S/390 Parallel Enterprise Server CMOS Cryptographic Coprocessor

by R. J. Easter
E. W. Chencinski
E. J. D'Avignon
S. R. Greenspan
W. A. Merz
C. D. Norberg

As the Internet becomes the basis for electronic commerce, and as more businesses automate their data processing operations, the potential for unauthorized disclosure of sensitive data increases. On-line databases are becoming increasingly large and complex. Sensitive data is transmitted on communication lines and often stored off-line. As a result, the efficient, economical protection of enterprise-critical information is becoming increasingly important in many diverse application environments. The protection required to conduct commerce on the Internet, provide data confidentiality, and provide user authentication can be achieved only by cryptographic services and techniques. The high-speed, physically secure IBM S/390® CMOS Cryptographic Coprocessor for S/390 Parallel Enterprise Servers™, together with the IBM Integrated Cryptographic Service Facility (ICSF), an IBM licensed program for the OS/390® operating system, provides the ability to encrypt and decrypt data, generate and manage cryptographic keys, perform PIN operations, and perform other cryptographic functions dealing with data integrity, digital signatures, and key exchange.

Introduction

The IBM S/390* CMOS Cryptographic Coprocessor for S/390 Parallel Enterprise Servers* [1], herein called the S/390 Cryptographic Module (SCM), is an extension to the S/390 central processor unit (CPU) and can be considered as an additional execution element. It is implemented on a single CMOS chip that connects both physically and logically to a CPU, and is supported by the IBM Integrated Cryptographic Service Facility (ICSF) [2]. Depending on the model, one or two SCMs are provided. **Figure 1** shows a S/390 system architecture with two SCMs.

The SCM hardware implements the S/390 cryptographic architecture, which is an extension of the S/390 architecture [3]. Available only in the ESA/390* mode, it is an upward-compatible extension to the Integrated Cryptographic Facility (ICRF) provided on the bipolar S/390 Enterprise Servers. All functions to support data privacy, message authentication, personal identification, and key management (along with many extensions) are provided as they were for ICRF. These functions and the associated instructions are referred to as the direct attached crypto (DAC) operations. For DAC operations, each SCM appears to be attached to only one CPU in the configuration and functions as an integral part of that CPU. That is, the DAC instructions are performed synchronously with the instruction processing of that CPU.

0018-8646/99/$5.00 © 1999 IBM

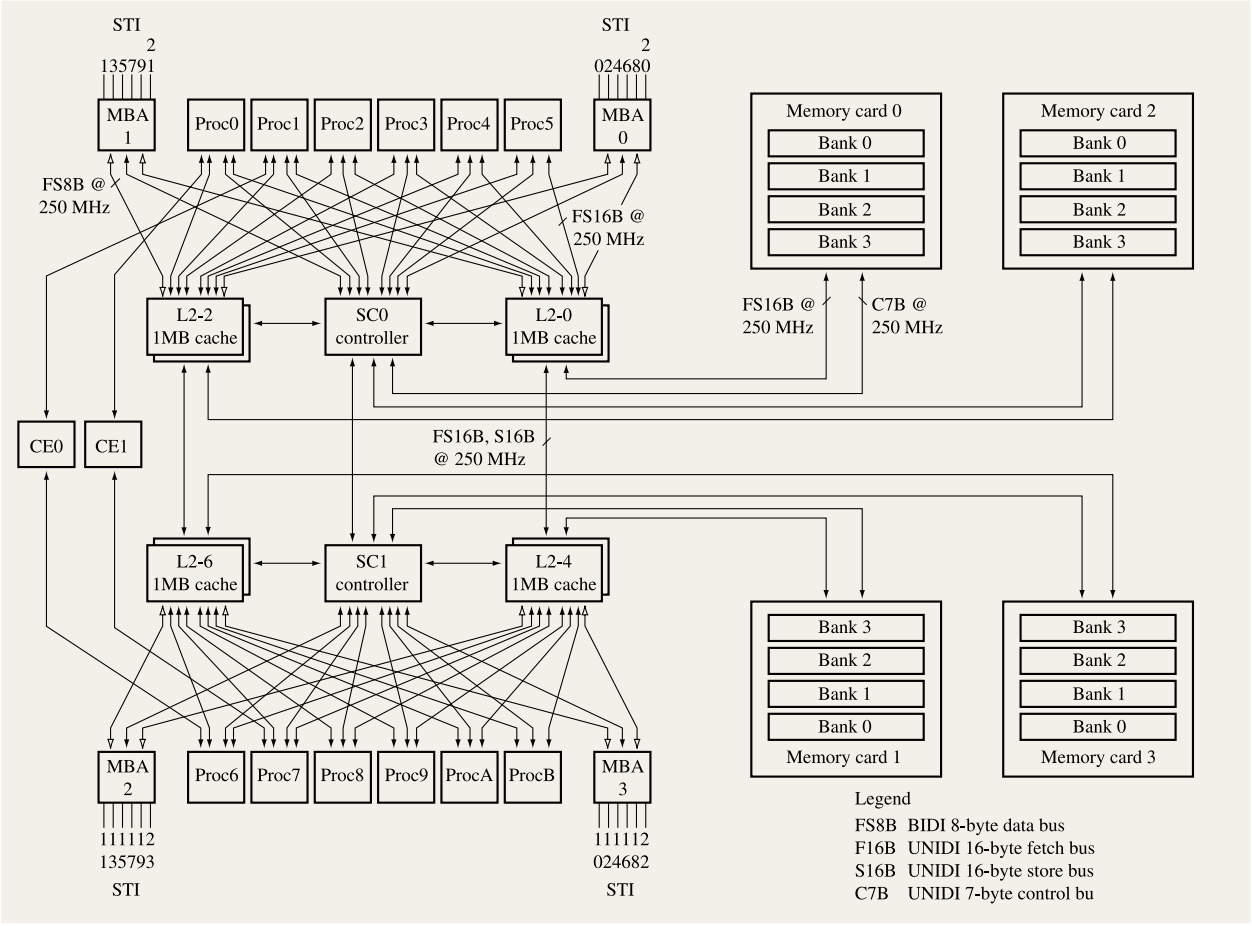


Figure 1

G5 system structure.

The SCM also includes a crypto asynchronous processor (CAP), which can perform operations concurrently with the DAC operations. These asynchronous operations include public key algorithm (PKA) functions, intended for use by application programs; and public key security control (PKSC) functions, which replace the physical security controls provided on ICRF. Communication with the crypto asynchronous processor is accomplished by means of a queued-message interface which provides communication among all CPUs and all crypto asynchronous processors in the configuration. This interface includes separate crypto asynchronous message (CAM) queues for each crypto asynchronous processor and instructions which permit all CPUs in the configuration to send and receive messages to and from all CAM queues.

The SCM implements the following cryptographic algorithms and functions to support the DAC architecture: Data Encryption Standard (DES) [4], Message Authentication (MAC), message digest, Secure Hash Standard (SHS) [5]; personal identification number (PIN) processing, pseudorandom number generation, key management functions, and control functions.

The SCM implements the following cryptographic algorithms to support the PKA and PKSC architecture: Rivest–Shamir–Adelman (RSA), Digital Signature Standard (DSS) [6], and Diffie–Hellman key agreement.

This paper discusses the key innovations in the design of the SCM:

- Cryptographic engine integration.
- Simultaneous synchronous and asynchronous operation execution.
- Logical and physical security.
- Key storage.
- Fault-tolerant design.
- FIPS 140-1 [7] validation.
- Performance.

Overview

Figure 2 represents a high-level overview of the SCM data flow. As is shown, the SCM chip attaches to the CPU through a dedicated hardware interface. Logically, the SCM is an execution-element extension of the CPU. The SCM is contained entirely in a single 12.7-mm × 12.75-mm CMOS 5X chip. This includes all instruction execution logic, nonvolatile key storage, and tamper-detection circuitry. The following are embodied on the chip:

- *CPU–SCM interface* This interface consists of a 36-bit (32-bit data, 4-bit parity) bidirectional data bus and 23 control lines.
- *Self-test and clock* This interface consists of 69 data and control lines.
- *Power and ground* This interface consists of two battery-backed-up unit (BBU) power pins, 376 power pins, and 351 ground pins.

As illustrated in Figure 2, within the secure boundary reside the input and output data buffers, execution controls, cryptographic algorithmic engines, and key storage, which includes DES master and auxiliary keys, signature keys, key part queues, and export control data. The entire implementation consists of hardware state machines. No imbedded software exists that contributes to instruction execution, key management, or security controls.

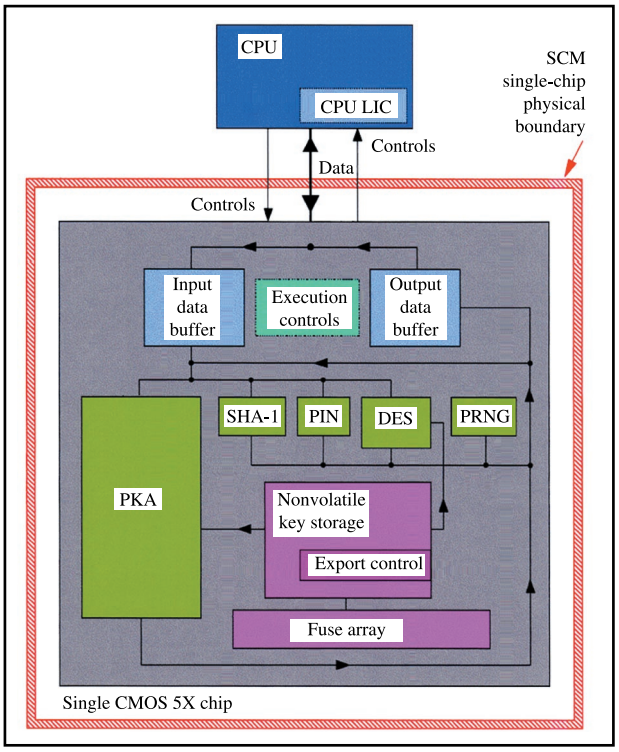


Figure 2

High-level overview of SCM dataflow.

Cryptographic engine integration

The three major elements in the SCM dataflow as shown in Figure 2 are the CPU interface, cryptographic engines (PKA, DES, SHA, and PIN), and key storage.

The CPU–SCM interface comprises the bidirectional data bus, control signals, and data input and output buffers. The CPU–SCM interface is controlled by the CPU's Licensed Internal Code (LIC). The CPU LIC initiates the cryptographic instructions and controls the data going into and out of the SCM; it does not control the execution of the instruction or the internal hardware state machines.

The majority of the dataflow logic is contained in the interfaces to the cryptographic engines. This includes working data registers and control logic that move data between engines and the CPU interface. The key logic contains working key registers which feed the cryptographic engines and perform key management operations. Integrated with the key logic is the key storage, which consists of battery-backed-up custom SRAMs that store keys, key parts, and configuration information.

Each of the cryptographic engines lies within the single chip and is controlled *entirely* by hardware state machines on the chip. For this reason, the SCM provides the maximum in security. The instruction execution cannot be altered and is embedded within the secure boundary of the chip. The state machines have also been optimized for hardware execution and provide a faster and more secure solution than a software implementation.

Functionally, the SCM chip can be divided into two parts: the direct attached crypto (DAC), and the crypto asynchronous processor (CAP), which encompasses the public key security control (PKSC) and public key algorithm (PKA) functions.

The DAC performs the synchronous SCM functions, which include DES-based encryption, hashing and MAC functions, PIN operations, key management and acceptance, and other key control functions. The DAC includes the CPU interface and the main controls and dataflow in the SCM. Once a cryptographic instruction is initiated by the CPU LIC, the data sent is stored in the input data buffer, and the main hardware state machine will control sending the data and keys (if needed) to the appropriate engine for processing. When the instruction is completed, the resulting data is loaded into the output data buffer and sent to the CPU when it is requested by the CPU LIC.

The crypto asynchronous processor (CAP) performs all of the public key security control (PKSC) and PKA operations. The CAP functions are performed asynchronously on the chip because of the large amount of time required to process these operations. Data is sent to and from the CAP via CPU LIC commands, which is a slightly different way of transmitting data from that used for the synchronous functions. Since these functions are run asynchronously, an interrupt is raised so that the CPU LIC recognizes when the CAP is done processing. The CPU LIC then requests that the resulting data be sent to the CPU directly from the CAP.

DAC cryptographic engines

- *DES*

The core synchronous encryption engine found in SCM is based on the Data Encryption Standard, or DES. The DES algorithm ciphers a 64-bit block of plaintext into a 64-bit block of ciphertext under the control of a 56-bit cryptographic key. The process of encryption consists of 16 separate rounds of encipherment, each round using a product cipher approach, or cipher function. These series of steps of substitutions and permutations in the algorithm determine its strength. Each round consists of three steps:

- The input block is split into two parts, a left half and a right half.
- The right half is then operated on using a cipher function.
- This output is combined (via XOR) with the left half.

The logical functions performed by the SCM DES hardware include the following:

- Encode and decode (electronic code book).
- Encipher and decipher (cipher block chaining).
- Translate.
- Triple encipher and triple decipher (two- and three-key CBC mode).
- MDC-2 and MDC-4.
- Message authentication (two- and three-key).

The DES algorithm is implemented entirely in hardware and performs two rounds in one machine cycle, for a total of eight machine cycles to cipher one doubleword of data. The DES engine for SCM has been validated under FIPS 46-2.

- *PIN*

A personal identification number (PIN) is a secret number assigned to or selected by the holder of a debit or credit card used in an electronic funds transfer (EFT) system. The PIN serves as the cardholder's electronic signature to authenticate his right of access to an account. In an interchange of EFT transactions, the PIN must be transmitted securely from the acquirer (the institution accepting the PIN) to the facility of the issuer for validation. Specific security requirements are placed on the acquirers in an interchange environment to ensure that an issuer's PINs are not compromised.

The PIN engine is designed to perform the basic banking functions of PIN generation, translation, and verification:

- PIN generation refers to the algorithmic creation of a PIN using the DES engine and a specific financial algorithm. The PIN is created from account data such that it can be consistently recreated, given the same input data.
- PIN translation refers to the electronic communication of a PIN from one institution to another for verification purposes. The PIN is embedded in a PIN block, which is the vehicle for transporting a PIN from one node to another. Once a PIN has been inserted into a PIN block, the block is enciphered under a PIN transport key before it is sent to the outside world. PIN blocks exist in many different formats. PIN translation allows the PIN block format, the enciphering key, or the format-padding parameters to be changed while translation occurs. The translation function allows two different institutions to communicate PIN information even though they may incorporate completely different security schemes.
- PIN verification refers to the authentication of a received PIN. This process involves extracting the received PIN from the PIN block and comparing it to a PIN algorithmically generated from associated account information.
- PIN offset is another function provided by the PIN engine. If the cardholder selects his own PIN, the offset function mathematically relates the user-selected PIN to the PIN created by PIN generation. This is done to ensure that the PIN verification process works correctly.

Each of the PIN functions provides a number of variations to support current EFT systems throughout the world. The SCM supports IBM, VISA, Interbank, ECI, ISO, and ANSI PIN formats.

- *SHA-1*

The Secure Hash Standard, or SHS, specifies a secure hash algorithm (SHA-1), which is implemented on the SCM.

In operation, a request for the SHA-1 engine may come either from the CPU via the Generate SHA-1 function or from the CAP for public key functions.

The 512 bits of request data arrive at the SHA-1 engine after passing through a staging register 64 bits at a time before being loaded into the 16-element-deep, 64-bit-wide register array. The register array not only serves as an accumulator for the incoming data, but also is used in the generation and accumulation of W0 through W79, as defined in the SHS standard. This generation is done while data is being fetched out for processing into the message digest in a pipelined fashion. The resulting 160-bit message digest is then sent to the SCM main dataflow to be sent back to the CPU or to the PKA engine for further processing. The SHA-1 engine for SCM has been validated under FIPS 180-1.

- *PRNG*

The SCM contains unique hardware for pseudorandom number generation (PRNG). Pseudorandom numbers can be requested by explicit instruction requests or for any of the other SCM DAC or CAP functions requiring a random number. Random numbers are used for many purposes such as key generation and padding. The PRNG algorithm employed is detailed in [8, 9]; this PRNG hardware includes

- A state machine.
- A free-running 64-bit incrementer (T register) running at the SCM clock frequency (approximately 110 MHz) that is also stored in battery-backed-up nonvolatile memory.
- A 128-bit randomization state register (S register) that is also stored in battery-backed-up nonvolatile memory. This 128-bit register is updated during idle times and during execution of a PKSC query command.

The PRNG state machine begins in any of four situations:

- During the pseudorandom number (PRN) initialization sequence initiated by ICSF.
- When one of the SCM functions that requires a pseudorandom number is received. This includes the DAC GPRN instruction and CAP operations that request a pseudorandom number.
- When the SCM is idle, the PRNG will run and update the S register continuously until another function request is received. The idle operation of the PRNG can be interrupted at any point; that is, if the PRNG is running while the SCM is idle, and a cryptographic instruction comes from the CPU, the PRNG is turned off and the cryptographic instruction is executed. The PRNG therefore has no performance impact on any of the other instructions.
- Following a power-on reset of the SCM, the S and T registers are restored from nonvolatile memory and the PRNG is launched. The contents of the T and the resulting S registers are then stored back into nonvolatile memory. This is a refresh of the PRN and occurs only if the PRN has been previously initialized.

- *CAP*

The CAP engine performs both the PKSC instructions, which provide the physical security controls, and the PKA instructions, which are used in implementing a public key infrastructure (PKI). The PKSC and PKA instructions rely on public key encryption and share the heart of the PKA engine, the modular multiplier.

PKSC instructions

There are two basic types of PKSC instructions, queries and signed requests. The queries request information from the SCM but do not change its state. Signed requests do alter the state of the SCM by performing such operations as creating transport keys, loading DES key parts, exporting encrypted keys, and setting permissions.

Signed requests are unique in that they may not necessarily complete when issued. The user can set up the PKSC engine to require multiple co-sign (COS) requests before completing the signed request. Signed requests are held in the pending command register (PCR) until either the required COS requests arrive or a new signed request is received. This wait could be as short as a few seconds or as long as several years (although in practice signed requests do not remain in the PCR for long.)

Prior to being placed in the PCR, all signed requests, including COS, must pass several tests: A valid RSA signature must be present, required fields must contain the proper values, and arithmetic fields must contain legal values. If any of the tests fail, the request is aborted, and a reply code indicating the failure is returned to the processor.

Query instructions always return data once the SCM is properly initialized. When a query instruction returns data, the reply message is signed so that the user can ensure that the reply came from the desired SCM and not from a malicious attacker.

PKA instructions

The PKA instructions, while using much the same hardware as PKSC instructions, execute quite differently. PKA instructions contain all of the data necessary to perform the operation; no data contained inside the SCM is required. Since the PKA instructions do not change the state of any architected facility, pre-testing of the instruction is not necessary. When a PKA instruction is received, its execution begins immediately. Tests are performed as convenient during the execution. If a test fails, the failure is recorded and execution continues. This has allowed the control logic for the PKA instructions to be simplified. A PKA instruction goes through the same

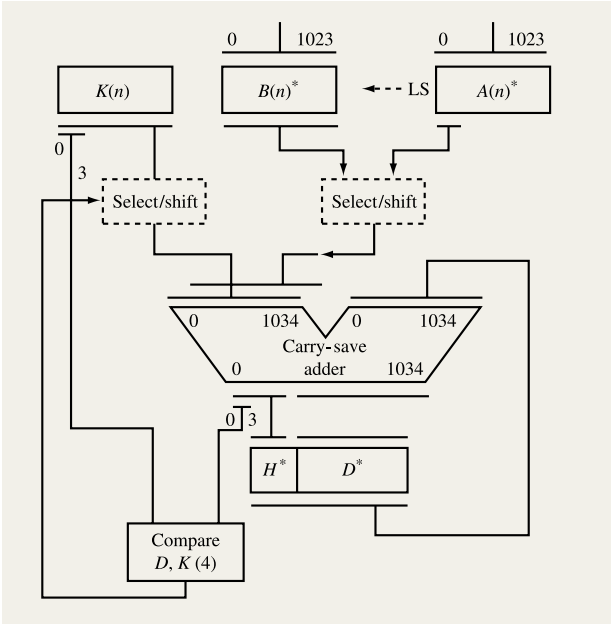


Figure 3

Modulo reduction multiplier.

sequence of states, independently of the results of the tests.

After the execution portion of the instruction completes, the reply message is built. When one of the execution tests fails, a failing reply message is generated. If all of the tests were successful, a normal reply message is returned with the final data. The normal reply message for PKA instructions includes all of the results of the operations which were performed. The instruction, when complete, leaves no data in the SCM.

Modulo arithmetic

RSA and other public key algorithms depend on modular arithmetic. Modular multiplication and modular exponentiation are the functions required.

To generate $A \times B \bmod(N)$, a modulo multiplier is used, in which the product $A \times B$ is reduced as the multiplication proceeds. Exponentiation, $A^B \bmod(N)$, is performed as repeated multiplications using a square and multiply algorithm.

Modulo multiplier

The multiplier is based on a paper by E. Brickell [10]. This multiplier calculates $A \times B \bmod(N)$ in a single multiply calculation, rather than calculating $A \times B$ and then reducing this product modulo N with a separate calculation. The multiplier is a delayed carry-save-type multiplier with some compare circuits that allow for the modulo reduction to take place during the multiply calculation.

Each register in Brickell's multiplier is split into a pair of registers containing the sum and carry bits from the addition operations. Brickell's design requires both sum and carry registers for the operands A and B and the intermediate results D. In order to save space in the SCM multiplier, operands A and B are not split into sum and carry registers. Instead, a carry-lookahead adder is used to add the sum and carry registers of D prior to reloading A and B. The effect is a savings of 2048 register bits at a cost of 16 cycles.

Figure 3 shows the organization of the multiplier. The operation of the multiplier is based on the basic principle of modulo reduction; that is, if Y is the modulo reduction of X modulo N and $X > N$, then $X = Y \bmod N$, $Y \leq N - 1$ can be calculated by repeatedly subtracting N from X until the result is less than N.

If N is the modulus, an integer number represented in k bits, and A and B are integer numbers that are each represented in k bits, then the product $A \times B$ is formed by using the binary representation of B in k bits:

$$B = b_0, b_1 2, b_2 2^2, \cdots, b_{k-1} 2^{k-1},$$

and using the A value,

$$A = a_0, a_1 2, a_2 2^2, \cdots, a_{k-1} 2^{k-1},$$

the result is accumulated in the accumulator D as follows ($\leftarrow$ indicates a left shift of one bit):

$D = 0$

do $i = 1$ to k

if $b_i = 1$, then $D = D + A$

$D = \leftarrow D$

end

As the product is accumulated, its magnitude approaches and eventually increases past the magnitude of the modulus N. If the appropriate point can be determined as the product magnitude increases past the magnitude of the modulus, the modulus value can be subtracted, thus reducing the product value modulo N at each appropriate point.

Subtracting the modulus is accomplished by precomputing a value $K = 2^k - N$, or the two's complement of N. When the compare circuitry determines the appropriate point at which to subtract the modulus, the K value is added to the product.

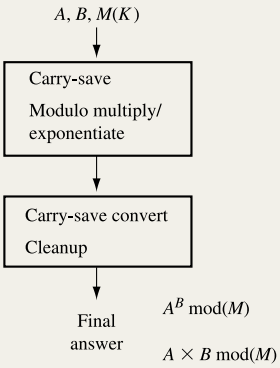


Figure 4

Modulo calculation.

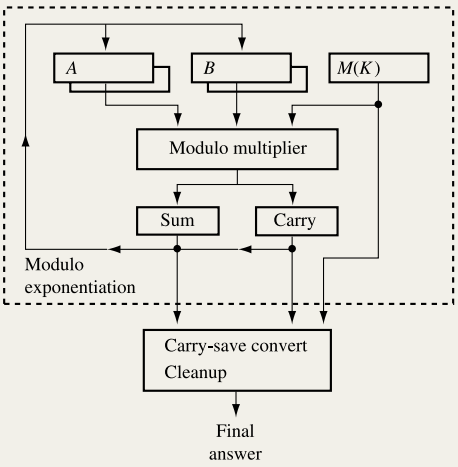


Figure 5

Modulo multiply/exponentiate.

Modulo multiplication and exponentiation

The functional flow of the modulo engine is shown in **Figure 4**. The engine incorporates a modulo multiplier, and some conversion and cleanup hardware. The multiplier performs the modulo multiplication and exponentiation functions, and the convert and cleanup hardware converts the result into the proper binary format to return to the application. Note that the convert and cleanup are done only at the end of an exponentiation or multiply calculation, not during the intermediate steps of a calculation.

Multiply and exponentiate

The modulo multiplier uses the carry-save adder described previously in the subsection on modulo arithmetic. The adder is 1035 bits wide and accepts input from the A, B, and K registers as well as the previous results, in carry-save format. The adder performs

$$D_i = 2(D_i + B + K)$$

for $i = j + 1$, and compares D to K as one step. The compare involves the high-order four bits of the D register and the appropriate value of K.

The exponentiation (M^E) is performed using the following algorithm. If M^{14} is to be calculated,

E = 14, or '1110' b,

and the powers

M^2, M^3, M^6, M^7, and M^{14}

would be calculated. Note that all of the calculations involve either multiplying the previous result by itself (squaring) or multiplying the previous result by M (for example, $M^3 = M^2 \times M$). This is shown in **Figure 5**.

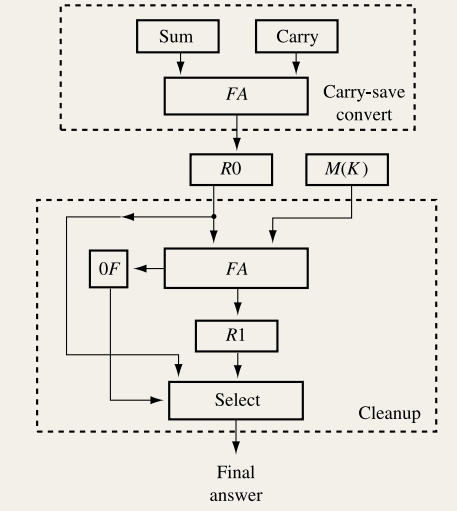


Figure 6

Convert and cleanup.

Convert and cleanup

The convert hardware, as shown in **Figure 6**, takes the carry-save representation of the result and converts it to a binary string. This is done by adding the result sum and carry values to form a single binary value. This result may be one bit larger than the block size because of overflow.

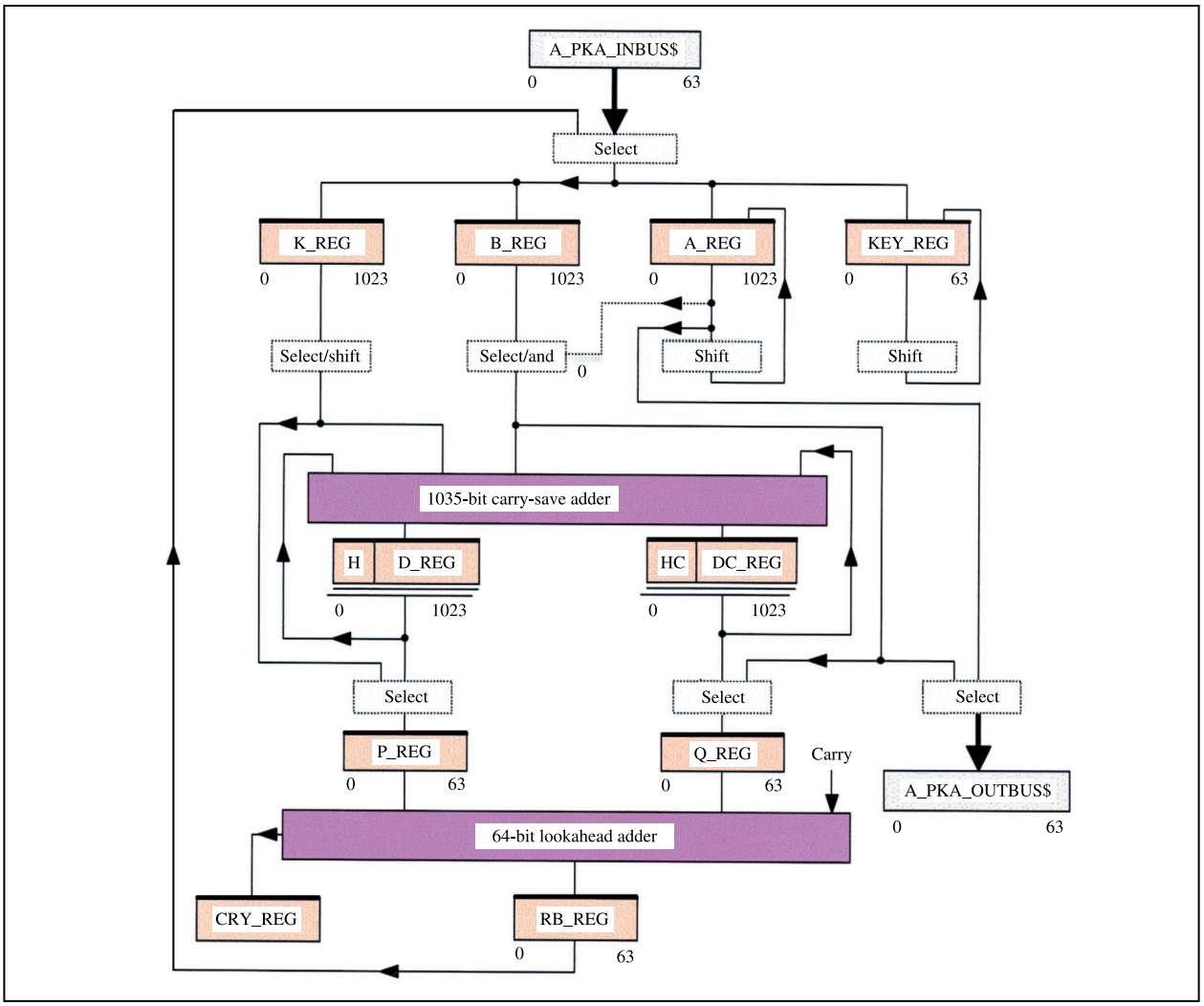


Figure 7

PKA engine dataflow.

The cleanup hardware ensures that the result formed by the binary conversion, $R0$, is smaller than the modulus. The result $R0$ may contain the real answer R and the modulus M, $R + M$. The check is done by adding K to the value $R0$ and checking for the overflow of 2^k:

$$R1 = R0 + K$$

or

$$R1 = R0 + 2^k - M;$$

then,

$$R1 = 2^k + R0 - M,$$

so that if

$$R0 - M \geq 0,$$

there will be an overflow and $R = R1$; otherwise, there is no overflow, and $R = R0$.

PKA engine dataflow

Figure 7 illustrates the PKA engine dataflow. The engine performs the necessary arithmetic, as indicated previously, to perform the algorithmic operations of RSA, Digital Signature Standard, and Diffie–Hellman key agreement.

Simultaneous synchronous and asynchronous execution

As described, the SCM may perform two types of operations: DAC and CAP. The DAC operations include 95 functions defined under ESA/390*. All DAC functions

are performed synchronously with respect to the issuing CPU. The SCM acts as an integral execution unit of the CPU. The CPU hardware, along with the CPU LIC, performs the initial instruction decode, setup, exception testing, and operand fetching. The instruction operation code and all operand data (general-purpose register data and/or storage data) are then transferred to the SCM over the data bus for actual instruction execution within the SCM. Depending on the operation, the CPU LIC will receive and store any results from the SCM and wait for an end-of-operation signal from the SCM which signals that execution is complete. CPU LIC will then complete any CPU ending steps before signaling that execution is complete and the CPU is available for execution of additional instructions.

The CAP operations include the functions of the public key security control and public key algorithms. Some of these operations may take milliseconds to execute; if run synchronously, they would severely impede the performance of the CPU. For this reason, the PKA and PKSC operations are run asynchronously. Communication between the CPU and the SCM for these asynchronous operations is executed by means of messages which pass through a queue and are sent to and from the SCM. The CPU LIC uses special commands to send an asynchronous request message to the SCM and receive the reply message when the execution of the operation is complete. While the CPU LIC does use the same interface as the DAC in sending/receiving these messages, these special commands route the messages to the asynchronous processor. This implementation supports simultaneous processing of both synchronous and asynchronous operations in the SCM.

All asynchronous functions are initiated by the CPU through the synchronous DAC. Once all data has been sent to the CAP for asynchronous processing, the DAC enters a state in which it may again accept synchronous operations. Within asynchronous operations, the CAP will request services of the DAC to perform functions such as hashing, DES encryption, random number generation, and other synchronous operations. The DAC must be idle (not performing a synchronous function) to be able to accept a CAP request for service. If the DAC is busy, the CAP will wait for services until the synchronous operation has completed.

The same is true for synchronous requests from the processor. If the DAC is busy with a request from the CAP, the DAC will reject all synchronous operations from the processor with a busy (inhibit) status. The processor will attempt to reissue the synchronous operation to the SCM until it is accepted (after the DAC has completed the CAP request).

There is a delicate balance between the synchronous and asynchronous functions. The design of the SCM controls was carefully devised so that synchronous and asynchronous functions do not interfere with each other. This is extremely important because the functions share the DES engine, SHA-1 engine, PRNG, dataflow, and interface with the processor unit. This sharing of elements is important in order to optimize area usage and to reduce redundancy in logic. However, this arrangement posed several challenges, such as sharing resources to minimize performance degradation, restoring working registers to the correct state, and maintaining the integrity of the operation such that they are logically independent. Any collision between synchronous and asynchronous functions would be catastrophic.

Key storage

Integrated within the chip are six custom nonvolatile arrays (C-SRAMs). These arrays must prevent the critical data contained in them from being lost when power is interrupted on either a scheduled or an unscheduled basis. In addition to the 2.5 volts supplied from the power supplies while ac power is active, two signal I/O lines carry external regulated battery voltage to the C-SRAM array cells, maintaining the data even when the system power drops. The system power supply uses a switch to alternate between battery power and the normal standby power supply. The switch selects the highest voltage, so that if the standby supply is active, it always supplies power. If the standby power supply fails, the battery supplies the power. A lithium battery cell is used to achieve the necessary shelf life. There is a low-battery-voltage sensor to signal the need for battery replacement. The power implementation includes the redundancy of two power cards, each with batteries to reduce the probability of a total power failure.

There are two sizes of C-SRAM: The first is 64 elements by 73 bits and is used for the key buffer, key part, signature-object master key, and receive-object master key arrays; the second is 512 elements by 73 bits, which is used twice to make a logical 1024-element PKA array. Each element contains 64 data bits, 8 error-correction code bits, and a complement-indication bit. To protect against ionizing radiation attacks, the data held in the nonvolatile arrays is periodically complemented, that is, read, inverted, and rewritten to the array. This method ensures that no imprinting of data can occur in the array cells. Complementing occurs approximately every 1.5 hours.

The C-SRAMs contain customer keys, key parts, configuration, export control, and other substantial data. This data may be stored for long periods of time and may be stored under battery power. Since it is essential that errors in the array be detected to prevent data integrity problems, a strong method of error detection is employed.

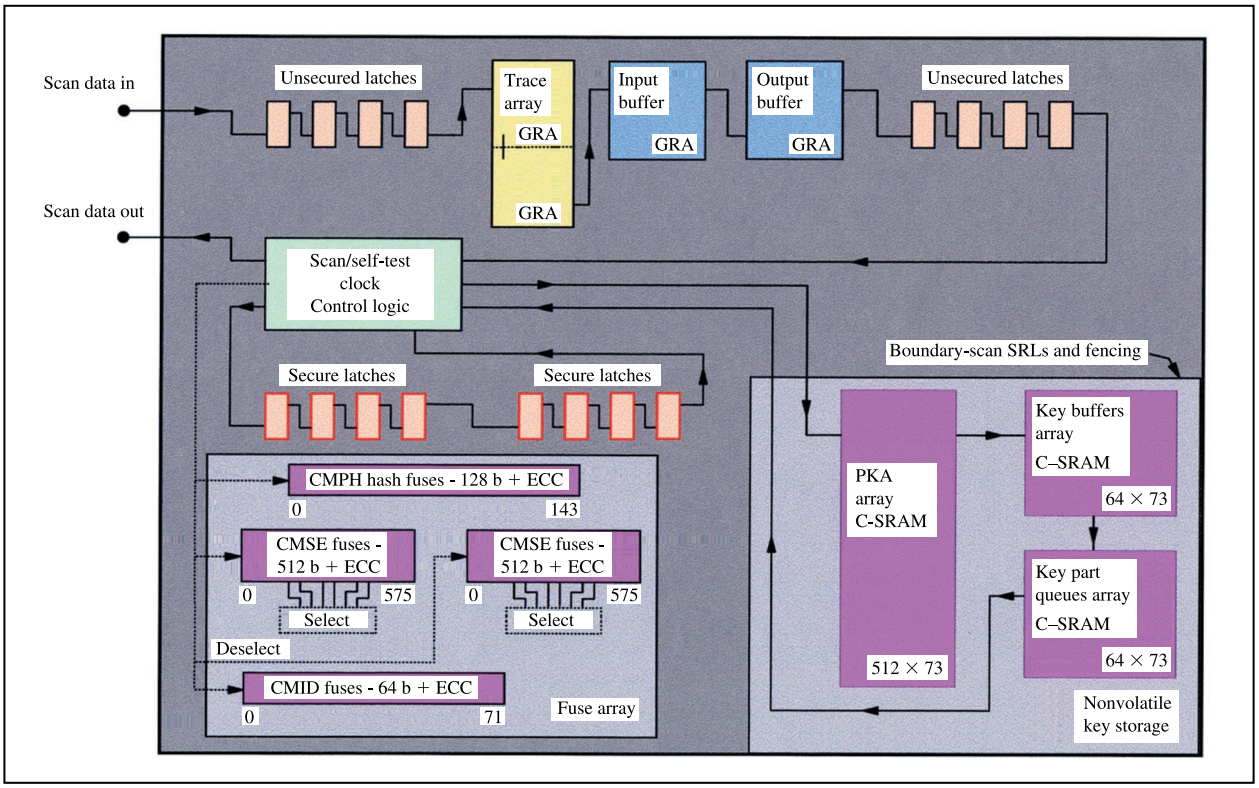


Figure 8

SCM scan and self-test configuration.

To deter unnecessary outages due to array failures, a technique of correcting the error improves reliability. The error-correction code (ECC) implemented for the C-SRAMs corrects single-bit errors and detects two-bit errors. The ECC is generated on the 64-bit data in the array plus the complement indication bit and the array address parity. All elements in the array are checked, including the address decoder. The ECC is generated and stored for each element in the array as it is written. When an element is read from the array, the ECC from the array is checked with the ECC generated on the output data of the array.

Logical and physical security

Regardless of a chip's architectural function, the chip must support basic infrastructure requirements; therefore, these support functions are typically found in any VLSI chip design. For SCM, many of these functions are modified to meet FIPS 140-1 security requirements, as described in the following subsections.

- *Resets*

For most VLSI chips, resets are performed using the basic scan chain structure. For SCM, however, the majority of the registers are hardware-reset, and scan-chain operations are inhibited.

- *Chip scan and test security*

The SCM chip has unique security requirements for its scan and test design. **Figure 8** gives an overview of the SCM from a chip-scan and self-test perspective. The design must allow for debugging of problems during early system bringup. However, data processed on chips that have been shipped to customers must be incontrovertibly secure. Additionally, these chips must achieve the same high manufacturing test coverage level as the rest of the chips in the S/390 systems.

The first step in achieving the required security was to carefully identify the latches which may contain information that must be protected (called secure latches) and the latches which will always contain data that it is

permissible to access for debug (called unsecured latches). The scan chain of the chip is divided such that all of the secure latches are in one section and all of the unsecured latches are in a separate section. A chip that is "shippable" to a customer has unique personalization embedded during manufacturing that will cause it to treat these two scan rings quite differently. However, a chip built for use in internal bringup treats both scan rings as if they were unsecured. No unsecured chip can be shipped, because the manufacturing test patterns produce completely different signatures for secure and unsecured chips. An unsecured chip would therefore fail the manufacturing test.

The SCM arrays that contain secure information can be accessed by scanning secure latches that connect to their data, address, and control pins. They are therefore accessible for debug, but are *not* accessible on a chip that has been built to be shippable to a customer.

Thorough manufacturing tests require access to all SRLs and arrays. To enable these tests, the chip has facilities that precisely monitor whether it has been put into one of the test modes. When the chip passes into or out of a manufacturing test mode, these facilities create reset signals and generate clocks that clear out all latches. While these resets are in process, the chip "fences off" the usual input pins controlling clocking and other basic operations, so that external manipulation of these pins cannot prevent the resets from completing. In concert with this security reset mechanism, there are controls that fence off the arrays under all conditions except system operation and array test. Array test is performed by an on-chip engine that never latches array data values, only pass/fail results. Finally, clocks are gated off directly when the chip is put into various test modes to ensure that test observation registers can never load customer data.

The SCM uses this combination of fencing, resetting, and clock blocking on the basis of the mode and any transitions of modes to enforce rock-solid security practices while allowing extensive manufacturing test coverage.

- *Clocking*

The SCM chip has unique requirements in its clocking design. It is attached to a CPU chip which runs on clocks that are higher in frequency than the SCM chip clocks. The temperature, sort points, and design vintage of the attached CPU may be quite different in the various machines to which the SCM chip is attached. The SCM-chip-to-CPU-chip interface must be synchronous, however, so that both chips will start and stop together. We therefore designed the SCM chip to run at programmable "gear ratios" which allow it to operate at a clock frequency that is a multiple of the clock frequency of the corresponding CPU.

The SCM, the CPU, and other central chips in an S/390 system receive a reference oscillator signal (REF_OSC). Each chip then uses a phase-locked loop (PLL) to generate a higher-frequency clock, which rises in sync with the REF_OSC signal. This design results in tightly synchronized clocks across all chips. The system clock chip drives the basic critical signals to all chips that control their clocking and scanning. The challenge for the SCM chip was to use the same existing signals as any other chip, but to interpret them in a way that allows synchronous operation with no data integrity exposure at a frequency that may not be an integer divisor of the REF_OSC signal.

The PLL in a chip that does not implement gearing has only one output frequency. The REF_OSC frequency is multiplied by a programmable control to create an internal PLL frequency, and is then divided down by a programmable integer to create the output frequency. The output frequency is sent through a typical clock distribution path and routed back to the PLL feedback, where it arrives at the same time as the trigger clock signal would arrive at a typical latch/trigger pair (SRL). The PLL uses the phase difference of the rising REF_OSC versus this feedback clock to control the oscillator that creates the internal PLL frequency, thereby driving the phase difference to zero. The SCM PLL, in contrast to this, was designed to drive out two frequencies: the nest frequency, which is generated in exactly the same manner as on other chips in the system, and the SCM frequency, which is simply a different multiple of the internal PLL frequency. Only the nest frequency is actually routed from a typical end distribution point to the PLL feedback pin, so only the nest frequency is precisely tuned to the REF_OSC. However, the SCM-frequency clock distribution is carefully designed to closely match the nest clock distribution on an SCM chip, so that the skew between the two clocks is minimized.

The SCM uses the nest-frequency clock section to interpret signals to and from the clock chip. The nest clock is also used for a single signal to each attached CPU that allows the CPUs to predict the proper cycles during which to transfer information on the SCM interface. The SCM frequency is used for all cryptographic processing and for command and data communications to and from the attached CPUs. The SCM uses *both* clocks to determine the relative alignment of the clocks and thereby control the gearing.

The alignment of the two clocks is determined through a series of SRLs that look for a signature sequence pattern of the value of the REF_OSC when observed at SCM latch-time points. When the signature sequence is detected, indicating the concurrent rising-edge alignment, a ring-shift register is preset to a single 1 in the proper position. This ring shifter is clocked by nest clocks.

The position of the single 1 indicates the current clock alignment at any time. The state of this ring-shift register is used to interpret the clock controls in the proper way to ensure that clocks are never chopped short, and never occur when they would cause a data integrity exposure due to clocks having already stopped on the CPU side of the interface.

The CPU chip uses the synchronizing signal from the SCM to preset its own ring shifter, and then decodes the state of its ring shifter to determine the proper cycle to load its SCM interface registers.

This gearing design, through the mechanisms outlined above, provides the flexibility required to include the SCM chip in systems of highly varying performance characteristics.

Reliability, availability, and serviceability

The reliability/availability/serviceability (RAS) strategy is to continue the S/390 objective of providing continuous reliable operation (CRO). The SCM, as part of the S/390 hardware structure, incorporates many mechanisms to fulfill the objective of CRO.

Error detection is fundamental in ensuring the integrity of data. Errors must be detected at the time of failure, contained, and isolated to the smallest entity possible to make recovery reasonable and to enable accurate field-replaceable unit (FRU) isolation. Error detection and fault isolation (ED/FI) mechanisms must detect either stuck or intermittent hardware faults. Once an error is detected, effective recovery mechanisms ensure continued functional availability. Isolation of an error to an FRU ensures rapid field serviceability.

The S/390 Parallel Enterprise Server design requirement for hardware logic chip design and implementation is to have an error-detection coverage of single bit faults of at least 95% and isolation to a single chip.

For the SCM, the coverage requirement of 95% was exceeded, with a coverage of 97.3%. This coverage was implemented using various well-known logic design error-detection concepts [10–12]:

- Byte parity on all internal and external data buses.
- Sequence, invalid states, and duplication on logic control state machines.
- Duplication of DES engines.
- Parity prediction and carry checking on adders and ALUs.
- Residue checking on the modular exponentiation engine.
- Parity on general register arrays.
- Double-bit error detection and single-bit correction on C-SRAM arrays and fuse arrays.
- Single-bit error scrubbing every three hours on C-SRAM arrays.
- 100% fault isolation on chip logic interface.
- Starting with G5, the physical connection of each SCM to two CPUs (as indicated in Figure 1). If the primary CPU to which the SCM is attached fails, the recovery action will logically reassign the SCM to the second physical CPU.

All of the above tests are continuous; at every machine clock cycle, the various hardware error checkers check the entire complement of hardware logic. For internal isolation purposes, the SCM contains more than 600 individual error latches. Therefore, when an error is detected, examination of the active error latches can assist in isolation of the source of the error. This is useful information for post-error failure analysis.

- *Error injection*

Implementation of ED/FI is critical to meet CRO, but mechanisms must be implemented as well to test and verify the logic. Starting with the G5 Enterprise Server, additional hardware was implemented in the SCM to enable the injection of different types of errors for recovery testing and validation. This error injection provides the ability during hardware test to verify that the hardware and external machine-level recovery code are designed and implemented correctly in reporting, recovering, and responding to errors.

The setup for error injection uses the same CPU-to-SCM interface and is set up using various options available via the CPU LIC. A setup is done such that an error (stuck or intermittent) is encountered and reported when certain preset conditions are met. This can be based on a trap event, a multiple occurrence of the trap event, and a cycle offset as well. Single- and double-bit errors can be loaded into the C-SRAMs to test both correctable and noncorrectable situations.

- *Recovery*

Once an error is detected, various recovery actions can be taken to clear the error. For intermittent errors, it is important that the hardware can be recovered and continue to be used within the system. The SCM has numerous recovery actions. Errors are of the following types:

- Arrays
  - Single-bit correctable error, either intermittent or stuck.
  - Multibit error, either intermittent or stuck.
- Non-array logic
  - Single-bit intermittent or stuck error.
  - Hang condition.

When an error is detected, the SCM enters a *failure* state. Logging of entries in the hardware trace array is stopped, hardware state machines are quiesced, the CPU/SCM

interface is fenced, and the error state is reported to the CPU. The CPU LIC will initiate a recovery action. Depending on the activity of the SCM at the time of the error, it may be able to reset and recover the SCM. If this is possible, it is done in a way that is transparent to the application. If the error is such that the SCM cannot be recovered, the SCM will remain in the failure state and be removed from the CEC configuration, and a maintenance action will be requested.

If the error is a correctable error, this information is automatically logged and recorded for threshold activity. If numerous correctable errors occur within a set period of time, the SCM will enter the failure state and a repair action will be requested.

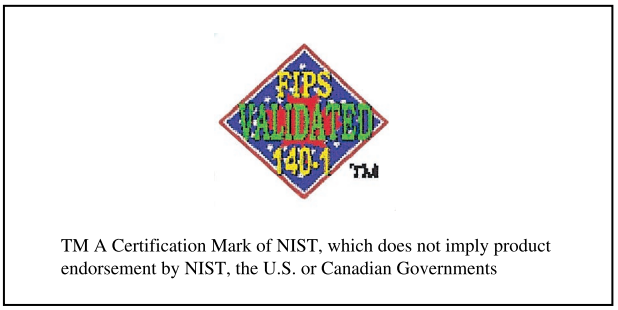
FIPS 140-1 security compliance

Federal Information Processing Standard 140-1, Security Requirements for Cryptographic Modules [7], "specifies the security requirements that are to be satisfied by a cryptographic module utilized within a security system protecting unclassified information within computer and telecommunication systems (including voice systems). The standard provides four increasing, qualitative levels of security: Level 1, Level 2, Level 3, and Level 4. These levels are intended to cover the wide range of potential applications and environments in which cryptographic modules may be employed. The security requirements cover areas related to the secure design and implementation of a cryptographic module. These areas include basic design and documentation, module interfaces, authorized roles and services, physical security, software security, operating system security, key management, cryptographic algorithms, electromagnetic interference/electromagnetic compatibility (EMI/EMC), and self-testing."

The SCM was successfully validated to comply with the rigorous requirements of the FIPS 140-1 standard and was awarded certification on January 7, 1999, at an overall Level 4 (**Figure 9**). The entire list of validated products can be found at the referenced URL [13]. The cryptographic module physical boundary defined by FIPS 140-1 as it applies to the SCM is indicated in Figure 2 as all of the logic and function encompassed within the red boundary. As a prerequisite to FIPS 140-1 validation, all FIPS-approved implemented cryptographic algorithms must also be validated against their respective standards. The SCM implements and has had validated the FIPS standard algorithms of DES, SHS, and DSS. As part of the validation, a rigorous number of defined tests must be passed [14].

The SCM security policy [15] provides a description of the cryptographic module, including all hardware, software, and firmware components, and defines the roles and services provided. A number of unique characteristics differentiate the SCM with respect to the validation:

1. The SCM contains no software, firmware, or operating system. Implementation is entirely in hardware.
2. The SCM is a single-chip module implementation.
3. Power-up tests and conditional tests are implemented using the internal RAS characteristics of the module rather than explicit known-answer tests.



TM A Certification Mark of NIST, which does not imply product endorsement by NIST, the U.S. or Canadian Governments

Figure 9

IFIPS 140.1 certification logotype.

Performance

The first design objective for the SCM was to meet the strict security requirements for cryptographic operations and a rich set of functions. The second design objective was to meet the S/390 RAS requirements. The third design objective was to equal and exceed the performance criteria set by the previous bipolar Integrated Cryptographic Facility (ICRF) for symmetric key functions and obtain reasonable performance for the new asymmetric key functions. Performance was optimized in the following ways:

- *CPU direct-attach coprocessor* Rather than an I/O-attached coprocessor, the SCM is attached directly to the host CPU. This direct connection to the CPU, and integration as a logical execution engine, ensures minimal latency to instruction decoding, setup, operand fetching, and storing and execution. The high-performance Generation 6, 637-MHz S/390 CPU provides the direct storage interface and CPU LIC services for the SCM.
- *Hardware-only design* As described, execution of all operations within the SCM is controlled and managed by hardware state machines. This provides highly optimized instruction execution with minimal idle or setup cycles.
- *Cryptographic engine integration* As described, the integration of the various cryptographic engines with

Figure 10

Photograph of the S/390 SCM, showing relative size.

very tightly controlled data-path structures provides optimum channeling of data between I/O and engines within the SCM.

- *Technology* The SCM is manufactured in IBM CMOS 5X [16] technology. This is a high-performance technology whose attributes include the following: supply voltage, 2.5 V; 0.25-μm L_{eff}; lithography generation, 0.5 μm; five metal levels. For I/O the chip uses a solder-ball grid-array die for high-performance packaging configurations.
- *Cycle time* As the CPU design has progressed using newer technologies, the SCM has remained in CMOS 5X technology. This has created a cycle-time differential between the CPU and the SCM. Therefore, the SCM internal cycle time varies as an integer multiplier of the direct-attached CPU. For the recently announced S/390 Generation 6 Parallel Enterprise Server, the SCM operates at 127 MHz.

Some examples of algorithmic function execution rates at the SCM hardware level at 127 MHz include

- DES encryption CBC mode: 102 megabytes per second.
- RSA 1024-bit signature generate: 78.5 transactions per second.
- RSA 1024-bit signature verify: 53000 transactions per second ⟨3-bit exponent⟩.
- RSA 1024-bit signature verify: 23000 transactions per second ⟨17-bit exponent⟩.

Summary

The SCM (**Figure 10**) provides a complete set of cryptographic algorithms within a secure architecture on a single-chip implementation. It has been designed and validated to meet the highest level of security, Level 4, as defined in NIST FIPS 140-1. With its introduction on the IBM G3 Enterprise Server, the SCM became the fastest, smallest, and most secure coprocessor ever developed.

Acknowledgment

Design of the SCM would not have been possible without the combined efforts of the ESA/390 crypto architecture team, the SCM hardware team, the SCM physical design team, the SAK test team, engineering test, and the Integrated Cryptographic Services (ICSF) development team. Development was also a global effort, with contributions from IBM Denmark and IBM Boeblingen.

References

1. IBM Corporation, *S/390 Parallel Enterprise Server and OS/390 Reference Guide*, Order No. G326-3070-07, October 1998; available through IBM branch offices.
2. IBM Corporation, *OS/390, Integrated Cryptographic Service Facility: Overview Version 2 Release 6*, Fourth Edition, Order No. GC23-3972-03, September 1998; available through IBM branch offices.
3. IBM Corporation, *Enterprise Systems Architecture/390: Principles of Operation*, Fifth Edition, Order No. SA22-7201-04, June 1997; available through IBM branch offices.
4. *Data Encryption Standard (DES)*, Federal Information Processing Standards Publication 46-2, National Institute of Standards and Technology, Washington, DC, December 30, 1993.
5. *Secure Hash Standard*, Federal Information Processing Standards Publication 180-1, National Institute of Standards and Technology, Washington, DC, April 17, 1995.
6. *Digital Signature Standard (DSS)*, Federal Information Processing Standards Publication 186-1, National Institute of Standards and Technology, Washington, DC, December 15, 1998.
7. *Security Requirements for Cryptographic Modules*, Federal Information Processing Standard 140-1, National Institute of Standards and Technology, Washington, DC, January 11, 1994.
8. P. C. Yeh and R. M. Smith, Sr., "S/390 CMOS Cryptographic Coprocessor Architecture: Overview and Design Considerations," *IBM J. Res. Develop.* **43**, 777–794 (1999, this issue).
9. P. C. Yeh, R. M. Smith, Sr., and R. S. DeBellis, "Pseudorandom Number Generator," filed as patent docket PO997046; "Pseudorandom Number Generator with Backup and Restoration," filed as patent docket PO997049; "Pseudorandom Number Generator with Normal and Test Modes of Operation," filed as patent docket PO997058.
10. E. F. Brickell, "A Fast Modular Multiplication Algorithm with Application to Two Key Cryptography," *Advances in Cryptology* (*Proceedings of CRYPTO 82*), Plenum Press, New York, 1983, pp. 51–60.
11. D. C. Bossen and M. Y. Hsiao, "Model for Transient and Permanent Error-Detection and Fault-Isolation Coverage," *IBM J. Res. Develop.* **26**, 67–77 (January 1982).
12. "ED/FI: A Technique for Improving Computer System RAS," presented at the 11th IEEE International Symposium on Fault Tolerant Computing, 1981.
13. *http://csrc.ncsl.nist.gov/cryptval/140-1/1401val.htm*.

14. *Derived Test Requirements for FIPS PUB 140-1, Security Requirements for Cryptographic Modules*, National Institute of Standards and Technology, Washington, DC, March 1995.
15. P. C. Yeh, "Security Policy for IBM S/390 CMOS Cryptographic Coprocessor," *http://www.ibm.com/security/html/secpol.html*.
16. IBM Corporation, *CMOS 5X Logic Products Data Book*, Order No. SA14-2204-01, January 1996; available through IBM branch offices.

Received November 3, 1998; accepted for publication July 22, 1999

Randall J. Easter *IBM System/390 Division, 522 South Road, Poughkeepsie, New York 12601 (reaster@us.ibm.com).* Mr. Easter received a B.S. degree in electrical engineering from Pennsylvania State University in 1978, joining IBM that same year. He was involved in the ES/3090 CPU I-element, E-element, and Control Store hardware design. He was the lead engineer on the ES/3090 Vector Coprocessor hardware design team. Later he became team leader and lead engineer for the ES/3090 Integrated Cryptographic Feature (ICRF) Coprocessor. In the early 1990s, he was team leader for the G3/G4 CMOS Cryptographic Coprocessor. Mr. Easter is a Senior Engineer and is currently involved in future cryptographic hardware development. He is an author of twelve filed patents and has received an IBM Division Award, an IBM Outstanding Technical Achievement Award, and, recently, two IBM Outstanding Innovation Awards.

Edward W. Chencinski *IBM System/390 Division, 522 South Road, Poughkeepsie, New York 12601 (chencins@us.ibm.com).* Mr. Chencinski received a B.S. degree in electrical engineering from Lehigh University in 1980, joining IBM that same year. He was involved in the ES/3090 SCE and Expanded Storage hardware design. For the ES/9000, he was involved in the hardware design of the Logic Support Element. In the early 1990s, Mr. Chencinski joined the G3/G4 CMOS Cryptographic Coprocessor hardware design team, focusing on pervasive functions, simulation, and timing. Mr. Chencinski is a Senior Engineer; he is currently involved in future cryptographic hardware development. He is an author of one filed patent and one pending patent and author of several journal articles.

Edward J. D'Avignon *IBM System/390 Division, 522 South Road, Poughkeepsie, New York 12601 (davignon@us.ibm.com).* Mr. D'Avignon received a B.S. degree in computer engineering in 1988 from Syracuse University. He joined IBM that same year and was involved in vector hardware design. In the early 1990s, he joined the G3/G4 CMOS Cryptographic Coprocessor design team, focusing on the PKA engine design. He is an Advisory Engineer currently involved in future cryptographic hardware development. Mr. D'Avignon serves as Vice President of Tau Beta Pi, the engineering honor society. He is an author of one filed patent and three pending patents.

Seth R. Greenspan *IBM System/390 Division, 522 South Road, Poughkeepsie, New York 12601 (sgreensp@us.ibm.com).* Mr. Greenspan received a B.S. degree in electrical engineering in 1988 from the University of Connecticut. Since joining IBM in 1988, he has been involved in the hardware design of the ES/9000 Instruction Execution Element and the Integrated Cryptographic Feature (ICRF). He was a lead designer for the G3/G4/G5/G6 CMOS Cryptographic Coprocessor. Mr. Greenspan is an Advisory Engineer and is currently involved in S/390 cryptographic hardware development. He has received two IBM Outstanding Technical Achievement Awards.

William A. Merz *IBM System/390 Division, 522 South Road, Poughkeepsie, New York 12601 (merz@us.ibm.com).* Mr. Merz received a B.S. degree in electrical engineering in 1981 from the New York Institute of Technology, joining IBM that same year. He was involved in the ES/3090 diagnostic software

development and design, and he also contributed to the ES/3090 Integrated Cryptographic Feature (ICRF). In the early 1990s, Mr. Merz was involved with the hardware design for the G3/G4 CMOS Cryptographic Coprocessor RSA engine. He is currently involved in development of microcode for the PCI cryptographic processor. Mr. Merz is an author of one filed patent.

Clark D. Norberg *IBM System/390 Division, 522 South Road, Poughkeepsie, New York 12601 (cnorberg@us.ibm.com).*
Mr. Norberg received a B.S. degree in electrical engineering in 1977 from the University of Buffalo. He joined IBM in 1977 and was involved in the IBM 3031 channel and attached processor design. He worked on the ES/3090 IOP hardware design team. Later he became team leader and lead engineer for the ES/9000 Integrated Offload Processor (IOP). In the early 1990s, Mr. Norberg joined the G3/G4 CMOS Cryptographic Coprocessor design team focusing on simulation. He was the design and integration team leader for the G5 CMOS Cryptographic Coprocessor design. Mr. Norberg is a Senior Engineer; he is currently involved in future cryptographic hardware development. He is an author of several journal articles and has received two IBM Outstanding Technical Achievement Awards.