Cryptographic system enhancements for the IBM System z9

T. W. Arnold
A. Dames
M. D. Hocker
M. D. Marik
N. A. Pellicciotti
K. Werner

IBM has offered hardware-based cryptographic processors for its mainframe computers for nearly thirty years. Over that period, IBM has continued to update both the hardware and software, providing added features, higher performance, greater physical security, and improved management features. This commitment continues with the System z9™, as demonstrated by the two improvements described in this paper. The first part of the paper describes enhancements to the System z9 to configure and control cryptographic features. The second part describes a new method for the cryptographic coprocessors to securely manage keys which are distributed to remote devices that are not necessarily in secure or well-controlled environments.

Introduction

Encryption is a vital part of today's business processes and information systems. Transactions sent across networks must be protected from eavesdropping and alteration. Data files on Internet-connected servers must be protected from malicious hackers. Secure Sockets Layer (SSL) traffic must be encrypted at high speeds. The list of areas that benefit from encryption grows every year.

IBM mainframe systems have long been designed with this need for encryption in mind. Today's IBM System z* offers a number of standard and optional hardware-based encryption features to satisfy nearly all customer application encryption requirements. In addition, the System z hardware and software provide the features necessary to easily manage the cryptographic configuration, and in a manner that is integrated with the other System z management facilities.

IBM continuously adds support for new customer requirements, and this generation of System z adds two important improvements to existing cryptographic facilities based on such requirements. The first improvement provides a flexible way to configure cryptographic hardware. The second provides improved cryptographic key management, targeted principally at loading of encryption keys in remotely located automatic teller machines (ATMs).

Cryptographic configuration feature

The cryptographic hardware available on System z9* consists of the message security assist functions and the Crypto Express2 (CEX2) feature. The message security assist, also called CP (central processor) assist for cryptographic functions (CPACF), is available on every CPU of the system. The CEX2 feature makes use of the 4764 PCI-X Cryptographic Coprocessor. Two coprocessors are plugged into the PCI-X (Peripheral Component Interconnect Extended) slots in the Hydra 3 book, which is a physical, electrical, and logical adaptation layer. The number 4764 references the model number (or machine type) of the cryptographic coprocessor card when sold for platforms other than the System z. The 4764 designation is used in this paper to refer to the individual cards [1]. The CEX2 feature was first introduced on IBM System z990 and called the CEX2 coprocessor.

The cryptographic configuration enhancement that is introduced on System z9 permits the 4764 PCI-X cryptographic adapter to be configured to run in one of two different modes:

- *Accelerator mode* (i.e., the fast path), which provides a function that is similar to that provided by the PCI cryptographic accelerator (PCICA).
- *Coprocessor mode* (i.e., the normal path), which provides a function that is functionally similar to the PCI-X cryptographic coprocessor (PCIXCC).

The difference between these two modes is illustrated in **Figure 1.** The term *RSA* in Figure 1 refers to the

0018-8646/07/$5.00 © 2007 IBM

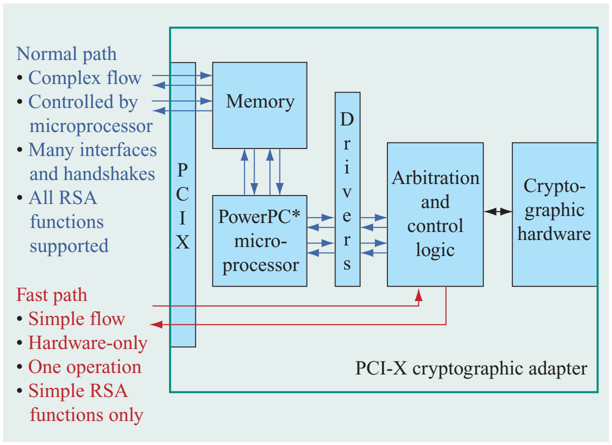

Normal path
• Complex flow
• Controlled by microprocessor
• Many interfaces and handshakes
• All RSA functions supported
Fast path
• Simple flow
• Hardware-only
• One operation
• Simple RSA functions only
PCIX
Memory
PowerPC* micro-processor
Drivers
Arbitration and control logic
Crypto-graphic hardware
PCI-X cryptographic adapter

Figure 1

Fast-path and normal-path interfaces.

Table 1 Cryptographic feature history prior to the IBM System z9.

Cryptographic feature	*Date introduced*	*IBM System z model*
PCICC	06/1999	9672 G5 9672 G6 z800 and z900
PCICA	10/2001	z800 and z900 z890 and z990
PCIXCC	09/2003	z890 and z990
Crypto Express2 (CEX2)	01/2005	z890 and z990

encryption algorithm developed in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman.

History of the System z cryptographic feature

For decades, large IBM computing systems, of which the most recent is known as System z, have contained specific hardware to support cryptographic functions. Recent cryptographic features have tended to be less tightly coupled to the central processor (CP), with the cryptographic features functioning somewhat like I/O. These features are in contrast to CPACF, which has fast but less secure hardware closely associated with CPs.

The earliest of the I/O-like cryptographic features, the PCI cryptographic coprocessor (PCICC), used the Hydra 1 book package with one 4758 PCI cryptographic adapter. This feature was introduced on the 9672 G5, an early model of the IBM System z (**Table 1**).

Performance for Secure Sockets Layer (SSL) [2] was significantly improved with the introduction of the PCICA on System z900. The PCICA supports only unencrypted RSA keys and is designed for cryptographic acceleration. It uses multiple IBM-designed cryptographic chips identical to those used in the 4758 adapter.

The 4764 PCI-X cryptographic adapter is supported on the following System z servers:

- Systems z890 and z990, where it is called the PCIXCC. This is packaged in a Hydra 1.75 book containing a single 4764 PCI-X cryptographic adapter.
- Systems z890 and z990, where it is also called the Crypto Express2 Coprocessor (CEX2C). This is packaged in a Hydra 3 book containing two 4764 PCI-X cryptographic adapters.
- System z, where it is called the CEX2. This is packaged in a Hydra 3 book with two 4764 PCI-X cryptographic adapters, with additional accelerator hardware enabled.

In the current System z9 processor complex, up to eight CEX2 books can be installed in the I/O cages. The I/O cage provides high-bandwidth I/O slots to enable a greater number of I/O ports in the system.

The CEX2 can be run in two modes, the coprocessor mode (CEX2C) and the accelerator mode (CEX2A). The CEX2 has functionally replaced the PCICC, the PCICA, and the PCIXCC to provide a lower total cost of ownership with enhanced performance. These latter cryptographic features are not supported on System z9.

Motivation for the introduction of the cryptographic configuration feature

Prior to the advent of System z9, and particularly prior to the introduction of the 4764 PCI-X cryptographic adapter, System z utilized a variety of IBM-developed cryptographic products. In System z9, the 4764 PCI-X cryptographic adapter is retained as the only secure cryptographic feature.

Because earlier cryptographic products are no longer being manufactured, product lines are being simplified, and the CEX2 feature is superior to previous products in terms of performance improvements and hardware acceleration, it has become desirable to support only one cryptographic feature, and this requires a means to configure the CEX2 to run either in coprocessor or accelerator mode.

The cryptographic configuration feature, discussed in forthcoming sections of this paper, was designed to provide a high level of security, high-performance acceleration of RSA public key operations, robustness, and ease of use.

Cryptographic configuration feature design

Overview

The CEX2 feature consists of a Hydra 3 book package containing two 4764 PCI-X cryptographic adapters [1]. The 4764 PCI-X cryptographic adapter incorporates two different communication paths for host access. With the introduction of the PCIXCC on System z990, only the coprocessor mode was available for use. Subsequently, when the CEX2 feature was introduced on System z9, both paths to the 4764 PCI-X cryptographic adapter were available for use, although they could only be used one at a time. A system operator may configure the communication path using the manual controls of the support element (SE). The SE is a dedicated workstation (e.g., laptop computer) supplied with each System z9 to provide a console for monitoring and operating the System z9. This configuration is communicated to the Hydra 3, which provides for the setup, initialization, and use of the appropriate communication path to the 4764 PCI-X cryptographic adapter.

SE, i390, and Hydra implementation

Figure 2 illustrates a high-level view of the different components that are involved in the configuration, initialization, diagnostic, monitoring, and functional usage of the CEX2 feature. Also shown is the possibility of supporting a variety of different IBM operating systems. Each 4764 PCI-X cryptographic adapter (CEX2C/CEX2A) is designed to support a maximum of 16 logical partitions. In Figure 2, i390 and millicode designate firmware code layers, providing System z architecture instruction support and an interface to the hardware. The Processor Resource/Systems Manager* (PR/SM*) provides support for logical partitioning. The SE is a firmware code layer that is directly attached to the server itself. It provides a console for monitoring and operating the system. The HMC is a single point of control for multiple systems, including the System z9, zSeries*, and S/390* platforms. The HMC communicates with each server through the SE, allowing for the remote operation of the server through the SE.

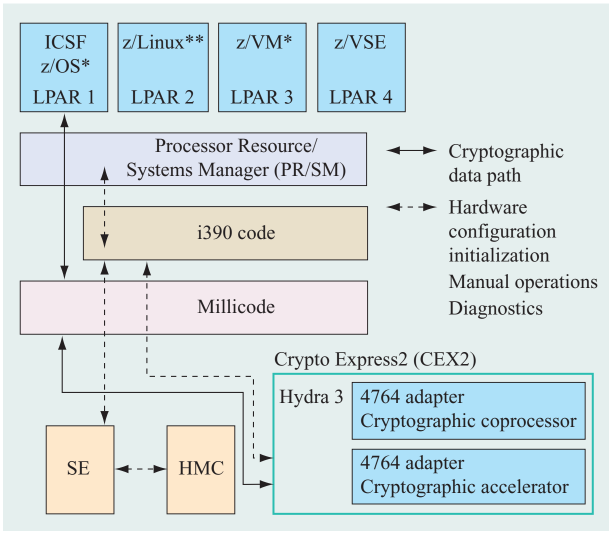
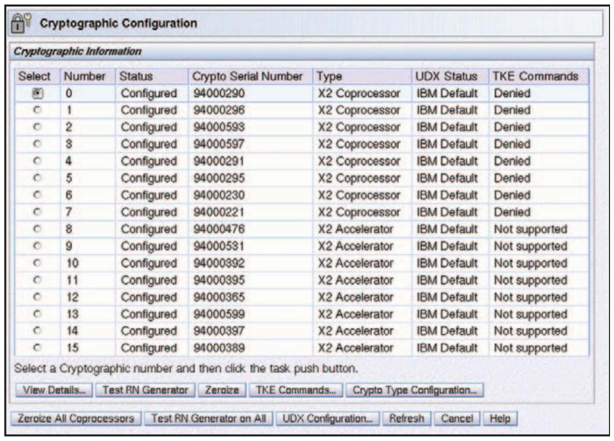


Figure 2

Components associated with the CEX2 feature. The solid arrows represent data paths, and the dashed arrows represent signal paths used for hardware configuration, initialization, manual operations, and diagnostics. (HMC: hardware management console; SE: support element; ICSF: Integrated Cryptographic Service Facility; VSE: Virtual Storage Extended, an operating system; LPAR: logical partition.)



Select	Number	Status	Crypto Serial Number	Type	UDX Status	TKE Commands
●	0	Configured	94000290	X2 Coprocessor	IBM Default	Denied
○	1	Configured	94000296	X2 Coprocessor	IBM Default	Denied
○	2	Configured	94000593	X2 Coprocessor	IBM Default	Denied
○	3	Configured	94000597	X2 Coprocessor	IBM Default	Denied
○	4	Configured	94000291	X2 Coprocessor	IBM Default	Denied
○	5	Configured	94000295	X2 Coprocessor	IBM Default	Denied
○	6	Configured	94000230	X2 Coprocessor	IBM Default	Denied
○	7	Configured	94000221	X2 Coprocessor	IBM Default	Denied
○	8	Configured	94000476	X2 Accelerator	IBM Default	Not supported
○	9	Configured	94000531	X2 Accelerator	IBM Default	Not supported
○	10	Configured	94000392	X2 Accelerator	IBM Default	Not supported
○	11	Configured	94000395	X2 Accelerator	IBM Default	Not supported
○	12	Configured	94000365	X2 Accelerator	IBM Default	Not supported
○	13	Configured	94000599	X2 Accelerator	IBM Default	Not supported
○	14	Configured	94000397	X2 Accelerator	IBM Default	Not supported
○	15	Configured	94000389	X2 Accelerator	IBM Default	Not supported

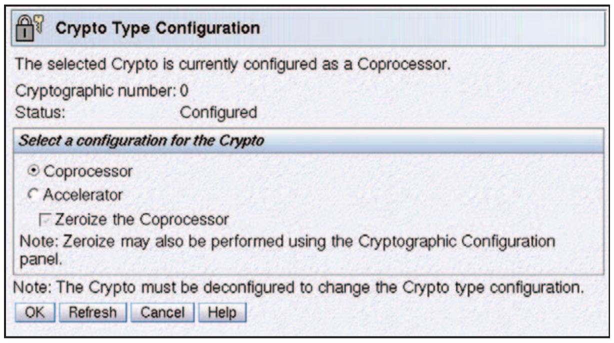
Select a Cryptographic number and then click the task push button.

Figure 3

Cryptographic configuration panel.

Support element implementation

Using the cryptographic configuration panel (**Figure 3**) on the SE, the system operator can configure the 4764 PCI-X cryptographic adapter (referred to as "Crypto" on the SE panels) to run as either a coprocessor or an accelerator. The cryptographic configuration panel allows the user to specify many configuration options and functions pertaining to the CEX2 cards installed on the system. For example, a user-defined extension (UDX) file containing custom firmware can be imported and activated, trusted key entry (TKE) commands can be permitted or denied for a particular 4764, or a 4764 can be zeroized. *Zeroizing* is the process of erasing application-level security-relevant data items (SRDIs) within the card; however, note that zeroizing leaves the 4764 operational [3]. The zeroize function is typically used in preparation for removal of the card from the system.


Crypto Type Configuration

The selected Crypto is currently configured as a Coprocessor.

Cryptographic number: 0

Status: Configured

Select a configuration for the Crypto

[✓ Coprocessor]

Accelerator

Zeroize the Coprocessor

Note: Zeroize may also be performed using the Cryptographic Configuration panel.

Note: The Crypto must be deconfigured to change the Crypto type configuration.

Figure 4

Cryptographic type configuration panel.

The cryptographic configuration panel in Figure 3 shows the status and configuration of all the 4764 PCI-X cryptographic adapters, in addition to the cryptographic type configuration for every cryptographic adapter installed on the system. "X2 Coprocessor" designates a cryptographic adapter that has been configured to run as a coprocessor, and "X2 Accelerator" designates a cryptographic adapter that has been configured to run as an accelerator.

The cryptographic type configuration panel (**Figure 4**) is used by the system operator to change the cryptographic type configuration of a cryptographic adapter. The configuration of the cryptographic adapter can be changed only when the adapter is offline.

If the cryptographic type configuration is being changed from that of a coprocessor to that of an accelerator, the option to zeroize the coprocessor is available to ensure that the SRDIs are removed before the adapter is used as an accelerator. When a CEX2 feature is added to the system, either concurrently or nonconcurrently, the adapter is initially configured as a coprocessor by default.

The cryptographic type configuration data is stored on the SE. The stored data includes the configuration setting and the option to zeroize the coprocessor. The next time the adapter is brought online, the SRDIs will first be erased, if requested, and then the adapter will be initialized to run in the selected mode.

Cryptographic type configuration data also persists during system power-off and power-on, partition reactivation with different activation profiles, configure off and on action, and when new cryptographic code is loaded. The data also persists during the zeroizing of SRDIs and during the replacement of the CEX2 feature. The CEX2 feature can be replaced as part of a repair action performed by a service representative.

i390 firmware implementation

The interface function to communicate with the CEX2 is accomplished using a queue structure, which consists of a set of 16 queues per cryptographic adapter, allocated in the hardware system area (HSA). HSA is a logical area of central storage, not addressable by application programs, used to store firmware and control information. Each queue consists of eight data elements.

Communication from an application within an operating system to the 4764 PCI-X cryptographic adapter is accomplished via these queues, using architected ESA/390 instructions. [Enterprise Systems Architecture/390* (ESA/390) was introduced in the 1990s and is the IBM mainframe computing design and successor of System/370*.] A queue must be assigned and configured to a logical partition, using the customize image profiles manual control on the SE. Aside from the 16 queues used for communications with an application, an additional queue is provided for maintenance commands.

i390, the central electronic complex (CEC) firmware layer for cryptography, provides support for the following:

- The communication path for maintenance commands (SE-related cryptographic requests and manual operations to the CEX2 feature).
- Maintenance of the queue structure for initialization, configuration, and error and recovery handling.
- Additional support in the queue control structure, which is required to provide Hydra firmware the appropriate configuration and zeroization information for use during initialization of the CEX2 feature.

CEX2 implementation

In the System z9, the 4764 PCI-X cryptographic adapter is incorporated in a Hydra 3 version of a System z common I/O package (CIOP) [4]. **Figure 5** is a photograph of the Hydra 3 CEX2 book, which contains the two 4764 PCI-X cryptographic adapters. The Hydra 3 book comprises two essentially separate, highly reliable intelligent data paths to the 4764 cards, each path having a PowerPC embedded processor within an application-specific integrated circuit (ASIC) with PCI-X bus connectivity to the 4764 and a proprietary self-timed interface (STI) bus connection to the rest of the CEC processor complex.

The firmware contained within the book is an adaptation layer between the 4764 and the CEC. The firmware transports the data requests to and from the 4764 with transport layer introspection as necessary. In addition, the Hydra 3 firmware handles exceptions, error recovery, and updating of firmware for the 4764, and it is the local repository of log messages produced by the firmware running on the 4764.

As shown in **Figure 6**, the four principal components of the Hydra 3 firmware that provides the functionality of the 4764 are the following:

- A component called the host interface, which handles communications with the i390/millicode and data transport via queuing structures contained in system memory.
- A miniboot state engine, which handles the specialized tasks associated with transporting revised or new cryptographically signed code into the 4764.
- A device driver (DD) component, which handles low-level hardware communications with the 4764 card itself, along with reset functions required by the PCI-X bus and by the distinct 4764 hardware.
- A maintenance component, which handles external requests, called maintenance element requests, primarily invoked by manual operations accomplished using the SE console to view status, provide zeroization, and perform related functions.

Firmware components also provide miscellaneous utility support and initialization functions for the Hydra hardware. With the exception of the error logging and recovery, these additional components are not described in any detail in this paper, but these features include Hydra initialization, error logging and recovery, concurrent update of the Hydra firmware, support for tasks associated with localized error determination, and support for many minor functions such as timers and storage allocation routines required by a typical embedded processor application.

The overall structure of the firmware remains unchanged, with respect to the Systems z890 and z990, with the addition of the CEX2A capability to the CEX2 feature. The principal changes to the firmware structure involved the handling of the additional architected clear key request blocks, changes to the device driver to support the additional hardware data path for the fast path, and certain changes associated with security.

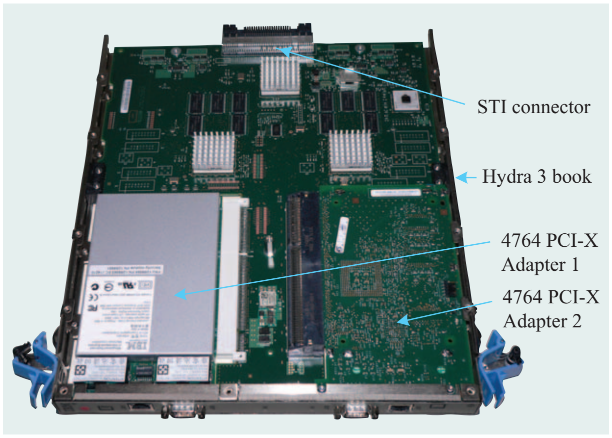


Figure 5

Hydra 3 book package without cover.

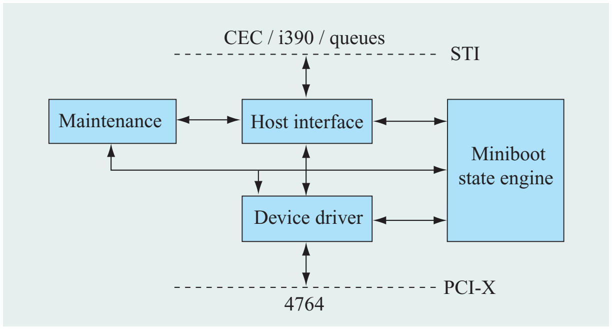


Figure 6

Overview of the firmware in Hydra used in conjunction with the CEX2.

Host interface

The host interface component of the firmware is the gateway for inbound cryptographic requests; it has several tasks, some of which are new or changed for fast-path support in the CEX2:

- The host interface provides rudimentary syntax checking of cryptographic requests (primarily inbound requests, but also outbound requests) in order to verify that the data constructs are not faulty. For instance, data bounds are examined to ensure that invalid storage accesses do not occur. The new fast-path CEX2A clear-key format of the request block required unique checks to be added. Note that Hydra host interface syntax checks are primarily intended to avert problems in transporting data requests. The firmware contained within the 4764 itself for normal-path requests, or the field-programmable gate array (FPGA)/cryptographic chip for fast-path requests, has the responsibility of implementing complete command checking.
- The host interface limits requests to either normal-path (CEX2C) or fast-path (CEX2A) requests. Although the 4764 supports both fast-path and normal-path requests simultaneously, for System z the Hydra 3 is configured to support only CEX2C or CEX2A requests at any one time. This structure

is enforced at the host interface part of the Hydra 3 firmware.
- A zeroization step was added to the host interface for the support of fast-path. The i390 firmware sets a state bit indicating to the Hydra firmware that a zeroization of the application-level SRDIs within the 4764 is required. This state is examined by Hydra after exiting from reset but prior to notifying i390 that inbound requests can be accepted. If the zeroization state is set, a command is constructed to zeroize the application-level SRDIs within the 4764.
- Additional data is collected for the resource measurement facility (RMF). The RMF provides an indication to the customer of the utilization rate of the cryptographic facility. On the basis of the RMF data, the customer can, for example, make an informed decision about resource allocation, such as decisions associated with plans to install additional CEX2 features.

Miniboot

The specialized miniboot state engine was not modified for support of the fast path. The Hydra-side miniboot state engine interacts with an equivalent miniboot partner within the 4764 so as to transport appropriately signed firmware from the host side to the firmware within the 4764 secure boundary in a manner that is resistant to potentially nefarious activity. Thus, for example, a signed firmware object is examined only by specialized code, which in turn runs only after a reset. The reset ensures that there is no vestige of previously running and potentially dubious firmware that may taint the examination of a signature for a signed object.

Fast-path is a hardware-only path through the 4764 and has no firmware component that requires the use of miniboot. The only exception would occur if a change to an application were necessary within the 4764 in order to set the appropriate enablement registers for fast-path.

Device driver

Of all of the components within the Hydra cryptographic firmware, the device driver underwent the largest change with respect to Systems z890 and z990. For Hydra cryptographic firmware, the device driver implementation performs tasks that are larger than those typically performed in other related platforms. One primary reason for this is that for the Hydra-based cryptographic adapters, the bulk of the Hydra-embedded controller application code is unchanged with the exception of the device driver. Thus, certain functions which normally would reside in a higher layer of the coding structure are subsumed by the various device drivers in order to reduce overall development resource utilization.

For the support of fast-path, the main firmware changes fell into several categories:

- Device driver support for an additional direct memory access (DMA) data path. Modular math (MM) requests (i.e., the clear-key requests used in fast-path) have a separate DMA path into the 4764. Different routing and registers are used for this purpose and require support by the device driver.
- Because the normal- and fast-path hardware DMA channels are different, as described above, additional checking prior to using the hardware is necessary. In all cases that include the normal path, the hardware is verified as operational prior to use for customer operations. For fast-path, the additional checking code uses “known answer tests” (KATs), which are used to verify the end-to-end correctness of that path from the device driver through the 4764 cryptographic ASIC and back to the device driver. Each KAT test is a representative cryptographic request with a known answer. The request is sent to the 4764 via the fast-path DMA channel, and the reply must be verified as being the same as the known answer. A KAT failure renders the CEX2 unusable.

The fast-path hardware returns specialized error codes depending on the problem encountered by the fast-path hardware in the 4764. The device driver translates these specialized error codes into return codes, as described in the System z9 architecture documents.

Error handling and recovery

Hydra firmware generates various trace data during normal operations. If a failure is encountered, this trace data is a valuable tool for determining the causes leading to the failure. In addition to trace data, error-state information is collected at the time of fault detection and saved to a log file in order to provide sufficient information to determine and repair the root cause of the failure. In no case is security-relevant data, such as the contents of a customer request, saved into logged data. Fast-path support has its own set of new trace entries and provides the collection of error-state information. These features are added to the overall Hydra firmware.

Improved remote key distribution

In the next few sections of this paper, we discuss improved methods for remote key distribution within the framework of the IBM Common Cryptographic Architecture (CCA), a carefully architected set of cryptographic functions and application programming interfaces (APIs) that provide both general-purpose functions and a broad set of functions designed specifically to secure financial transactions.

With the introduction of the System z9, the IBM CEX2 coprocessor feature (CEX2C) adds new methods for securely transferring symmetric encryption keys to remote devices, such as automated teller machines (ATMs), PIN entry devices, and point-of-sale terminals. (The term *symmetric encryption* refers to a type of encryption in which the same key is used to encrypt and decrypt the message.) The coprocessor feature may also be used to transfer symmetric keys to another cryptographic system of any type, such as a different kind of hardware security module (HSM) in an IBM or non-IBM computer server. These new methods for transferring symmetric encryption keys are added to the IBM CCA API, which is the programming interface used with the CEX2C as well as cryptographic features for other IBM servers. On System z servers, the Integrated Cryptographic Service Facility (ICSF) component of z/OS provides the CCA API software.

These new methods for transferring keys are especially important for banks, because for initial key distribution, they replace expensive operations by humans with network transactions that can be processed quickly and inexpensively. The new CCA features allow applications to support the recently approved ANSI X9.24-2 standard, which was driven by banks and ATM vendors and defines acceptable methods for this kind of key distribution. New models of ATMs are being deployed today with features to support this standard.

It has always been difficult to exchange keys between HSMs that have different architectures without compromising security. Since different vendors attach security attributes to keys in different ways, it has often been necessary to remove the attributes when exchanging keys, which can expose the keys to misuse. The new features provide a way to translate CCA keys to other attribute styles without removing the security attributes.

These novel and flexible methods support a variety of requirements, fulfilling the new needs of the banking community while simultaneously making significant interoperability improvements to related cryptographic key management functions. For the purposes of our discussion, the ATM scenario is used to illustrate the operation of the new methods. However, other uses of this method are also valuable.

Definitions

The following definitions will be useful to keep in mind for the remainder of this paper.

- *ATM* – An automated teller machine, used to perform banking transactions.
- *Master key* – A key stored in a secure cryptographic device for the purpose of encrypting keys, to be used in that device, which are stored externally in unprotected storage. The CEX2C coprocessor has two master keys—a Data Encryption Standard (DES) master key used to protect DES and Triple Data Encryption Standard (TDES) keys, and a public-key algorithm (PKA) master key used to protect RSA [5] keys and other public-key objects.
- *MKVP* – A master key verification pattern. This is a cryptographically calculated hash of the cleartext (unencrypted) value of a master key, which can be used to verify that the correct key value is used without disclosing information about any bits of the key itself.
- *Key-encrypting key (KEK)* – A symmetric key that is used to encrypt a key for secure transport to another device over unprotected paths. Both devices must have the same KEK key value so that one can encrypt a key with it and the other can decrypt the key after it is received.
- *Internal key* – A key that is intended for use on the local cryptographic device. This key is encrypted with a master key associated with the cryptographic device.
- *External key* – A key that is for exchange with another cryptographic device. This key is encrypted with a transport key, also called a key-encrypting key (KEK). The KEK is shared with the other device to which the key may be transmitted.
- *Variant* – A value used to modify a key value. The variant is generally a binary string of the same length as the key, and it is exclusive-ORed with the key value to produce a variant key that is used for some cryptographic operation. Variants are often used to produce versions of a base key that are intended for specific purposes.
- *MAC* – A message authentication code. This is a cryptographically computed checksum that uses a cryptographic key to produce a fixed-length hash of a variable-length message string. The MAC changes if any portion of the message is changed, or if the wrong key is used.
- *TDES* – An abbreviation for Triple-DES, the version of the DES encryption algorithm that uses either 128-bit or 192-bit encryption keys. The effective length of these keys is 112 bits and 168 bits, because one bit of each byte is used for parity.

Remote key loading

Remote key loading refers to the process of installing symmetric encryption keys from a central administrative site into a remotely located device. This entails two phases of key distribution. The first involves distribution of initial KEKs to a newly installed device. The second phase involves distribution of operational keys or replacement KEKs, enciphered under a KEK currently installed in the device.

As we have discussed, we use an ATM as an example to demonstrate the key-loading process. A new ATM has none of the bank's keys installed when it is delivered from the manufacturer. The process of securely loading the first key is difficult. This has typically been done by loading the first KEK into each ATM manually, in multiple cleartext key parts. In this process, two separate people must carry key part values to the ATM and load them manually. Once inside the ATM, these parts are combined to form the actual KEK. In this manner, neither of the two people has the entire key, protecting the key value from disclosure or misuse. This method is labor-intensive and error-prone, making it expensive for the banks.

New techniques have been developed to define acceptable methods for loading these keys using public key cryptographic techniques, which allow the banks to load the initial KEKs without sending a person to the ATMs. These new methods make the process quicker, more reliable, and much less expensive for the banks. The new cryptographic features added to the IBM CEX2C provide flexible and novel methods for the creation and use of the special key forms that are needed for remote key distribution of this type. In addition, they provide ways to surmount longstanding barriers to secure key exchange with non-IBM cryptographic systems.

Once the ATM is in operation, the bank can install new keys as needed by sending them enciphered under a KEK that it installed at an earlier time. This is conceptually straightforward, but the cryptographic architecture in the ATMs is often different from that of the host system sending the keys, and it is difficult to export the keys in a form understood by the ATM. For example, cryptographic architectures often enforce key usage restrictions, in which a key is associated with data that describes limitations on how the key can be used for encrypting data, for encrypting keys, for operating on message authentication codes (MACs), etc. The encoding of these restrictions, and the method used to bind them to the key itself, differ among cryptographic architectures, and it is often necessary to translate the format to that understood by the target device before a key can be transmitted. It is difficult to do this without reducing security in the system, for example, by making it possible to arbitrarily change key usage restrictions. The methods described here provide a mechanism through which the system owner can securely control these translations, preventing the majority of attacks that could be mounted by modifying usage restrictions.

The primary vehicle supporting these new methods is the *trusted block*, which is described in the following sections.

Trusted blocks

The trusted block is the central data structure that supports all remote key loading functions. It provides great power and flexibility, but this means that it must be designed and used with care in order to have a secure system. This security is provided through several features of the design.

- Dual control is used to create a trusted block. In other words, two separate people must cooperate in order to create a usable block.
- The trusted block includes cryptographic protection that prevents any modification after it is created.
- A number of fields in the trusted block rules offer the ability to limit how the block is used, reducing the risk of using it in unintended ways or with unintended keys.

These features are covered in more detail in later sections.

The trusted block is the enabler that requires secure approval for its creation, then enables the export or generation of DES and TDES keys in a wide variety of forms as approved by the administrators who created the trusted block. For added security, the trusted blocks themselves may be created on a separate system, such as an IBM System x platform with a 4764 cryptographic coprocessor card, where that system is locked in a secure room. Trusted blocks can subsequently be imported to the System z server, where they are used to support applications.

In CCA, API functions are called *verbs* or *services*. Two new CCA verbs have been developed to manage and use trusted blocks. The verb *Trusted_Block_Create* (TBC) creates a trusted block, and the verb *Remote_Key_Export* (RKX) uses a trusted block to generate or export DES keys according to the parameters in the trusted block.

Overview of trusted block elements

The trusted block consists of several parts, some of which are required while others are optional. **Figure 7** shows an overview of the contents of a trusted block, with some details omitted for brevity. The following is a description of some of the elements that are depicted in the figure.

Public key

This contains an RSA public key and its attributes. For distribution of keys to a remote ATM, this is the root certification key for the ATM vendor, and it is used to verify the signature on public-key certificates for specific individual ATMs. In this example, the trusted block also contains rules that are used to generate or export symmetric keys for the ATMs. The trusted block may also be used simply as a trusted public key container, and in this case the public key in the block is used in general-purpose cryptographic functions such as digital signature verification.

The public key attributes contain information on key usage restrictions. This is used to securely control which operations are permitted to use the public key. If desired, the public key can be restricted to use for only digital signature operations, or for only key-management operations.

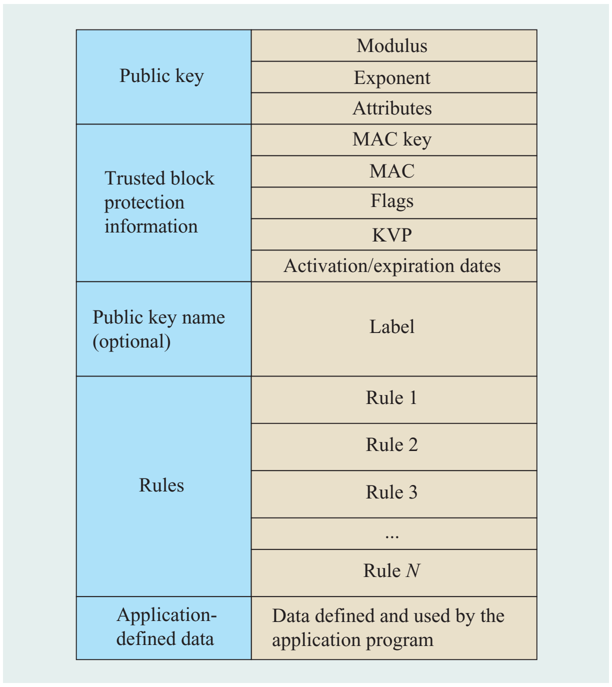
Trusted block protection information

This section contains information that is used to protect the trusted block contents from modification. A cipher block chaining (CBC)-mode MAC is calculated over the trusted block using a randomly generated TDES key, according to the method provided by the ISO 16609 standard. (Cipher block chaining is a method that uses an encryption algorithm to securely encrypt data that is longer than the size of the block that the algorithm fundamentally encrypts.) The MAC key itself is encrypted and embedded in the block. For the internal form of the block, the MAC key is encrypted with a variant of the CEX2C PKA master key. For the external form, the MAC key is encrypted with a fixed variant of a key-encrypting key. The MKVP field contains the MKVP for the PKA master key that was used, and the field is filled with binary zeros if the trusted block is in external format.

Note that the trusted block can optionally contain an expiration date and an activation date. The activation date is the first day on which the block can be used, and the expiration date is the last day when the block can be used. If these dates are present, the *date-checking* flag in the trusted block indicates whether the coprocessor should check the dates using its internal clock. In the case of the CEX2C in a System z server, the clock is not set; thus, date checking must be performed by an application program before using the trusted block. This feature is still valuable, because storing the dates in the block itself securely maintains the association between the trusted block and its activation and expiration dates.

The flags field contains the following boolean flags:

- The *active* flag indicates whether the trusted block is active and ready for use. This is the basis for enforcing dual control over creation of the block. One person creates the block, but in an inactive state. Subsequently, a second person must approve the block, which causes the active flag to be turned on.



Public key	Modulus
	Exponent
	Attributes
Trusted block protection information	MAC key
	MAC
	Flags
	KVP
	Activation/expiration dates
Public key name (optional)	Label
Rules	Rule 1
	Rule 2
	Rule 3
	...
	Rule *N*
Application-defined data	Data defined and used by the application program

Figure 7

Trusted block overall layout.

- The *date-checking* flag indicates whether the CEX2C should check the expiration and activation dates for the trusted block. If this flag is off, date checking must be performed outside the device if it is required. This is done so that the trusted block can be used with devices that have their own internal clocks, as well as with devices that do not. As mentioned in the case of the CEX2C, the date-checking flag is turned off, because the CEX2C internal clock is not set when it is used in a System z.

Trusted block name (public key name)

This field optionally contains a text string that is a key label name for the trusted block. It is included in the block for use by an external system, such as a host computer, and not by the card itself. In the System z environment, the label can be checked by RACF to determine whether use of the block is authorized. It is possible to disable use of trusted blocks that have been compromised or must be removed from use for other reasons. One approach is to publish a revocation list containing the key names for the blocks that must not be used. Code in the host system can check each trusted block before it is used in the cryptographic coprocessor in order to ensure that the name from that block is not in the revocation list. Other methods can also be used.

Table 2 Contents of a rule.

Field name	*Required?*	*Description*
Rule ID	Yes	An eight-character name for the rule.
Operation	Yes	Indicator of whether rule is to generate a new key or export an existing key.
Generated key length	Yes	For key generation, indicates the length of the key.
Key-check algorithm ID	Yes	Algorithm to be used to compute the optional key-check value (KCV). Options are • No KCV to be computed. • Encrypt zeros with the key. • Compute MDC2 hash of the key.
Symmetric-encrypted output format	Yes	Format of the symmetric-encrypted key output. Options are • CCA format key token. • RKX token (see section on the RKX key token).
Asymmetric-encrypted output format	Yes	Format of the optional asymmetric-encrypted key output. (Key is encrypted with RSA.) Options are • No asymmetric-encrypted key. • Encrypt in PKCS1.2 format. • Encrypt in RSAOAEP format.
Transport key variant	No	A variant to be applied to the transport key before it is used to encrypt the key being generated or exported.
Transport key control vector	No	A CCA control vector to be applied to the transport key before it is used to encrypt the key being generated or exported. In CCA, this method is used to define permitted uses for the key.
Transport key rule reference	No	Rule ID for the rule that must have been used to generate the transport key, if that key is an RKX token.
Export key length limits	Yes	When a key is being exported, this indicates the minimum and maximum lengths of the key that can be exported with this rule.
Output key variant	No	A variant to be applied to the generated or exported key before it is encrypted.
Export key rule reference	No	Rule ID for the rule that must have been used to generate the key being exported, if that key is an RKX token.
Export key control vector restrictions	No	Masks and templates that can be used to restrict the possible control-vector values that a key can have when being exported with RKX. Applies only if the key is a CCA key token. This can be used to control the types of CCA keys that can be processed using the rule.
Export key label template	No	The application program can identify the key to be exported using a key label, which is a name for the key. The rule can optionally contain a key label template, which is matched against the host-supplied key label, using wild cards so that the template can match a set of related key labels. The operation is accepted only if the supplied label matches the wild-card template in the rule.

Rules

A variable number of *rules* can be included in the block. Each rule contains information on how to generate or export a symmetric key, including values for variants to be used in order to provide keys in the formats expected by systems with differing cryptographic architectures. Use of the rules is described in sections of this paper that cover key generation and export using the RKX function. **Table 2** summarizes the mandatory and optional values that are part of each rule.

Application-defined data

The trusted block can hold data that is defined and understood only by the host application program. This data is included in the protected contents of the trusted block, but it is not used or examined in any way by the coprocessor. By including its own data in the trusted block, an application can guarantee that the data is not changed in any way, because it is protected in the same way as other trusted block contents.

Changes to the CCA API

We have made the following changes to the CCA API in order to support remote key loading using trusted blocks:

- A new API *Trusted_Block_Create (TBC)* has been developed to securely create trusted blocks under dual control.
- A new API *Remote_Key_Export (RKX)* has been created to generate or export DES and TDES keys under control of the rules contained in a trusted block.
- The *Digital_Signature_Verify (DSV)* API has been enhanced. It can now verify digital signatures using the RSA public key contained in a trusted block, in addition to ordinary CCA RSA keys.
- The *PKA_Key_Import (PKI)* API has been enhanced. This API is used to import an RSA key into the CCA domain. The enhancement also allows the API to import a trusted block that is in external form, producing an internal-format trusted block ready to be used in the local system.
- The *PKA_Key_Token_Change (KTC)* API has been enhanced. When the master key is changed, this API is used to re-encipher the internal RSA keys that were encrypted under the old master key, producing keys that are instead encrypted under the new master key. The API has been enhanced so that it can also update trusted blocks, which is accomplished by re-enciphering the MAC key embedded in the block.

RKX key token

CCA normally uses key tokens that are designed solely for the purposes of protecting the key value and carrying metadata associated with the key to control its use by CCA cryptographic functions. The remote key-loading design introduces a new type of key token called an *RKX token*. The purpose of this token is somewhat different, and its use is connected directly with the RKX verb added to the CCA command set for remote key loading.

The RKX token uses a special structure that binds the token to a specific trusted block, allowing sequences of RKX calls to be bound together as if they were an atomic operation. This allows a series of related key-management operations to be performed using the RKX verb. These capabilities are made possible by incorporating three features into the RKX key token structure:

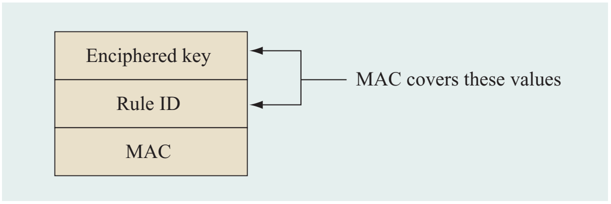


Figure 8

Simplified conceptual view of the RKX token structure. The arrows indicate that a MAC is computed over the encrypted key and the rule ID. The MAC, which ensures that a block of data has not been modified, provides integrity for the values of the enciphered key and the rule ID.

- The key is enciphered using a variant of the MAC key that is in the trusted block. A fixed, randomly derived variant is applied to the key before it is used. As a result, the enciphered key is protected against disclosure because the trusted block MAC key is itself protected at all times.
- The key token structure includes a rule ID for the trusted block rule that was used to create the key. A subsequent call to the RKX function can use this key with a trusted block rule that references this rule ID, effectively chaining use of the two rules together in a secure fashion.
- A MAC is computed over the encrypted key and the rule ID, using the same MAC key that is used to protect the trusted block itself. This MAC guarantees that the key and the rule ID cannot be modified without detection, providing integrity and binding the rule ID to the key itself. In addition, the MAC verifies only if the RKX token is used with the same trusted block that created the token, thus binding the key to that specific trusted block.

Figure 8 shows a simplified conceptual view of the RKX token structure.

Using trusted blocks

The following examples illustrate how trusted blocks are used with the new CCA functions.

Creating a trusted block

Figure 9 illustrates the steps used to create a trusted block. In particular, this shows the two-step process that

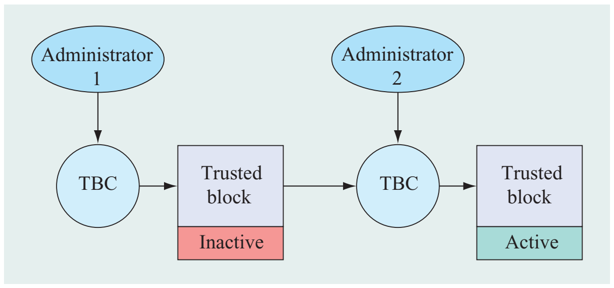


Figure 9

Trusted block creation and activation. This figure outlines the set of operations required to create a usable trusted block. The arrows represent input and output parameters for the TBC functions.

requires action by two different administrators in order to create the trusted block. Trusted blocks are structures that could be abused in order to circumvent security if an attacker could create them with undesirable settings. The requirement for two separate people makes it impossible for a single dishonest employee, for example, to create such a block. A trusted block cannot be used for any operations until it is in the active state.

The trusted block is always created in *external* form, which means that it is protected by a transport key (KEK). When the trusted block is complete, it is imported to the system where it will be used. The system importation is performed because the most secure way to create trusted blocks is on a separate computer which is located in an access-controlled area. Any number of trusted blocks can be created in order to meet different needs of application programs.

Exporting keys with RKX

Figure 10 shows the process required for using a trusted block in order to export a DES or TDES key. This high-level representation illustrates the main steps of the process. First, the RKX function is called with the following information:

- A trusted block that is in the active state. The block defines how the export operation is to be processed and includes values such as variants to apply to the keys.
- A transport key, which is a KEK or an RKX token that is used to encrypt the key being exported.
- The key to be exported (shown in Figure 10 as the export key). The export key can be in one of two forms. In particular, it can be either a CCA key token or an RKX token, as described in the section on the RKX key token.
- A key-encrypting key, indicated in Figure 10 as the export key KEK. This is used only if the export key is a CCA key token that is in external form, where it is encrypted under a KEK. In this case, the export key KEK is the key required by the coprocessor in order to decrypt the export key and obtain its cleartext value.
- A public-key certificate, which is optional. If it is included, it contains the certified public key for a specific ATM. The certificate is signed with the private key of the ATM vendor; the corresponding public key is contained in the trusted block so that this certificate can be validated. The public key contained in the certificate can be used to encrypt the exported key.

The processing steps are simple at a high level, but there are many options and significant complexity in the details.

- The trusted block itself is validated. This includes several types of validation:
  - Cryptographic validation, which uses the MAC that is embedded in the block, in which the MAC key is decrypted using the coprocessor master key. The MAC is then verified using that key. This verifies that the block has not been corrupted or tampered with, and it also verifies that the block is intended for use with this coprocessor because the MAC verifies successfully only if the master key is correct.
  - Consistency checking and field validation, in which the validity of the structure itself is checked, and all values are verified to be within defined ranges.
  - Checking of fields in the trusted block to see whether all requirements are met for use of this trusted block. For example: The trusted block must be in the active state. The current date must be between the activation date and the expiration date (inclusive), if those are present in the trusted block, and if the related flag indicates that date checking should be performed in the coprocessor card.
- Input parameters to the RKX function are validated against rules defined for them within the trusted block. For example, the rule can restrict the length of the key to be exported, or the rule can restrict the control vector (CV) values for the key to be exported, so that only certain key types can be exported with that rule.
- The export key is decrypted, and then the rules embedded in the trusted block are used to modify

that key to produce the desired output key value. For example, the trusted block can contain a variant to be exclusive-ORed with the export key before that key is encrypted. Variants are used in many non-IBM cryptographic systems to provide key separation so that a key cannot be used for the wrong purpose.

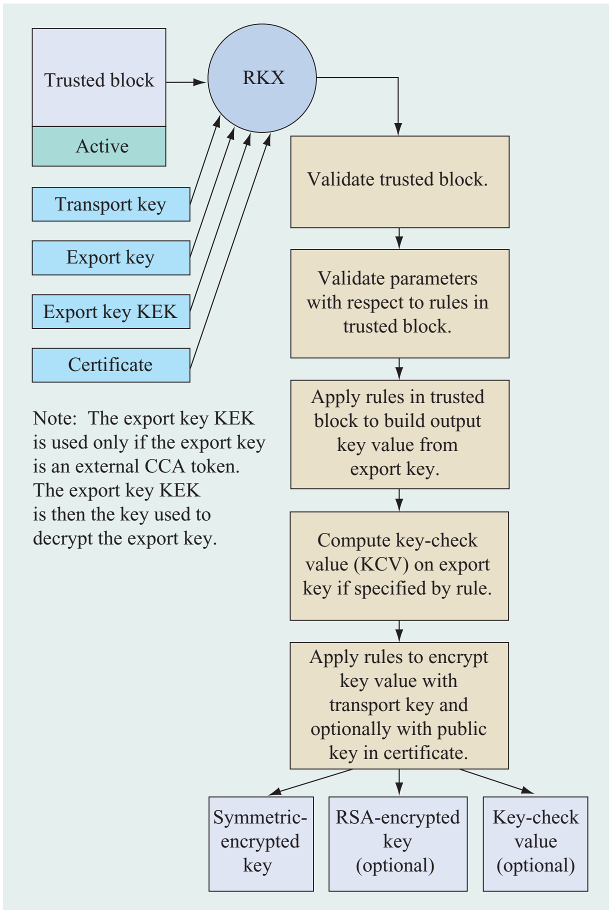
- A key check value (KCV) can be optionally computed for the export key. The trusted block permits one of two key-check algorithms to be used: encrypting binary zeros with the key, or computing an MDC2 hash of the key. The KCV is returned as an output of the RKX function. (MDC2 stands for a modification detection code cryptographic-hashing algorithm and is based on the DES encryption algorithm.)
- The export key, which may have been modified with a variant according to the rules in the trusted block, is enciphered with the transport key. The rules can specify that the key should be created in one of two formats, a CCA key token or the new RKX token described previously in this paper. With proper choice of rule options, the RKX token can be used to create keys that can be used in non-CCA systems. The key value can be extracted from the CCA token, resulting in a generic encrypted key, with variants and other options as defined in the rule.

Two optional fields in the trusted block may modify the transport key before it is used to encrypt the export key. In particular, the trusted block can contain a CCA control vector (CV) to be exclusive-ORed with the transport key before that key is used to encrypt the export key. This exclusive-OR process is the standard way CCA applies a CV to a key. Additionally, the trusted block can also contain a variant to be exclusive-ORed with the transport key prior to its use. If a variant and control vector are both present in the trusted block, the variant is applied first, and then the CV is applied.

The export key can optionally be encrypted with the RSA public key contained in the certificate parameter to RKX, in addition to encrypting it with the transport key as described above. These two encrypted versions of the export key are provided as separate outputs of the RKX function. The trusted block allows a choice of encrypting the key in either PKCS1.2 format or PKCSOAEP format. Both of these formats provide standard ways of formatting keys in order to encrypt them with a public-key algorithm such as RSA.

Generating keys with RKX

The process for using a trusted block in order to generate a new DES or TDES key is similar to the process described in Figure 10 for exporting keys. In particular, the RKX function is called with the following information:



Trusted block
Active
RKX
Transport key
Export key
Export key KEK
Certificate
Note: The export key KEK is used only if the export key is an external CCA token. The export key KEK is then the key used to decrypt the export key.
Validate trusted block.
Validate parameters with respect to rules in trusted block.
Apply rules in trusted block to build output key value from export key.
Compute key-check value (KCV) on export key if specified by rule.
Apply rules to encrypt key value with transport key and optionally with public key in certificate.
Symmetric-encrypted key
RSA-encrypted key (optional)
Key-check value (optional)

Figure 10

Exporting keys using a trusted block.

- A trusted block which is in the active state, as discussed for the case of exporting keys. The block defines how the key-generation operation is to be processed.
- A transport key, which is a CCA KEK or an RKX token that is used to encrypt the key being generated.
- A public-key certificate, which is optional. If it is included, it contains the certified public key for a specific ATM. The certificate is signed with the private key of the ATM vendor; the corresponding public key is contained in the trusted block so that this certificate can be validated. The public key contained in the certificate can be used to encrypt the generated key.

Most of the processing steps are the same as those described previously for key export. In particular,

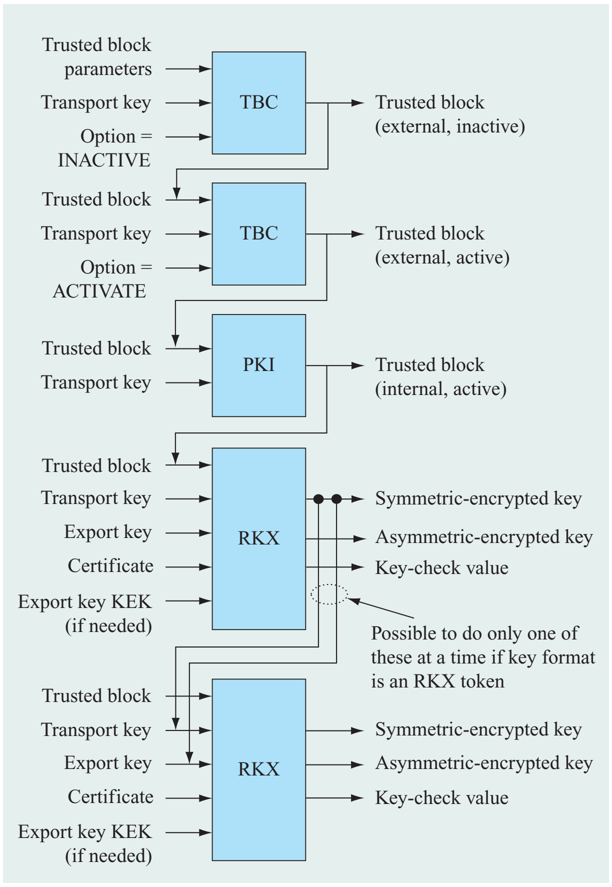


Figure 11

Typical flow of verbs for remote key export. (TBC: trusted block create; PKI: PKA key import; RKX: remote key export.)

- Validation of the trusted block and input parameters is performed as previously described for export.
- The DES or TDES key to be returned by the RKX function is randomly generated. The trusted block indicates the length for the generated key.
- The output key value is optionally modified by a variant as described above for export, and then encrypted in the same way as for export, using the transport key and optionally the public key in the certificate parameter.
- The key-check value (KCV) is optionally computed for the generated key using the same method as for an exported key.

Overall CCA function flow

The new and modified CCA functions are used together to create trusted blocks, and then to generate or export keys under the control of those trusted blocks. **Figure 11** summarizes the flow of the CCA functions to show how they are used. In the upper three blocks in the figure, a trusted block is created, approved, and then imported to the system where it will be used. In the remaining blocks, this trusted block is used to control the generation or export of DES or Triple-DES keys to be used on other systems or devices.

Usage example

The scenario described below shows how these CCA functions might be combined in a real-life application to distribute a key to an ATM and keep a copy for local use. Some of the terminology used reflects typical terms used in ATM networks. Our example illustrates a fairly complex real-world key distribution scenario, in which the following values are produced:

- A *terminal master key (TMK)*, which is the root KEK used by the ATM to exchange other keys. The TMK is produced in two forms: encrypted under the ATM public key, so that it can be sent to the ATM, and as an RKX token that is used in subsequent calls to the RKX verb to produce other keys.
- A *key-encrypting key (KEK1)*, which is encrypted under the TMK in a form that can be understood by the ATM.
- A *PIN-encrypting key (PINKEY)*, which is used by the ATM to encrypt customer-entered PINs, and by the System z host to verify those PINs. The PINKEY is produced in two forms: encrypted under KEK1 in a form that can be understood by the ATM, and as a CCA internal key token with the proper PIN key control vector, encrypted under the CCA DES master key and suitable for use with the System z CEX2C coprocessor.

Seven steps are required to produce these keys using the new verbs. These steps use a combination of five rules, which would be contained in a single trusted block. We refer to those rules as GEN1, GEN2, EXP1, EXP2, and EXP3, referring respectively to the terms *generate* and *export*.

1. Use RKX with rule GEN1 to generate a TMK for use with the ATM. The key is output in two forms:
   a. ePu(TMK): Output is encrypted under the ATM public key, supplied in the certificate parameter. The encrypted TMK is transmitted to the ATM.
   b. RKX(TMK): Output is an RKX token, suitable for subsequent input to the RKX function.
2. Use RKX with rule GEN2 to generate a key-encrypting key KEK1 as an RKX token, RKX(KEK1).

3. Use RKX with rule GEN2 to generate a PIN key PINKEY as an RKX token, RKX(PINKEY).
4. Use RKX with rule EXP1 to export KEK1 encrypted under the TMK as a CCA token, using a variant of zeros applied to the TMK. (The term *variant* is defined in the "Definitions" section of this paper.) This produces eTMK(KEK1). The encrypted KEK1 is transmitted to the ATM.
5. Use RKX with rule EXP2 to export PINKEY encrypted under KEK1 as a CCA token, using a variant of zeros applied to KEK1. This produces eKEK1(PINKEY). The encrypted PINKEY is transmitted to the ATM.
6. Use RKX with rule EXP3 to export PINKEY under KEK2, an existing CCA key-encrypting key on the local System z server. This produces eKEK2(PINKEY), with the CCA control vector for a PIN key.
7. Use the key import (KIM) function to import the PINKEY, produced in Step 6, into the local system as an operational key. This produces eMK(PINKEY), a copy of the key encrypted under the local DES master key and ready for use by CCA PIN API functions.

Conclusion

In designing the cryptographic configuration feature discussed in the first part of this paper, we have made a significant effort to ensure that the firmware stack did not adversely affect data throughput in the high-performance CEX2A data path. Owing to the efficient 4764 hardware interface design and care in design and implementation of the Hydra firmware, the impact of the Hydra firmware on the full-capacity throughput performance is essentially negligible.

The cryptographic configuration feature allows unique cryptographic hardware to be run in different cryptographic modes, thus reducing the number of different types of hardware required. Furthermore, the cryptographic configuration feature permits flexible or adaptable usage, depending on customer needs for the System z9 CEX2 feature.

Returning to the discussion of remote key distribution in the second part of this paper, we note that the changes we have introduced to CCA solve one new problem and one longstanding problem. The changes allow the secure distribution of initial keys to ATMs and other remote devices using public-key techniques, in a flexible way that can support a wide variety of different cryptographic architectures. The changes also make it far easier, and far more secure, to send keys to non-CCA systems when those keys are encrypted with a triple-DES key-encrypting key. These changes make it easier for customers to develop more secure systems, and they enable IBM CCA cryptographic systems to evolve and support the new methods being used for key transport in the banking and finance industry.

The new CCA services are complex, and IBM offers an alternative for customers who do not have the time or skills to develop application programs that call these functions directly. In particular, the IBM Distributed Key Management System (DKMS) has higher-level functions to generate the trusted blocks as well as an API to set up and perform the sequences of calls to the RKX service. This offers a much simpler interface for an application programmer. More information on DKMS is available on the IBM Security Solutions Web site [6].

*Trademark, service mark, or registered trademark of International Business Machines Corporation in the United States, other countries, or both.

**Trademark, service mark, or registered trademark of Linus Torvalds in the United States, other countries, or both.

References

1. T. W. Arnold and L. P. Van Doorn, "The IBM PCIXCC: A New Cryptographic Coprocessor for the IBM eServer*," *IBM J. Res. & Dev.* **48**, No. 3/4, 475–487 (2004).
2. A. O. Freier, P. Karlton, and P. C. Kocher, "The SSL Protocol, Version 3.0" (Internet Draft), Transport Layer Security Working Group, March 1996; see *http://wp.netscape.com/eng/ssl3/ssl-toc.html*.
3. NIST Security Computer Division,"IBM eServer Cryptographic Coprocessor Security Module Model 4764-001 Firmware (MiniBoot) Version 1.16 Security Policy"; see *http://csrc.nist.gov/cryptval/140-1/140sp/140sp524.pdf*.
4. D. J. Stigliani, Jr., T. E. Bubb, D. F. Casper, J. H. Chin, S. G. Glassen, J. M. Hoke, V. A. Minassian, J. H. Quick, and C. H. Whitehead, "IBM eServer z900 I/O Subsystem," *IBM J. Res. & Dev.* **46**, No. 4/5, 421–445 (2002).
5. R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Commun. ACM* **21**, No. 2, 120–126 (1978).
6. IBM Corporation, "Security Solutions: IBM Distributed Key Management System (DKMS)"; see *http://www.ibm.com/security/products/prod_dkms.shtml*.

Received March 22, 2006; accepted for publication April 26, 2006; Internet publication January 31, 2007

Todd W. Arnold *IBM Systems and Technology Group, 8501 IBM Drive, Charlotte, North Carolina 28262 (arnoldt@us.ibm.com).* Mr. Arnold is a Senior Technical Staff Member in the IBM Systems and Technology Group. He received a B.S. degree in electrical engineering from Case Western Reserve University in 1978. Joining IBM, he initially worked on optical character recognition systems for high-speed check-processing systems. Since 1985, he has worked in development of high-security cryptographic products, with a particular emphasis on the requirements of the banking and finance industry. He was responsible for the development of the first IBM smart-card product, which won the "Most Innovative Smart Card Product of 1989" award at the European Smart Card Application Technology (ESCAT) conference. Mr. Arnold received an IBM Outstanding Innovation Award for this work in 1989. He is an author or coauthor of eight patents and has contributed to several ANSI and ISO standards in the area of security. Mr. Arnold is a member of the Institute of Electrical and Electronics Engineers.

Anne Dames *IBM Systems and Technology Group, 8501 IBM Drive, Charlotte, North Carolina 28262 (annedames@us.ibm.com).* Ms. Dames is a Senior Software Engineer in the IBM Systems and Technology Group. She received a B.S. degree in mathematics from Johnson C. Smith University in 1984, joining IBM that same year. She initially worked in product engineering on dot matrix printer products including the IBM 4214, IBM 5224, IBM 5225, and IBM 3287. She later joined the IBM Proprinter development group, which produced printers for the personal computer market. In 1990, she joined the Application Solutions laboratory and worked on the first IBM home banking solution. Since 1993, Ms. Dames has worked in development of high-security cryptographic products, with a particular emphasis on development of the IBM Common Cryptographic Architecture host APIs and cryptographic firmware. In 2005, she received an M.S. degree in computer science, with a concentration in interdisciplinary networking with emphasis on information security, from the University of North Carolina at Charlotte.

Michael D. Hocker *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (hocker@us.ibm.com).* Mr. Hocker is an Advisory Engineer in the IBM Systems and Technology Group. He joined IBM in 1977 after receiving a B.S.E.E. degree with honors from Virginia Polytechnic Institute and State University. His first assignment involved firmware design for the 308x series Processor Controller Power/Thermal Interrupt Handler, followed by continued work with the large-systems processor controller for bipolar processors. After a period of time writing software for the OSA Support Facility, Mr. Hocker became increasingly involved with the System z I/O subsystem embedded firmware (device driver and ancillary components) that support the 4758 cryptographic adapter. Current work includes responsibility for the application-specific components of all versions of the System z Hydra I/O subsystem cryptographic embedded firmware, and responsibility for future development of those components. He is a co-author of 15 patents and several publications, and the recipient of an IBM Outstanding Technical Achievement Award for design, development, and implementation efforts for the 4764 System z I/O subsystem firmware.

Mark D. Marik *IBM Systems and Technology Group, 8501 IBM Drive, Charlotte, North Carolina 28262 (mdmarik@us.ibm.com).* Mr. Marik is a Senior Engineer in the IBM Systems and Technology Group. After receiving a B.S. degree in electrical engineering from the University of Kentucky in 1981, he joined IBM, working initially in testing and development of communication microcode for the IBM 4704-2 banking display. He subsequently focused on device firmware development in the IBM 4736 ATM, IBM 3624 ATM high-speed SDLC adapter, IBM 3892 Document Processor/Sorter, IBM 4772 Banking Statement Printer, and IBM 4779 Magnetic Card Reader Encoder Terminal. He also worked on several custom OEM projects. Since 1997, he has worked in development of high-security banking and financial cryptographic products, with an emphasis in embedded CP/Q and Linux device-driver development for communications and cryptographic ASICs, as well as embedded application performance enhancements using feedback-directed program restructuring. He is an author or co-author of six patents and five technical disclosure bulletins.

Nancy A. Pellicciotti *IBM Systems and Technology Group, 1701 North Street, Endicott, New York 13760 (napell@us.ibm.com).* Ms. Pellicciotti is an Advisory Software Engineer in the IBM Systems and Technology Group. She received a B.S. degree in mathematics and computer science from Clarkson University and joined IBM in Endicott, New York, in 1983. She started her career in the support processor area of Endicott Processor Development, contributing to the delivery of the 9370 and 9371 systems. She continued in System z hardware development, working on components of the Support Element (SE) and later on the Hardware Management Console (HMC) that supports System z servers that ranged from the 9221, 9672, 9673, and 9674 to the z900, z990, and the recently shipped z9* server. She began her work on the Crypto team in 2000 and is currently the Crypto focal point for SE and HMC functionality, most recently supporting the Crypto Express2 for the z9 server.

Klaus Werner *IBM Systems and Technology Group, IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (wernerk@de.ibm.com).* Mr. Werner is a Senior Development Engineer in the IBM Systems and Technology Group, working in processor firmware development. He studied communications engineering at the University of Cooperative Education of Stuttgart and received his Diplom-Ingenieur degree in 1984. He joined IBM that same year to work in the bring-up and test of CMOS processors. Mr. Werner has worked on projects in a variety of areas including RAS, system-feature development, and processor firmware development for the IBM 9221, CMOS G1 to G6 processors, z900, z990, and z9. In 1995 he joined the Crypto team, working on System z cryptographic support for G3 through System z9. Mr. Werner is currently the System z Firmware Focal Point for Crypto.