SecureCore

SP Processor Architecture Reference Manual

Princeton University

Version 1.0 – 8/11/2008
Jeffrey Dwoskin and Ruby Lee
(with feedback from NPS & ISI)

Princeton University Department of Electrical Engineering
Technical Report CE-L2008-008

Table of Contents **Page**

1 Overview

This manual documents the architecture extensions for Authority-mode SP, to be added to an existing base processor. Refer to [Dwoskin07] and [Lee05] for further details about the architectural features and their intended uses. It also provides the interface for the software SP Emulation Module for the SecureCore demo, as well as the TSM Interface for a key management application for the SecureCore demo.

Architectural Specifications

Current specification is for:
- Single-core
- Single-threaded execution
- Single active or suspended CEM thread – Authority-mode SP
- 64-bit word size (see Appendix A for 32-bit)

Note: Multi-core and simultaneous multithreading (i.e. hyper-threading) are open issues.

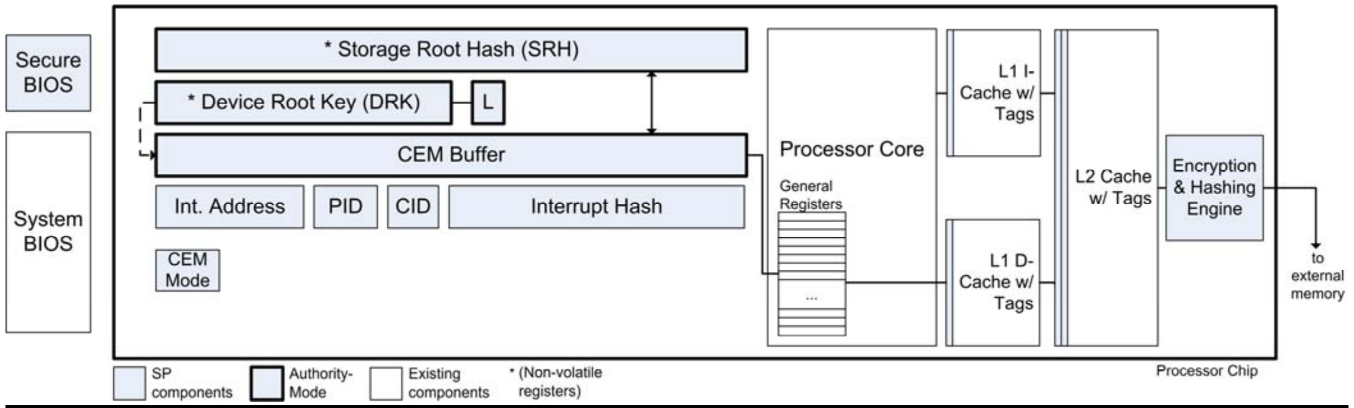


Figure 1: Basic SP Architecture Block Diagram for an authority-mode SP device

2 SP Registers

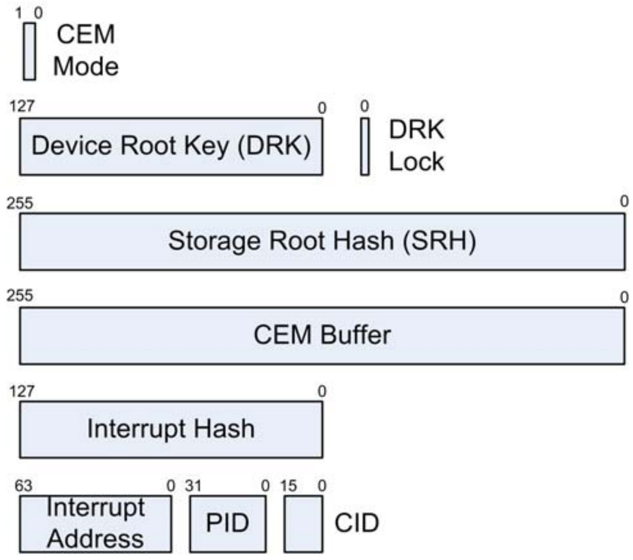


Figure 2: SP Registers

Little-endian bit-addressing is used. In the rest of this document, the 'right half' of a register refers to the low-order bits (e.g. bits 0-127 of the SRH), and the left-half refers to the high-order bits (e.g. bits 128-255 of the SRH).

CEM Registers

CEM_Mode (2 bits)

Indicates CEM state for Authority-mode SP. Authority mode can be in one of three states: normal (non-CEM), suspended CEM, or active CEM.

CEM_Mode	Authority CEM
00	Normal (Non-CEM)
01	Active CEM
10	Suspended CEM
11	*Reserved*

Device Root Key (DRK) (128 bits – Non-volatile) – *Authority-mode*

Device Root Key. Master device key for Authority-mode CEM protection and for deriving keys. Writable only through DRK.set instruction when DRK_Lock == 0. Not directly readable via instruction set.

DRK_Lock (1 bit) – *Authority-mode*

DRK lock bit. Cleared to 0 on reboot. Set to 1 using DRK.lock instruction in Secure BIOS. Once the DRK_Lock register is set, it cannot be cleared to 0 without rebooting. The DRK.set instruction raises an exception if DRK_Lock == 1.

Storage Root Hash (SRH) (256 bits – Non-volatile) – *Authority-mode*

Storage Root Hash. On-chip root hash for Authority-mode TSM storage tree. Used by Authority-mode TSM to verify integrity of its protected data structures. Accessible only through SRH.get & SRH.set instructions when in Active Authority-CEM mode.

CEM_Buffer (256 bits) – *Authority-mode*

Intermediate register used by the CEM instructions so the SRH register can be set atomically and for reading and writing SP registers which are larger than the word-size of the general registers. Used for accessing newly created derived keys and for retrieving and atomically setting the SRH register. Accessible only through GR.set and GR.get instructions when in Active Authority CEM mode.

The size of the CEM_Buffer register is implementation specific, but has to be at least 256 bits wide.

Interrupt_Hash (128 bits) – *Authority-mode*

Hash of general purpose registers for suspended Authority-mode CEM thread. *(May be expanded to include the Initialization Vector (IV) for register encryption.)*

Interrupt_Address, PID, & CID (64 + 32 + 16 bits) – *Authority-mode*

Interrupt Address (64 bits), PID (Process ID) (32 bits), and CID (Compartment ID) (16 bits) which triggers return to Active CEM mode from Suspended CEM mode for Authority mode. The values are saved when an interrupt occurs during CEM, and the processor hardware checks the stored values every time a return from interrupt is detected, triggering return to CEM on a match..

The PID is used to distinguish OS contexts (i.e. processes), and the CID is used to distinguish Hypervisor/Kernel compartments (i.e. partitions or virtual machines). The PID & CID are assumed to be available in the base architecture and are saved with the Interrupt Address by SP to prevent accidental return to CEM. If the OS or Hypervisor does not use the base architecture's PID and CID registers, or if they are used improperly, a false match using only the virtual address may trigger a return from CEM which will then fail integrity checking.

The PID & CID are also assumed to be used by the base architecture to tag cache lines and reduce unnecessary flushing of the cache for isolation of processes and compartments. Without the tags, the cache needs to be flushed on every process context switch or compartment switch, whereas the tags allow cache values from a different context to remain in cache but be inaccessible from other contexts.

3 SP Instructions

Summary of SP Instructions

	Instr. Class	Mnemonic	Operation	Restrictions	Explanation
Authority-mode CEM	Initialize	drk.set.*sel* Rs1, Rs2	DRK[sel] ← Rs1||Rs2	drk_lock == 0	Sets the selected word of the DRK register.
		drk.lock	DRK_Lock = 1	*none*	Locks the DRK register. drk.set can no longer be used until the next reboot.
	Master Root Secrets / CEM Register Access	drk.derive Rs1, Rs2	CEM_Buffer ← Derive(DRK, Rs1||Rs2)	CEM_Mode == 01 (Active Auth CEM)	Derives a new key from the DRK, using the contents of Rs1 and Rs2 as a nonce. Result stored in the lower 128-bits of the CEM Buffer register.
		srh.get	CEM_Buffer ← SRH	CEM_Mode == 01	Copies the SRH register into the CEM Buffer register.
		srh.set	SRH ← CEM_Buffer	CEM_Mode == 01	Atomically copies the CEM_Buffer into the SRH register.
		gr.get.*sel* Rs1, Rs2	CEM_Buffer[sel] ← Rs1||Rs2	CEM_Mode == 01	Retrieves two words from general registers into a selected region of the CEM Buffer.
		gr.set.*sel* Rd	Rd ← CEM_Buffer[sel]	CEM_Mode == 01	Sets a general register with the selected word of the CEM Buffer register.
	CEM	begin_cem.a	CEM_Mode = 01	CEM_Mode == 00 (Normal)	Enter active Authority CEM mode for next instruction. CIC checking using DRK begins.
		* end_cem	CEM_Mode = 00	CEM_Mode == 01	Exits Active CEM mode and returns to Normal.
	Secure Memory	* secure_load Rd, Rs, imm	Rd ← Mem[Rs + imm]	CEM_Mode == 01	Secure load from memory. Reads from a cache line with "secure data" flag set, or from main memory, decrypting and MAC verification with the appropriate device key (DRK).
		* secure_store Rd, Rs. imm	Rd → Mem[Rs + imm]	CEM_Mode == 01	Secure store to memory. Writes to a cache line with the "secure data" flag set. Eviction from cache will cause

					encryption and MAC generation using the appropriate device key (DRK).

[*] Authority-mode SP instruction also used for User-mode SP

We assume a base instruction set with a RISC architecture, such as MIPS. Slight modifications might be necessary for implementation on other architectures, especially in regards to memory access semantics.

We assume a general register file with 32 general purpose registers of 64-bit word size. Instructions with Rd, Rs parameters can use any of the GPRs, numbered R0-R31. R0 is hard wired to binary zero. We assume two read ports and one write port on the register file.

The CEM_Buffer register is an implementation-dependent storage location that is at least 256 bits wide. The term "X.get" in the instruction mnemonics refers to copying X to the CEM_Buffer, where X = gr or srh. The term "X.set" in the instruction mnemonics refers to copying from the CEM_Buffer to X, where X = gr or srh.

The subop "sel" in the instruction mnemonics refers to the selected word in the CEM_Buffer which is read, or the selected two words in the CEM_Buffer (or DRK) which are written. When single words are selected, as in gr.set.sel, sel = 0, 1, 2 or 3 for 64-bit words, and sel = 0, 1,…, or 7 for 32-bit words. When two words are selected, as in gr.get.sel and drk.set.sel, the even indicies are used. Hence, sel = 0 or 2 for 64-bit words, and sel = 0, 2, 4 or 6 for 32-bit words.

<u>Instruction Details</u>

sample

Sample Instruction Name

Format:	mnemonic Rd,Rs1,Rs2
Description:	Each instruction detail page describes the semantics and usage of each SP instruction. The Format section shows the mnemonic, subops, and arguments for the instruction. Instructions can produce one register result, written to Rd, the destination register. Instructions may take parameters Rs1 and Rs2, the source registers. Rd, Rs1, and Rs2 can be chosen independently from any of the general registers R0…R31, where R0 is hard-wired to binary zero. Memory instructions may also take an *immediate* argument.
Exceptions:	*Sample Exception* – If the instruction can generate any exceptions, they are listed here.

DRK Set

Format: drk.set.*sel* Rs1,Rs2

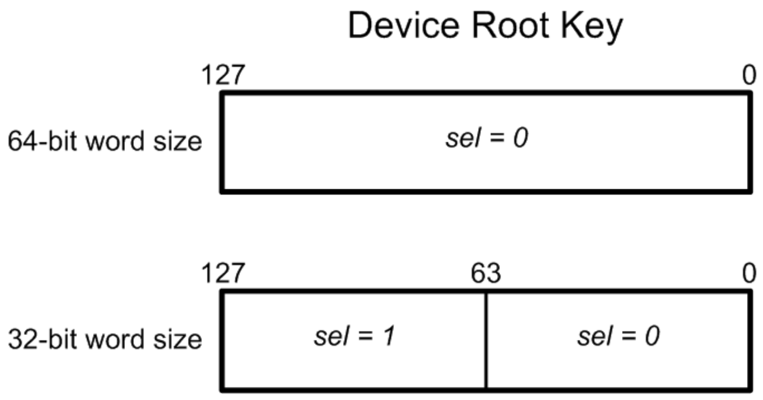
Description: Sets the selected portion of the DRK register with the concatenation of Rs1 and Rs2. Requires DRK_Lock == 0.

sel = 0 for 64-bit registers, since all 128-bits of the drk register can be written in one instruction.

sel = 0 or 2 for 32-bit registers, to indicate whether the right (sel = 0) or left (sel = 2) half of the drk is written to.

Note: DRK_Lock is automatically reset to 0 on each reboot or power cycle. Typically the Secure BIOS, regular BIOS or OS/Hypervisor kernel will allow DRK initialization and then lock the register before loading other software.

Exceptions: *SP Exception (Initialization)* – Raised if DRK_Lock == 1.



DRK Lock

Format: drk.lock

Description: Sets DRK_Lock = 1, and prevents using the drk.set instruction until the next reboot.

Note: The DRK_Lock register is automatically reset to 0 on each reboot or power cycle and cannot be cleared to 0 by any instruction.

On boot, the Secure BIOS will either:

a. Perform device initialization, generating a new DRK and writing it with the drk.set instruction, followed by the drk.lock instruction, before proceeding to normal boot. Typically this is only performed in a trusted depot by the authority.
b. Skip device initialization, execute the drk.lock instruction and proceed to a normal boot.

The normal boot proceeds by continuing with the regular system BIOS, which also executes drk.lock as an added precaution. Normal software will therefore not have an opportunity to overwrite the DRK register. Access to the Secure BIOS is required in order to attack the DRK by overwriting it.

Exceptions: None

DRK – Derive Key

Format: drk.derive Rs1,Rs2

Description: Generates a new key that is derived from the DRK and the 128-bit nonce formed by concatenating Rs1 and Rs2. The 128-bit result is stored in the rightmost (low-order) 128-bits of the CEM Buffer, with the remaining 128-bits set to zero.

The derived key is generated by the hardware hashing engine which reads the DRK without revealing its contents. A cryptographic MAC is computed over the nonce, keyed with the DRK. Repeated generation of derived keys is guaranteed to always give the same result when used with the same DRK and same nonce. The MAC algorithm used is implementation specific.

Note: See Appendix A for the operation of drk.derive for processors with 32-bit word size.

Exceptions: *CEM Exception (Access)* – Raised if not in an active authority-mode CEM: CEM_Mode != 01 (Active CEM).

srh.get

SRH Get

Format:	srh.get
Description:	Copy the entire SRH register into the rightmost (low-order) 256 bits of the CEM_Buffer register.
Exceptions:	*CEM Exception (Access)* – Raised if not in an active authority-mode CEM: CEM_Mode != 01 (Active CEM).

SRH Set

Format:	srh.set
Description:	Stores the contents of the rightmost (low-order) 256 bits of the CEM_Buffer register into the SRH register. The entire SRH register is set atomically, in a single cycle.
Exceptions:	*CEM Exception (Access)* – Raised if not in an active authority-mode CEM: CEM_Mode != 01 (Active CEM).

gr.get.*sel*

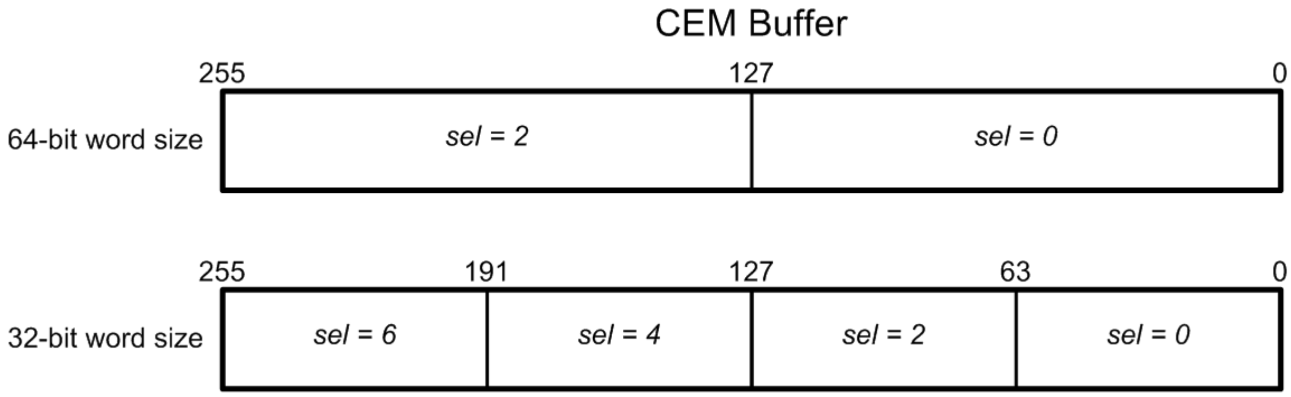
GR Get to CEM Buffer

Format: gr.get.*sel* Rs1, Rs2

Description: Retrieves Rs1 and Rs2 from general registers and stores the concatenation of the registers into the selected 128-bit region, *sel*, of the CEM_Buffer register.

sel = 0 or 2 for 64-bit words, or sel = 0, 2, 4 or 6 for 32-bit words, where 0 indicates the rightmost (low-order) region.

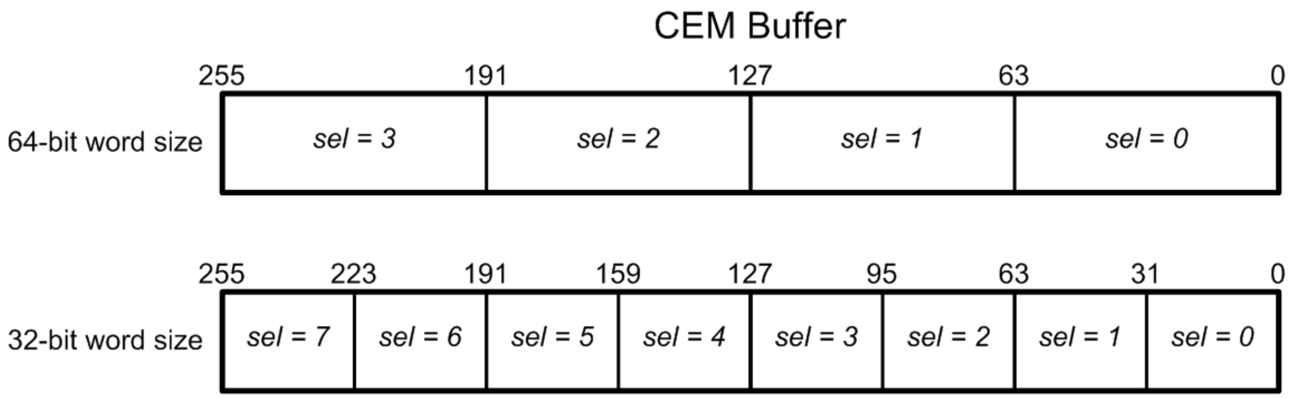
Exceptions: *CEM Exception (Access)* – Raised if not in an active authority-mode CEM: CEM_Mode != 01 (Active CEM).



gr.set.*sel*

GR Set from CEM Buffer

Format:	gr.set.*sel* Rd
Description:	Sets a general register, Rd, from the selected word, *sel*, of the CEM_Buffer register. sel = 0, 1, 2, or 3 for 64-bit words, or sel = 0, 1, …, or 7 for 32-bit words, where 0 indicates the rightmost (low-order) word.
Exceptions:	*CEM Exception (Access)* – Raised if not in an active authority-mode CEM: CEM_Mode != 01 (Active CEM).



Begin CEM – Authority-mode

Format:	begin_cem.a
Description:	Sets CEM_Mode = 01, entering Authority-mode CEM for the next instruction. The subsequent instructions will be fetched with CIC checking using the DRK. Note: When CEM_Mode = 01, if an instruction fetch triggers a cache miss, requiring bringing in a new instruction cache line from off-chip memory, code integrity checking (CIC) does MAC-verification using the DRK as the key. If MAC verification passes, the instruction cache line is loaded into the cache and tagged as "Secure Instruction"; otherwise it is not loaded into the cache and a CEM Exception *(Code Integrity)* is rasied. If a cache hit occurs for the instruction fetch, the "Secure Instruction" tags is required. If these tags are not set, the fetch is treated as a cache miss.
Exceptions:	*CEM Exception (Busy)* – Raised if CEM_Mode != 00 (Normal). Authority-mode CEM cannot be started if another CEM thread is active or if another authority-CEM thread is suspended.

End CEM

Format:	end_cem
Description:	Exits the active CEM thread. Sets CEM_Mode = 00 (Normal). The subsequent instructions will be fetched without CIC checking.
Exceptions:	*CEM Exception (Access)* – Raised if not in an Active CEM mode: CEM_Mode != 01 (Active CEM).

Secure Load

Format: secure_load Rd,Rs,imm

Description: Secure load from memory using displacement mode, where the effective address is calculated as Rs+immediate. Fetches the word from memory into register Rd. Must be in Active CEM mode (CEM_Mode == 01).

Loading from secure memory is dependent on the state of the cache tags:

Hardware Hit/Miss	**CEM Cache Tags**	**Dirty**	**Process**
Hit	“Secure Data”	X	Previously verified cache line. Requested word fetched from cache into the specified general register.
“Hit”	!“Secure Data”	0	Treat as cache miss.
“Hit”	!“Secure Data”	1	Evict dirty cache line and then treat as a cache miss.
Miss	N/A	N/A	Encrypted cache line and corresponding MAC fetched from off-chip memory. MAC verification performed. If verification passes, “Secure Data” tag is set, the line is decrypted and stored in cache, and requested word fetched to the general register. If verification fails, CEM Exception (Data Integrity) is raised, the line is not stored in cache, and no data is written to the register.

Note: When a cache-line with “Secure Data” tag set is later evicted from the cache, it will first be encrypted and a MAC calculated, both using the DRK, before being written to off-chip memory.

Note: If a cache-line with “Secure Data” tag set is later accessed using regular rather than secure load/store instructions (or for a secure or regular instruction fetch), it will first be evicted and then re-fetched into the cache in encrypted state.

Note: Memory accesses must be word-aligned (otherwise Unaligned Address Exception occurs).

Exceptions: *CEM Exception (Access)* – Raised if not in Active CEM mode: CEM_Mode != 01 (Active CEM).

CEM Exception (Data Integrity) – Raised if MAC verification fails when fetching a secure data cache line from off-chip memory. If requested line is in cache, but the required “Secure Data” cache tag is not set, the line will first be evicted (if necessary) and then re-fetched from memory to perform MAC check.

Secure Store

Format: secure_store Rd,Rs,imm

Description: Secure store to memory using displacement mode, where the effective address is calculated as Rs+immediate. Stores the word from register Rd into memory. Must be in Active CEM mode (CEM_Mode == 01).

Storing to secure memory is dependent on the state of the cache tags:

Hardware Hit/Miss	**CEM Cache Tags**	**Dirty**	**Process**
Hit	"Secure Data"	X	Previously verified cache line. Requested word written into cache from the specified general register.
"Hit"	!"Secure Data"	0	Treat as cache miss.
"Hit"	!"Secure Data"	1	Evict dirty cache line and then treat as a cache miss.
Miss	N/A	N/A	Encrypted cache line and corresponding MAC fetched from off-chip memory. MAC verification performed. If verification passes, "Secure Data" tag is set, the line is decrypted and stored in cache, and requested word is written to cache from the general register. If verification fails, CEM Exception *(Data Integrity)* is raised, the line is not stored in cache, and the word is not written.

Note: When a cache-line with the "Secure Data" tag set is later evicted from the cache, it will first be encrypted and a MAC calculated, both using the DRK, before being written to off-chip memory.

Note: If a cache-line with the "Secure Data" tag set is later accessed using regular rather than secure load/store instructions (or for a secure or regular instruction fetch), it will first be evicted and then re-fetched into the cache in encrypted state.

Note: Memory accesses must be word-aligned (otherwise Unaligned Address Exception occurs).

Exceptions: *CEM Exception (Access)* – Raised if not in Active CEM mode: CEM_Mode != 01 (Active CEM).

CEM Exception (Data Integrity) – Raised if MAC verification fails when fetching a secure data cache line from off-chip memory. If requested line is in cache, but the required “Secure Data” cache tag is not set, the line will first be evicted (if necessary) and then re-fetched from memory to perform MAC check.

4 Modifications to Base ISA

The following instructions and operations have modified semantics from the base ISA to support SP and CEM.

Summary of Modified Instructions and Operations

	Component	Mnemonic/ Name	Operation	Restrictions	Explanation
Modifications to Base ISA	Execution	Instruction Fetch	Execution pipeline ← Mem[Program Counter]	*none*	The next instruction to be executed is fetched into the processor. If in Active CEM mode, the "secure instruction" tag is required. In all other CEM modes, no "secure" tag is permitted. If a cache-line does not have the correct tags, it must first be evicted and then refetched.
	Normal Memory Access	load Rd, Rs, imm	Rd ← Mem[Rs + imm]	*none*	Normal (non-secure) load from memory. Reads unprotected data from a cache line which has no CEM cache tags set. If the cache line has a "secure" tag set, the line will first be evicted and then refetched in encrypted state without the "secure" tag.
		store Rd, Rs. imm	Rd → Mem[Rs + imm]	*none*	Normal (non-secure) store to memory. Writes unprotected data to a cache line without the CEM cache tags set. If the cache line has a "secure" tag set, the line will first be evicted and then refetched in encrypted state without the "secure" tag.
	Interrupt Handling	Interrupt	CEM_Mode == 01 (Active CEM) → CEM_Mode == 10 (Suspended CEM)	*none*	If an interrupt occurs during Active CEM mode, secure the register state (encrypt registers with DRK, save Interrupt Hash and Interrupt Address) and enter Suspended CEM mode.

		Return From Interrupt (RFI)	CEM_Mode == 10 (Suspended CEM) → CEM_Mode == 01 (Active CEM)	*none*	If in Suspended CEM mode when returning from an interrupt and the saved Interrupt Address matches the current return address, verify and restore the register state and resume Active CEM mode.

Details of Modifications to Base ISA
ifetch

Instruction fetch

Format: *N/A – implicit operation*

Description: Instruction fetch occurs as part of the execution pipeline as the next instruction is retrieved from memory at the address pointed to by the Program Counter. Instructions are fetched from L1-instruction cache, possibly resulting in misses to L2 cache or main memory.

Fetching instruction from memory is dependent on the CEM mode and the state of the cache tags:

CEM_Mode	**HW Hit/ Miss**	**CEM Cache Tags**	**Process**
00 (Normal) *or* 10 (Suspended CEM)	Hit	*none*	Normal execution – no Code Integrity Checking (CIC).
	“Hit”	“Secure Instruction” or “Secure Data”	The cache line will first be evicted (if dirty) and refetched as if a miss.
	Miss	N/A	Instructions are fetched without Code Integrity Checking, and no CEM cache tags are set.
01 (Active CEM)	Hit	“Secure Instruction”	Normal CEM execution – with Code Integrity Checking.
	“Hit”	*none* or “Secure Data”	The cache line will first be evicted (if dirty) and refetched as if a miss.
	Miss	N/A	Cache line is fetched from off-chip memory with Code Integrity Checking, performing MAC-verification using the DRK. If MAC verification passes, the line is loaded into the cache, tagged as “Secure Instruction”, and the requested instruction is executed; otherwise the line is not loaded into the cache and a CEM Exception *(Code Integrity)* is raised.
11 (*Reserved*)	X	X	N/A

Note: Memory accesses must be word-aligned (otherwise Unaligned Address Exception occurs).

Exceptions: *CEM Exception (Code Integrity)* – Raised if MAC check fails when fetching a secure instruction cache line from memory.

Normal Load

Format: load Rd,Rs,imm

Description: Normal load from memory using displacement mode, where the effective address is calculated as Rs+immediate. Fetches the word from memory into register Rd. Available in all CEM modes.

Loading from normal memory is dependent on the state of the cache tags:

Hardware Hit/Miss	**CEM Cache Tags**	**Process**
Hit	*none*	Cache line is unprotected. The requested word is fetched from cache into the specified general register.
“Hit”	“Secure Data”	Data cache line protected by CEM. The cache line must first be evicted, causing it to be encrypted and have a MAC calculated, both using the DRK, before being written to off-chip memory. It is then refetched into cache in encrypted state as a miss.
“Hit”	“Secure Instruction”	Verified and tagged CEM instructions must be evicted from cache and then treated as a miss.
Miss	N/A	The cache line is fetched from memory without protection or verification, and no CEM cache tags are set. The requested word is then fetched from cache into the specified general register.

Note: Memory accesses must be word-aligned (otherwise Unaligned Address Exception occurs).

Exceptions: None

Normal Store

Format: store Rd,Rs,imm

Description: Normal store to memory using displacement mode, where the effective address is calculated as Rs+immediate. Stores the word from register Rd into memory. Available in all CEM modes.

Storing to normal memory is dependent on the state of the cache tags:

Hardware Hit/Miss	**CEM Cache Tags**	**Process**
Hit	*none*	Cache line is unprotected. Requested word written into cache from the specified general register.
“Hit”	“Secure Data”	Data cache line protected by CEM. The cache line must first be evicted, causing it to be encrypted and have a MAC calculated, both using the DRK, before being written to off-chip memory. It is then refetched into cache in encrypted state as a miss.
“Hit”	“Secure Instruction”	Verified and tagged CEM instructions must be evicted from cache and then treated as a miss.
Miss	N/A	The cache line is fetched from memory without protection or verification, and no CEM cache tags are set. The requested word is then written into cache from the specified general register.

Note: Memory accesses must be word-aligned (otherwise Unaligned Address Exception occurs).

Exceptions: None

Interrupt

Format: *N/A – implicit operation due to software trap, exception, or hardware fault*

Description: The actions taken when an interrupt occurs are dependent on the CEM mode:

CEM_Mode	**Interrupt handling**
00 (Normal) *or* 10 (Suspended CEM)	Normal interrupts handling. Control of the execution is transferred to the corresponding interrupt handling routine in the supervisor or kernel.
01 (Active CEM)	CEM interrupt protection is triggered before execution is transferred to the supervisor or kernel. Registers are encrypted in place, and a hash is taken and stored in the Interrupt Hash register on-chip. The return address of the program counter is stored in the Interrupt Address register (along with the current PID & CID), and the CEM mode is changed from Active CEM to Suspended CEM. See Section 6.1.3 for details.
11 (*Reserved*)	N/A

Exceptions: None

Return From Interrupt

Format: *N/A – dependent on base ISA – may be explicit return from interrupt (rfi) instruction, or may be implicit on a regular return or jump instruction.*

Description: The actions taken when a return from interrupt occurs are dependent on the CEM mode:

CEM_Mode	Interrupt handling
00 (Normal)	Normal return from interrupt. Control of the execution is transferred to the corresponding return location specified by the supervisor or kernel.
01 (Active CEM)	N/A. Interrupt handling does not take place in Active CEM mode. Any returns or jumps while in Active CEM must be to other verified code, tagged as "Secure Instruction", and CEM mode remains unchanged.
10 (Suspended CEM)	Upon any Return From Interrupt (RFI), the return address (and current PID & CID) is checked against the suspended CEM state. A match triggers the reverse process to verify the Interrupt Hash and then restore (decrypt) the register state. An RFI that does not match the Interrupt Address is presumably not returning to the suspended CEM thread and will continue silently, remaining in Suspended CEM mode. If an RFI matches, but the Interrupt Hash does not verify correctly, a CEM Exception *(Register Integrity)* is raised.
11 (*Reserved*)	N/A

Exceptions: *CEM Exception (Registry Integrity)* – Raised if hash check fails for general registers (when computed hash does not match the contents of the Interrupt Hash register), when resuming Active CEM mode from Suspended CEM mode.

5 SP Exceptions/Faults

Table 1: SP Exceptions/Faults

Name	Val	Description
SP Exception (Initialization)	1	Raised if attempting to execute drk.set when the DRK_Lock register is set to 1.
CEM Exception (Access)	2	Raised if a CEM-only instruction is executed while not in Active CEM mode.
CEM Exception (Busy)	3	Raised if trying to enter authority-mode CEM when another CEM thread is already active or when another CEM thread is suspended.
CEM Exception (Code Integrity)	4	Raised if MAC check fails when fetching a secure instruction cache line from memory.
CEM Exception (Data Integrity)	5	Raised if MAC check fails when fetching a secure data cache line from memory.
CEM Exception (Register Integrity)	6	Raised if hash check fails for general registers (when computed hash does not match the contents of the Interrupt Hash register), when resuming Active CEM mode from Suspended CEM mode.
SP Exception (Not Implemented)	7	Raised when trying to access a feature not implemented on the device. (May also be used for user-mode SP instructions on a device where only authority-mode is implemented.)
SP Exception (Virtualization)	8	Raised for errors during CEM Save/Restore virtualization instructions. It can be caused by a protection ring access control violation, an improper CEM state during the call (e.g. trying to save or restore while in Active CEM), or by restoring an invalid state. (*Note: CEM Save/Restore not yet defined for authority mode*)

6 Operation

6.1 *CEM (Concealed Execution Mode) for TSM Protection*

The SP Hardware directly protects temporary data handled by the TSM code. This is called Concealed Execution Mode (CEM). Code Integrity Checking is also performed for all TSM instruction cache lines.

Below, we describe the CEM protection mechanisms.

6.1.1 Code Integrity Checking (CIC)

Authority-mode CEM is triggered by the corresponding Begin_CEM instruction, entering Active CEM mode for the next instruction. During CEM execution, Code Integrity Checking is used, verifying the hash of each TSM instruction before execution. A keyed-hash (MAC) is pre-computed and stored with each cache line of TSM code, either in-line with the code (as shown in Figure 3) or in a separate area of memory. The hash is keyed with the DRK for authority mode. The hash is computed over the instructions of the cache line and the virtual address of the start of the line. As each cache line of TSM code is loaded, the hash is recomputed by the on-chip hashing engine and compared to the stored value. Upon verification, the line is tagged as “Secure Instruction” in cache. Subsequent fetches from cache with the “Secure Instruction” tag set need not be verified again. If the cache line’s MAC does not verify correctly upon loading from memory, a CEM Exception *(Code Integrity)* is raised. If the cache line is in cache without the correct “Secure Instruction” tag, it is first evicted and then reloaded for MAC verification.

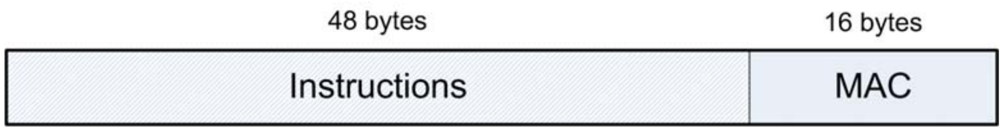


Figure 3: Sample cache line with embedded MAC for CIC checking.

6.1.2 CEM Secure Memory

Data in memory is protected explicitly by the TSM using Secure_Load and Secure_Store instructions to access sensitive data. Similarly to Code Integrity Checking, secure data lines are protected in cache by a “Secure Data” tag when written with a Secure_Store instruction. When a tagged cache line is evicted, it is first encrypted before writing to memory, and a MAC is computed with the DRK and stored separately in memory (in a predetermined location). A Secure_Load that causes a cache line to be read in from memory triggers integrity checking of the MAC and then decryption, setting the “Secure Data” tag on the new cache line. Any regular load/store operations on a “Secure Data” cache line will cause the line to be evicted and then re-read from memory as ciphertext that is does not have the “Secure Data” tag set in cache.

6.1.3 CEM Interrupt Handling

During Active CEM mode, any interrupt, trap or exception will trigger CEM interrupt protection before execution is transferred to the supervisor or kernel. Registers are encrypted as a single block using the on-chip encryption & hashing engine; the resulting ciphertext is divided and placed back into the registers. A hash is then taken of the result and stored in the Interrupt Hash register on-chip. The return address of the program counter is stored in the Interrupt Address register along with the current PID & CID, and the CEM mode is changed from Active CEM to Suspended CEM. Upon each subsequent Return From Interrupt (RFI) instruction, the return address (and current PID & CID) is checked against the suspended CEM state. A match triggers the reverse process to verify the Interrupt Hash and then restore (decrypt) the register state. An RFI that does not match the Interrupt Address is presumably not returning to the suspended CEM thread and will continue silently, remaining in Suspended CEM mode. If an RFI matches, but the Interrupt Hash does not verify correctly, a CEM Exception *(Register Integrity)* is raised.

See [Lee05] and [Dwoskin07] for more details.

6.1.4 Cache-tag Details

All cache lines store tags to indicate CEM protection (secure/normal) and CEM instruction/data (implied in L1 split cache). The combined tags indicate either "Secure Instruction", "Secure Data", or "Normal" The instruction/data bits are currently used only for secure CEM cache lines and are ignored when the "secure" tag == normal (0). *(Note: However, we leave open the possibility that these bits could be used in the future to distinguish instructions and data even for non-secure lines, to implement addition protection outside of CEM.)*

If one or both caches are physically tagged, an additional virtual-address tag is added. This virtual tag is from the virtual address that was used to correctly verify the cache line's MAC as it was read in. When reading/writing a "secure"-tagged cache line from a physically-tagged cache for CEM instructions or CEM data, the virtual tag must also match. Otherwise the line is first evicted and then reloaded from main memory as if the line were not in cache. (This prevents certain splicing attacks on CEM code and data.) When evicting dirty cache lines of CEM data, the virtual tag is also used for MAC computation.

The virtual tags in each cache may also contain the Process ID, ensuring that each process can only access its own tagged data in cache. Otherwise that entire cache must be flushed by the OS on each context switch. ** This is not an SP/CEM security requirement, but is necessary for proper isolation by the OS. **

<u>L1 instruction cache lines</u>
"secure" tag (1 bit): 1 = secure instruction, 0 = normal instruction

<u>L1 data cache lines</u>
"secure" tag (1 bit): 1 = secure data, 0 = normal data

<u>L2 shared cache lines</u>
"secure" tag (1 bit): 1 = secure, 0 = normal
instruction/data tag (1 bit): 1 = instruction, 0 = data

6.2 SP Module

Figure 4 shows the datapath for the new SP registers and the Encryption/Hashing Engine in an SP module. Code Integrity Checking (secure instruction fetch) uses the Encryption/Hashing Engine and the DRK to process instructions as they are fetched from off-chip memory into on-chip caches. Similarly the engine processes secure data lines for secure load/store as they move between the caches and memory. On interrupts, the engine performs encryption/decryption and integrity checking of the entire register file, saving the Interrupt Hash, Interrupt Address, PID, and CID in the corresponding SP registers when the interrupt occurs and checking them for return to CEM.

Other SP registers are used explicitly for new SP instructions. General registers are used to access the CEM Buffer and to set the DRK. They are also the source for the nonce in generating derived keys, which are then stored in the lower half of the CEM Buffer. The SRH is accessed by copying to or from the CEM Buffer in one operation. The DRK Lock is set directly by an instruction, and the CEM Mode is set by hardware in response to Begin/End CEM instructions, interrupts, or returns from interrupt.

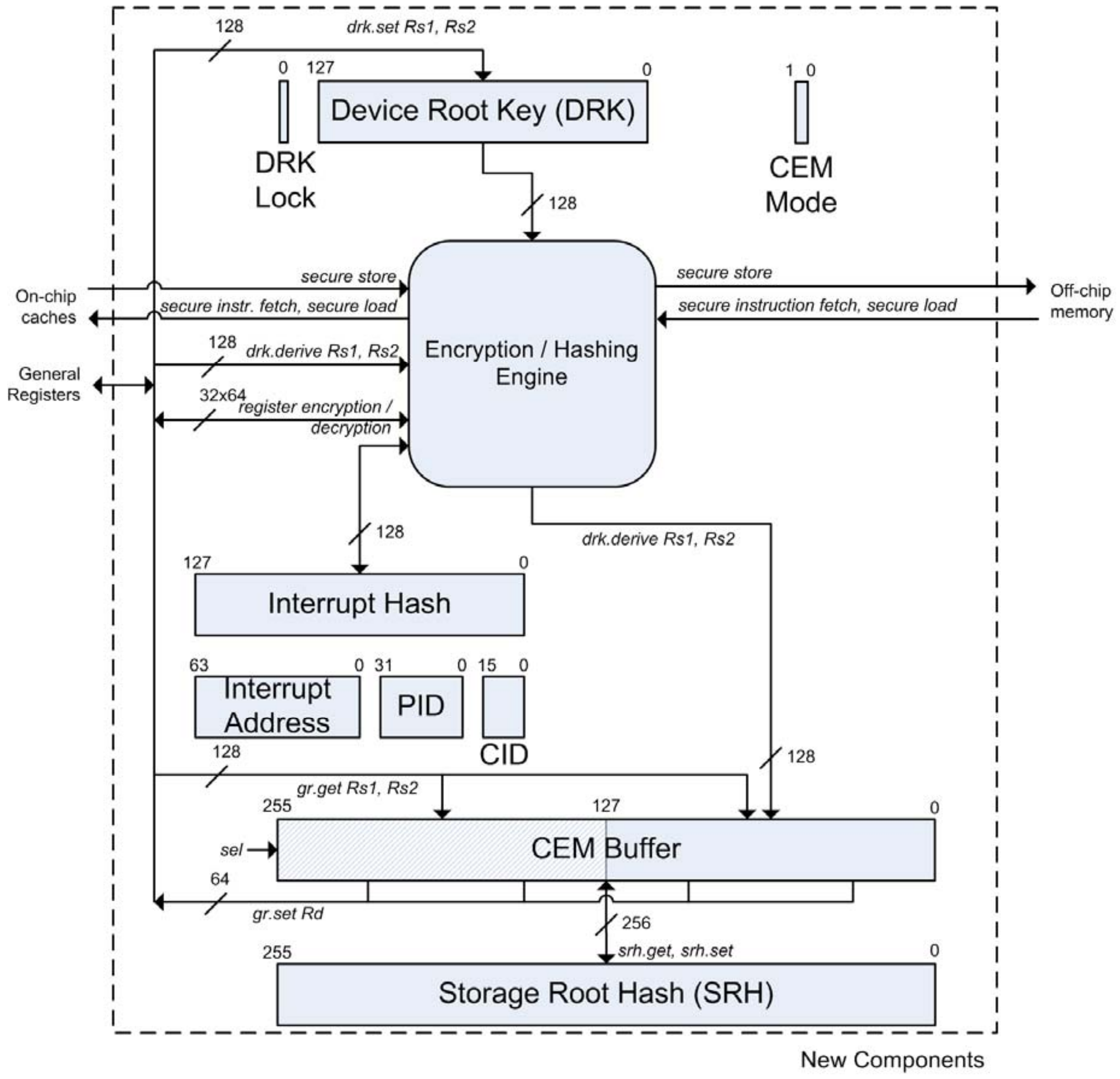


Figure 4: Datapath for new Authority-mode SP module

6.3 Hierarchical Ring Protection

This is not currently part of SP, but is assumed to be present in the base architecture. For x86, we assume the presence of rings to provide at least user- and supervisor- modes, as well as VT support for negative rings to protect the TML.

7 References

[Lee05] Ruby B. Lee, Peter C. S. Kwan, John Patrick McGregor, Jeffrey Dwoskin, and Zhenghong Wang, Architecture for Protecting Critical Secrets in Microprocessors, *Proceedings of the 32nd International Symposium on Computer Architecture*, Madison, Wisconsin, June 2005.

[Dwoskin07] Jeffrey Dwoskin and Ruby Lee, "Hardware-rooted Trust for Secure Key Management and Transient Trust," to appear at ACM CCS 2007

[SecureCore] Ganesha Bhaskara, Timothy E. Levin, Thuy D. Nguyen, Cynthia E. Irvine, Terry V. Benzel, Jeffrey Dwoskin, Ruby Lee, *Virtualization and Integration of SP Services in SecureCore*, University of California, Information Sciences Institute Technical Report ISI-TR-623, September 2006

Appendix A – 32-bit Word-size

Word size is implementation-specific and affects the semantics of some CEM instructions. Software can determine the word size from the processor model for compatibility.

With 32-bit word-size, the follow changes are required:

- Setting the DRK requires selecting either the upper or lower 64-bit (2-word) portion of the register using *sel* == 0 or 1. (See Section 3.)
- Accessing the CEM Buffer register will take place with 32-bit words: 32-bits for gr.set using *sel* == 0, 1, 2, …, or 7 and 64-bits for gr.get using *sel* == 0, 2, 4, or 6. (See Section 3.)
- Creating derived keys will take the 128-bit nonce from the CEM Buffer register rather than from two register parameters. (See modified instruction details below.)

DRK – Derive Key *(32-bit word-size)*

Format:	drk.derive
Description:	Generates a new key that is derived from the DRK and a 128-bit nonce located in the rightmost (low-order) 128-bits of the CEM Buffer. The 128-bit result is stored back into the rightmost (low-order) 128-bits of the CEM Buffer, overwriting the nonce, with the remaining 128-bits set to zero. To emulate the behavior of the 64-bit version of drk.derive, first gr.get is called twice to copy the nonce from general registers into the rightmost 4 words of the CEM Buffer. Then drk.derive is called (without parameters) to generate the derived key. The derived key is generated by the hardware hashing engine which reads the DRK without revealing its contents. A cryptographic MAC is computed over the nonce, keyed with the DRK. Repeated generation of derived keys is guaranteed to always give the same result when used with the same DRK and same nonce. The MAC algorithm used is implementation specific. Note: See Section 3 for the operation of drk.derive for processors with 64-bit word size.
Exceptions:	*CEM Exception (Access)* – Raised if not in an active authority-mode CEM: CEM_Mode != 01 (Active CEM).