The IBM PCIXCC: A new cryptographic coprocessor for the IBM eServer

T. W. Arnold
L. P. Van Doorn

IBM has designed special cryptographic processors for its servers for more than 25 years. These began as very simple devices, but over time the requirements have become increasingly complex, and there has been a never-ending demand for increased speed. This paper describes the PCIXCC, the new coprocessor introduced in the IBM z990 server. In many ways, PCIXCC is a watershed design. For the first time, a single product satisfies all requirements across all IBM server platforms. It offers the performance demanded by today's Web servers, it supports the complex and specialized cryptographic functions needed in the banking and finance industry, and it uses packaging technology that leads the world in resistance to physical or electrical attacks against its secure processes and the secret data it holds. Furthermore, it is programmable and highly flexible, so that its function can be easily modified to meet new requirements as they appear. These features are possible because of innovative design in both the hardware and embedded software for the card. This paper provides an overview of that design.

Introduction

IBM has a rich history in the field of cryptography and in developing products that implement cryptographic functions. In the 1970s, IBM developed the Lucifer algorithm [1], which was the basis for the Data Encryption Standard (DES), the most widely used cryptographic algorithm in the world for more than 20 years. Immediately after the National Institute of Standards and Technology (NIST), then the National Bureau of Standards, adopted DES as a standard, IBM began designing and delivering products that enabled customers to protect their data. High-speed-DES channel-attached encryption units such as the IBM 3845 and 3848 were designed for S/370* mainframe users, and banking-oriented products, such as the 3600 finance system, were designed to meet the specific needs of banks and other financial institutions.

As technology improved and customer use of encryption increased in sophistication, IBM continued to develop new, leading-edge cryptographic products. In the 1980s, the 3600 finance system was replaced with the 4700 system, which included improved cryptographic functions for the banking industry. In the mid-1980s, the IBM finance industry product organization began research into advanced security technologies, including smart cards and high-security, high-performance DES-based cryptographic coprocessors for use in personal computers. This research led to the development of the transaction security system (TSS) product family, which was introduced in 1989. TSS included a fully integrated set of security components that could be used individually or together to provide an unprecedented level of security. The components included several technologies that are still important today.

TSS saw the introduction of the IBM Common Cryptographic Architecture (CCA), a carefully architected set of cryptographic functions and application programming interfaces (APIs) that provide both general-

0018-8646/04/$5.00 © 2004 IBM

purpose functions and a broad set of functions designed specifically to secure financial transactions. CCA introduced strong key typing and related key-management functions which prevented many attacks that had been possible with earlier systems based only on the basic cryptographic algorithms themselves [2, 3].

The TSS product family included the following components:

- A highly secure cryptographic coprocessor card (the 4755).
- The personal security card (PSC), an innovative smart card implementing a subset of the CCA and able to integrate securely with the other TSS components to support user authentication, secure data storage, and portable access rights to determine a user's authority when using the other components.
- A high-security smart card reader (the 4754) which served as the reader for the PSC and could also operate as a cryptographic processor.
- A channel-attached cryptographic processor (the 4753), which used the 4755, 4754, and personal security card to provide cryptographic functions for mainframe systems.
- A signature dynamics biometric pen and processor, which identified users on the basis of the movement of the pen as they signed their names with normal ink on any paper document.

A fundamental part of the TSS hardware was secure, tamper-resistant packaging technology to ensure that sophisticated attackers would not be able to break into the cryptographic devices to extract keys and other secret data, insert Trojan horses, or otherwise modify the function of the devices. This technology and the research that led to it formed much of the basis for the NIST FIPS 140 Standard [4], which is used to rigorously test such security today.

Following the introduction of TSS and CCA, the technology was adopted for use on other IBM platforms. The 4755 cryptographic adapter card was the basis for a cryptographic feature on the AS/400* midrange system. The S/390* mainframe developers designed a high-speed CCA processor with an Integrated Cryptographic Feature (ICRF) [5] that provided similar capabilities in their system. This gave customers a compatible, highly secure cryptographic architecture across the entire range of servers, from personal computers to mainframes.

As the TSS product family aged, IBM developed the 4758 PCI cryptographic coprocessor, the first of a new generation of coprocessors. The 4758 again provided CCA functions in an industry-leading tamper-resistant package [6, 7]. It was the first product ever to reach the highest level of protection, NIST FIPS 140-1 Level 4. IBM supported the 4758 on all of its server families, from Intel**-based personal computers through the RS/6000*, AS/400, and S/390 systems.[1] Later, 4758 technology was refreshed with faster cryptographic hardware and other improvements, again supported on all of the IBM server platforms.

With the introduction of the CMOS-based IBM mainframe systems, the ICRF was replaced with a very fast single-chip cryptographic processor, the *Cryptographic Coprocessor Feature*, or CCF. This became a standard, no-cost component of all zSeries CMOS systems, giving zSeries customers the advantage of free, high-performance, secure cryptographic functions.

The IBM-developed cryptographic chip in the 4758 was also used to develop a cryptographic accelerator card, directed mainly toward secure-sockets-layer (SSL) [8] performance. The accelerator is supported on all server platforms except for xSeries*.

Lessons learned

Our experience with the earlier products taught us many lessons about what our customers want and what the products have to do. This section highlights some important issues that led to our formulation of the requirements and subsequent design of the new cryptographic coprocessor.

Flexibility

Cryptographic applications are constantly evolving, and no single set of cryptographic functions can meet the needs of all customers over the life of a product. Each year, there are new application areas and new standards that dictate cryptographic product features. In addition, many customers have requirements that are unique to their geography or to their application area. For example, many countries have local banking regulations that impose cryptographic requirements that are different from those of the rest of the world.

Our experience with these applications has taught us that *flexibility* is an essential feature for our cryptographic coprocessors. We must not design the product with a fixed set of functions; it must be easy to add new functions over time. Furthermore, for specific customers we must have the ability to add special functions to the coprocessor that would not be in the standard product function set.

The need for updates to the cryptographic functions over time implies a related requirement: The coprocessor software should be able to be updated in the field. However, since this is a highly secure device, that process is much more complex than it is for most products. The coprocessor requires special features for secure software update so that cryptographic functions can be changed

[1] In S/390 and later zSeries*, the 4758 is called the *PCI cryptographic coprocessor*, or *PCICC*.

with no exposure to Trojan horses or other security attacks.

Performance

Today's systems require high-performance cryptographic functions. Traditional applications, such as automatic teller machine personal information number (ATM PIN) processing have always required high-performance symmetric-key cryptography, but this is even more important today as customers aggregate applications onto a smaller number of higher-performance servers. Each server must have the ability to process more transactions per second than with earlier generations. Public-key cryptographic functions such as RSA (named after its inventors, R. Rivest, A. Shamir, and L. Adleman) [9], are critical to SSL-based transactions and to digital signature applications. Use of these functions has dramatically increased in recent years, with a corresponding rise in throughput requirements. These algorithms rely on complex mathematical operations that use very large numbers, and performing them in software on the main processors of the server can easily consume most of the CPU capacity, leaving little for the application itself. Thus rises the need to implement the cryptographic functions in a *coprocessor*, which can execute these functions independently while the server CPU is free to perform other functions.

Satisfying two goals: security and acceleration

There are two fundamentally different goals for a cryptographic coprocessor. One is to simply accelerate the complex cryptographic algorithms in order to increase system throughput beyond that possible when the algorithms are executed in server software. In this environment, there is no need for the coprocessor to provide any special level of security beyond what existed when the processing was done without the coprocessor.

The second environment has security as its primary goal, although high performance is also very important. Banking systems and certificate authorities are examples of this environment. It is essential that all cryptographic keys be securely protected from disclosure, modification, and misuse. It is equally important to guarantee that there can be no tampering with the operation of the security functions executed by the coprocessor. These requirements dictate a physically and logically secure processing environment that is protected against tampering, eavesdropping, and other attacks.

Multiplatform support

A given class of application is not necessarily tied to a single server platform. For example, ATM PIN processing may be done on a zSeries mainframe, a pSeries* AIX* server, or an iSeries* system. In fact, in some environments, the same application may be implemented on two or more platforms. We have learned that many customers place a high value on compatible cryptographic functions across all of their platforms, so that they can design and implement an application one time, then port it to other platforms where they want it to run.

For many applications, this means having an industry-standard cryptographic API, such as PKCS#11 [10], on each platform. For more specialized applications, such as those in banking and finance, however, a specialized API, such as the IBM CCA, may be required. (See the section on the Common Cryptographic Architecture for a description of CCA in the PCIXCC coprocessor.)

Export control

For a cryptographic coprocessor to be valuable, one must be able to obtain it for a server. This may seem obvious, but cryptography has a stormy history when it comes to export control. For many years, cryptographic coprocessors were classified by the United States government as military weapons, and export was highly restricted and tightly regulated. It was very difficult to develop such a product in the United States and sell it to customers in other countries. IBM learned that it is essential to design all cryptographic products with provisions for complying with export regulations.

In recent years, export controls on cryptographic products have been almost entirely removed, and the need for special export control features has been greatly reduced. Government rules can change at any time, however, and it is still important to design all products so that they can accommodate new regulations that may be introduced.

Security certifications

Cryptography and security are difficult areas, requiring specialized knowledge. Many customers realize that they do not have this depth of knowledge and they wisely choose to trust recognized experts to help them select products that are well designed and adequately meet the security needs of their application. The easiest way to do this is to choose a product that has been certified to an appropriate security standard by an independent expert organization. If the standard is well known and has been scrutinized by experts in the field, certification under that standard provides a high degree of assurance that the product will offer good security in the area covered by the standard.

One of the most important contemporary standards related to cryptographic coprocessors is NIST FIPS 140, a standard for evaluating the physical and logical security of a hardware- or software-based cryptographic module. The hardware and low-level firmware of the current generation of IBM cryptographic coprocessors have been certified at

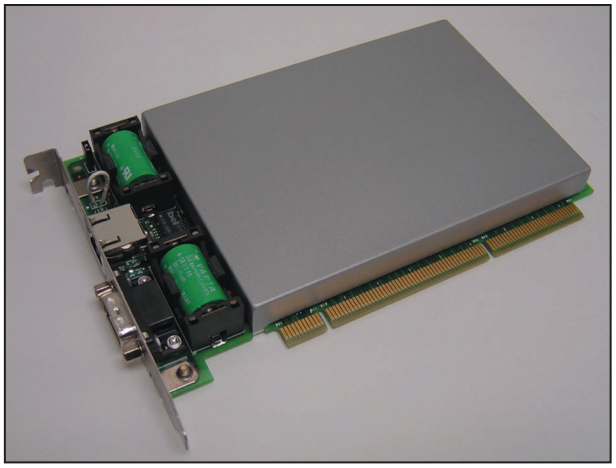

Figure 1

PCIXCC card.

the highest level of FIPS 140-1, and the new coprocessor has been designed to conform to the newer version, FIPS 140-2.

The PCIX cryptographic coprocessor

The following sections describe the new PCIX cryptographic coprocessor (PCIXCC) (**Figure 1**), which IBM designed to replace its current cryptographic coprocessors. The PCIXCC builds on the technology used in those earlier products and on the lessons described above that we learned with those products.

Goals of the PCIX cryptographic coprocessor design

The design goals for the PCIXCC were based on known customer requirements, experience with earlier products, and projections of future customer needs. These goals are summarized below.

1. The coprocessor should have the capability to replace all cryptographic features currently available on all of the IBM server platforms. It must provide the features of both a high-speed cryptographic accelerator and a high-security coprocessor, which were previously available only in separate products. While IBM may not deploy the PCIXCC on all of its servers, the goal of the coprocessor design was to support that possibility and to have the necessary features. The products that the PCIXCC could replace include the following:
   - *4758 PCI cryptographic coprocessor.* The 4758 is available today in forms for all IBM server platforms: on the zSeries as the PCICC feature, on the iSeries as feature 4801 or 4802, on the pSeries as feature 4963, and on the xSeries as machine type 4758. It provides highly secure cryptography supporting the CCA API on all platforms and PKCS#11 on xSeries and pSeries systems.
   - *zSeries Cryptographic Coprocessor Feature (CCF).* The CCF is a single-chip cryptographic coprocessor embedded in all IBM CMOS mainframe systems.
   - *IBM cryptographic accelerator card.* This PCI card provides acceleration for cryptographic operations, especially for the RSA algorithm that is critical to SSL performance. It is supported on zSeries, iSeries, and pSeries servers.
2. The coprocessor must be highly flexible and programmable so that features can be added throughout the product lifetime as standards and customer requirements evolve. A programming toolkit should provide the ability for special functions to be added for specific customers, with those added functions executing with the same security as the standard functions supplied by IBM. There must be a security architecture that ensures that the card is trusted and has not been modified, and that any software updates are trusted.
3. Both the physical packaging and the low-level software must be designed to comply with FIPS 140-2 at Level 4, the highest security level.
4. The coprocessor must be in the form of a PCI-X card, providing a high-speed connection in any of the IBM server systems.
5. The card must provide high-performance implementations of algorithms that include DES and Triple-DES (TDES) encryption [11], DES and TDES Message Authentication Codes (MACs) [12, 13], NIST Advanced Encryption Standard (AES) [14], RSA and other public-key algorithms, and the SHA-1 [15] and MD5 [16] hashing algorithms. In addition, it must provide a cryptographically strong hardware-based random number generator.
6. High-performance host-to-card communications is difficult to achieve and can significantly degrade overall coprocessor performance when the majority of requests include only a small amount of data. This is because the communications overhead required for each transaction to the card can easily be larger than the time it takes the card to perform the requested operation. It is important that the PCIXCC design include innovative high-speed paths to alleviate this problem and improve overall coprocessor throughput.
7. As a secure programmable coprocessor, internal software is used to orchestrate the cryptographic operations and to handle parameter checking, data formatting, and part of the communications control. It is essential that the software running inside the card

execute as quickly as possible. The secure packaging limits total power dissipation, so the embedded microprocessor and related components must be selected carefully to maximize performance while minimizing power use.

8. The implementation should reuse as much as possible from earlier products, while replacing those parts that must be changed to meet the new goals, such as higher performance.
9. The programming API at the host interface should maintain compatibility with previous coprocessors so that host application programs do not have to be modified.
10. The coprocessor should have high reliability and built-in error checking for both hardware and software, consistent with the reliability of other components of IBM servers.

Overview of the design

This section provides a brief overview of the PCIXCC card hardware and software. Subsequent sections treat each area in greater detail.

Hardware overview

The PCIXCC hardware is implemented in the form of a PCI-X adapter card, with a secure module containing all security-related components. The module is designed to meet the stringent security requirements of the FIPS 140-2 standard at Level 4. The internal components include a processor subsystem consisting of an IBM PowerPC* 405GPr microprocessor operating at 266 MHz, with 64 MB of dynamic random-access memory (DRAM), 16 MB of flash-erasable programmable read-only memory (flash EPROM) for persistent data storage, and 128 KB of static complementary metal oxide semiconductor (CMOS) RAM backed up with battery power when the card is powered off. The microprocessor serves as the primary controller of card operations. It orchestrates operation of the special-purpose hardware in the card and implements communications with the host and the cryptographic API functions that comprise the external interface from host application programs to the card.

Other critical hardware components include an IBM-developed custom cryptographic chip which internally we call *Otello*, a hardware-based cryptographic-quality random-number source, and a field-programmable gate array (FPGA) code-named *Rigoletto* which implements all of the interface and "glue" logic, including a sophisticated hardware assist to support high-performance communications to the card and between the PowerPC processor and cryptographic engines inside the card. The secure packaging technology is designed to detect or prevent all known physical attacks that might allow determined adversaries to extract secret data or tamper with the execution of critical operations within the card.

The hardware is designed with extensive integrated error checking to ensure that no hardware failures result in undetected errors in customer data. Otello includes hardware redundancy in all cryptographic algorithms so that any errors will be detected and will prevent erroneous results. Both Otello and Rigoletto feature error checking on all data paths and other logic. The 64-MB DRAM uses error-correcting logic to automatically correct any single-bit errors and to stop on multibit errors. Software includes additional error checking to verify the health of the PowerPC processor itself and to perform additional memory checking.

Software overview

The software is implemented as a layered design with a bootstrap loader at the lowest level and an application program at the highest level in the hierarchy. The card uses an embedded Linux** operating system that provides a subset of the features normally found in desktop or server Linux systems. All software except the bootstrap loader can be securely upgraded in the field, and each layer can be replaced individually. The application program at the top of the hierarchy implements the IBM CCA cryptographic API, which provides the functions accessible to application programs and administrative software running in the host system.

Details of the hardware design

This section provides information about the hardware design of the PCIXCC card and the special-purpose logic that gives it some of its unique characteristics.

Card overall design

The PCIXCC is a 3.3-V PCI-X bus adapter card with all security-related components encased in a tamper-responding security module (Figure 1). **Figure 2** is a block diagram of the card that includes the components in the secure module and those attached to the mother card.

Processor subsystem

The control processor inside the PCIXCC secure module is an IBM PowerPC 405GPr microprocessor running at a clock speed of 266 MHz. Integrated peripheral devices on the processor chip include both Ethernet and serial port interfaces, which are attached to standard connectors at the back edge of the card. Memory includes 64 MB of error-correcting (ECC) DRAM, 16 MB of flash EPROM, and 128 KB of static RAM powered by a battery when the card is powered off. In addition, a real-time clock module maintains the date and time for use by the PowerPC microprocessor. An internal PCI bus is used to

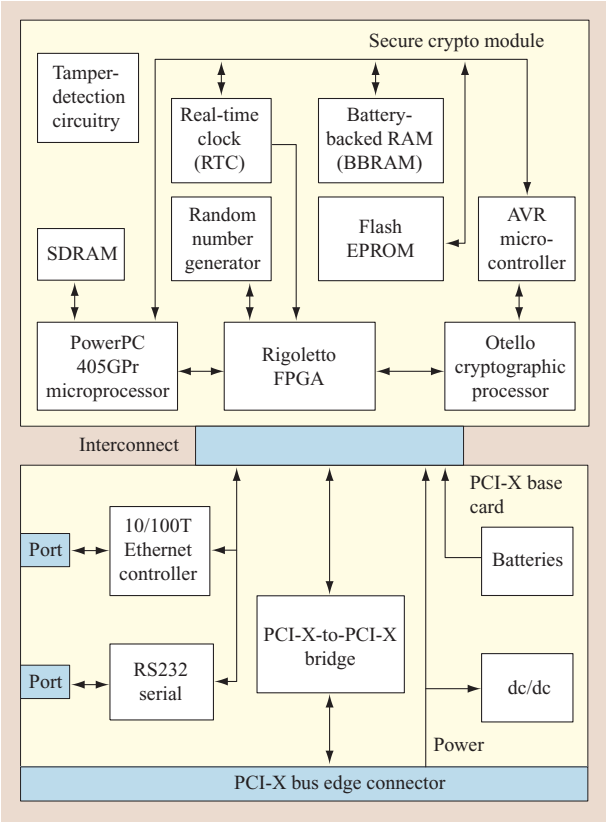

Figure 2

PCIXCC hardware.

interconnect the processor and other hardware devices on the card.

Tamper detection

The secure module on the PCIXCC card is designed with industry-leading tamper-detection features. The security-related electronic components are wrapped in a flexible mesh with narrow, embedded, overlapping conductive lines that prevent any 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. This is done by zeroizing the battery-backed static RAM; all sensitive data is stored either directly in the static RAM or in flash memory and encrypted under a 168-bit TDES key that is itself stored in that static RAM. If that key is destroyed, all encrypted data in the flash memory is rendered unusable. Other special circuits sense attacks that can cause imprinting in the static RAM. *Imprinting* is a process that can permanently burn data into the RAM, so that the same data appears each time the RAM chip is powered on. Different data can be written to the chip while it is operating, but the next time it is powered on, the originally imprinted data appears again as the initial memory content. Imprinting can be caused by exposing the memory to either very low temperatures or X-rays, and the tamper circuitry detects either of these and zeroizes the memory before imprinting can occur. Finally, there are attacks that are driven by manipulating the power-supply voltages to the card, and these conditions are also detected to prevent the attacks from succeeding.

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. The trust model demands that higher layers in the software hierarchy must not be able to modify operation of the lower layers or tamper with security-related data owned by those lower layers. To accomplish this, the card uses a separate 8-bit security microcontroller that keeps track of the security state of the card and blocks access by higher layers to the memory they must not be allowed to access. This independent processor is required, since code of any trust level running in the PowerPC processor would otherwise be able to access any memory region regardless of the trust level associated with its contents.

Otello cryptographic engine

Otello is an IBM-designed custom chip that provides fast hardware implementation of the essential cryptographic algorithms used by the PCIXCC card. The Otello chip is divided into two cryptographic algorithm sections. The symmetric-key cryptography and hashing unit[2] (SKCH) provides the DES, TDES, and AES symmetric encryption algorithms and the SHA-1 and MD5 secure hashing algorithms. In addition to data encryption, the DES implementation includes both single-DES and TDES MAC support, conforming to ANSI X9.9 and ANSI X9.19. The public-key unit provides modular math functions that are used to provide algorithms such as RSA. In addition, Otello contains an add-on interface, an interface to the PowerPC microprocessor, an interface to communicate with the Atmel AVR** security microcontroller, and an interface to the hardware random-number source. **Table 1** shows performance on the available cryptographic algorithms.

Rigoletto field-programmable gate array

The Rigoletto FPGA contains the logic for all interfaces between the host server, the PowerPC microprocessor, and

[2] Symmetric-key cryptographic algorithms are those that use an identical key for encrypting data and for decrypting that same data. A cryptographic hash function generates a fixed-length fingerprint, called the *hash*, from a variable-length input block. It should be infeasible to learn anything about the input block from the hash, and it should be infeasible to find a second set of input data that produces the same hash value.

Table 1 Crypto-chip performance.

Function	*Performance* (MB/s)
DES (56-bit key)	200
TDES (168-bit key)	67
AES (128-bit key)	185
AES (192-bit key)	156
AES (256-bit key)	136
SHA-1	198
MD5	239
RSA (1024-bit CRT key)	3300 operations per second

the Otello cryptographic chip. Since both the host server and the PowerPC microprocessor interface directly with the FPGA in order to talk to each other or to request cryptographic services, the FPGA is the key component for all internal and external programming interfaces.

The Rigoletto FPGA provides two fundamentally different communication paths for host-to-card transactions—*normal path* and *fast path*.

Normal path

In this mode, host requests are transferred via scatter-gather direct memory access (DMA) directly into DRAM memory in the card. Multiple requests can be buffered in memory, and so are already in the card when resources are available to process them. Once a request is in the card, software in the PowerPC microprocessor determines what function has been requested and executes that function with a combination of PowerPC software and calls to the on-card hardware. This method provides good performance and can—with the flexibility of this software-based approach—implement any required functions.

Fast path

The fast path is a novel design that provides very high performance for public-key cryptographic functions.[3] It gives the host server a direct hardware path to the Otello public-key cryptographic engine so that data does not have to stop in PowerPC memory and no software is involved. Logic in the Rigoletto FPGA arbitrates requests to the Otello chip so that normal-path and fast-path operations can be intermixed with no interference. The fast-path design supports operations using cleartext RSA keys, or using wrapped RSA keys that are encrypted under a TDES fast-path master key securely stored inside the tamper-resistant module.

[3] The Otello cryptographic chip is designed to support symmetric and hashing operations in fast path, in addition to public key operations. The logic currently implemented in the Rigoletto FPGA, however, does not provide a path from the host to those Otello functions. This could be added in the future with different FPGA programming.

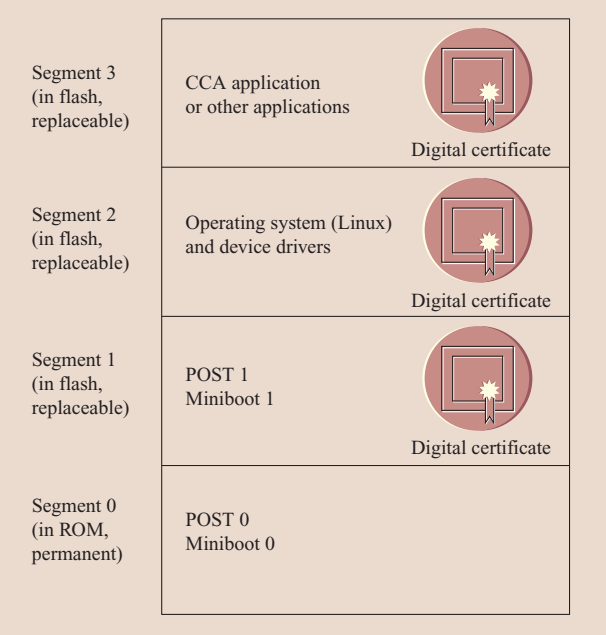

Figure 3

Software layers of the PCIXCC.

Rigoletto includes sophisticated logic and first-in first-out (FIFO) buffers to control and accelerate communications between the host and card and between the PowerPC microprocessor and the Otello cryptographic chip. The PowerPC microprocessor has very little to do in software. Special-purpose control logic manages almost all aspects of the scatter-gather DMA transfers. This results in communications throughput that is more than an order of magnitude greater than that of the earlier 4758 card.

Random-number generator

The card includes two cryptographic-quality hardware random-number generators. The entropy is obtained from electrical noise from a semiconductor junction. Each of the two random-number sources provides random bits at a rate of 128 Kb/s.

Details of the software design

The software that runs on the PowerPC 405GPr in the coprocessor is divided into four separate components (**Figure 3**):

- *Segment 0* contains POST (power-on self-test) 0 and Miniboot 0, stored in a region of flash EPROM that is unalterable once the card leaves the factory. POST 0 contains the small, low-level hardware self-test and setup. Miniboot 0 is the lowest-level software for control of loading software into segments 1, 2, and 3.

- *Segment 1* contains POST 1 and Miniboot 1. These are extensions to the POST and Miniboot in Segment 0, but have the important distinction that they can be securely reloaded after the card has been manufactured. Thus, Segment 0 holds the minimum required POST and Miniboot functions, while Segment 1 contains the majority. This is done to minimize the chances that a critical error will occur in code that cannot be updated in the field.
- *Segment 2* contains the operating system and device drivers. The PCIXCC card uses an open-source embedded Linux operating system. Special device drivers have been written to allow the operating system and application program to use the unique hardware inside the card.
- *Segment 3* contains the application program that runs on the PowerPC 405GPr to give the card the cryptographic API functions seen by host programs. In the current PCIXCC card, this application program implements the functions of the IBM CCA API. However, the card is designed to allow future developers to write other application programs to execute in the card. These could replace CCA or run concurrently with it, allowing the card to provide multiple sets of API functions to host application programs.

POST and Miniboot

The purpose of POST is to test and initialize all hardware in the coprocessor card, including the PowerPC 405GPr processor, the cryptographic engines, the communications interfaces, and all other logic. It must be written with the assumption that any hardware in the card may be faulty, and it must prevent use of the card if there are serious faults. It must report problems that are found, unless the problems themselves make reporting impossible.

The PCIXCC POST executes a sequence of tests, each identified by a *checkpoint*. The checkpoint is a numeric value that indicates which test is underway at any given time. If POST halts due to a failure, the checkpoint number is available in an interface register that can be read by the host computer system. This allows the host to identify the test that has failed.

The purpose of Miniboot is to control the secure loading of new software into Segments 1, 2, and 3. Since a large portion of Miniboot resides in Segment 1, this means that it has the ability to reload itself; this is done in a failsafe manner to ensure that an operational copy will always be present in the coprocessor card. The Miniboot code-loading architecture is a fundamental component of the PCIXCC FIPS 140 Level 4 design; it provides assurance that any software executing in the card has not been tampered with, and that it was created by IBM or someone approved by IBM to do so. Each segment has control over what software can be loaded into the next segment, and all segments are protected with digital signatures that can be verified back to a root key securely managed by IBM. In addition, Miniboot provides a facility called *outbound authentication* (OA), which can be used by applications inside or outside the card in order to verify at any time that the card is a genuine, untampered IBM cryptographic coprocessor. This is a valuable feature, because it makes it possible, even remotely, to verify that a coprocessor is not an insecure hardware or software replica of the PCIXCC and that the PCIXCC has not been modified with the addition of Trojan horses or other changes that would violate its security.

Linux operating system and device drivers

The PCIXCC cryptographic coprocessor was conceived as an improved version of its predecessor, the 4758, with the goal of minimizing software changes. The decision to replace the embedded processor, which is based on the Intel486** architecture, with an embedded PowerPC 405GPr had some major consequences, however, especially for the operating system.

The 4758 embedded operating system is CP/Q, a message-passing-based microkernel developed by IBM. Using CP/Q in the PCIXCC would have made it easier for developers to port the existing 4758 firmware, but we eventually decided against it for several reasons. First, CP/Q was a home-grown embedded operating system for which development had stopped many years ago. As the only remaining CP/Q user, it was up to the 4758 development team to maintain the O/S, which consumed development resources that could be better used elsewhere. Second, there was no version of CP/Q for the PowerPC 405 family, and the porting process would have been long and expensive. Finally, there were performance problems with the CP/Q microkernel-based architecture. IBM Research had already experimented with Linux on the 4758 and, partly on this basis, we decided to use Linux inside the PCIXCC. An additional motivation for this was that Linux has a large existing user base, including a very active embedded development community. For example, the Linux kernel comes with good flash memory support, and most significantly, it had an existing and actively maintained port for the embedded PowerPC 405. By adopting Linux as the card O/S, the PCIXCC development team could focus more of its energy on the unique PCIXCC components.

The length of time available to port existing 4758 CCA firmware to the new environment turned out to be relatively short. To make it as easy as possible, we tried to mimic as much of the Linux run-time environment available on ordinary workstations as made sense in a small embedded environment. This included shared library support, C and C++ run-time support, thread support, and software floating-point support.

Initially we considered using one of the available commercial embedded Linux distributions, but we quickly dismissed that option. These distributions tend to focus on much larger systems and come with a large number of base components, such as *sed* and *awk*, which are overkill for a small embedded system. Instead, we created our own package that consisted of a bootstrap loader, a standard Linux kernel, a small set of utility programs that initialize the system and drivers, and a minimal set of shared libraries to provide C and C++ run-time support.

We used open-source components as much as we could. The bootstrap loader is a slightly modified version of *ppcboot*, and the kernel is a slightly modified version of the standard Linux kernel. The minor modifications add support for PowerPC microprocessor differences and card-specific peripherals that are used during system bringup, such as the clock. All other device drivers for crypto and communication peripherals are loaded dynamically into the kernel once the kernel is up and running. The reason for using dynamically loaded device drivers is that they provide a strong separation between the open-source license used by the kernel and the license under which the device drivers are available. This separation was useful when we were still debating the appropriate licensing scheme under which to release the software. In the end, we decided to provide all device driver modules under the same license as the Linux kernel, the GNU General Public License (GPL).

CCA cryptographic application program

The following sections describe the CCA architecture and the CCA application program that runs in Segment 3 on the PCIXCC cryptographic coprocessor.

The Common Cryptographic Architecture

The Common Cryptographic Architecture (CCA) API is the programming interface used by host application programs to access the PCIXCC card to perform cryptographic or administrative functions. CCA provides both generic cryptographic functions and a rich set of functions specifically designed to support the unique needs of the banking and finance industries.

CCA is a hardware-based cryptographic architecture. It is hardware-based in the sense that it must be implemented in a secure environment, where there is protection against disclosure or modification of secret CCA data and protection against modification of the execution of the CCA functions themselves. CCA must operate as a black box in which the inputs, outputs, and operations are well-defined in an external API, but where the data and functions within the box are protected against outside entities. From this perspective, the PCIXCC platform is ideal because its design is based on FIPS 140-2 Level 4 requirements. The secure module on the card provides precisely the kind of protection needed to secure an API such as CCA.

One of the security principles of CCA is that no application keys can ever appear in cleartext form outside the secure module. It is possible for a customer to load initial keys in cleartext form, but once those keys have been imported into the module, it is impossible to reveal them in cleartext form any more. Once keys are in the CCA environment, the API is designed to guarantee that they will always be securely encrypted. The preferred way to generate keys is to let the CCA module generate them so that they leave the module already in encrypted form, but this is not always practical when keys must be interchanged with non-CCA systems. To accommodate both this and legacy key-entry paradigms, CCA also provides functions to load DES keys in multiple cleartext key parts. These functions also provide dual control to ensure that different people are responsible for each of the separate key parts.

In general, under CCA, all application keys are encrypted under a TDES master key held securely inside the protected module. The keys can then be stored outside the module without having to be concerned about their security; they cannot be attacked because the master key used to encrypt them is itself secure inside the tamper-protected module and will be zeroized if there is any attempted attack. This allows the customer to store a number of application keys, with the number limited only by storage on hard disk or other media. For convenience, all CCA host application software provides built-in key storage databases that are managed by CCA itself. The customer can refer to keys by an identifying string called a *key label*, and CCA will use the label to locate the actual encrypted key token and send that token to the PCIXCC card for use in the requested CCA function. Alternatively, the customer can use his own database or other software to store and manage the encrypted key tokens.

There is one exception to the application keys stored outside the secure module: CCA supports *retained RSA keys*, in which the RSA key pair is generated inside the secure module, and only the public key is ever allowed to leave the secure environment. The private key remains inside the module and is never allowed to leave in any form. This is designed to meet the strict demands of some standards, which require assurance that the private key can exist only in a single cryptographic module. By doing so, non-repudiation is greatly strengthened; if a private key can exist only in one cryptographic device, it provides assurance that any digital signature computed using that private key can have originated only at the system in which that device is installed. In the PCIXCC, retained RSA private keys are stored in the flash memory inside the secure module. Like all CCA data stored in that memory, they are securely encrypted under a TDES key

that is destroyed if there is any attempt to tamper with the device.

For DES keys, CCA enforces strong separation of key usage with *control vectors* (CVs)—bit strings that are cryptographically bound to the key itself so that they cannot be removed or modified without rendering the key unusable. The bits in the control vector specify the possible uses of the key in great detail. This prevents the many attacks that are otherwise possible by using a key for an inappropriate function. The control vectors serve a number of purposes, including the following:

- They enable or disable the use of the key for specific cryptographic functions, such as
  - Enciphering or deciphering data.
  - Generating a message authentication code (MAC).
  - Verifying a MAC.
  - Importing a key from another cryptographic system.
  - Exporting a key to another cryptographic system.
  - Processing financial PIN transactions.
  - Processing Europay**-MasterCard**-Visa** (EMV) smart-card-based transactions [17].
- They are used to define the length of the key and its position for multilength DES keys. For example, the control vector may define a value as one of the following:
  - A single-length (56-bit) DES key.
  - The left half of a double-length (112-bit) DES key.
  - The right half of a double-length (112-bit) DES key.
- They contain a number of fields to protect against specific cryptographic attacks.

CCA systems are designed to be highly resistant to insider attacks, even by system programmers or the design team that developed the hardware and software. This is facilitated by the consistent use of strong key separation and well-architected discrete cryptographic functions. Many complex operations are implemented in CCA as atomic functions to ensure that no unprotected intermediate results will ever appear outside the secure module. For banking and finance applications and others that require strong assurances of security, these features are of paramount importance. These CCA design principles can be contrasted with pure software implementations of cryptographic functions or with cryptographic accelerators that provide only the raw encryption algorithms. Such implementations are extremely valuable in applications such as SSL that have short-lived data and encryption keys, but they cannot provide the kind of security required for ATM PIN processing, digital signatures, or many other functions with strong security requirements defined by the application itself, by standards, or by legal documents.

Today's CCA implementation includes functions in the following broad categories:

- 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.
- MD5 and SHA-1 hashing.
- Processing Europay-MasterCard-Visa (EMV) smart-card-based transactions.
- An integrated role-based access control system to securely control which CCA functions are authorized for different users.

The above are implemented in the form of roughly 75 separate API functions, many of which have a wide variety of processing options.

The CCA implementation in the PCIXCC coprocessor

CCA is implemented in the PCIXCC in software that runs in Segment 3 (see Figure 3). It is implemented as an application program under the embedded Linux operating system. The CCA software calls special Linux device drivers in order to access the cryptographic hardware and the communications interface that permits it to interact with application programs running on the host computer system.

The preceding-generation IBM z900 servers have CCA implemented with a combination of two hardware coprocessors. Every machine has a standard single-chip CCA processor called the Cryptographic Coprocessor Feature (CCF). CCF provides a limited subset of CCA functions, but is extremely fast for symmetric cryptographic algorithms. In addition to the CCF, customers can purchase an optional feature called PCICC[4], which is the direct predecessor of the PCIXCC. The PCICC feature contains the 4758 PCI cryptographic coprocessor card with a special version of its CCA software tailored for the specific needs of the zSeries platform. While its performance on symmetric algorithms is much lower than the CCF, the PCICC has better RSA performance and a much broader set of CCA functions, and its programmability allows it to support new requirements and special custom features that arise over time. In addition to CCA, the z900 offers a cryptographic accelerator feature called the PCI Cryptographic

[4] The initialization PCICC stands for PCI cryptographic coprocessor, as opposed to the new PCIXCC, which is the PCI-X cryptographic coprocessor. PCI and PCI-X are the card interface bus technologies used on the two cards.

Accelerator (PCICA). The PCICA is strictly an accelerator to speed up cryptographic algorithms. It does not have any security to protect keys or data, and it does not support CCA.

With the PCIXCC, we have a single coprocessor card that can replace the CCF, the PCICC, and the PCICA features in the zSeries systems. One card contains all of the necessary cryptographic functions, performance, programmability, and tamper resistance for these systems. Similarly, the PCIXCC card has the capability to replace both the 4758 card (used in iSeries, pSeries, and xSeries servers) and the cryptographic accelerator card (used in iSeries and pSeries systems).

CCA is structured as a multithreaded application in which requests from the host are dispatched to one of several worker threads. This architecture allows the I/O (crypto and communication) to be overlapped by computations and provide a traditional blocking programming model. For example, some threads may be suspended waiting for RSA operations to complete, while others may be communicating with the host.

Toolkit and debugger

Like the PCICC and 4758, the new PCIXCC coprocessor is designed to support custom programming, a feature that makes it possible to add special features users might need or request. The CCA API can be extended with new functions using the user-defined extensions (UDX) toolkit. The toolkit provides the necessary tools to write and debug CCA extensions that execute inside the secure module on the card with the same industry-leading security as the standard CCA functions provided by IBM. It also provides tools to add new functions to the host API software so that they can be called by application programs.

Performance

Preliminary performance measurements show that the PCIXCC meets or exceeds all of its performance goals. Its performance compared with that of the predecessor PCICC card is excellent, and the card meets all of the goals for replacement of the PCICC, PCICA, and CCF in zSeries servers.

Table 2 shows performance for the PCIXCC in a z990 system compared with a PCICC in the z900. Each machine had a single coprocessor card, but note that multiple cards can be used, and performance scales approximately linearly with the number of cards. All tests were performed using the normal path mode of communications. No measurements are shown for the fast path mode.

The results shown in Table 2 were measured from real host server application programs running on top of the host operating system and the CCA API software. These operations all used fully secured keys, and complex operations such as PIN translate performed the entire function as an atomic operation inside the secure module. No cleartext keys or other sensitive cleartext information appear outside the secure boundary of the coprocessor.

Table 2 Performance comparison of the PCIXCC in a z990 system and PCICC in the z900 system.

Operation	*Performance* (operations per second)		*Ratio*
	PCICC (4758)	*PCIXCC*	
Generate 1024-bit RSA digital signature	104	1172	11.3
Generate 2048-bit RSA digital signature	42	458	10.9
Encipher 1024-byte blocks of data (single-DES)†	166	1183	7.1
Encipher 1024-byte blocks of data (TDES)	168	1145	6.8
Generate RSA key, 1024-bit CRT†† format	0.28	1.8	6.4
Generate wrapped DES key	125	1007	8.1
Export (wrap) DES key	190	1214	6.4
PIN translate	106	1075	10.1
PIN encrypt	147	1488	10.1
Derive diversified key	131	1048	8.0

† Note that the DES and Triple-DES numbers are measured on an IBM xSeries server running Linux, and not on the z990 and z900 systems. This is because no PCICC data is available for these functions from the zSeries systems.
†† Chinese remainder theorem.

Future possibilities

The PCIXCC is an extremely flexible, high-performance, highly reliable cryptographic coprocessor. Today, it is available only in IBM zSeries servers using the CCA API, but the design makes it easy to consider a variety of other systems and applications in the future.[5] Possibilities include the following areas:

- The coprocessor card could be supported on other IBM platforms, such as iSeries, pSeries, or xSeries.
- In addition to CCA, other APIs could be supported, such as PKCS#11. Multiple APIs could be supported in the card simultaneously.
- The card could be offered through original equipment manufacturer (OEM) channels so that other companies could sell it as their product. They could use the development toolkit to differentiate their product version from IBM versions of the card, but they would still have the benefit of the hardware design, reliability, and the embedded Linux operating system.

[5] None of these possibilities is in any way a commitment by IBM to develop or offer such a product or feature.

- The card was designed with an integrated Ethernet port so that it can be used as a *controlled network interface*. The combination of the Ethernet port, the cryptographic hardware, the programmability, and the tamper-responsive envelope make the card an attractive solution to implement secure network protocols in potentially hostile environments.

The fact that the card is fully programmable with complete security for loading and executing code opens almost limitless possibilities for its use.

Concluding remarks

In this paper we have described the new IBM PCI-X Cryptographic Coprocessor, a flexible, high-performance cryptographic processor for server systems. The PCIXCC is now available as a feature on IBM z990 servers.

We have described the overall design of the PCIXCC and contrasted it with its predecessors, particularly the 4758 card, also known as the PCICC in zSeries systems. The lessons we learned with the 4758 contributed significantly to the innovative design of the PCIXCC, where new approaches resulted in a typical performance improvement of 10× greater than the 4758. The new design provides a high degree of upward compatibility with predecessor products so that users benefit from the improved performance and reliability without having to change their application programs.

The flexibility and secure design of the PCIXCC make it an ideal candidate for servers requiring the highest possible level of security for their keys and cryptographic operations, while still obtaining very high levels of performance in real-world applications.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or service mark of Intel Corporation or its subsidiaries, Linus Torvalds, Atmel Corporation, Europay International, MasterCard International Incorporated, Visa International, or SET Secure Electronic Transaction LLC in the United States and/or other countries.

References

1. H. Feistel, "Cryptography and Computer Privacy," *Scientific American* **228,** No. 5, 15–23 (May 1973).
2. D. G. Abraham, G. M. Dolan, G. P. Double, and J. V. Stevens, "Transaction Security System," *IBM Syst. J.* **30,** No. 2, 206–229 (1991).
3. D. B. Johnson and G. M. Dolan, "Transaction Security System Extensions to the Common Cryptographic Architecture," *IBM Syst. J.* **30,** No. 2, 230–243 (1991).
4. *Security Requirements for Cryptographic Modules,* Federal Information Processing Standard 140-2, National Institute of Standards and Technology, Washington, DC, May 25, 2001.
5. R. M. Smith, Sr., and P. C. Yeh, "Integrated Cryptographic Facility of the Enterprise Systems Architecture/390: Design Considerations," *IBM J. Res. & Dev.* **36,** No. 4, 683–694 (July 1992).
6. IBM Corporation, *IBM PCI Cryptographic Coprocessor, CCA Basic Services Reference and Guide,* Release 2.41 for IBM 4758 Models 002 and 023; see *http://www-3.ibm.com/security/cryptocards/html/library.shtml*.
7. S. W. Smith and S. Weingart, "Building a High-Performance, Programmable Secure Coprocessor," *Computer Networks* (Special Issue on Computer Network Security) **31,** 831–860 (April 1999).
8. A. Freier, P. Karlton, and P. Kocher, "The SSL Protocol, Version 3.0," Transport Layer Security Working Group, November 1996; see *http://wp.netscape.com/eng/ssl3/draft302.txt/*.
9. 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 (February 1978).
10. *Cryptographic Token Interface Standard*, PKCS #11 v2.11, RSA Laboratories, November 2001; see *http://www.rsasecurity.com/rsalabs/pkcs/pkcs-11/index.html*.
11. *Data Encryption Standard (DES)*, Federal Information Processing Standard 46-3, National Institute of Standards and Technology, Washington, DC, October 1999.
12. *Financial Institution Message Authentication (Wholesale),* American National Standard X9.9, American Banker's Association, May 1996.
13. *Financial Institution Retail Message Authentication,* American National Standard X9.19, American Banker's Association, May 1996.
14. *Advanced Encryption Standard (AES),* Federal Information Processing Standard 197, National Institute of Standards and Technology, Washington, DC, November 2001; see *http://csrc.nist.gov/CryptoToolkit/aes/*.
15. *Secure Hash Standard*, Federal Information Processing Standard 180-1, National Institute of Standards and Technology, Washington, DC, April, 1995; see *http://www.itl.nist.gov/fipspubs/fip80-1.htm*.
16. R. Rivest, "The MD5 Message-Digest Algorithm," MIT Laboratory for Computer Science and RSA Data Security, Inc., April 1992; see *http://www.ietf.org/rfc/rfc1321.txt/*.
17. Europay, MasterCard, Visa information and specifications; see *http://www.emvco.com/*.

Received September 22, 2003; accepted for publication December 3, 2003; Internet publication April 6, 2004

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 at the IBM Laboratory in Charlotte, North Carolina. After receiving a B.S. degree in electrical engineering from Case Western Reserve University in 1978, he joined IBM, where 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 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. He received an IBM Outstanding Innovation Award for this work in that same year. He is an author or co-author 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.

Leendert P. Van Doorn *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (leendert@us.ibm.com).* Dr. Van Doorn is a Research Staff Member at the IBM Thomas J. Watson Research Center, where he runs the Secure Systems Department. He received his Ph.D. degree in computer science from the Vrije Universiteit in Amsterdam. In 1998 he joined IBM, where he has worked on physically secure coprocessors, wireless network security, and secure operating systems. More recently, he has been working on the integration of Trusted Computing Group technology into Linux and other operating systems and on a secure hypervisor. Before joining IBM, he worked on the capability-based distributed operating system *Amoeba* and on the extensible operating system *Paramecium*. Dr. Van Doorn is a member of IEEE, ACM, and Usenix.