R5,B'0011',WkRetcde		01350000
	MVI	WkRetcde+2,3	Set reason	01360000

```

      B      Sha800                                01370000
*                                           01380000
Sha800 DS      OH                                01390000
      MVI     WkRetcde+3,12      Set retcode 12    01400000
*                                           01410000
Sha900 DS      OH                                01420000
      MVC     0(4,R8),WkRetcde   Store RC into callers parmlist 01430000
      LM      R0,R15,WkSaveA     Reload registers   01440000
      BR      R14                Return to caller   01450000
*                                           01460000
      DROP    R12                CPACF020          01470000
      DROP    R13                #Work             01480000
      LTORG                               01490000
*                                           01500000
      TITLE   '***** CPACF020 : EQUATES *****' 01510000
      DS      OF                                01520000
SHA1   DS      OXL(SHA1END-SHA1ST)              01530000
SHA1ST DC      XL4'67452301'                    01540000
      DC      XL4'EFCDA889'                    01550000
      DC      XL4'98BADCFE'                    01560000
      DC      XL4'10325476'                    01570000
      DC      XL4'C3D2E1F0'                    01580000
      DC      XL4'00000000'                    01590000
      DC      XL4'00000000'                    01600000
      DC      XL4'00000000'                    01610000
      DC      XL4'00000000'                    01620000
      DC      XL4'00000000'                    01630000
SHA1END EQU    *                                01640000
*                                           01650000
SHA256 DS      OXL(SHA256EN-SHA256ST)            01660000
SHA256ST DC     XL4'6A09E667'                    01670000
      DC     XL4'BB67AE85'                    01680000
      DC     XL4'3C6EF372'                    01690000
      DC     XL4'A54FF53A'                    01700000
      DC     XL4'510E527F'                    01710000
      DC     XL4'9B05688C'                    01720000
      DC     XL4'1F83D9AB'                    01730000
      DC     XL4'5BE0CD19'                    01740000
      DC     XL4'00000000'                    01750000
      DC     XL4'00000000'                    01760000
SHA256EN EQU    *                                01770000
*                                           01780000
R0      EQU    0                                01790000
R1      EQU    1                                01800000
R2      EQU    2                                01810000
R3      EQU    3                                01820000
R4      EQU    4                                01830000
R5      EQU    5                                01840000
R6      EQU    6                                01850000
R7      EQU    7                                01860000
R8      EQU    8                                01870000
R9      EQU    9                                01880000
R10     EQU    10                               01890000
R11     EQU    11                               01900000
R12     EQU    12                               01910000
R13     EQU    13                               01920000
R14     EQU    14                               01930000
R15     EQU    15                               01940000
*                                           01950000
      PRINT   NOGEN                             01960000

```

EJECT				01970000
TITLE	***** CPACF020 : WORK AREA *****			01980000
#Work DSECT				01990000
WkStart EQU	*			02000000
WkOsSave DS	18F		OS save area	02010000
WkLen DS	F		Length	02020000
WkSp1 DS	F		Subpool	02030000
*				02040000
WkPhase DS	CL4		Phase name	02050000
WkSaveA DS	16F		Subroutine save area 1	02060000
*				02070000
WkParms DS	5F		Parm addresses	02080000
*				02090000
WkRetcde DS	F		Return Code	02100000
*				02110000
WkKLMDPM DS	XL40		Max length of KLMD parameter block	02120000
*				02130000
WkEnd EQU	*			02140000
*				02150000
END				02160000

---

## REXCP020 program

Example A-5 is a REXX program that can be used to invoke the CPACF020 program in Example A-4.

*Example: A-5 Program to invoke CPACF020*

/* REXX */	00010000
/*-----*/	00020000
/*	00030000
/* Invoke CPACF020	00040000
/*	00050000
/*-----*/	00060000
	00070000
/* trace i	00080000
say ' ';	00090000
say ' ';	00100000
say ' ';	00110000
say ' ';	00120000
say '-----';	00130000
say ' ';	00140000
say 'Running CPACF020 to generate HASH values'	00150000
say ' ';	00160000
	00170000
parm1 = '00000000'X	00180000
parm2 = '02'X  ', '  '0030'X	00190000
parm3 = COPIES('00'X,48)	00200000
parm4 = COPIES('00'X,32)	00210000
	00220000
address linkpgm 'CPACF020',	00230000
'parm1',	00240000
'parm2',	00250000
'parm3',	00260000
'parm4'	00270000
	00280000
/* check the return and reason codes */	00290000
	00300000

```

if (parm1 <> '00000000'X) then do
    say 'CPACF020 FAILED : RC =' c2x(parm1);
    say ' ';
    say c2X(parm1)
    say c2X(parm2)
end ;
else do ;
    say 'CPACF020 OK '
    say 'Text (part1)...'||c2x(SUBSTR(parm3,01,16))
    say 'Text (part2)...'||c2x(SUBSTR(parm3,17,16))
    say 'Text (part3)...'||c2x(SUBSTR(parm3,33,16))
    say ' '
    if SUBSTR(parm2,1,1) = '01'X then ,
        say 'HASH value (SHA-1)...'||c2X(SUBSTR(parm4,01,20))
    if SUBSTR(parm2,1,1) = '02'X then do
        say 'HASH value (SHA-256)...'||c2X(SUBSTR(parm4,01,16))
        say ' '||c2X(SUBSTR(parm4,17,16))
    end
    say 'CPACF020 completed'
end ;

exit

```

## REXBOWH program

Example A-6 is a REXX program that can be used to invoke the CSNBOWH (One Way Hash) functions of ICSF.

*Example: A-6 Program to invoke CSNBOWH*

```

/* Rexx */
/*-----*/
/*
/*      Invoke CSNBOWH
/*
/*-----*/
/* ***** */
/* initialize parameters for common use
/*
/*
/* trace i
/*
say ' ';
say ' ';
say ' ';
say ' ';
say '-----';
say ' ';
say 'Running CSNBOWH to generate hash value'
say ' ';

ex_rc = '00000000'X
ex_rs = '00000000'X
exit_data_length = '00000000'X
exit_data        = ' '
rule_array_count = '00000002'x
rule_array       = 'SHA-1  ONLY  '
text length      = '00000030'x

```

```

text                = COPIES('00'X,48)                                00002900
chain_vect_len      = '00000080'x                                    00003000
chain_vect          = COPIES('00'X,128)                              00003100
if SUBSTR(rule_array,1,8) = 'SHA-1 ' then hash_len = '00000014'X    00003200
if SUBSTR(rule_array,1,8) = 'SHA-256 ' then hash_len = '00000020'X  00003300
hash                = COPIES('00'X,32)                               00003400
                                                             00003500
address linkpgm 'CSNBOWH',                                           00003600
                    'ex_rc',                                           00003700
                    'ex_rs',                                           00003800
                    'exit_data_length',                                00003900
                    'exit_data',                                       00004000
                    'rule_array_count',                                00004100
                    'rule_array',                                       00004200
                    'text_length',                                     00004300
                    'text',                                             00004400
                    'chain_vect_len',                                  00004500
                    'chain_vect',                                       00004600
                    'hash_len',                                        00004700
                    'hash',                                             00004800
/* check the return code */                                           00004900
                                                             00005000
if (ex_rc <> '00000000'X) | (ex_rs <> '00000000'X) then do          00005100
    say 'CSNBOWH FAILED : RC =' c2x(ex_rc);                          00005200
    say '          RS =' c2x(ex_rs);                                  00005300
    say ' ' ;                                                         00005400
end ;                                                                 00005500
else do ;                                                            00005600
    say 'CSNBOWH OK '                                                00005700
    say 'Text (part1)...'||c2x(SUBSTR(text,01,16))                  00005800
    say 'Text (part2)...'||c2x(SUBSTR(text,17,16))                  00005900
    say 'Text (part3)...'||c2x(SUBSTR(text,33,16))                  00006000
    say ' ' ;                                                         00006100
    if SUBSTR(rule_array,1,8) = 'SHA-1 ' then ,                      00006200
        say 'HASH value (SHA-1)...'||c2x(SUBSTR(hash,01,20))        00006300
    if SUBSTR(rule_array,1,8) = 'SHA-256 ' then do                  00006400
        say 'HASH value (SHA-256)...'||c2x(SUBSTR(hash,01,16))      00006500
        say '          '||c2x(SUBSTR(hash,17,16))                  00006600
    end                                                              00006700
    say ' ' ;                                                         00006800
    say 'CSNBOWH completed'                                          00006900
end ;                                                                00007000
                                                             00007100
exit                                                                00007200

```

---

## REXBSYE program

Example A-7 is a REXX program that can be used to invoke the CSNBSYE function of ICSF to perform clear key encryption.

*Example: A-7 Program to invoke CSNBSYE*

```

/* REXX */                                                            00000100
/*-----*/                                                           00000200
/*                                                     */ 00000300
/* Invoke CSNBSYE                                     */ 00000400
/*                                                     */ 00000500
/*-----*/                                                           00000600

```



```

/* ***** */
/* initialize parameters for common use */
/* */

/* trace i */
say ' ';
say ' ';
say ' ';
say ' ';
say '-----';
say ' ';
say 'Running CSNBSYE to perform AES encryption'
say ' ';

ex_rc = '00000000'X
ex_rs = '00000000'X
exit_data_length = '00000000'X
exit_data = ' '
rule_array_count = '00000004'x
rule_array = 'AES CBC KEY-CLR INITIAL '
key_length = '00000018'x
key_identifier = '000102030405060708090A0B0C0D0E0F'X
key_identifier = key_identifier||'101112131415161718191A1B1C1D1E1F'X
key_parms_len = '00000004'x
key_parms = ' '
block_size = '00000010'x
init_vector_len = '00000010'x
init_vector = '00000000000000000000000000000000'X
chain_data_len = '00000020'x
chain_data = COPIES('00'X,32)
clear_text_len = '00000030'x
clear_text = COPIES('00'X,48)
cipher_text_len = '00000030'x
cipher_text = COPIES('00'X,48)
opt_data_len = '00000004'x
opt_data = '00000000'X

address linkpgm 'CSNBSYE',
               'ex_rc',
               'ex_rs',
               'exit_data_length',
               'exit_data',
               'rule_array_count',
               'rule_array',
               'key_length',
               'key_identifier',
               'key_parms_len',
               'key_parms',
               'block_size',
               'init_vector_len',
               'init_vector',
               'chain_data_len',
               'chain_data',
               'clear_text_len',
               'clear_text',
               'cipher_text_len',
               'cipher_text',
               'opt_data_len',
               'opt_data'

```

```

00000700
00000800
00000900
00001000
00001100
00001200
00001300
00001400
00001500
00001600
00001700
00001800
00001900
00002000
00002100
00002200
00002300
00002400
00002500
00002600
00002700
00002800
00002900
00003000
00003100
00003200
00003300
00003400
00003500
00003600
00003700
00003800
00003900
00004000
00004100
00004200
00004300
00004400
00004500
00004600
00004700
00004800
00004900
00005000
00005100
00005200
00005300
00005400
00005500
00005600
00005700
00005800
00005900
00006000
00006100
00006200
00006300
00006400
00006500
00006600

```

/* check the return code */	00006700
	00006800
if (ex_rc <> '00000000'X)   (ex_rs <> '00000000'X) then do	00006900
say 'CSNBSYE FAILED : RC =' c2x(ex_rc);	00007000
say '                          RS =' c2x(ex_rs);	00007100
say ' ' ;	00007200
end ;	00007300
else do ;	00007400
say 'CSNBSYE OK '	00007500
say 'Clear_Text (part1)...' c2x(SUBSTR(clear_text,01,16))	00007600
say 'Cipher_Text (part1)...' c2x(SUBSTR(cipher_text,01,16))	00007700
say ' ' ;	00007800
say 'Clear_Text (part2)...' c2x(SUBSTR(clear_text,17,16))	00007900
say 'Cipher_Text (part2)...' c2x(SUBSTR(cipher_text,17,16))	00008000
say ' ' ;	00008100
say 'Clear_Text (part3)...' c2x(SUBSTR(clear_text,33,16))	00008200
say 'Cipher_Text (part3)...' c2x(SUBSTR(cipher_text,33,16))	00008300
say ' ' ;	00008400
say 'CSNBSYE completed'	00008500
end ;	00008600
	00008700
exit	00008800

---

## REXBSYED program

Example A-8 is a REXX program that invokes CSNBSYE and CSNBSYD services of ICSF to encrypt and decrypt a string of data.

*Example: A-8 Program to invke CSNBSYE and CSNBSYD*

/* REXX */	00000100
/*-----*/	00000200
/*	*/ 00000300
/* Invoke CSNBSYE	*/ 00000400
/*	*/ 00000500
/*-----*/	00000600
	00000700
/* ***** */	00000800
/* initialize parameters for common use	*/ 00000900
/*	*/ 00001000
	00001100
/* trace i	*/ 00001200
say ' ';	00001300
say ' ';	00001400
say ' ';	00001500
say ' ';	00001600
say '-----';	00001700
say ' ';	00001800
say 'Running CSNBSYE to perform AES Encryption'	00001900
say ' ';	00002000
	00002100
ex_rc = '00000000'X	00002200
ex_rs = '00000000'X	00002300
exit_data_length = '00000000'X	00002400
exit_data = ' ';	00002500
rule_array_count = '00000004'x	00002600
rule_array = 'AES CBC KEY-CLR INITIAL '	00002700
key_length = '00000010'x	00002800

```

key_identifier = '000102030405060708090A0B0C0D0E0F'X      00002900
key_parms_len  = '00000004'x                                00003000
key_parms      = ' '                                          00003100
block_size     = '00000010'x                                  00003200
init_vector_len = '00000010'x                                  00003300
init_vector     = '00000000000000000000000000000000'X      00003400
chain_data_len  = '00000020'x                                  00003500
chain_data      = COPIES('00'X,32)                            00003600
clear_text_len  = '00000030'x                                  00003700
clear_text      = COPIES('00'X,48)                            00003800
cipher_text_len = '00000030'x                                  00003900
cipher_text     = COPIES('00'X,48)                            00004000
opt_data_len    = '00000004'x                                  00004100
opt_data        = '00000000'X                                  00004200
                                                         00004300
address linkpgm 'CSNBSYE',                                     00004400
               'ex_rc',                                         00004500
               'ex_rs',                                         00004600
               'exit_data_length',                             00004700
               'exit_data',                                     00004800
               'rule_array_count',                             00004900
               'rule_array',                                    00005000
               'key_length',                                    00005100
               'key_identifier',                                00005200
               'key_parms_len',                                00005300
               'key_parms',                                     00005400
               'block_size',                                    00005500
               'init_vector_len',                              00005600
               'init_vector',                                   00005700
               'chain_data_len',                               00005800
               'chain_data',                                   00005900
               'clear_text_len',                               00006000
               'clear_text',                                   00006100
               'cipher_text_len',                              00006200
               'cipher_text',                                  00006300
               'opt_data_len',                                 00006400
               'opt_data'                                       00006500
/* check the return code */                                     00006600
                                                         00006700
if (ex_rc <> '00000000'X) | (ex_rs <> '00000000'X) then do    00006800
  say 'CSNBSYE FAILED : RC =' c2x(ex_rc);                     00006900
  say '          RS =' c2x(ex_rs);                             00007000
  say ' ' ;                                                    00007100
end ;                                                         00007200
else do ;                                                     00007300
  say 'CSNBSYE OK '                                           00007400
  say 'Clear_Text (part1)...'|c2x(SUBSTR(clear_text,01,16))    00007500
  say 'Cipher_Text (part1)...'|c2x(SUBSTR(cipher_text,01,16))  00007600
  say ' ' ;                                                    00007700
  say 'Clear_Text (part2)...'|c2x(SUBSTR(clear_text,17,16))    00007800
  say 'Cipher_Text (part2)...'|c2x(SUBSTR(cipher_text,17,16))  00007900
  say ' ' ;                                                    00008000
  say 'Clear_Text (part3)...'|c2x(SUBSTR(clear_text,33,16))    00008100
  say 'Cipher_Text (part3)...'|c2x(SUBSTR(cipher_text,33,16))  00008200
  say ' ' ;                                                    00008300
  say 'CSNBSYE completed'                                       00008400
end ;                                                         00008500
                                                         00008600
/*-----*/                                                  00008700
/*                                                            */  00008800

```

```

/* Invoke CSNBSYD */ 00008900
/* */ 00009000
/*-----*/ 00009100
00009200
/* ***** */ 00009300
/* initialize parameters for common use */ 00009400
/* */ 00009500
00009600
/* trace i */ 00009700
say ' '; 00009800
say ' '; 00009900
say ' '; 00010000
say ' '; 00010100
say '-----'; 00010200
say ' '; 00010300
say 'Running CSNBSYD to perform AES Decryption' 00010400
say ' '; 00010500
00010600
ex_rc = '00000000'X 00010700
ex_rs = '00000000'X 00010800
exit_data_length = '00000000'X 00010900
exit_data = ' ' 00011000
rule_array_count = '00000004'x 00011100
rule_array = 'AES CBC KEY-CLR INITIAL ' 00011200
key_length = '00000010'x 00011300
key_identifier = '000102030405060708090A0B0C0D0E0F'X 00011400
key_parms_len = '00000004'x 00011500
key_parms = ' ' 00011600
block_size = '00000010'x 00011700
init_vector_len = '00000010'x 00011800
init_vector = '00000000000000000000000000000000'X 00011900
chain_data_len = '00000020'x 00012000
chain_data = COPIES('00'X,32) 00012100
/* clear_text_len = '00000030'x */ 00012200
/* clear_text = COPIES('00'X,48) */ 00012300
/* cipher_text_len = '00000030'x */ 00012400
/* cipher_text = COPIES('00'X,48) */ 00012500
opt_data_len = '00000004'x 00012600
opt_data = '00000000'X 00012700
00012800
address linkpgm 'CSNBSYD', 00012900
'ex_rc', 00013000
'ex_rs', 00013100
'exit_data_length', 00013200
'exit_data', 00013300
'rule_array_count', 00013400
'rule_array', 00013500
'key_length', 00013600
'key_identifier', 00013700
'key_parms_len', 00013800
'key_parms', 00013900
'block_size', 00014000
'init_vector_len', 00014100
'init_vector', 00014200
'chain_data_len', 00014300
'chain_data', 00014400
'cipher_text_len', 00014500
'cipher_text', 00014600
'clear_text_len', 00014700
'clear_text', 00014800

```

'opt_data_len',	00014900
'opt_data'	00015000
/* check the return code */	00015100
	00015200
if (ex_rc <> '00000000'X)   (ex_rs <> '00000000'X) then do	00015300
say 'CSNBSYD FAILED : RC =' c2x(ex_rc);	00015400
say '                  RS =' c2x(ex_rs);	00015500
say ' ' ;	00015600
end ;	00015700
else do ;	00015800
say 'CSNBSYD OK '	00015900
say 'Cipher_Text (part1)...' c2x(SUBSTR(cipher_text,01,16))	00016000
say 'Clear_Text (part1)...' c2x(SUBSTR(clear_text,01,16))	00016100
say ' ' ;	00016200
say 'Cipher_Text (part2)...' c2x(SUBSTR(cipher_text,17,16))	00016300
say 'Clear_Text (part2)...' c2x(SUBSTR(clear_text,17,16))	00016400
say ' ' ;	00016500
say 'Cipher_Text (part3)...' c2x(SUBSTR(cipher_text,33,16))	00016600
say 'Clear_Text (part3)...' c2x(SUBSTR(clear_text,33,16))	00016700
say ' ' ;	00016800
say 'CSNBSYD completed'	00016900
end ;	00017000
	00017100
exit	00017200

---

Archived

## Programs used in sysplex testing

This appendix provides the source of the REXX programs used to test the new sysplex support functions.

**Note:** The code supplied here has not been subjected to any formal IBM test and is distributed on an “AS IS” basis without any warranty either express or implied. The implementation of any of the techniques described or used herein is a customer responsibility and depends on the customers operational environment. While each item may have been reviewed for accuracy in a specific situation and may run in a specific environment, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

## REXBKRC program

Example B-1 is a REXX program that we used to generate a single DES key and write it into the CKDS using the Key Record Create and Key Record Write functions of ICSF. The key was then used to test the Key Record Read function from different z/OS images in the Sysplex.

*Example: B-1 Program to generate a single DES key and write it into the CKDS*

```

/* REXX */                                00000100
/*-----*/                               00000200
/* Create key for test                      */ 00000300
/*                                          */ 00000400
/*                                          */ 00000500
/*                                          */ 00000600
/*-----*/                               00000700
                                          00000800
/* ***** */                             00000900
/* initialize parameters for common use    */ 00001000
/*                                          */ 00001100
                                          00001200
/* trace i */                             00001300
say '-----';                          00001400
say ' ';                                00001500
say ' (Re)Creating key LENNIE.SYSPLEX.TEST002' 00001600
say ' ';                                00001700
                                          00001800
zero = '00000000'x;                     00001900
                                          00002000
/* ***** */                             00002100
/* Key Record Delete (CSNBKRD)             */ 00002200
/*                                          */ 00002300
/*                                          */ 00002400
/* ***** */                             00002500
                                          00002600
say ' ';                                00002700
say 'Key Record Delete (CSNBKRD) ';       00002800
say ' (Conditional) - delete if found '    00002900
say ' ';                                00003000
                                          00003100
ex_rc = 'FFFFFFF'x;                       00003200
ex_rs = 'FFFFFFF'x;                       00003300
exit_data_length = '00000000'x;           00003400
exit_data = ' ';                          00003500
/* Set Key Label */                       00003600
                                          00003700
key_label = LEFT('LENNIE.SYSPLEX.TEST002',64,' ') 00003800
                                          00003900
rule_array_count = '00000001'x;           00004000
rule_array = LEFT('LABEL-DL',8,' ');       00004100
                                          00004200
/** Delete key using CSNBKRD              */ 00004300
                                          00004400
address linkpgm 'CSNBKRD',                 00004500
              'ex_rc',                     00004600
              'ex_rs',                     00004700
              'exit_data_length',          00004800
              'exit_data',                 00004900
              'rule_array_count',          00005000
              'rule_array',                 00005100
              'key_label';                  00005200

```



```

/* check the return and reason codes */
if (ex_rc <> zero | ex_rs <> zero) then do
    say 'Key record delete FAILED' : RC = c2x(ex_rc);
    say ' : RS = c2x(ex_rs);
    say ' ' ;
end ;
else do ;
    say 'Key record delete is OK' : RC = c2x(ex_rc);
    say ' : RS = c2x(ex_rs);
    say ' ' ;
end ;

/* ***** */
/* Keyed Record Generate (CSNBKGN) */
/* */

say '-----';
say 'Keyed Record Generate (CSNBKGN)';
say ' ';

ex_rc = 'FFFFFFF'x;
ex_rs = 'FFFFFFF'x;

exit_data_length = '00000000'x;
exit_data = '';
key_form = LEFT('OPOP',4,' ');
key_length = LEFT('SINGLE',8,' ');
key_type1 = LEFT('MAC',8,' ');
key_type2 = LEFT('MACVER',8,' ');
kek_key_ident1 = LEFT('00'X,64,'00'X);
kek_key_ident2 = LEFT('00'X,64,'00'X);
generated_key1 = LEFT('00'X,64,'00'X);
generated_key2 = LEFT('00'X,64,'00'X);

address linkpgm 'CSNBKGN',
               'ex_rc',
               'ex_rs',
               'exit_data_length',
               'exit_data',
               'key_form',
               'key_length',
               'key_type1',
               'key_type2',
               'kek_key_ident1',
               'kek_key_ident2',
               'generated_key1',
               'generated_key2';

say ' ';
key_token = generated_key1

/* check the return and reason codes */
if (ex_rc <> zero | ex_rs <> zero) then do
    say 'Key generate(KGN) FAILED' : RC = c2x(ex_rc);
    say ' : RS = c2x(ex_rs);
    say ' ' ;

```

```

00005300
00005400
00005500
00005600
00005700
00005800
00005900
00006000
00006100
00006200
00006300
00006400
00006500
00006600
00006700
00006800
00006900
00007000
00007100
00007200
00007300
00007400
00007500
00007600
00007700
00007800
00007900
00008000
00008100
00008200
00008300
00008400
00008500
00008600
00008700
00008800
00008900
00009000
00009100
00009200
00009300
00009400
00009500
00009600
00009700
00009800
00009900
00010000
00010100
00010200
00010300
00010400
00010500
00010600
00010700
00010800
00010900
00011000
00011100
00011200

```

```

exit;
end ;
else do ;
    say 'Key generate(KGN) OK' : RC =' c2x(ex_rc);
    say ' : RS =' c2x(ex_rs);
    say ' ' ;
    say 'Generated key token in hex:' ;
    hexkt = c2x(key_token) ;
    plkt = substr(hexkt,1,16) ;
    mkvp = substr(hexkt,17,16) ;
    crgrm1 = substr(hexkt,33,16) ;
    crgrm2 = substr(hexkt,49,16) ;
    icv = substr(hexkt,65,32) ;
    crgrm3 = substr(hexkt,97,16) ;
    p2kt = substr(hexkt,113)
    say ' first bytes :' plkt ' Master Key VP :' mkvp ;
    say ' cryptogram Key1 :' crgrm1 ' cryptogram Key2 :' crgrm2 ;
    say ' Control Vector :' icv ;
    say ' cryptogram Key3 :' crgrm3 ' last bytes :' p2kt ;
    say ' ' ;
end ;

/* ***** */
/* Key record Create (CSNBKRC) */
/* */

say '-----';
say 'Key Record Create (CSNBKRC)';
say ' ';

ex_rc2 = 'FFFFFFF'x;
ex_rs2 = 'FFFFFFF'x;

exit_data_length = '00000000'x;
exit_data = ' ';

address linkpgm 'CSNBKRC',
               'ex_rc2',
               'ex_rs2',
               'exit_data_length',
               'exit_data',
               'key_label';

say ' ' ;

/* check the return and reason codes */

if (ex_rc2 <> zero | ex_rs2 <> zero) then do;
    say 'Key Record Create (KRR) fail' : RC =' c2x(ex_rc2);
    say ' : RS =' c2x(ex_rs);
    say ' ' ;
    exit;
end ;
else do ;
    say 'Key Record Create (KRR) OK' : RC =' c2x(ex_rc2);
    say ' : RS =' c2x(ex_rs);
end ;

```

```

/* ***** */
/*      Key Record Write (CSNBKRW)      */
/*      */
/*      */
/* ***** */
00017300
00017400
00017500
00017600
00017700
00017800
00017900
say '-----';
say 'Key Record Write (CSNBKRW)';
say ' ';
00018000
00018100
00018200
ex_rc      = 'FFFFFFF'x;
00018300
ex_rs      = 'FFFFFFF'x;
00018400
exit_data_length = '00000000'x;
00018500
exit_data     = '';
00018600
00018700
00018800
00018900
00019000
00019100
00019200
00019300
00019400
00019500
00019600
00019700
00019800
00019900
00020000
00020100
00020200
00020300
00020400
00020500
00020600
00020700
00020800
00020900
00021000
00021100
00021200
00021300
00021400
00021500

/** Export key by CSNBKRW */
address linkpgm 'CSNBKRW',
               'ex_rc',
               'ex_rs',
               'exit_data_length',
               'exit_data',
               'key_token',
               'key_label';

/* check the return and reason codes */

say ' ';

if (ex_rc <> zero | ex_rs <> zero) then do
    say 'Key record Write (KRW) FAILED : RC =' c2x(ex_rc);
    say '                               : RS =' c2x(ex_rs);
    say ' ';
    exit;
end ;
else do ;
    say 'Key record Write (KRW) is OK : RC =' c2x(ex_rc);
    say '                               : RS =' c2x(ex_rs);
    say ' ';
end ;

exit

```

## REXBKRD program

Example B-2 is a REXX program that can be used to delete the single DES key from the CKDS. The different z/OS images in the sysplex will later check if they find the key that is missing in the in-storage copy of the CKDS.

*Example: B-2 Program to delete the single DES key from the CKDS*

```

/* REXX */                                00000100
/*-----*/                                00000200
/* Delete key for test                      */ 00000300
/*                                          */ 00000400
/*                                          */ 00000500
/*                                          */ 00000600
/*-----*/                                00000700
                                          00000800
/* ***** */                                00000900
/* initialize parameters for common use    */ 00001000
/*                                          */ 00001100
                                          00001200
say '-----';                            00001300
say ' ';                                  00001400
say ' Deleting key LENNIE.SYSPLEX.TEST002' 00001500
say ' ';                                  00001600
                                          00001700
zero = '00000000'x;                      00001800
                                          00001900
/* ***** */                                00002000
/* Key Record Delete (CSNBKRD)             */ 00002100
/*                                          */ 00002200
/*                                          */ 00002300
/* ***** */                                00002400
                                          00002500
say ' ';                                  00002600
say 'Key Record Delete (CSNBKRD) ';        00002700
say ' (Conditional) - delete if found '    00002800
say ' ';                                  00002900
                                          00003000
ex_rc = 'FFFFFFF'x;                       00003100
ex_rs = 'FFFFFFF'x;                       00003200
exit_data_length = '00000000'x;           00003300
exit_data = '';                           00003400
/* Set Key Label */                       00003500
                                          00003600
key_label = LEFT('LENNIE.SYSPLEX.TEST002',64,' ') 00003700
                                          00003800
rule_array_count = '00000001'x;           00003900
rule_array = LEFT('LABEL-DL',8,' ');      00004000
                                          00004100
/*** Delete key using CSNBKRD             */ 00004200
                                          00004300
address linkpgm 'CSNBKRD',                00004400
              'ex_rc',                    00004500
              'ex_rs',                    00004600
              'exit_data_length',         00004700
              'exit_data',               00004800
              'rule_array_count',         00004900
              'rule_array',              00005000
              'key_label';                00005100
                                          00005200

```



## REXBKRR program

Example B-3 is a REXX program that can be used to check if the z/OS images have access to the latest version of the key record.

*Example: B-3 Program to read the key record*

```

/* REXX */                                00000100
/*-----*/                               */ 00000200
/* Read key token                          */ 00000300
/*                                         */ 00000400
/*                                         */ 00000500
/*                                         */ 00000600
/*-----*/                               */ 00000700
                                         00000800
/* ***** */                             */ 00000900
/* initialize parameters for common use    */ 00001000
/*                                         */ 00001100
                                         00001200
/* trace i */                             00001300
say '-----';                           00001400
say ' ';                                00001500
say ' Read key LENNIE.SYSPLEX.TEST002'    00001600
say ' ';                                00001700
                                         00001800
zero = '00000000'x;                     00001900
                                         00002000
/* ***** */                             */ 00002100
/* Key Record Read (CSNBKRR)               */ 00002200
/*                                         */ 00002300
/*                                         */ 00002400
/* ***** */                             */ 00002500
                                         00002600
say ' ';                                00002700
say 'Key Record Display (using CSNBKRR) '; 00002800
say ' ';                                00002900
                                         00003000
ex_rc = 'FFFFFFF'x;                      00003100
ex_rs = 'FFFFFFF'x;                      00003200
exit_data_length = '00000000'x;          00003300
exit_data = ' ';                          00003400
/* Set Key Label */                      00003500
                                         00003600
key_label = LEFT('LENNIE.SYSPLEX.TEST002',64,' ') 00003700
key_token = LEFT('00'x,64,'00'x)         00003800
                                         00003900
/**** Read key using CSNBKRR */           00004000
                                         00004100
address linkpgm 'CSNBKRR',               00004200
              'ex_rc',                    00004300
              'ex_rs',                    00004400
              'exit_data_length',         00004500
              'exit_data',                00004600
              'key_label',                00004700
              'key_token';                00004800
                                         00004900
/* check the return and reason codes */    00005000
                                         00005100
say ' ';                                00005200
                                         00005300

```

```

if (ex_rc <> zero | ex_rs <> zero) then do                                00005400
    say 'Key record read FAILED' : RC =' c2x(ex_rc);                    00005500
    say ' : RS =' c2x(ex_rs);                                           00005600
    say ' ' ;                                                            00005700
    end ;                                                                00005800
else do ;                                                                00005900
    say 'Key record read is OK' : RC =' c2x(ex_rc)                      00006000
    say ' : RS =' c2x(ex_rs);                                           00006100
    say ' ' ;                                                            00006200
    hexkt = c2x(key_token) ;                                             00006300
    plkt = substr(hexkt,1,16) ;                                          00006400
    mkvp = substr(hexkt,17,16) ;                                          00006500
    crgrm1 = substr(hexkt,33,16) ;                                       00006600
    crgrm2 = substr(hexkt,49,16) ;                                       00006700
    icv = substr(hexkt,65,32) ;                                          00006800
    crgrm3 = substr(hexkt,97,16) ;                                       00006900
    p2kt = substr(hexkt,113) ;                                           00007000
    say ' first bytes : ' plkt ' Master Key VP : ' mkvp ;              00007100
    say ' cryptogram Key1 : ' crgrm1 ' cryptogram Key2 : ' crgrm2 ;      00007200
    say ' Control Vector : ' icv ;                                       00007300
    say ' cryptogram Key3 : ' crgrm3 ' last bytes : ' p2kt ;            00007400
    say ' ' ;                                                            00007500
    end ;                                                                00007600
                                                                00007700
exit                                                                    00007800

```

---

Archived



# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

## IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 134. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Exploiting S/390 Hardware Cryptography with Trusted Key Entry*, SG24-5455
- ▶ *S/390 Crypto PCI Implementation Guide*, SG24-5942
- ▶ *zSeries Crypto Guide Update*, SG24-6870
- ▶ *IBM @server zSeries 990 (z990) Cryptography Implementation*, SG24-7070
- ▶ *zSeries TKE V4.2 Update*, SG24-6499

## Other publications

These publications are also relevant as further information sources:

- ▶ *z/OS Cryptographic Services ICSF Trusted Key Entry PCIX Workstation User's Guide*, SA23-2211
- ▶ *z/OS ICSF Overview*, SA22-7519
- ▶ *z/OS ICSF System Programmer's Guide*, SA22-7520
- ▶ *z/OS Cryptographic Services Integrated Facility Administrator's Guide*, SA22-7521
- ▶ *z/OS ICSF Application programmer's Guide*, SA22-7522
- ▶ *z/OS ICSF TKE PCIX Workstation - User's Guide*, SA23-3211
- ▶ *Maintenance Information for Desktop Consoles*, GC28-6847
- ▶ *z/Architecture Principles of Operation*, SA22-7832
- ▶ *zSeries 990 Service Guide*, G229-9039
- ▶ *System z9 Processor Resource/Systems Manager Planning Guide*, SB10-7041

## Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ IBM Systems Journal article  
<http://domino.research.ibm.com/tchjr/journalindex.nsf/>
- ▶ System z9 Cryptographic Coprocessors Performance  
<http://www.ibm.com/servers/eserver/zseries/security/cryptography.html>

## How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

[ibm.com/redbooks](http://ibm.com/redbooks)

## Help from IBM

IBM Support and downloads

[ibm.com/support](http://ibm.com/support)

IBM Global Services

[ibm.com/services](http://ibm.com/services)

# Index

## A

abend codes 64  
accelerator 32  
access control points 81  
Adjunct Processor 28  
AES 10

## C

CCA Command Processors 68  
CCF 77  
CEX2A 26, 49  
CICS Wait List 73  
CKDS 9  
clear key internal token 52  
clear keys 19  
code segments 29  
component trace 63  
Control Domain Index 40  
CP Assist for Cryptographic function 2  
CPACF 2, 77  
CPACF020 22  
Crypto Express2 77  
Crypto modules 77  
Cryptographic  
    candidate List 40  
    coprocessors 77  
    domains 26  
    function support 2  
    processors 2  
    throughput 43  
Cryptographic Key Data Set 9  
Cryptographic Online List 40  
CSFEUTIL 53, 58  
CSFKEYS 20  
CSFPCI 68  
CSFPUTIL 63  
CSFSERV 20  
CSNBOWH 48  
CSNBRNG 19, 48  
CSNBSYD 18  
CSNBSYE 18–19  
CSNDPKD 3, 5  
CSNDPKE 3, 5

## D

DES 2

## E

EMV 2000 3  
Ethernet LAN 78

## F

FC0800 77  
FC0855 6  
FC0859 6  
FC0860 77  
FC0861 77  
FC0863 6, 77  
FC0868 77  
FC0887 6  
FC0888 6  
Feature Code 3863 2, 14  
FIPS 140-2 2, 32

## H

hash 17  
HCR7706 46  
HCR7708 46  
HCR770A 46  
HCR770B 46  
HCR7720 46  
HCR7730 18, 46  
HSA 27

## I

IBM 4753 4  
IBM 4758-2 Cryptographic Adapter 77  
IBM 4764 Cryptographic Adapter 77  
IBM 4764-PCIXCC 26  
IBM Systems Journal 26  
ICSF 9  
IDCAMS 50  
Image Profile 39

## J

Joan Daemen 20

## K

Key Part Import 55  
Key Record  
    Create 55  
    Delete 55  
    Read 20, 50  
    Write 50, 55  
Key Token Build 49  
KGUP 50  
    TYPE(CLRAES) 50  
    TYPE(CLRDES) 50  
KLMD 2, 16  
KM 2, 15  
KMAC 2, 16  
KMC 2, 16  
KMID 2

## **L**

LIC Release 47  
    January 2005 47  
    May 2004 47  
Linux 11, 30  
Load Operational Key 55

## **M**

MAC 18–19  
May 2004 LIC Release 47  
Message-Security Assist 14, 17  
MRP 49  
multiple CKDS 60

## **O**

OS/390 V2R9 32

## **P**

PCHID 28  
PCICA 26, 77  
PCICC 32  
PCI-X 3  
PCIXCC 26, 77  
PKA Key Storage 80  
PKDS 9  
PKDSCACHE 62  
PowerPC 405 GPr 29  
Private Key Data Set 9  
PRNG 10, 18–19  
PSC 80  
Public Key Algorithm 3

## **Q**

Query 16

## **R**

Reconfiguration 32  
    zeroize 33–34  
Redbooks Web site 134  
    Contact us x  
Retained key support 4  
REXBKRC 57  
REXBKRD 57  
REXBKRR 58  
REXBSYE 22  
REXBSYED 22  
REXCP020 22  
RMF records 21  
RSA 3

## **S**

SHA-1 17  
SHA-256 10, 18, 21  
SHAREOPTIONS 53  
SMF Type 82 63  
SSL 2

STI 27  
Stub 70  
stub 69, 72  
SYSICSF 55  
sysplex support 52  
SYSPLEXCKDS 54, 61  
SYSZCKDS 55  
SYSZCKT 55

## **T**

tamper detection 31  
    imprinting 32  
    zeroizing 32

## **TKE**

4764initialize.cni 90  
applications tasks 83  
configuration tasks 85  
Cryptographic Adapter initialization 89  
DNS 93  
enablement 79  
group 36  
host 76  
maintenance tasks 85  
service applications tasks 85  
Smart Card Utility Program 91  
system clock 88  
system management tasks 84  
TCP/IP setup 91  
trusted key entry tasks 83  
utilities tasks 84  
welcome panel 82  
TKE Smart Card readers 78  
TKE V2 79  
TKE V3 80  
TKE V4 80  
TKE V5.0 75  
TKECM 78, 80

## **U**

UDX 4, 67  
    activation 72  
    callable service 69  
    function code identifier 68  
Usage Domain Index 40

## **V**

Vincent Rijmen 20  
VPN 2

## **Z**

z/VM V5.1 10–11  
z800 32  
ZERO-PAD 49









# z9-109 Crypto and TKE V5 Update



**Redbooks**

**The enhanced CPACF functions of System z9, with code samples**

**The Accelerator mode of the Crypto Express 2 card**

**The new TKE Version 5**

This IBM Redbook provides detailed information on the implementation of hardware cryptography in the new System z9, along with the new version of the Trusted Key Entry (TKE) workstation that is required when a TKE is to manage System z9 cryptographic coprocessors. It also addresses the CKDS sysplex support delivered in ICSF HCR7730, which is not dependent on the use of a System z9.

It is expected that the reader is familiar with zSeries hardware cryptography implementation and the purpose and usage of the TKE workstation.

## **INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION**

### **BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:**  
**[ibm.com/redbooks](http://ibm.com/redbooks)**

SG24-7123-00

ISBN 0738494178