



Intel® Trust Domain CPU Architectural Extensions

343754-003US
AUGUST 2026

Notices & Disclaimers

Intel Corporation ("Intel") provides these materials as-is, with no express or implied warranties.

All products, dates, and figures specified are preliminary, based on current expectations, and are subject to change without notice. Intel does not guarantee the availability of these interfaces in any future product. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

The products described might contain design defects or errors known as errata, which might cause the product to deviate from published specifications. Current, characterized errata are available on request.

Intel technologies might require enabled hardware, software, or service activation. Some results have been estimated or simulated. Your costs and results might vary.

No product or component can be absolutely secure.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted that includes the subject matter disclosed herein.

No license (express, implied, by estoppel, or otherwise) to any intellectual-property rights is granted by this document.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice.

Copies of documents that have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting <http://www.intel.com/design/literature.htm>.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

Revision History

Revision	Description	Date
343754-001	Initial release.	September 2020
343754-002	Intel Persistent SEAMLDR module support and other updates.	May 2021
343754-003	Full rewrite, including updated list of SEAMOPS leaves and SEAM-only MSRs, and their related data structures and variables.	August 2026

CHAPTER 1

SECURE ARBITRATION MODE (SEAM)

1.1	Modes of Operation	1-1
1.1.1	SEAM Calls	1-2
1.1.2	SEAM Returns	1-2
1.1.3	SEAM Host-Guest Transitions	1-2

CHAPTER 2

SEAM AND TME-MK

CHAPTER 3

OPERATION IN SEAM

3.1	SEAM Root Operation	3-1
3.2	SEAM Non-Root Operation	3-1

CHAPTER 4

SEAM MEMORY

4.1	SEAM Range	4-1
4.2	Memory Protections	4-1
4.3	Memory Type	4-2

CHAPTER 5

SEAM INSTRUCTIONS

SEAMCALL—Enter SEAM Root Operation	5-2
SEAMRET—Exit SEAM Root Operation	5-4
TDCALL—Call SEAM Module	5-6
SEAMOPS—Invoke SEAM Operation	5-7

CHAPTER 6

SEAM DATA

6.1	Data Structures	6-1
6.2	Platform-Scope Variables	6-5
6.3	Core-Scope Variables	6-6
6.4	Constants	6-6

CHAPTER 7

SEAMOPS LEAF FUNCTIONS

Leaf 0 – CAPABILITIES	7-2
Leaf 1 – SEAMREPORT	7-3
Leaf 2 – SEAMDB_CLEAR	7-5
Leaf 3 – SEAMDB_INSERT	7-6
Leaf 4 – SEAMDB_GETREF	7-7
Leaf 5 – SEAMDB_REPORT	7-8
Leaf 6 – SEAMVERIFYREPORT	7-10
Leaf 7 – TDSTATE_ENUM	7-11
Leaf 8 – TDSTATE_LOAD	7-12
Leaf 9 – TDSTATE_SAVE_CLEAR	7-13
Leaf 10 – UPDATESVNMAC	7-15
Leaf 11 – SEAMGETKEY	7-16
Leaf 15 – FEATURES	7-18
Helper Functions	7-20

CHAPTER 8
SEAM MSRS

MSR 1400H: IA32_SEAMRR_BASE – SEAM Range Base	8-1
MSR 1401H: IA32_SEAMRR_MASK – SEAM Range Mask	8-1
MSR 1402H: MSR_SEAMEXTEND – Read and Write CR_SEAMEXTEND	8-1
MSR 98H: MSR_WBINVDP – Write-Back-Invalidate Partial	8-2
MSR 99H: MSR_WBNOINVDP– Write-Back-No-Invalidate Partial	8-3
MSR 9AH: MSR_INTR_PENDING	8-4
MSR 550H: MSR_SEAM_SAI_MODE	8-5

CHAPTER 9
SEAM ISA ENUMERATION

CHAPTER 10
DEBUG AND MONITORING

10.1	Probe Mode	10-1
10.2	Core Telemetries	10-1
10.3	Machine Checks	10-1
10.4	Unbreakable SEAM Shutdown	10-2

APPENDIX A
TRUST DOMAIN EXTENSIONS

A.1	TDX Modules	A-1
A.2	TDX Configuration and Enabling	A-1
A.3	TDX SVNs	A-1

APPENDIX B
GLOSSARY

TABLES

		PAGE
6-1	SEAMEXTEND Data Structure	6-1
6-2	SEAMREPORT Data Structure	6-2
6-3	REPORTMACSTRUCT Data Structure.....	6-2
6-4	REPORTTYPE Data Structure.....	6-2
6-5	TEE_TCB_INFO Data Structure	6-3
6-6	TEE_TCB_SVN Data Structure.....	6-3
6-7	SEAMDB_RECORD Data Structure	6-4
6-8	TDSTATE Data Structure.....	6-4
6-9	SEAMKEYREQUEST Data Structure	6-4
6-10	KEYCONTEXT Data Structure.....	6-5
6-11	SEAMOPS Error Codes	6-7
6-12	Configuration Constants	6-7
6-13	Configuration Constants	6-7
7-1	SEAMOPS Leaf Functions	7-1
7-2	SEAM ISA Features Bitmap.....	7-18
8-1	MSR_SEAMEXTEND Bit Definitions	8-1
8-2	MSR_WBINVDP Bit Definitions.....	8-2
8-3	MSR_WBNOINVDP Bit Definitions.....	8-3
8-4	MSR_INTR_PENDING Bit Definitions	8-4
8-5	MSR_SEAM_SAI Bit Definitions	8-5

FIGURES

	PAGE
Figure 1-1. SEAM Modes of Operation.....	1-2

CHAPTER 1

SECURE ARBITRATION MODE (SEAM)

This document specifies secure arbitration mode (SEAM), an extension of the virtual-machine extensions (VMX) that allow running secure virtual machines.

The SEAM architecture is a foundation for Intel's Trusted Domains Extensions (TDX) architecture. Appendix A provides details about some aspects of the TDX architecture.

1.1 MODES OF OPERATION

The SEAM architecture defines a protected mode of operation called SEAM, which is an extension of VMX operation. Like VMX operation, SEAM includes root and non-root operations:

- **SEAM root operation** is the SEAM extension of VMX root operation. The software running in SEAM root operation is called a **SEAM module**. There are two sub-modes to SEAM root operation:
 - **Loader mode**: At most one logical processor can execute in this sub-mode at any time.
 - **Module mode**: Multiple logical processors can execute in this sub-mode in parallel.
- **SEAM non-root operation** is the SEAM extension of VMX non-root operation. The software running in this SEAM non-root operation is called **SEAM guest**. Multiple logical processors can execute in this mode concurrently.

NOTE

In the TDX architecture, software running in module-mode SEAM root operation is called a **TDX module** (sometimes SEAM module), while software running in loader-mode SEAM root operation is called a **Persistent SEAM Loader (P-SEAMLDLDR)**. The goal of the TDX module is to support SEAM guests called **Trust Domains (TDs)**. The goal of the P-SEAMLDLDR is to install and update TDX modules. The P-SEAMLDLDR itself is installed by launching an authenticated code module (ACM) called a **Non-Persistent SEAM Loader (NP-SEAMLDLDR)**.

Certain platform operations may be required to enable SEAM; until they are performed, SEAM is not enabled. Once SEAM is enabled, it may or may not be ready for execution in loader mode or in module mode (again, possibly based on certain platform operations). Details regarding this enabling are provided in Chapter 9, "SEAM ISA Enumeration."

The following diagram shows the VMX and SEAM software components, the modes in which they operate, and the transitions between these execution modes.

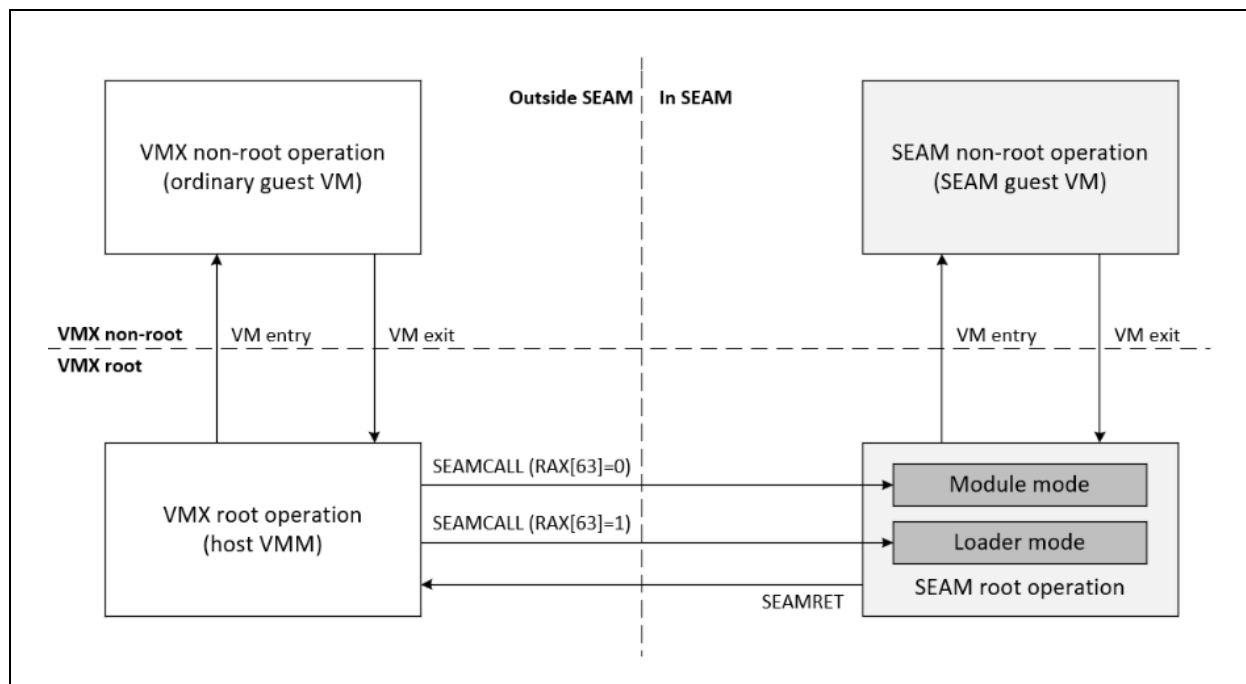


Figure 1-1. SEAM Modes of Operation

1.1.1 SEAM Calls

Processor transitions from VMX root operation to SEAM root operation are called **SEAM calls**. SEAM calls are initiated by executing the SEAMCALL instruction. A SEAM call operates in a manner similar to that of SMM VM exits in response to execution of the VMCALL instruction in VMX root operation (see Section “SMM VM Exits” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C). A SEAM call uses a **SEAM-transfer VMCS** that resides within the SEAM range of the physical-address space (see Section 4.1, “SEAM Range”).

SEAM calls use RAX[63] to determine the SEAM root operation sub-mode to be entered. When RAX[63] is 0, the transition is into module-mode SEAM root operation; otherwise, it is into loader-mode SEAM root operation. An attempt to enter loader-mode SEAM root operation fails if another logical processor is in SEAM having entered in loader mode.

1.1.2 SEAM Returns

Processor transitions from SEAM root operation back to VMX root operation are called **SEAM returns**. SEAM returns are initiated by executing the SEAMRET instruction. A SEAM return operates in a manner similar to that of VM entries that return from SMM (see Section “VM Entries that Return from SMM” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C). Like SEAM calls, SEAM returns use a SEAM-transfer VMCS.

A SEAM return from loader-mode SEAM root operation ensures that all data for any VMCS that the logical processor used in SEAM (including the SEAM-transfer VMCS) are in memory.

1.1.3 SEAM Host-Guest Transitions

Processor transitions between SEAM root operation and SEAM non-root operation are similar to processor transition between VMX root operation and VMX non-root operation. A VMCS used to transition between SEAM root and non-root operation, and to control SEAM non-root operation, is called **SEAM-guest VMCS**.

A SEAM-guest VMCS supports SEAM-specific fields and controls. In general, those fields and controls are ignored outside SEAM operation.

The SEAM-specific VMCS fields are the following:

- **SEAM-guest KeyID.** This is a VM-execution control field that contains the key identifier (KeyID) assigned for the key with which the SEAM guest's private pages are encrypted in memory.
- **SEAM shared EPT pointer.** This is a VM-execution control field that contains the physical address of the base of shared EPT PML4 table or PML5 table (depending on how EPT is configured).

The SEAM-specific VMX controls are the following:

- **SEAM guest-physical address width.** This is a VM-execution control that determines, for SEAM non-root operation, the width of guest-physical addresses translated by EPT and also the position of the SHARED bit in guest-physical addresses. See Chapter 4, "SEAM Memory."
- **Allow SEAM-guest telemetry.** This is a VM-entry control that allows collection of core telemetry in SEAM non-root operation.

Any processor that supports SEAM also supports the SEAM-specific VMCS fields and controls identified above. Support for the SEAM-specific VMX controls above is enumerated normally by the IA32_VMX_PROCBASED_CTLSS3 MSR (for "SEAM guest-physical address width") and by IA32_VMX_ENTRY_CTLSS MSR (for "allow SEAM-guest telemetry").

SEAM guests must run with EPT enabled. VM entry in SEAM root operation will fail if the "enable EPT" VM-execution control is not set in the SEAM-guest VMCS.

CHAPTER 2

SEAM AND TME-MK

SEAM interoperates with Total Memory Encryption-Multi-Key (TME-MK). This chapter explains the specifics of TME-MK related to that interoperation.

The current configuration of TME-MK can be inferred from the contents of the IA32_TME_ACTIVATE MSR (MSR address 982H). The following fields are relevant:

- If IA32_TME_ACTIVATE[1:0] = 11B, Total Memory Encryption (TME) is active.
- IA32_TME_ACTIVATE.MK_TME_KEYID_BITS[35:32] specifies the number of bits in a physical address that are interpreted as a key identifier (KeyID). If TME is active and this value is greater than 0, TME-MK is active.
- IA32_TME_ACTIVATE.TDX_RESERVED_KEYID_BITS[39:36] specifies the number of KeyID bits that can be set only in SEAM-private KeyIDs (see Section 3.2, "SEAM Non-Root Operation"). This value never exceeds IA32_TME_ACTIVATE.MK_TME_KEYID_BITS.

Let m be the value enumerated by CPUID.80000008H:EAX[7:0]; $k = \text{IA32_TME_ACTIVATE.MK_TME_KEYID_BITS}$; and $p = \text{IA32_TME_ACTIVATE.TDX_RESERVED_KEYID_BITS}$. Then in any physical address, bits $m-1:m-k$ form a key identifier that is used to select an encryption key (only bits $(m-k)-1:0$ are used to address memory). The PCONFIG instruction can be used to specify the encryption key associated with each key identifier. A k -bit key identifier is a **SEAM-private KeyID** if it sets any of bits $k-1:k-p$; thus, a physical address contains a SEAM-private KeyID if it sets any of bits $m-1:m-p$.

The IA32_MKTME_KEYID_PARTITIONING MSR (MSR address 87H) enumerates the number of key identifiers available for use:

- IA32_MKTME_KEYID_PARTITIONING.NUM_MKTME_KIDS[31:0] enumerates the number of TME-MK key identifiers available for general use. These are numbered 1 to NUM_MKTME_KIDS (0 is not considered a TME-MK key identifier).
- IA32_MKTME_KEYID_PARTITIONING.NUM_TDX_PRIV_KIDS[63:32] enumerates the number of SEAM-private key identifiers. These are numbered NUM_MKTME_KIDS + 1 to NUM_MKTME_KIDS + NUM_TDX_PRIV_KIDS.

CHAPTER 3

OPERATION IN SEAM

This chapter provides details for SEAM root operation and SEAM non-root operation.

3.1 SEAM ROOT OPERATION

Processor behavior in SEAM root operation is similar to processor behavior in VMX root operation, with the following differences:

- Software can execute SEAMRET and SEAMOPS instructions. (Execution of these instructions outside SEAM root operation results in a #UD exception.) As noted in Section 1.1.2, execution of SEAMRET causes a SEAM return to VMX root operation. The SEAMOPS and SEAMRET instructions are described in Chapter 5.
- Software can access a special set of SEAM-only MSRs. An attempt to access a SEAM-only MSR outside SEAM operation results in a #GP(0) exception. (Some may be accessible only in SEAM root operation.) Chapter 8 identifies the SEAM-only MSRs.
- An attempt to execute the VMXOFF instruction or the VMCALL instruction in SEAM root operation results in a #GP(0) exception. Software in SEAM cannot use VMXOFF to leave VMX operation, nor can it use VMCALL to invoke an SMM-transfer monitor using an SMM VM exit.
- Entry to SEAM root operation (by either a SEAM call or a VM exit from SEAM non-root operation) blocks NMIs. Execution of IRET in SEAM root operation unblocks NMIs normally, as do appropriately configured executions of ERETS and ERETU.
- System-management interrupts (SMIs), uncore machine checks and machine-check SMIs (MSMIs) are masked in SEAM root operation. If pending, they are delivered after leaving SEAM root operation.
- Triple faults and machine checks caused by poisoned cache-line consumption in SEAM root operation cause unbreakable shutdown (see Section 10.4, “Unbreakable SEAM Shutdown”).
- TLB entries that cache address translations are tagged with an “in SEAM” bit, to separate them from address translations cached in VMX root operation.
- Memory type calculation is impacted by SEAM-private KeyID. See details in Section 4.3.

3.2 SEAM NON-ROOT OPERATION

Processor behavior in SEAM non-root operation is largely the same as that in VMX non-root operation, with the following differences:

- System-management interrupts (SMIs) and machine-check SMIs (MSMIs) that occur in SEAM non-root operation cause a VM exit instead of being delivered normally. VM exits caused by SMIs are reported with exit reason 5 (I/O SMI) or 6 (Other SMI) and qualification 0. VM exits caused by MSMIs are reported with exit reason 6 (Other SMI) and qualification 1. The SMI or MSMI remains pending following the VM exit and will be delivered following the next SEAM return.
- TLB entries that cache address translations are tagged with an “in SEAM” bit, to separate them from address translations cached in VMX non-root operation. Software running in VMX root operation can invalidate them using INVEPT instruction but not using INVVPID instruction.
- Virtual interrupt delivery ignores bits corresponding to vectors 0 through 30 in the virtual interrupt-request register (VIRR) in the virtual-APIC page when computing the requesting, virtual interrupt (RVI). See Section “Virtual-Interrupt Delivery” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C.
- Memory-type calculation for accesses to VMCS-linked data structures is impacted by SEAM-private KeyID. See details in Section 4.3.

The remainder of this section explains the changes in SEAM non-root operation with regard to the use of guest-physical addresses.

As noted in Section 1.1.3, VM entry to SEAM non-root operation requires EPT to be enabled. Thus, all addresses used by guest software as physical addresses (e.g., addresses in the guest's paging structures) are treated as guest-physical addresses and are thus translated by EPT. Translation by EPT in SEAM non-root operation differs somewhat from that in ordinary VMX non-root operation.

In SEAM non-root operation, the width of guest-physical addresses (GPAW) depends on the EPT page-walk length and the value of the "SEAM guest-physical address width" VM-execution control bit. The GPAW is 52 bits if the EPT page-walk length is 5 and "SEAM guest-physical address width" is 1; otherwise, it is 48 bits.

If the guest-physical address width is 48 and the CPU's native physical address width (MAXPHYADDR, as reported by CPUID.80000008H:EAX[7:0]) is 48 bits or more, an access to a guest-physical address that sets bits in the range 51:48 causes an EPT violation. Such an EPT violation always causes a VM exit, even if the "EPT-violation #VE" VM execution control bit is set to 1. (This is not the case with 5-level EPT outside SEAM operation. In SEAM operation, this occurs with 5-level paging as long as the "SEAM guest-physical address width" VM-execution control is 0.)

If the guest-physical address width is 48 and the CPU's native physical address width is less than 48, then an access to a guest-physical address that sets bits in the ranges 51:48 and 46:MAXPHYADDR causes a page fault exception (#PF). The error code will set bit 3, indicating that a reserved bit was set.

The uppermost bit in a guest-physical address (bit 47 or bit 51 as appropriate) is defined to be the address's "SHARED" bit. Guest-physical addresses that clear this bit are **SEAM-guest private addresses**; those that set this bit are **SEAM-guest shared addresses**.

Certain situations in SEAM non-root operation require SEAM-guest private addresses:

- Instruction fetches are limited to SEAM-guest private addresses. An attempt to fetch an instruction from a SEAM-guest shared address results in a page fault exception (#PF), regardless of the values of the "no-execute enable" (NXE) bit in IA32_EFER MSR and the value of the "execute-disable" (XD) bit in the leaf paging structure used to map the page.
- CR3 must contain a SEAM-guest private address. An attempt to load CR3 with a SEAM-guest shared address results in #GP(0).
- With PAE paging, each of the PDPTE registers must contain a SEAM-guest private address. An attempt to load any PDPTE register with a SEAM-guest shared address results in #GP(0).
- The guest paging structures must be located at SEAM-guest private addresses. If linear-address translation encounters a paging-structure entry that references a paging structure with a SEAM-guest shared address, a page fault (#PF) occurs. The error code will set bit 3, indicating that a reserved bit was set.

Translation of guest-physical addresses differs for private and shared addresses:

- Translation of a SEAM-guest private address uses EPT paging structures located using the EPTP field in the VMCS.
 - Any physical address used to access an EPT paging structure is modified as follows. Let k be the value of IA32_TME_ACTIVATE.MK_TME_KEYID_BITS[35:32] and M be the value enumerated by CPUID.80000008H:EAX[7:0]. Before being used to access memory, bits $M-1:M-k$ of the physical address are replaced by bits $k-1:0$ of the SEAM-guest KeyID field in the VMCS.
 - The final physical address (to which the original SEAM-guest private address is being translated) is also modified in the manner described in the previous paragraph.
- Translation of a SEAM-guest shared address uses EPT paging structures located using the SEAM shared EPT pointer in the VMCS.
 - Bits 7:0 of the EPTP field (not the SEAM shared EPT pointer) control the translation of the guest-physical address. Note that bits 11:0 of the SEAM shared EPT pointer field are reserved and must be set to 0.
 - Any physical address used to access an EPT paging structure should not use a SEAM-private KeyID (see Chapter 2). If translation of a SEAM-guest shared address would access an EPT paging structure with a SEAM-private KeyID, an EPT misconfiguration occurs.
 - If the final physical address (to which the original SEAM-guest shared address is being translated) contains a SEAM-private KeyID, the access is aborted, meaning that writes are ignored and reads return values with all bits set to 1.

CHAPTER 4

SEAM MEMORY

This chapter specifies access controls and cryptographic measures applied to protect software running in SEAM root operation (SEAM modules) and SEAM non-root operation (SEAM guests).

4.1 SEAM RANGE

The SEAM range is a physical memory range sequestered for SEAM modules. The range is allocated using range register MSRs IA32_SEAMRR_BASE (MSR address 1400H) and IA32_SEAMRR_MASK (MSR address 1401H). The existence of these MSRs is enumerated by bit 15 of IA32_MTRRCAP (MSR address FEH). Processors that support SEAM architecture return IA32_MTRRCAP bit 15 as 1. For more information about these MSRs, see Chapter 2 in Volume 4 of the Intel® 64 and IA-32 Architectures Software Developer's Manual.

The SEAM range is divided into a **module sub-range** and a **loader sub-range**. The module sub-range resides at the bottom of the SEAM range (i.e., it starts at platform physical address (IA32_SEAMRR_BASE & ~FFFH)). This range contains SEAM transfer VMCSs, which are used in transitions into and out of module-mode SEAM root operation. The SEAM-transfer VMCS for logical processor x resides at byte offset $(x + 1) * 4096$, where x is x2APIC ID of the logical processor, as returned by CPUID.1FH.0:EDX.

The loader sub-range is allocated at the top of the SEAM range, and is defined by per-core CRs CR_SEAMLDR_RR_BASE and CR_SEAMLDR_RR_MASK. This range contains a single SEAM transfer VMCS, at byte offset 4096, which is used in transitions into and out of loader-mode SEAM root operation.

Bit 11 of IA32_SEAMRR_MASK MSR indicates whether the SEAM range is valid and protected. When this bit is 1, the processor provides the SEAM range with cryptographic and access-control protections as follows:

- Outside SEAM root operation, accesses to the physical addresses in the SEAM range are aborted, meaning that writes are ignored and reads return values with all bits set to 1.
- The following items apply in SEAM root operation:
 - Accesses to the module sub-range of the SEAM range are allowed.
 - Accesses to the loader sub-range of the SEAM range are allowed only in loader-mode SEAM root operation; otherwise, they are aborted (see above).
 - Instruction fetches from physical addresses outside the SEAM range are prevented by the processor (the response to such attempts is implementation-specific).
- The contents of the SEAM range are encrypted by the TME-MK engine. The encryption key is machine specific. It is typically an "ephemeral" key that is generated each time the platform is restarted.

4.2 MEMORY PROTECTIONS

Outside SEAM operation, registers holding physical addresses cannot be loaded with physical addresses with SEAM-private KeyIDs (see Section 3.2, "SEAM Non-Root Operation"):

- Attempts to load control registers and MSRs with a physical address with a SEAM-private KeyID results in a general-protection exception (#GP(0)). This does not apply to loads of the IA32_RTIT_OUTPUT_BASE MSR (using WRMSR, WRMSRLIST, WRMSRNS, or XRSTORS), as that MSR is checked at another time (see below).
- Linear-address translation without EPT that encounters a paging-structure entry with a physical address with a SEAM-private KeyID results in a page fault.
- Guest-physical-address translation with EPT that encounters an EPT paging-structure entry with a physical address with a SEAM-private KeyID results in an EPT misconfiguration.
- VM entry fails if it would use or load a physical address (from a VMCS field other than the SEAM shared EPT pointer) that contains a SEAM-private KeyID. Note that the VMWRITE instruction doesn't fail on attempt to set a VMCS field with a physical address with a SEAM-private KeyID.

- The value of the physical address loaded into IA32_RTIT_OUTPUT_BASE MSR is checked when processor trace is enabled (when IA32_RTIT_CTL.TraceEn is set to 1), as is the physical address of a new output buffer when it is loaded from the table of physical addresses (ToPA). Attempts to load a PT output address with a physical address with a SEAM-private KeyID results in a machine-dependent behavior: either PT gets disabled with an error logged, or PT output gets silently aborted (see below).
- Attempts to load IA32_APIC_BASE with physical address that has a non-zero KeyID results in a general-protection exception (#GP(0)).
- VMX instructions that accept physical address operands return an instruction-specific failure indication if provided with a physical address with a SEAM-private KeyID, as if the physical address had a reserved bit set to 1.

If, despite the limitations identified above, there is an access using a physical address with a SEAM-private KeyID (perhaps because an MSR had been loaded before TME was activated), the access is aborted, meaning that writes are ignored and reads return values with all bits set to 1.

A cache line that was written with a physical address with a SEAM-private KeyID cannot be read using a physical address with a different KeyID. The memory-protection mode supported by a particular processor is machine-dependent.

The INVD instruction invalidates the processor's internal caches without writing modified data back to memory. Since this may compromise the operation of SEAM software that may have modified the data, the INVD instruction causes a general-protection fault (#GP) if executed when the SEAM range is valid and protected (see Section 4.1).

For information about memory protection modes, see Intel® Architecture Memory Protections for Confidential Computing.¹

4.3 MEMORY TYPE

When running in SEAM operation, the memory type is determined as follows:

- When CR0.CD is set, the memory type will be UC, except for special operations that explicitly force a memory type (e.g., fast strings, MOVDIR64, etc.); these continue to operate with their respective special memory type in the same way they execute outside of SEAM non-root operation.
- The memory type of memory accesses in SEAM non-root operation (EPT is enabled) when CR0.CD is not set are calculated as follows:
 - The EPT (either shared EPT or secure EPT, depending on the SHARED bit of the guest physical address) provides EPT memory type.
 - For linear accesses, the PAT memory type is taken as usual from the IA32_PAT MSR according to the "Ignore PAT" bit in EPT.
- The memory type of memory accesses in SEAM root operation and of memory accesses to VMCS-linked data structures (e.g., MSR bitmap) in SEAM non-root operation (EPT is disabled), when CR0.CD is not set, are calculated as follows:
 - If MTRRs are disabled (IA32_MTRR_DEF_TYPE[11] = 0), the MTRR memory type is UC. If MTRRs are enabled, the MTRR memory type depends on KeyID: if it is SEAM-private KeyID (access to SEAM private memory), the MTRR type is WB, else it is taken from the MTRR register as usual.
 - For linear accesses, the PAT memory type is taken as usual from the IA32_PAT MSR.

1. This document can be found at <https://www.intel.com/content/www/us/en/content-details/869103/intel-architecture-memory-protections-for-confidential-computing.html>.

CHAPTER 5

SEAM INSTRUCTIONS

The SEAM architecture includes the following instructions:

- **SEAMCALL.** Execution causes a SEAM call (see Section 1.1.1) to enter SEAM root operation. It can be used by virtual-machine monitors (operating in VMX root operation) to call SEAM-module services.
- **TDCALL.** Execution causes a VM exit from SEAM non-root operation to SEAM root operation. It can be used by SEAM guests to request SEAM-module services.
- **SEAMOPS.** Execution invokes a SEAM-specific operation. It is used by SEAM modules to execute such operations.
- **SEAMRET.** Execution causes a SEAM return (Section 1.1.2) that leaves SEAM root operation. It is used by SEAM modules to return to the calling virtual-machine monitor.

An attempt to execute a SEAM instruction on a processor that does not support SEAM instruction set (enumerated by bit 5 of IA32_VMX_PROCBASED_CTRL3 MSR being 0; see Chapter 9, “SEAM ISA Enumeration”) results in an invalid-opcode exception (#UD).

SEAMCALL—Enter SEAM Root Operation

Opcode	Instruction	Op/En	64-bit Mode	Compat/Leg Mode	Description
66 0F 01 CF	SEAMCALL	Z0	Valid	Invalid	Transitions to SEAM root operation.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

Description

SEAMCALL is a privileged instruction that causes a SEAM call by passing control to software running in SEAM root operation. The instruction can be executed only in 64-bit mode in VMX operation outside SEAM operation and outside SMM. In VMX non-root operation, execution causes a VM exit with exit reason 76 (SEAMCALL) and exit qualification 0.

Execution when the SEAM range is not valid (IA32_SEAMRR_MASK[11] = 0) or immediately following execution of MOV to SS or POP SS instruction results in a general-protection exception (#GP).

The instruction uses RAX[63] to select a sub-mode of SEAM root operation: if this value is 1, the instruction transitions into loader-mode SEAM root operation; otherwise, it transitions into module-mode SEAM root operation. The instruction fails with VMFailInvalid indication (see Section “Conventions” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C) if SEAM is globally disabled; or if a loader-mode SEAM root operation is called when another logical processor is in SEAM having entered in loader mode.

If there is no failure, SEAMCALL uses the SEAM-transfer VMCS to effect a SEAM call (see Section 1.1.1).

A SEAM call is similar to an SMM VM exit invoked by VMCALL that transitions from VMX root operation into SMM VMX root operation (see Section “SMM VM Exits” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C), with the following differences:

- The exit reason is 76 (SEAMCALL) and the exit qualification is always 0.
- The current VMCS pointer is loaded with the value of the SEAM transfer VMCS pointer.
- The VMCS link pointer of the SEAM transfer VMCS is loaded with the current VMCS pointer at the time SEAMCALL commenced.
- SMBASE is not saved.

Operation

If not in VMX operation or in SMM operation or in SEAM operation

 Raise #UD;

Else if IA32_EFER.LMA = 0 or CS.L = 0

 Raise #UD;

Else if CPL > 0

 Raise #GP(0);

Else if in VMX non-root operation

 Perform VM exit(“Basic Reason” = 76, “Qualification” = 0); // to VMX root operation

Else if IA32_SEAMRR_MASK.VALID = 0 or events blocked by MOV-SS

 Raise #GP(0);

Else if CR_GLOBAL_TEE_DISABLE = 1 // SEAM is globally disabled

 Return VMFailInvalid;

Else if RAX[63] = 0

 // Check if module-mode SEAM root operation is ready for execution

 If CR_SEAMEXTEND.SEAM_READY = 0

 Return VMFailInvalid;

 Else

 // Enter module-mode SEAM root operation

```

        Set TMP_CVP := (IA32_SEAMRR_BASE & ~FFFH) + (x + 1) * 4096;
    Endif;
Else // RAX[63] = 1
    // If another logical processor is executing in loader-mode SEAM operation
    // Or loader-mode SEAM root operation is not ready for execution
    // Return VMFailInvalid
    Acquire CR_SEAMLDR_MUTEX;
    If CR_SEAMLDR_MUTEX could not be acquired
        Return VMFailInvalid;
    Else
        If CR_SEAMEXTEND.P_SEAMLDR_READY = 0
            Release CR_SEAMLDR_MUTEX;
            Return VMFailInvalid;
        Else
            // Enter loader-mode SEAM root operation
            Enter loader mode; // opens loader sub-range
            Set TMP_CVP := CR_SEAMLDR_RR_BASE + 4096;
        Endif;
    Endif;
Endif;
Set RFLAGS.CF,OF,SF,PF,AF,ZF := 0;
Enter SEAM mode; // opens SEAM range
Set TMP_CVP → VMCS link pointer := Current VMCS pointer;
Set Current VMCS pointer := TMP_CVP;
Perform SEAM call to SEAM root operation (with "Basic Reason" = 76, "Qualification" = 0);
Block NMI and SMI signals;

```

Flags Affected

See the Operation section above and Section "Conventions" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C.

Protected Mode Exceptions

#UD SEAMCALL is not recognized in protected mode.

Real-Address Mode Exceptions

#UD SEAMCALL is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD SEAMCALL is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD SEAMCALL is not recognized in compatibility mode.

64-Bit Mode Exceptions

#UD

- If executed outside VMX operation.
- If executed in system-management mode (SMM).
- If executed in SEAM operation.
- If preceded by LOCK prefix.

#GP(0)

- If the current privilege level is not 0.
- If the SEAM range is not valid.
- If events blocked by MOV SS or POP SS instructions.

SEAMRET—Exit SEAM Root Operation

Opcode	Instruction	Op/En	64-bit Mode	Compat/Leg Mode	Description
66 0F 01 CD	SEAMRET	Z0	Valid	Invalid	Transitions out of SEAM root operation.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

Description

SEAMRET is a privileged instruction that causes a SEAM return by passing control to software running in VMX root operation. The instruction can be executed only in 64-bit mode in SEAM root operation.

SEAMRET works like VM entries that return from SMM and transition from SMM VMX root operation to VMX root operation (see Section “VM Entries that Return from SMM” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C), with the following differences:

- It starts with the current VMCS pointer being a SEAM-transfer VMCS pointer.
- SMBASE is not loaded.
- A SEAM return from loader-mode SEAM root operation invalidates all address translations.

Operation

If not in SEAM root operation or IA32_EFER.LMA = 0 or CS.L = 0

Raise #UD;

Else if CPL > 0

Raise #GP(0);

Else

Let TMP_CVP := Current VMCS pointer;

Perform SEAM return to VMX root operation;

If not failed // no VMFailValid, VMFailInvalid, or “Failed VM entry” VM exit

Set Current VMCS pointer := TMP_CVP → VMCS link pointer;

If SEAMRET commenced in loader-mode SEAM operation

Invalidate all address translations;

Set current VMCS pointer := NULL;

Exit loader mode; // closes loader sub-range

Release CR_SEAMLDR_MUTEX;

Endif;

Exit SEAM mode; // closes SEAM range

Endif;

Endif;

Flags Affected

See the Operation section above and Section “Conventions” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C.

Protected Mode Exceptions

#UD SEAMRET is not recognized in protected mode.

Real-Address Mode Exceptions

#UD SEAMRET is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD SEAMRET is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD SEAMRET is not recognized in compatibility mode.

64-Bit Mode Exceptions

#UD If executed outside SEAM root operation.
If preceded by LOCK prefix.

#GP(0) If the current privilege level is not 0.

TDCALL—Call SEAM Module

Opcode	Instruction	Op/En	64-bit Mode	Compat/Leg Mode	Description
66 0F 01 CC	TDCALL	Z0	Valid	Invalid	Call SEAM module from SEAM guest.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

Description

TDCALL is a privileged instruction that allows software running in SEAM non-root operation to call a SEAM module in SEAM root operation. The instruction can be executed only in SEAM non-root operation or VMX non-root operation.

In SEAM non-root operation or VMX non-root operation, execution causes a VM exit with exit reason 77 (TDCALL) and exit qualification 0.

Operation

If not in SEAM non-root operation and not in VMX non-root operation

Raise #UD;

Else if CPL > 0

Raise #GP(0);

Else

Perform VM exit("Basic Reason" = 77, "Qualification" = 0);

Flags Affected

None.

Protected Mode Exceptions

- #UD If executed outside VMX non-root operation and outside SEAM non-root operation.
If preceded by LOCK prefix.
- #GP(0) If the current privilege level is not 0.

Real-Address Mode Exceptions

- #UD If executed outside VMX non-root operation and outside SEAM non-root operation.
If preceded by LOCK prefix.

Virtual-8086 Mode Exceptions

- #UD If executed outside VMX non-root operation and outside SEAM non-root operation.
If preceded by LOCK prefix.
- #GP(0) Otherwise.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

Same as protected mode exceptions.

SEAMOPS—Invoke SEAM Operation

Opcode	Instruction	Op/En	64-bit Mode	Compat/Leg Mode	Description
66 0F 01 CE	SEAMOPS	Z0	Valid	Invalid	Invoke SEAM operation.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

Description

SEAMOPS is a privileged instruction that allows software running in SEAM root operation to invoke SEAM operations (leaf functions of the instruction). Software selects the leaf function by setting the appropriate value in RAX. Other registers may have leaf-specific purposes. The instruction can be used only in 64-bit mode.

Execution of a SEAMOPS function may lock the value of CR_CPUSVN if not already locked. The set of SEAMOPS functions that lock CR_CPUSVN is implementation-specific.

Uses of the address-size override prefix (67H) with certain SEAM operations may reduce the address size of the linear addresses to 32 bits. The set of SEAM operations that are affected by address-size override prefix is implementation-specific.

Operation

If not in SEAM root operation or IA32_EFER.LMA = 0 or CS.L = 0 or preceded by LOCK prefix

Raise #UD;

If CPL > 0

Raise #GP(0);

If the SEAMOPS leaf indexed by RAX is not supported

Raise #GP(0);

Invoke the SEAMOPS leaf indexed by RAX;

Flags Affected

The impact on flags is leaf-specific. See leaf descriptions.

Protected Mode Exceptions

#UD SEAMOPS is not recognized in protected mode.

Real-Address Mode Exceptions

#UD SEAMOPS is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD SEAMOPS is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD SEAMOPS is not recognized in compatibility mode.

64-Bit Mode Exceptions

#UD If executed outside SEAM operation.

If preceded by LOCK prefix.

#GP(0) If the current privilege level is not 0.

If the SEAMOPS leaf indexed by RAX is not supported.

CHAPTER 6

SEAM DATA

This chapter specifies SEAM data used by SEAM-only ISA: data structures, variables, and constants.

6.1 DATA STRUCTURES

This section specifies the layout of the data accessed and modified by SEAM operations and MSRs.

CPUSVN

A 16-byte data structure that contains security version numbers (SVNs) of updatable CPU components. The layout of this structure is implementation-specific.

SEAMEXTEND

A 136-byte data structure that contains measurements, configuration, TCB information and controls for SEAM modules.

Table 6-1. SEAMEXTEND Data Structure

Field Name	Offset (bytes)	Size (bytes)	Description
VALID	0	2	A bitmap that indicates which 64-bit chunks, in the first 128 bytes of the SEAMEXTEND structure, contain valid information.
Reserved	2	6	Should be set to 0.
TEE_TCB_SVN	8	16	The TEE_TCB_SVN of the active SEAM module.
MRSEAM	24	48	The SHA-384 hash of the code image of the active SEAM module.
MRSIGNER	72	48	The SHA-384 hash of the signature of the provider of the active SEAM module. Note: If the active SEAM module was provided by Intel, this field is zero and is marked invalid.
ATTRIBUTES	120	8	The configuration attributes of the SEAM module. Note: If the active SEAM module was provided by Intel, this field is zero and is marked invalid.
SEAM_READY	128	1	A flag that indicates whether the SEAM module is ready for execution (1) or not (0).
SEAM_UNDER_DEBUG	129	1	A flag that indicates whether SEAM module, SEAM loader and SEAM guests can be debugged by ITP (1) or not (0).
P_SEAMLDR_READY	130	1	A flag that indicates whether the SEAM loader is ready for execution (1) or not (0).
Reserved	131	5	Should be set to 0.

SEAMREPORT

A 495-byte data structure that contains measurement and configuration information about a SEAM entity (SEAM guest or SEAM module) and information about the TCB of the entity.

Table 6-2. SEAMREPORT Data Structure

Field Name	Offset (bytes)	Size (bytes)	Description
REPORTMACSTRUCT	0	256	Report data. See REPORTMACSTRUCT definition.
TEE_TCB_INFO	256	239	Report security information. The SHA-384 hash of this field is included in REPORTMACSTRUCT.TEE_TCB_INFO_HASH.

REPORTMACSTRUCT

A 256-byte data structure that contains measurement and configuration information of a SEAM module or a SEAM guest.

Table 6-3. REPORTMACSTRUCT Data Structure

Field Name	Offset (bytes)	Size (bytes)	Description
REPORTTYPE	0	4	Report header. See REPORTTYPE definition.
Reserved	4	12	Reported as 0.
CPUSVN	16	16	The value of CR_CPUSVN at the time the report was generated. See CPUSVN definition.
TEE_TCB_INFO_HASH	32	48	SHA-384 hash of the report's security information. See TEE_TCB_INFO definition.
TEE_INFO_HASH	80	48	SHA-384 of additional TEE-specific data. All 0's if no TEE-specific data is reported.
REPORT_DATA	128	64	Additional TEE-specific data. All 0's if no TEE-specific data is reported.
Reserved	192	32	Reported as 0.
MAC	224	32	MAC of all above REPORTMACSTRUCT fields. Computed using HMAC-SHA-256 algorithm with CR_REPORT_MAC_KEY.

REPORTTYPE

A 4-byte data structure that contains type information for REPORTMACSTRUCT.

Table 6-4. REPORTTYPE Data Structure

Field Name	Offset (bytes)	Size (bytes)	Description
TYPE	0	1	TEE Type: 0: SGX Enclave 1-127: Reserved 128-255: SEAM type
SUBTYPE	1	1	Type-specific subtype.
VERSION	2	1	Type-specific version. Must be zero for type 0 (SGX).
Reserved	3	1	Must be 0.

TEE_TCB_INFO

A 239-byte data structure that contains measurements, configuration and TCB information about the SEAM modules that created and generated the report.

When the TEE being reported is a SEAM guest, then the TEE_TCB_SVN, MRSEAM, MRSIGNER and ATTRIBUTES fields contain information about the SEAM module that created the SEAM guest, while the TEE_TCB_SVN2 field contains information about the SEAM module that generated the report (i.e., SEAM reports generated for the same SEAM guest by different SEAM modules may contain different TEE_TCB_SVN2 values).

Table 6-5. TEE_TCB_INFO Data Structure

Field Name	Offset (bytes)	Size (bytes)	Description
VALID	0	4	A bitmap of the 64-bit chunks of this TEE_TCB_INFO structure that contain valid information.
Reserved	4	4	Reported as 0.
TEE_TCB_SVN	8	16	The TEE_TCB_SVN of the SEAM module that was active when the reported TEE was created (e.g., for TDs, it is the TEE_TCB_SVN of the TDX module that created the TD). See TEE_TCB_SVN definition.
MRSEAM	24	48	The SHA-384 hash of the code image of TEE-creating SEAM module (e.g., for TDs, it identifies the SEAM module that created the TD).
MRSIGNER	72	48	The SHA-384 hash of the signature of the provider of the TEE-creating SEAM module (e.g., for TDs, it identifies the TD-creating TDX module provider). Note: If the SEAM module was provided by Intel, this field is zero and is marked invalid.
ATTRIBUTES	120	8	The configuration attributes of the TEE-creating SEAM module. Note: If the SEAM module was provided by Intel, this field is zero and is marked invalid.
TEE_TCB_SVN2	128	16	The TEE_TCB_SVN of the SEAM module that was active when the reported TEE was generated (e.g., for TDs, it's the TEE_TCB_SVN of the TDX module that generated the TD report). See TEE_TCB_SVN definition.
Reserved	144	95	Reported as 0.

TEE_TCB_SVN

A 16-byte data structure that contains the security version numbers (SVNs) of a SEAM module.

Table 6-6. TEE_TCB_SVN Data Structure

Field Name	Offset (bytes)	Size (bytes)	Description
SEAMSVN	0	2	SEAM module SVN. The internal structure of this 16-bit value depends on SEAM module's vendor. For Intel SEAM modules, the SEAMSVN value is partitioned as follows: - Bits 15:8 denote module's generation. - Bits 7:0 denote module's security level within a generation. Thus, Intel SEAM module with SEAMSVN x is more secure than Intel SEAM module with SEAMSVN y if and only if $x[15:8] = y[15:8]$ and $x[7:0] > y[7:0]$.
LAST_PATCH_SE_SVN	2	1	The security SVN of the MCU patch that was installed when this SEAM module was loaded.
LATE_SEAMLDR_SE_SVN	3	1	The security SVN of the SEAMLDR module that installed this SEAM module. Value 0 indicates that late SEAMLDR SE SVNs are not supported.
Reserved	4	12	Reported as 0.

SEAMDB_RECORD

A 128-byte data structure that contains information about installed SEAM module.

Table 6-7. SEAMDB_RECORD Data Structure

Field Name	Offset (bytes)	Size (bytes)	Description
NONCE	0	32	A 256-bit nonce for this entry.
TEE_TCB_SVN	32	16	The TEE_TCB_SVN of the SEAM module that inserted this entry. See TEE_TCB_SVN definition.
MRSEAM	48	48	The SHA-384 hash of the code image of the SEAM module that inserted this entry.
ATTRIBUTES	96	8	The configuration attributes of the SEAM module. Note: This field is set to all 0's when the SEAM module was provided by Intel.
Reserved	104	24	Should be set to 0.

TDSTATE

A 96-byte data structure that holds partial CPU state.

The partial CPU state can be loaded into CPU registers using SEAMOPS leaf 8 (TDSTATE_LOAD) and saved-cleared using SEAMOPS leaf 9 (TDSTATE_SAVE_CLEAR). The layout version and the byte size of the TDSTATE structure loaded by and saved those SEAMOPS leaves is specified by SEAMOPS leaf 6 (TDSTATE_ENUM).

In the table below, the version column denotes, for each field, in which TDSTATE version it was introduced.

Table 6-8. TDSTATE Data Structure

Field Name	Offset (bytes)	Size (bytes)	TDSTATE Version	Description
DR0	0	8	0	Debug breakpoint 0.
DR1	8	8	0	Debug breakpoint 1.
DR2	16	8	0	Debug breakpoint 2.
DR3	24	8	0	Debug breakpoint 3.
DR6	32	8	0	Debug status register.
CR2	40	8	0	Page fault linear address.
DS_AREA	48	8	0	DS save area (MSR 600H).
STAR	56	8	0	System call target address (MSR C0000081H).
LSTAR	64	8	0	IA-32e mode system call target address (MSR C0000082H).
KERNEL_GS_BASE	72	8	0	Swap target of base address of GS (MSR C0000102H).
TSC_AUX	80	8	0	Auxiliary TSC (MSR C0000103H).
FMASK	88	8	0	System call flag mask (MSR C0000084H).

SEAMKEYREQUEST

A 128-byte data structure that holds a key request. It is used by SEAMOPS leaf 11 (SEAMGETKEY).

Table 6-9. SEAMKEYREQUEST Data Structure

Field Name	Offset (bytes)	Size (bytes)	Description
CPUSVN	0	16	The CPUSVN value for the key to be derived. The SVNs in this field should not be numerically higher than the corresponding SVNs in CR_CPUSVN. See definitions of CPUSVN and CR_CPUSVN.
TEE_TCB_SVN	16	16	The TEE_TCB_SVN for the key to be derived. Ignored if the requested key is not associated with any specific SEAM module or is requested for a specific SEAMDB entry. See definition of TEE_TCB_SVN.

Table 6-9. SEAMKEYREQUEST Data Structure

Field Name	Offset (bytes)	Size (bytes)	Description
SEAMDB_NONCE	32	32	The 256-bit nonce associated with the SEAMDB entry whose index is given in the SEAMDB_INDEX fields, or 0 if the key is not requested for a SEAMDB entry. See definition of SEAMDB_RECORD.
SEAMDB_INDEX	64	8	The index of the SEAMDB entry for which the key is requested or all-1s if the key is not requested for any SEAMDB entry.
KEYNAMESPACE	72	2	A software-specified value that the SEAM module can use to separate keys between key name spaces. For example, a TDX Module may use different KEYNAMESPACE values to separate S4 migration keys and TD sealing keys.
KEYSIZE	74	1	The size of the requested key: either 0 = 128-bit key or 1 = 256-bit key (values 2-255 are reserved and must not be used).
Reserved	75	5	Must be 0.
KEYINFO	80	48	Key-specific additional information, which SEAM modules can use to separate keys within a key name space. For example, a TDX Module may associate keys with TDs by setting this field with the SHA-384 measurement of the TD for which the key is requested.

KEYCONTEXT

A 68-byte data structure that holds context for a key request. It is used by SEAMOPS leaf 11 (SEAMGETKEY) to describe the context of the requested key in the key derivation string.

Table 6-10. KEYCONTEXT Data Structure

Field Name	Offset (bytes)	Size (bytes)	Description
OWNER	0	2	The TCB component that requested the key: 0: SEAM loader 1: SEAM module 2: SGX - Other values are reserved.
KEYNAME	2	2	Owner-specific key name space.
EXTRATCB	4	16	Additional set of SVNs (e.g. in TEE_TCB_SVN format).
EXTRAPROVIDER	20	48	Additional information about TCB provider (e.g. SEAM module signer).

6.2 PLATFORM-SCOPE VARIABLES

This section specifies platform-scope SEAM variables.

CR_CPUSVN

A 128-bit variable that holds the current CPUSVN of the CPU.

CR_GLOBAL_TEE_DISABLE

A boolean flag. When set to 1, disables the SEAMCALL, ENCLU, and ENCLS instructions in the platform. Set by the CPU in cases that put the confidentiality or integrity of TEEs at risk (for example, machine checks). Initialized to 0.

CR_SEAMEXTEND

A 136-byte variable that holds the measurement and configuration details of the current SEAM environment (see definition of SEAMEXTEND data structure). Initialized by NP-SEAMLDR ACM (see Section 1.1).

CR_REPORT_MAC_KEY

A 128-bit or 256-bit variable that holds current report key. The report key is used for signing SEAM reports and authenticating them. The size of the report key is implementation-dependent. Initialized at patch load time with a random number.

CR_TEE_OWNER_EPOCH

A 128-bit or 256-bit variable that holds a platform-owner value. In CPUs that do not support 256-bit keys, CR_TEE_OWNER_EPOCH holds a 128-bit value. Initialized to 0.

CR_PFK1

A 128-bit or 256-bit variable that holds the platform root sealing key (PFK1). In CPUs that do not support 256-bit keys, CR_PFK1 holds a 128-bit key.

CR_SEAMDB

A variable that contains an array of n SEAMDB_RECORD entries, where n = CONST_SEAMDB_SIZE (see Section 6.4). Initialized to all 0's.

CR_SEAMDB_NEXT

A 64-bit variable that holds the index of the next free entry in CR_SEAMDB array. Initialized to 0 (indicating that CR_SEAMDB is empty).

CR_SEAMLDR_MUTEX

A boolean flag that indicates, when set to 1, that a single logical processor is running in loader-mode SEAM operation. When it is 0, no logical process is running in loader-mode SEAM operation. This variable is used by SEAMCALL and SEAMRET instructions to ensure that at most one logical processor is executing in loader-mode SEAM operation (see Chapter 5). Initialized to 0.

CR_PENDING_TEE_KEYS_REFRESH

A 64-bit variable that indicates which attestation resources should be refreshed. Specifically, bit 0 indicates that CR_REPORT_MAC_KEY requires refresh.

CR_SVNMAC_LOCK

A multiple-readers, single-writer lock. Used to synchronize updates to the security engine.

6.3 CORE-SCOPE VARIABLES

This section specifies platform-scope SEAM variables.

CR_SEAMLDR_RR_BASE

A 64-bit variable that holds the platform physical address of the loader sub-range of the SEAM range (see Section 4.1). Initialized at patch load time to the same value on all cores in the platform.

CR_SEAMLDR_RR_MASK

A 64-bit variable that holds the platform physical mask of the loader sub-range of the SEAM range (see Section 4.1). Initialized at patch load time to the same value on all cores in the platform.

6.4 CONSTANTS

This section specifies constants used and returned by SEAM operations and MSRs.

Error Codes

SEAMOPS functions return the following error codes.

Table 6-11. SEAMOPS Error Codes

Value	Name	Description
0	SEAMERR_SUCCESS	Operation completed successfully.
1	SEAMERR_BAD_INPUT	Illegal input argument.
2	SEAMERR_NO_ENTROPY	Insufficient entropy (transient error).
3	SEAMERR_DATABASE	Error on attempt to access SEAM module database.
4	SEAMERR_INVALID_SVN	The requested SVN is not supported by the CPU.
5	SEAMERR_INVALID_REPORT	The SEAM report is not valid.
6	SEAMERR_INVALID_KEYSIZE	The requested key size is not supported by the CPU.
7	SEAMERR_LOCK	Lock could not be acquired.

Configuration Constants

Table 6-12. Configuration Constants

Name	Description
CONST_SEAMDB_SIZE	The size of SEAM database, measured in number of entries.

Other Constants

Table 6-13. Configuration Constants

Name	Description
NULL	The constant FFFFFFFF_FFFFFFFFH.
NUM_CACHE_BLOCKS	Number of cache invalidation blocks (implementation dependent).

CHAPTER 7

SEAMOPS LEAF FUNCTIONS

The SEAMOPS instruction allows software running in SEAM root operation to invoke CPU functions called SEAMOPS leaves. The following table summarizes the SEAMOPS functions.

Table 7-1. SEAMOPS Leaf Functions

Leaf	Function	Description
0	CAPABILITIES	Enumerate available SEAMOPS leaves.
1	SEAMREPORT	Generate a SEAM report for current SEAM module.
2	SEAMDB_CLEAR	Clear the SEAM module database. Executable only in loader-mode SEAM operation.
3	SEAMDB_INSERT	Insert module details to the SEAM database. Executable only in loader-mode SEAM operation.
4	SEAMDB_GETREF	Get reference to the last module entry in the SEAM database.
5	SEAMDB_REPORT	Generate a SEAM report using a SEAM module entry in the SEAM database.
6	SEAMVERIFYREPORT	Verify a SEAM report.
7	TDSTATE_ENUM	Enumerate the capabilities of TD state operations.
8	TDSTATE_LOAD	Load a partial CPU state.
9	TDSTATE_SAVE_CLEAR	Save and clear a partial CPU state.
10	UPDATESVNREPORTKEY	Update Security Engine with new SEAM SVNs and Report Key.
11	SEAMGETKEY	Generate a SEAM key.
15	FEATURES	Enumerate SEAM features

Leaf 0 - CAPABILITIES

This operation enumerates all SEAMOPS leaves supported by the CPU.

Input Operands

Register	Description	Values
RAX	Leaf index	0

Output Operands

Register	Description	Values
RAX	Available SEAMOPS leaves	Bitmap of SEAMOPS leaf numbers. RAX[j] = 1 indicates that SEAMOPS leaf j is supported (RAX[0] is always 1).

Operation

Set RAX := bitmap of supported SEAMOPS leaves;

Flags Affected

None.

Error Codes

None.

Exceptions

None.

Leaf 1 - SEAMREPORT

This operation creates a signed SEAM report that contains measurement/configuration information of the SEAM environment, along with additional input data. SEAM reports can be created for the SEAM host or the SEAM guest.

The operation is used by SEAM modules in the process of creating a report of the SEAM module itself, or for a TEE (e.g., TD) hosted by the SEAM module. It fills a SEAMREPORT buffer with the measurements and configuration information of the TEE, the SEAM module, and any additional components in the TEE TCB.

The SEAMREPORT structure has a REPORTMACSTRUCT field that is integrity-protected with a MAC that covers TEE_TCB_INFO_HASH, TEE_INFO_HASH (when provided), and the additional 64-byte data. TEE_TCB_INFO_HASH is the SHA-384 hash of the TEE_TCB_INFO structure, which reflects the measurement/configuration of the SEAM module, and other elements in the TCB of all instances of the TEE hosted by the SEAM module. The VALID field contains a bitmap that indicates which fields have been populated, as not all fields apply to all reports. When used by SEAM modules to create SEAM guest reports, the TEE_INFO_HASH is expected to be a SHA-384 hash of measurement and configuration of the SEAM guest being reported by the SEAM module. When used to report the SEAM module itself, TEE_INFO_HASH may be a string of 0's.

The SEAM module can select the report type value. All valid values for REPORTTYPE.TYPE must have the high order bit set to 1. Failure to set this bit will result in an error.

The SEAM module can additionally provide 64-byte value to be included in the SEAM report. This allows the SEAM module to associate data for itself or for the SEAM guest it reports.

Input Operands

Register	Description	Values
RAX	Leaf index	1
RCX	Linear address of SEAMREPORT buffer	The address should be paging canonical and 1KB-aligned. The addressed buffer should be writable.
RDX	Report type	Upper 32-bits of RDX should be zero. Low 32-bits of RDX should contain a valid SEAM-defined REPORTTYPE value (see REPORT-TYPE specification).
R8	Linear address of a 64-byte report data buffers	The address should be paging canonical and 64B-aligned. The addressed buffer should be readable.
R9	Linear address of TEE_INFO_HASH buffer	The address should be paging canonical and 64B-aligned. The addressed buffer should be readable.

Output Operands

Register	Description	Values
RAX	Success/error code	See Error Codes.

Operation

```

If RCX[9:0] != 0 or R8[5:0] != 0 or R9[5:0] != 0
    Raise #GP(0);
If RDX[63:24] != 0 or RDX[7] = 0
    Error(SEAMERR_BAD_INPUT);
Clear TMP_TEE_TCB_INFO;
Set TMP_TEE_TCB_INFO.VALID := CR_SEAMEXTEND.VALID;
Set TMP_TEE_TCB_INFO.TEE_TCB_SVN := CR_SEAMEXTEND.TEE_TCB_SVN;
Set TMP_TEE_TCB_INFO.MRSEAM := CR_SEAMEXTEND.MRSEAM;
Set TMP_TEE_TCB_INFO.MRSIGNER := CR_SEAMEXTEND.MRSIGNER;
Set TMP_TEE_TCB_INFO.ATTRIBUTES := CR_SEAMEXTEND.ATTRIBUTES;
Set TMP_TEE_TCB_INFO.TEE_TCB_SVN2 := CR_SEAMEXTEND.TEE_TCB_SVN;
Clear TMP_REPORT_MAC_STRUCT;
Set TMP_REPORT_MAC_STRUCT.REPORTTYPE := RDX[31:0];

```

```

Set TMP_REPORT_MAC_STRUCT.CPUSVN := CR_CPUSVN;
Set TMP_REPORT_MAC_STRUCT.TEE_TCB_INFO_HASH := Sha384(TMP_TEE_TCB_INFO);
Set TMP_REPORT_MAC_STRUCT.TEE_INFO_HASH := ReadLinear(R9, 48);
Set TMP_REPORT_MAC_STRUCT.REPORT_DATA := ReadLinear(R8, 64);
TMP_REPORT_MAC_STRUCT.MAC := HmacSha256(CR_REPORT_MAC_KEY, TMP_REPORT_MAC_STRUCT);
WriteLinear(RCX, TMP_REPORT_MAC_STRUCT, 256);
WriteLinear(RCX + 256, TMP_TEE_TCB_INFO, 239);
Success();

```

Flags Affected

Flag ZF is cleared on success, else set to 1.

Flags CF, OF, SF, PF and AF are cleared.

Error Codes

SEAMERR_SUCCESS – on success.

SEAMERR_BAD_INPUT

- RDX contained an illegal REPORTTYPE value.

Exceptions

#GP(0)	Linear address in RCX, R8 or R9 is not paging canonical. RCX is not aligned on 1K-byte boundary. R8 or R9 is not aligned on 64-byte boundary.
#PF(fault code)	Linear address in R8 or R9 is not mapped as readable. Linear address in RCX is not mapped as writable.

Leaf 2 - SEAMDB_CLEAR

This loader-only operation clears the SEAM database.

Input Operands

Register	Description	Values
RAX	Leaf index	2

Output Operands

Register	Description	Values
RAX	Success/error code	0

Operation

If not in loader-mode SEAM operation

 Raise #GP(0);

If CR_SEAMDB_NEXT > 0

 Clear CR_SEAMDB entries 0 to CR_SEAMDB_NEXT - 1;

Set CR_SEAMDB_NEXT := 0;

Success();

Flags Affected

Flags CF, OF, SF, PF, AF and ZF are cleared.

Error Codes

SEAMERR_SUCCESS – on success.

Exceptions

#GP(0) Executed outside of loader-mode SEAM operation.

Leaf 3 - SEAMDB_INSERT

This loader-only operation adds an entry to the SEAM database and fills it with data in CR_SEAMEXTEND.

Input Operands

Register	Description	Values
RAX	Leaf index	3

Output Operands

Register	Description	Values
RAX	Success/error code	See Error Codes.

Operation

```

If not in loader-mode SEAM operation
    Raise #GP(0);
If CR_SEAMDB_NEXT = CONST_SEAMDB_SIZE
    Error(SEAMERR_DATABASE);
Set TMP_REPORT_MAC_KEY := GenerateRandom();
Set TMP_SEAMDB_ENTRY_NONCE := GenerateRandom();
If TMP_REPORT_MAC_KEY = 0 or TMP_SEAMDB_ENTRY_NONCE = 0
    Error(SEAMERR_NO_ENTROPY);
If !AcquireLock(CR_SVNMAC_LOCK, w)
    Error(SEAMERR_LOCK);
If CR_PENDING_TEE_KEYS_REFRESH[0] = 1
    Set CR_REPORT_MAC_KEY := TMP_REPORT_MAC_KEY;
    CR_PENDING_TEE_KEYS_REFRESH[0] := 0;
ReleaseLock(CR_SVNMAC_LOCK);
Clear TMP_SEAMDB_RECORD;
Set TMP_SEAMDB_RECORD.NONCE := TMP_SEAMDB_ENTRY_NONCE;
Set TMP_SEAMDB_RECORD.TEE_TCB_SVN := CR_SEAMEXTEND.TEE_TCB_SVN;
Set TMP_SEAMDB_RECORD.MRSEAM := CR_SEAMEXTEND.MRSEAM;
Set TMP_SEAMDB_RECORD.ATTRIBUTES := CR_SEAMEXTEND.ATTRIBUTES;
Set CR_SEAMDB[CR_SEAMDB_NEXT] := TMP_SEAMDB_RECORD;
Set CR_SEAMDB_NEXT := CR_SEAMDB_NEXT + 1;
Success();

```

Flags Affected

Flag ZF is cleared on success, else set to 1.

Flags CF, OF, SF, PF and AF are cleared.

Error Codes

SEAMERR_SUCCESS – on success.

SEAMERR_DATABASE

- SEAM database is full.

SEAMERR_NO_ENTROPY

- Random number generation failed.

Exceptions

#GP(0) Executed outside of loader-mode SEAM operation.

Leaf 4 - SEAMDB_GETREF

This operation returns index and nonce of the last entry inserted to SEAM database. These values can be used to refer to the same SEAM database entry at a later time, e.g., to create a SEAM report using SEAMOPS leaf 5 (SEAMDB_REPORT).

The operation returns error if the database is empty (CR_SEAMDB_NEXT = 0). On success, the operation returns last entry's index in R10 and last entry's nonce in R10-R14. On error, those registers are not modified. In all cases, the operation returns the total SEAM database capacity (CONST_SEAMDB_SIZE) in R15.

SEAM modules can therefore calculate the number of free SEAMDB slots N using the following formula: $N = R15 - ((RAX = 0) ? (R10 + 1) : 0)$.

Input Operands

Register	Description	Values
RAX	Leaf index	4

Output Operands

Register	Description	Values
RAX	Success/error code	See Error Codes.
R10	SEAM database entry index	The index of the last entry inserted into the SEAM database.
R14-11	SEAM database entry nonce	The 256-bit nonce of the last entry inserted into the SEAM database.
R15	SEAM database capacity	CONST_SEAMDB_SIZE.

Operation

```

Set R15 := CONST_SEAMDB_SIZE;
If CR_SEAMDB_NEXT = 0
    Error(SEAMERR_DATABASE);
Set R10 := CR_SEAMDB_NEXT - 1;
Set R11 := CR_SEAMDB[R10].NONCE[63:0];
Set R12 := CR_SEAMDB[R10].NONCE[127:64];
Set R13 := CR_SEAMDB[R10].NONCE[191:128];
Set R14 := CR_SEAMDB[R10].NONCE[255:192];

```

Flags Affected

Flag ZF is cleared on success, else set to 1.

Flags CF, OF, SF, PF and AF are cleared.

Error Codes

SEAMERR_SUCCESS – on success.

SEAMERR_DATABASE

- SEAM database is empty.

Leaf 5 - SEAMDB_REPORT

This operation works like SEAMOPS leaf 1 (SEAMREPORT), except that it fills in the TEE_TCB_INFO field with information about the SEAM module referenced by SEAM database.

The operation is used by SEAM modules to fill reports for SEAM guests that were created by a previous SEAM module.

Input Operands

Register	Description	Values
RAX	Leaf index	5
RCX	Linear address of SEAMREPORT buffer	The address should be paging canonical and 1KB-aligned. The addressed buffer should be writable.
RDX	Report type	The upper 32 bits of RDX should be zero. The lower 32 bits of RDX should contain a valid SEAM-defined REPORTTYPE value (see REPORTTYPE specification).
R8	Linear address of a 64-byte report data buffers	The address should be paging canonical and 64B-aligned. The addressed buffer should be readable.
R9	Linear address of TEEINFOHASH buffer	The address should be paging canonical and 64B-aligned. The addressed buffer should be readable.
R10	SEAM database entry index	The SEAM database index of the SEAMDB_RECORD entry associated with the guest-creating module (the SEAM module when the SEAM guest was created). Should be less than CR_SEAMDB_NEXT.
R14-11	SEAM database entry nonce	The 256-bit nonce stored in the SEAMDB_RECORD entry associated with the guest-creating module (R11 contains the low 64 bits of the nonce).

Output Operands

Register	Description	Values
RAX	Success/error code	See Error Codes.

Operation

If RCX[9:0] != 0 or R8[5:0] != 0 or R9[5:0] != 0

Raise #GP(0);

If RDX[63:24] != 0 or RDX[7] = 0

Error(SEAMERR_BAD_INPUT);

If R10 >= CR_SEAMDB_NEXT

Error(SEAMERR_DATABASE);

Set TMP_NONCE := CR_SEAMDB[R10].NONCE;

If TMP_NONCE[63:0] != R11 or TMP_NONCE[127:64] != R12 or TMP_NONCE[191:128] != R13 or TMP_NONCE[255:192] != R14

Error(SEAMERR_DATABASE);

Clear TMP_TEE_TCB_INFO;

Set TMP_TEE_TCB_INFO.VALID := CR_SEAMEXTEND.VALID | 30000H;

Set TMP_TEE_TCB_INFO.TEE_TCB_SVN := CR_SEAMDB[R10].TEE_TCB_SVN;

Set TMP_TEE_TCB_INFO.MRSEAM := CR_SEAMDB[R10].MRSEAM;

Set TMP_TEE_TCB_INFO.MRSIGNER := CR_SEAMDB[R10].MRSIGNER;

Set TMP_TEE_TCB_INFO.ATTRIBUTES := CR_SEAMDB[R10].ATTRIBUTES;

Set TMP_TEE_TCB_INFO.TEE_TCB_SVN2 := CR_SEAMEXTEND.TEE_TCB_SVN;

Clear TMP_REPORT_MAC_STRUCT;

Set TMP_REPORT_MAC_STRUCT.REPORTTYPE := RDX[31:0];

```

Set TMP_REPORT_MAC_STRUCT.CPUSVN := CR_CPUSVN;
Set TMP_REPORT_MAC_STRUCT.TEE_TCB_INFO_HASH := Sha384(TMP_TEE_TCB_INFO);
Set TMP_REPORT_MAC_STRUCT.TEE_INFO_HASH := ReadLinear(R9, 48);
Set TMP_REPORT_MAC_STRUCT.REPORT_DATA := ReadLinear(R8, 64);
Set TMP_REPORT_MAC_STRUCT.MAC := HmacSha256(CR_REPORT_MAC_KEY, TMP_REPORT_MAC_STRUCT);
WriteLinear(RCX, TMP_REPORT_MAC_STRUCT, 256);
WriteLinear(RCX + 256, TMP_TEE_TCB_INFO, 239);
Success();

```

Flags Affected

Flag ZF is cleared on success, else set to 1.

Flags CF, OF, SF, PF and AF are cleared.

Error Codes

SEAMERR_SUCCESS – on success.

SEAMERR_BAD_INPUT

- RDX contains an illegal REPORTTYPE value.

SEAMERR_DATABASE

- R10 contains an out-of-bound SEAMDB index.
- R14-R11 value doesn't match entry nonce of indexed SEAMDB entry.

Exceptions

#GP(0)	Linear address in RCX, R8 or R9 is not paging canonical. RCX is not aligned on 1K-byte boundary. R8 or R9 is not aligned on 64-byte boundary.
#PF(fault code)	Linear address in R8 or R9 is not mapped as readable. Linear address in RCX is not mapped as writable.

Leaf 6 – SEAMVERIFYREPORT

This operation verifies a REPORTMACSTRUCT, which is the bottom part of a SEAM report. The operation verifies the report in two steps:

1. It verifies that the report's CPUSVN matches the current CPU's set of SVN's, which is stored in CR_CPUSVN.
2. It verifies that the report's MAC matches the HMAC-SHA-256 hash of the first 224 bytes of the report, where the hash is computed using the current CPU's report key, which is stored in CR_REPORT_MAC_KEY.

Input Operands

Register	Description	Values
RAX	Leaf index	6
RBX	Linear address of REPORTMACSTRUCT buffer	The address should be paging canonical and 256B-aligned. The addressed buffer should be readable.

Output Operands

Register	Description	Values
RAX	Success/error code	See Error Codes.

Operation

```

If RBX[7:0] != 0
    Raise #GP(0);
Set TMP_REPORT_MAC_STRUCT := ReadLinear(RBX, 256);
If TMP_REPORT_MAC_STRUCT.CPUSVN != CR_CPUSVN
    Error(SEAMERR_INVALID_SVN);
Set TMP_MAC = HmacSha256(CR_REPORT_MAC_KEY, TMP_REPORT_MAC_STRUCT);
If TMP_MAC != TMP_REPORT_MAC_STRUCT.MAC
    Error(SEAMERR_INVALID_REPORT);
Success();

```

Flags Affected

Flag ZF is cleared on success, else set to 1.

Flags CF, OF, SF, PF and AF are cleared.

Error Codes

SEAMERR_SUCCESS – on success.

SEAMERR_INVALID_SVN

- The report was created for a CPUSVN other than current CPU's CPUSVN.

SEAMERR_INVALID_REPORT

- Report authentication failed (the report was modified, or signed with a different key).

Exceptions

#GP(0)	RBX is not a paging canonical linear address. RBX is not aligned on 256-byte boundary.
#PF(fault code)	The linear address in RBX is not mapped as readable.

Leaf 7 - TDSTATE_ENUM

This operation enumerates the size and layout of the TDSTATE data structure.

Input Operands

Register	Description	Values
RAX	Leaf index	7

Output Operands

Register	Description	Values
RAX	TDSTATE information	Bits 7:0 - Version. Bits 15:8 - Size in quadwords (64-bit blocks). Bits 63:16 - Reserved (0).

Operation

Set RAX[7:0] := TDSTATE version;
Set RAX[15:8] := TDSTATE size;
Set RAX[63:16] := 0;

Flags Affected

None.

Error Codes

None.

Exceptions

None.

Leaf 8 - TDSTATE_LOAD

This operation loads several CPU registers with the values read from a TDSTATE data structure.

The order of memory reads and register loads is implementation-specific. The CPU may read only 32 bits of memory when loading the 32-bit registers IA32_TSC_AUX and IA32_FMASK. Reserved bits are silently forced to their respective reserved values.

Input Operands

Register	Description	Values
RAX	Leaf index	8
RCX	Linear address of TDSTATE buffer	The address should be paging canonical and 8B-aligned. The addressed buffer should be readable.

Output Operands

None.

Operation

```

If RCX[2:0] != 0
    Raise #GP(0);
Set DR0 := ReadLinear(RCX, 8);
Set DR1 := ReadLinear(RCX + 8, 8);
Set DR2 := ReadLinear(RCX + 16, 8);
Set DR3 := ReadLinear(RCX + 24, 8);
Set DR6 := SetClearDr6ReservedBits(ReadLinear(RCX + 32, 8));
Set CR2 := ReadLinear(RCX + 40, 8);
Set IA32_DS_AREA := CpuCanonicalize(ReadLinear(RCX + 48, 8));
Set IA32_STAR := ReadLinear(RCX + 56, 8);
Set IA32_LSTAR := CpuCanonicalize(ReadLinear(RCX + 64, 8));
Set IA32_KERNEL_GSBASE := CpuCanonicalize(ReadLinear(RCX + 72, 8));
Set IA32_TSC_AUX := ReadLinear(RCX + 80, 8) & FFFFFFFFH;
Set IA32_FMASK := ReadLinear(RCX + 88, 8) & FFFFFFFFH;

```

Flags Affected

None.

Error Codes

None.

Exceptions

#GP(0)	Linear address RCX, RCX+8, ..., or RCX+88 is not paging canonical. RCX is not aligned on 8-byte boundary.
#PF(fault code)	Linear address RCX, RCX+8, ..., or RCX+88 is not mapped as readable.

Leaf 9 - TDSTATE_SAVE_CLEAR

This operation saves the values of several CPU registers into TDSTATE data structure and clears them.

The order register reads, memory writes and register clears is implementation-specific. It is guaranteed, however, that a register is cleared only after its previous value was saved to memory.

Input Operands

Register	Description	Values
RAX	Leaf index	9
RCX	Linear address of TDSTATE buffer	The address should be paging canonical and 8B-aligned. The addressed buffer should be writable.

Output Operands

None.

Operation

```

If RCX[2:0] != 0
    Raise #GP(0);
WriteLinear(RCX, DR0, 8);
Set DR0 := 0;
WriteLinear(RCX + 8, DR1, 8);
Set DR1 := 0;
WriteLinear(RCX + 16, DR2, 8);
Set DR2 := 0;
WriteLinear(RCX + 24, DR3, 8);
Set DR3 := 0;
WriteLinear(RCX + 32, DR6, 8);
Set DR6 := FFFFFFF0H;
WriteLinear(RCX + 40, CR2, 8);
Set CR2 := 0;
WriteLinear(RCX + 48, IA32_DS_AREA, 8);
Set IA32_DS_AREA := 0;
WriteLinear(RCX + 56, IA32_STAR, 8);
Set IA32_STAR := 0;
WriteLinear(RCX + 64, IA32_LSTAR, 8);
Set IA32_LSTAR := 0;
WriteLinear(RCX + 72, IA32_KERNEL_GSBASE, 8);
Set IA32_KERNEL_GS_BASE := 0;
WriteLinear(RCX + 80, IA32_TSC_AUX, 8);
Set IA32_TSC_AUX := 0;
WriteLinear(RCX + 88, IA32_FMASK, 8);
Set IA32_FMASK := 0;

```

Flags Affected

None.

Error Codes

None.

Exceptions

#GP(0) Linear address RCX, RCX+8, ..., or RCX+88 is not paging canonical.

	RCX is not aligned on 8-byte boundary.
#PF(fault code)	Linear address RCX, RCX+8, ..., or RCX+88 is not mapped as writable.

Leaf 10 - UPDATESVNREPORTKEY

This operation informs the underlying security engine of the SVNs and report key used by SEAM.

Input Operands

Register	Description	Values
RAX	Leaf index	10

Output Operands

None.

Operation

While (not AcquireLock(CR_SVNMAC_LOCK, R))

 // spin-wait until lock acquired

Store CR_REPORT_MAC_KEY, CR_SEAMEXTEND.TEE_TCB_SVN.SEAMSVN and CR_SEAMEXTEND.TEE_TCB_SVN.LAST_PATCH_SE_SVN in security engine;

ReleaseLock(CR_SVNMAC_LOCK);

Flags Affected

None.

Error Codes

None.

Exceptions

#GP(0) Executed outside of loader-mode SEAM operation.

Leaf 11 – SEAMGETKEY

This operation generates a 256-bit key according to the given key request. The key can be generated for the current or previously installed SEAM module, or for the current SEAM loader. Separate keys are generated for SEAM modules and for SEAM loaders.

Input Operands

Register	Description	Values
RAX	Leaf index	11
RCX	Linear address of SEAMKEYREQUEST buffer	The address should be paging canonical and 128B-aligned. The addressed buffer should be readable.
RCX	Linear address of 256-bit key buffer	The address should be paging canonical and 32B-aligned. The addressed buffer should be writable.

Output Operands

Register	Description	Values
RAX	Success/error code	See Error Codes.

Operation

```

If RCX[6:0] != 0
    Raise #GP(0);
Set TMP_KEYREQUEST := ReadLinear(RCX, 96);
If TMP_KEYREQUEST.RESERVED != 0
    Raise #GP(0);
If RDX[4:0] != 0
    Raise #GP(0);
Set TMP_INDEX := TMP_KEYREQUEST.SEAMDB_INDEX;
If TMP_INDEX = NULL
    If in loader-mode SEAM operation
        Clear TMP_TEE_TCB_SVN;
    Else
        Set TMP_TEE_TCB_SVN := CR_SEAMEXTEND[1023:0];
Else
    If TMP_INDEX >= CR_SEAMDB_NEXT
        or CR_SEAMDB[TMP_INDEX].NONCE != TMP_KEY_REQUEST.NONCE
        Error(SEAMERR_DATABASE);
    Set TMP_TEE_TCB_SVN := CR_SEAMDB[TMP_INDEX].TEE_TCB_SVN;
If TMP_KEYREQUEST.TEE_TCB_SVN[7:0] > TMP_TEE_TCB_SVN[7:0]
or TMP_KEYREQUEST.TEE_TCB_SVN[15:8] != TMP_TEE_TCB_SVN[15:8]
or TMP_KEYREQUEST.TEE_TCB_SVN[23:16] > TMP_TEE_TCB_SVN[23:16]
or TMP_KEYREQUEST.TEE_TCB_SVN[127:24] != 0
    Error(SEAMERR_INVALID_SVN);
If in loader-mode SEAM operation
    Set TMP_KEYCONTEXT.OWNER := 0;
Else
    Set TMP_KEYCONTEXT.OWNER := 1;
Set TMP_KEYCONTEXT.KEYNAMESPACE := TMP_KEYREQUEST.KEYNAMESPACE;
Set TMP_KEYCONTEXT.EXTRATCB := TMP_KEYREQUEST.TEE_TCB_SVN;
Set TMP_KEYCONTEXT.EXTRAPROVIDERS := CR_SEAMEXTEND.MRSIGNER;
If (TMP_KEYREQUEST.KEYSIZE = 0 and CPU doesn't support 128-bit sealing keys)
or (TMP_KEYREQUEST.KEYSIZE = 1 and CPU doesn't support 256-bit sealing keys)

```

```

or TMP_KEYREQUEST.KEYSIZE >= 1
    Error(SEAMERR_INVALID_KEYSIZE);
Set TMP_BASEKEY := GetBaseKey(TMP_KEYREQUEST.CPUSVN);
If GetBaseKey failed // the requested CPUSVN is not supported by the CPU
    Error(SEAMERR_INVALID_SVN);
// Collect entropy sources into key materials string
Set TMP_KEYMATERIALS := GenKeyMaterialsString(TMP_BASEKEY, CR_PFK1, CR_TEE_OWNER_EPOCH);
// Collect requested key parameters into derivation string
Set TMP_DERIVATION_STRING := GenKeyDerivationString(
    TMP_KEYREQUEST.KEYINFO, // key label
    TMP_KEYCONTEXT, // key context
    TMP_KEYREQUEST.KEYSIZE = 0 ? 128 : 256);
// derive the key from key materials and derivation string
TMP_KEY := HmacSha256(TMP_KEYMATERIALS, TMP_DERIVATIONSTRING);
If TMP_KEYREQUEST.KEYSIZE = 0
    Set TMP_KEY[255:128] := 0 // clear upper 128 bits
WriteLinear(RDX, TMP_KEY, 32);
Success();

```

Flags Affected

Flag ZF is cleared on success, else set to 1.

Flags CF, OF, SF, PF and AF are cleared.

Error Codes

SEAMERR_SUCCESS – on success.

SEAMERR_INVALID_SVN

- The requested key is for an unsupported CPUSVN.
- The SVN of requested key is not compatible with the SVN of the module for which the key is requested.

SEAMERR_DATABASE

- The requested key is for an unknown SEAM module.

Exceptions

#GP(0)	Linear address in RCX or RDX is not paging canonical. RCX is not aligned on 128-byte boundary. RDX is not aligned on 32-byte boundary.
#PF(fault code)	Linear address in RCX is not mapped as readable. Linear address in RDX is not mapped as writable.

Leaf 15 - FEATURES

This operation returns a bitmap that enumerates SEAM ISA features except SEAMOPS leaves (which are enumerated by SEAMOPS leaf 0 (CAPABILITIES)).

CPUs that do not support this SEAMOPS operation (i.e., SEAMOPS leaf 0 return $RAX[15] = 0$) support features as if the SEAM ISA feature bitmap (below) were all zeroes.

Input Operands

Register	Description	Values
RAX	Leaf index	15

Output Operands

Register	Description	Values
RAX	Bits 63:0 of SEAM ISA feature bitmap	See below.
RBX	Bits 127:64 of SEAM ISA feature bitmap	Reserved. Set to 0.
RCX	Bits 191:128 of SEAM ISA feature bitmap	Reserved. Set to 0.
RDX	Bits 255:192 of SEAM ISA feature bitmap	Reserved. Set to 0.

The following table specifies the set of SEAM ISA features enumerated in register RAX.

Table 7-2. SEAM ISA Features Bitmap

Bit(s)	Description	Value(s)
0	Reserved	0.
1	No WBINVDP MSRs	Returns 1 if MSR_WBINVDP and MSR_WBNOINVDP are not supported by the CPU, else 0. See definitions of MSR_WBINVDP and MSR_WBNOINVDP.
2	128-bit Keys	Returns 1 if SEAMOPS leaf 11 (SEAMGETKEY) supports generation of 128-bit sealing keys, else 0.
3	256-bit Keys	Returns 1 if SEAMOPS leaf 11 (SEAMGETKEY) supports generation of 256-bit sealing keys, else 0.
63:4	Reserved	0.

Operation

Clear RAX, RBX, RCX, RDX;

If the CPU doesn't support MSR_WBINVD and MSR_WBNOINVDP

$RAX[1] := 1$;

If the CPU supports 128-bit sealing keys

$RAX[2] := 1$;

If the CPU supports 256-bit sealing keys

$RAX[3] := 1$;

Success();

Flags Affected

Flags ZF, CF, OF, SF, PF and AF are cleared.

Error Codes

None.

Exceptions

None.

Helper Functions

This section specifies helper functions used by SEAMOPS functions.

Success()

This helper function clears RAX as well as RFLAGS bits CF, OF, SF, PF, AF, and ZF.

Error(ErrorCode)

This helper function sets RAX to ErrorCode; clears RFLAGS bits CF, OF, SF, PF and AF; and sets RFLAGS.ZF := 1.

CpuCanonicalize(LA)

This helper function converts the 64-bit linear address 'LA' into the equivalent 64-bit CPU-canonical linear address, and returns the result. The conversion is done by extending bit L-1 to bits 63:L, where L is the CPU-canonical address width enumerated by the CPUID.07H.0:ECX[16] bit, known as LA57 bit (if LA57 = 1 then L = 57, else L = 48).

SetClearDr6ReservedBits(Dr6)

This helper function sets reserved-1 bits and clears reserved-0 bits in the given 64-bit 'Dr6' value, and returns the result.

ReadLinear(SrcLA, Size)

This helper function reads 'Size' bytes from the buffer pointed by linear address 'SrcLA'. It may detect fault conditions related to linear memory reads (#PF if 'SrcLA' is not mapped as readable, and #GP if 'SrcLA' is not a paging-canonical linear address). If a fault condition was detected during the read operation, the operation is aborted and the fault is raised.

WriteLinear(DstLA, Src, Size)

This helper function writes 'Size' bytes from 'Src' buffer into the buffer pointed by linear address 'DstLA'. It may detect fault conditions related to linear memory reads (#PF if 'DstLA' is not mapped as writable, and #GP(0) if 'DstLA' is not a canonical linear address). If a fault condition was detected during the write operation, the operation is aborted and the fault is raised.

WritePhysical(DstPA, Src, Size)

This helper function writes 'Size' bytes from 'Src' buffer into the buffer pointed by physical address 'DstPA'. It doesn't raise faults.

Sha384(Src)

This helper function returns the 48-byte SHA-384 hash of 'Src' buffer.

HmacSha256(Key, Msg)

This helper function returns the 32-byte HMAC-SHA-256 hash value of the message in 'Msg' buffer using the key 'Key'.

GenKeyMaterialsString(KeyMaterial1, KeyMaterial2, KeyMaterial3)

This helper function returns a string that contains the three input key material strings 'KeyMaterial1-3'.

GenKeyDerivationString(KeyLabel, KeyContext, KeySize)

This helper function returns a derivation string for the key label string 'KeyLabel', the key context string 'KeyContext', and key size number 'KeySize' (which is either 128 or 256). The key derivation string is generated according to NIST SP 800-108 standard.

GetBaseKey(ReqCpuSvn)

This helper function returns a 256-bit CPU base key that was derived for the requested CPUSVN. On CPUs that do not support 256-bit keys, the helper function returns a 128-bit CPU base key. If the CPU cannot derive a valid base key for the requested CPUSVN, this helper function fails.

GenerateRandom()

This helper function returns a fresh 256-bit random number from the RNG. If there is not enough entropy, the value 0 is returned.

AcquireLock(Lock, Access)

This helper function tries to acquire a readers-writer lock for either shared read (R) or exclusive write (W). If the requested access is Read, the function returns FALSE if the lock is currently taken for Write; otherwise, it increments the number of readers holding the lock and returns TRUE. If the requested access is Write, the function returns FALSE if the lock is currently taken for Write or the number of readers holding the lock is not zero, else it marks the lock as taken for Write and returns TRUE.

ReleaseLock(Lock)

This helper function releases a readers-writer lock. If the caller is holding the lock as Reader, the function decrements the number of readers holding the lock. If the caller is holding the lock as Writer, the function marks the lock as free.

CHAPTER 8 SEAM MSRS

This chapter provides details of MSRs associated with the SEAM architecture.

NOTE

In this section, the term “RetVal” refers to the value read from the MSR, and the term “NewVal” refers to the value to be written to the MSR.

The first two MSRs establish the SEAM memory range. Because these MSRs cannot be written after the SEAM range is established, and because SEAM can be entered only after the SEAM range has been established, these MSRs can be written only outside SEAM. These MSRs can be read both outside SEAM and in SEAM.

MSR 1400H: IA32_SEAMRR_BASE - SEAM Range Base

This core-scope MSR contains the base physical address of the SEAM memory range. Verified at patch load time to hold the same value in all cores in the platform. See the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4 for details.

MSR 1401H: IA32_SEAMRR_MASK - SEAM Range Mask

This core-scope MSR contains the mask of address of the SEAM memory range. Verified at patch load time to hold the same value in all cores in the platform. See the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4 for details.

The remaining MSRs identified in this chapter are SEAM-only. Attempts to read or write them are not allowed outside SEAM (e.g., execution of RDMSR and WRMSR with one of these MSRs will cause #GP).

MSR 1402H: MSR_SEAMEXTEND - Read and Write CR_SEAMEXTEND

This write-only command MSR allows SEAM loader to read and write the platform-scope CR_SEAMEXTEND register. The MSR supports two commands:

- The COPY_IN command (selected by RAX[0] = 0) copies the contents of the input buffer into the platform-scope CR_SEAMEXTEND register. It also sets the LAST_PATCH_SE_SVN byte (2) to the minimum patch SE SVN across all packages in platform (the value of the corresponding byte of the input buffer is ignored).
- The COPY_OUT command (selected by RAX[0] = 1) copies the contents of CR_SEAMEXTEND register to the output buffer.

The MSR is supported by all CPUs that support SEAM (indicated by IA32_VMX_PROCBASED_CTL3[5] = 1).

The MSR can be used only when the processor reserved memory (PRM) is protected by the CPU (indicated by IA32_PMRM_PHYSBASE[11] = 1).

Table 8-1. MSR_SEAMEXTEND Bit Definitions

Bit(s)	Field	Value
0	Command	Commands: ▪ 0 = COPY_IN - Copy SEAMEXTEND data structure from input buffer to CR_SEAMEXTEND and set CR_SEAMEXTEND.TEE_TCB_SVN.LAST_PATCH_SE_SVN. ▪ 1 = COPY_OUT - Copy SEAMEXTEND data structure from CR_SEAMEXTEND to output buffer. See definitions of SEAMEXTEND (Table 6-1) and CR_SEAMEXTEND (Section 6.2).
7:1	Reserved	Must be 0.

Table 8-1. MSR_SEAMEXTEND Bit Definitions

Bit(s)	Field	Value
MAXPHYADDR–1:8	Address	Bits MAXPHYADDR–1:8 of the buffer physical address. Note: MAXPHYADDR is the physical address width enumerated by CPUID.80000008H:EAX[7:0].
63:MAXPHYADDR	Reserved	Must be 0.

Write Operation

If IA32_PRMRR_PHYSMASK[11] = 0 or not in loader-mode SEAM operation

Raise #GP(0);

If NewVal[63:MAXPHYADDR] != 0 or NewVal[7:1] != 0

Raise #GP(0);

Set TMP_PA := NewVal & ~FFH;

If NewVal[0] = 0

Set CR_SEAMEXTEND := ReadPhysical(TMP_PA, 136);

Let TMP_LATE_SESVN := Minimum MCU SE-SVN across all packages in platform;

Set CR_SEAMEXTEND.TEE_TCB_SVN[2] := TMP_LATE_SESVN;

Else

WritePhysical(TMP_PA, CR_SEAMEXTEND, 136);

Read Operation

// MSR is write-only

If in VM-exit MSR-store area

VMX abort;

Raise #GP(0);

MSR 98H: MSR_WBINVDP – Write-Back-Invalidate Partial

This thread-scope MSR writes back and invalidates a single cache block in the WBINVD domain to which the logical processor belongs. Cache blocks are indexed from 0 to NUM_CACHE_BLOCKS – 1, where NUM_CACHE_BLOCKS is the number of cache sub-blocks in the WBINVD domain. If the cache block is the top block (i.e., cache block index is NUM_CACHE_BLOCKS – 1) then the MSR performs a bus-fencing operation to guarantee that the current and past write-back-invalidate operations are completed.

Reads from the MSR return NUM_CACHE_BLOCKS.

The MSR can be used by SEAM modules to implement WBINVD instruction's functionality in steps, one block at a time. To write back and invalidate the entire caches in the underlying WBINVD domain, the module would perform the following loop:

For i = 0 to RDMSR(MSR_WBINVDP) – 1

WRMSR(MSR_WBINVDP, i);

The MSR is not supported by CPUs in which SEAMOPS leaf 15 (FEATURES) exists and returns RAX[1] = 1. Attempts to access this MSR in those CPUs results in #GP(0) exception.

Table 8-2. MSR_WBINVDP Bit Definitions

Bit(s)	Field	Value
63:0	Read: Number of blocks Write: Block index	Read: Return number of cache blocks in the underlying cache hierarchy. Write: Cache block index between. This value should be less than the number of cache blocks.

Read Operation

If in VM-exit MSR-store area

VMX abort;

If not in SEAM
 Raise #GP(0);
 RetVal := NUM_CACHE_BLOCKS;

Write Operation

If in VM-entry MSR-load area
 VM entry fails;
 If in VM-exit MSR-load area
 VMX abort;
 If not in SEAM
 Raise #GP(0);
 If NewVal >= NUM_CACHE_BLOCKS
 Raise #GP(0);
 Perform WBINVD operation on the cache block indexed by 'NewVal';
 If NewVal = NUM_CACHE_BLOCKS - 1
 Ensure that all previous per-block WBINVD operations are completed;

MSR 99H: MSR_WBNOINVDP- Write-Back-No-Invalidate Partial

This thread-scope MSR writes back (but does not necessarily invalidate) a single cache block in the WBINVD domain that the LP belongs to. Cache blocks are indexed from 0 to NUM_CACHE_BLOCKS - 1, where NUM_CACHE_BLOCKS is the number of cache sub-blocks in the WBINVD domain. If the cache block is the top block (i.e., cache block index is NUM_CACHE_BLOCKS - 1) then the MSR performs a bus-fencing operation to guarantee that the current and past write-back operations are completed.

Reads from the MSR return NUM_CACHE_BLOCKS.

The MSR can be used by SEAM modules to implement WBNOINVD instruction's functionality in steps, one block at a time. To write back the entire caches in the underlying WBINVD domain, the module would perform the following algorithm:

```
For i = 0 to RDMSR(MSR_WBNOINVDP) - 1
  WRMSR(MSR_WBNOINVDP, i);
```

The MSR is not supported by CPUs in which SEAMOPS leaf 15 (FEATURES) exists and returns RAX[1] = 1. Attempts to access this MSR in those CPUs results in #GP(0) exception.

Table 8-3. MSR_WBNOINVDP Bit Definitions

Bit(s)	Field	Value
63:0	Read: Number of blocks Write: Block index	Read: Return number of cache blocks in the underlying cache hierarchy. Write: Cache block index between. This value should be less than the number of cache blocks.

Read Operation

If in VM-exit MSR-store area
 VMX abort;
 If not in SEAM
 Raise #GP(0);
 RetVal := NUM_CACHE_BLOCKS;

Write Operation

If in VM-entry MSR-load area
 VM entry fails;
 If in VM-exit MSR-load area
 VMX abort;
 If not in SEAM

```

    Raise #GP(0);
If NewVal >= NUM_CACHE_BLOCKS
    Raise #GP(0);
Perform WBNOINVD operation on the cache block indexed by 'NewVal';
If NewVal = NUM_CACHE_BLOCKS - 1
    Ensure that all previous per-block WBNOINVD operations are completed;

```

MSR 9AH: MSR_INTR_PENDING

This thread-scope read-only MSR indicates the architectural events that are pending delivery. The events might be pending delivery for the following reasons:

- INTR – Due to RFLAGS.IF = 0, MOV-SS blocking, or STI blocking (same as in VMX root operation). Delivery of pending interrupt can be unblocked by setting RFLAGS.IF to 1.
- NMI – Due to NMI blocking (same as in VMX root operation). Can be unblocked by executing IRET, ERETS or ERETU instructions.
- SMI – Always blocked in SEAM root operation (unlike VMX root operation).

The MSR is supported by all CPUs that support SEAM (indicated by IA32_VMX_PROCBASED_CTL3[5] = 1).

Table 8-4. MSR_INTR_PENDING Bit Definitions

Bit(s)	Field	Value
0	INTR	1 if hardware interrupt is pending delivery, else 0.
1	NMI	1 if hardware non-maskable interrupt (NMI) is pending delivery, else 0.
2	SMI	1 if SMM interrupt (SMI or MSMI) is pending delivery, else 0.
3	UNSPEC1	1 if an unspecified event is pending delivery, else 0. Note: SEAM modules should not assume any specific meaning to this bit.
4	UNSPEC2	1 if an unspecified event is pending delivery, else 0. Note: SEAM modules should not assume any specific meaning to this bit.
63:5	Reserved	Return 0.

Read Operation

```

If in VM-exit MSR-store area
    VMX abort;
If not in SEAM
    Raise #GP(0);
Set RetVal := 0;
If INTR is pending
    Set RetVal[0] := 1;
If NMI is pending
    Set RetVal[1] := 1;
If SMI is pending
    Set RetVal[2] := 1;
If UNSPEC1 is pending
    Set RetVal[3] := 1;
If UNSPEC2 is pending
    Set RetVal[4] := 1;

```

Write Operation

```

// MSR is read-only
If in VM-entry MSR-load area
    VM entry fails;
If in VM-exit MSR-load area

```

VMX abort;
Raise #GP(0);

MSR 550H: MSR_SEAM_SAI_MODE

This thread-scope command MSR controls SEAM module's ability to access TDX-Connect registers (through MMIO interface). This ability is controlled by a SEAM-specific and thread-specific value called SEAM Security Attribute of Initiator (SEAM SAI).

The MSR is supported by all CPUs that support TDX Connect architecture. SEAM modules should not access this MSR when TDX Connect architecture is not supported.

Table 8-5. MSR_SEAM_SAI Bit Definitions

Bit(s)	Field	Value
0	SEAM_SAI	Setting to 1 enables SEAM module's access to TDX-Connect MMIO registers. Setting to 0 disables SEAM module's access to TDX-Connect MMIO registers.
63:1	Reserved	Must be 0.

Write Operation

If not in SEAM or in SEAM non-root operation or the CPU does not support the TDX Connect architecture

Raise #GP(0);

If NewVal[63:1] != 0

Raise #GP(0);

If NewVal[0] = 1

Set SEAM SAI;

Else

Clear SEAM SAI;

Read Operation

// MSR is write-only

If in VM-exit MSR-store area

VMX abort;

Raise #GP(0);

CHAPTER 9

SEAM ISA ENUMERATION

SEAM instructions are enumerated by bit 5 of `IA32_VMX_PROCBASED_CTL3` (MSR address 492H). As noted in Section 1.1.3, this bit enumerates support for the 1-setting of “SEAM guest-physical address width” VM-execution control. Since every processor that supports the SEAM instructions will also support this setting (and vice versa), enumeration of this setting serves to enumerate the instructions. (Software need not use the 1-setting of “SEAM guest-physical address width” to execute SEAM instructions.)

When this enumeration bit is read as 0, execution of the following instructions results in an invalid-opcode (`#UD`) exception: `SEAMCALL`, `SEAMRET`, `TDCALL`, `SEAMOPS`. Note that bit 5 enumerates both the ability to set the control bit and the availability of SEAM ISA.

`SEAMOPS` leaf functions are enumerated by leaf 0 of `SEAMOPS` instruction, which returns a bitmap of supported leaf functions (including leaf function 0 itself) in `RAX`. When `RAX` bit *i* is read as 0, execution of `SEAMOPS` instruction leaf *i* results in a general protection fault (`#GP(0)`) exception.

SEAM MSRS are enumerated as follows:

- `IA32_SEAMRR_PHYSBASE` and `IA32_SEAMRR_PHYSMASK` are enumerated by bit 15 of `IA32_MTRRCAP` (MSR address FEH).
- `MSR_SEAMEXTEND` and `MSR_INTR_PENDING` are enumerated by bit 5 of `IA32_VMX_PROCBASED_CTL3` (MSR address 492H).
- `MSR_WBINVD` and `MSR_WBNOINVD` are enumerated by bit 5 of `IA32_VMX_PROCBASED_CTL3` and `SEAMOPS` leaf function 15 (`FEATURES`).
- `MSR_SEAM_SAI_MODE` is not enumerated. It exists only on CPUs that support TDX Connect.

CPUs that support SEAM disable switching to the legacy xAPIC mode after BIOS or system software has put the local APIC in x2APIC mode. See information about the `IA32_XAPIC_DISABLE_STATUS` MSR at <https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/cpuid-enumeration-and-architectural-msrs.html>.

CHAPTER 10

DEBUG AND MONITORING

This chapter specifies SEAM interactions with debug and monitoring architectures.

10.1 PROBE MODE

In-Target Probe (ITP) is a post silicon tool targeted at providing control, visibility and tracking of a target system to increase efficiency and productivity of debug and validation. To debug software running in SEAM operation using ITP, the platform should be configured to use non-production keys (NPK) prior to BIOS_DONE (see Appendix A.1).

Configuring the platform to use NPK can be done using the red unlock or DAM unlock mechanisms. On CPUs that support NPK requests, the platform can also be configured to use NPK by setting of NPK_REQUEST bit (0) in SGX_DEBUG_MODE (MSR 503H). Platform use of NPK is enumerated in the NPK_ACTIVE bit (1) of SGX_DEBUG_MODE MSR.

If the platform is not configured with NPK (SGX_DEBUG_MODE.NPK_ACTIVE = 0), the SYSTEM_UNDER_DEBUG flag in CR_SEAMEXTEND is set to 0. CR_SEAMEXTEND.SYSTEM_UNDER_DEBUG = 0, affects ITP operation as follows:

- Attempts to break logical processor while running in SEAM root operation are blocked until the logical processor has exited SEAM root operation.
- Attempts to break logical processor while running in SEAM non-root operation cause VM exit with “Other SMI” reason (6) and qualification 0, allowing the SEAM module that handles this VM exit to exit SEAM root operation (The ITP break request is kept pending).
- ITP debug resources are disabled while running in SEAM operation.

When the platform is configured with NPK (SGX_DEBUG_MODE.NPK_ACTIVE = 1), CR_SEAMEXTEND.SYSTEM_UNDER_DEBUG is set to 1, and ITP works normally and can debug software running in SEAM operation.

10.2 CORE TELEMETRIES

Core Telemetry is part of Intel Platform Monitoring Technology (Intel PMT) capabilities. On CPUs that support this capability, the CPU collects and aggregates telemetry information from all logical processors.

By default, the CPU doesn't aggregate logical-processor telemetries collected while running in SEAM operation, but SEAM modules can allow aggregation of telemetries collected while the logical process is running in SEAM non-root operation (i.e. in SEAM guests). See Section “Intel® RDT Derivative Technologies” in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B for details.

10.3 MACHINE CHECKS

SEAM range protection (indicated by IA32_SEAMRR_MASK.VALID = 1) requires that the supported IA32_MCi_CTL bits, in all MCA banks, are set to 1, on all cores in platform. If any supported IA32_MCi_CTL bit is cleared when SEAM range is protected, the platform-scope CR_GLOBAL_TEE_DISABLE state variable is set to 1, causing all further SEAMCALL instructions to fail (see Chapter 5).

Machine check errors can be reported to logical as either machine check exception (MCE) or machine check system-management interrupt (MSMI), depending on MCA bank configuration. When running in SEAM root mode, poison consumption errors are always reported as MCEs regardless of MCA bank configuration.

When an uncorrected error reported as MCE is observed by the logical processor while running in SEAM non-root mode, it is handled normally according to CR4.MCE flag and VMCS exception bitmap. If the MCE is observed in SEAM root mode, it leads to unbreakable SEAM shutdown (see Section 10.4).

When an uncorrected error reported as MSMI is observed by the logical processor while running in SEAM root mode, it is kept pending until SEAM root mode is exited. If the MSMI is observed by the logical while running in SEAM non-root mode (or was already pending upon entry to SEAM non-root mode), then the MSMI results in a VM exit with “Other SMI” reason (6) and qualification 1 (see Section 3.2).

CPUs that support SEAM non-root (SEAM_NR) status bit (12) in IA32_MCG_STATUS MSR (enumerated by IA32_SEAM_CAP[12] = 1), set the SEAM_NR bit (12) of IA32_MCG_STATUS MSR to 1 if the error was observed by the logical processor while running in SEAM non-root mode. MCE error handler can use this bit to determine if the RIP pushed onto the error handler’s stack is the RIP at which the MCE exception (#MC) was delivered. See Section “Machine-Check Exception Handler” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B for details.

10.4 UNBREAKABLE SEAM SHUTDOWN

Shutdown conditions that happen in SEAM root operation lead to an unbreakable SEAM shutdown state. Upon entering this state, the logical processor clears the SEAM_READY and P_SEAMLDR_READY flags in the CR_SEAMEXTEND state variable (so that subsequent SEAMCALL instructions will fail in all logical processors of the platform).

Events that may lead to unbreakable SEAM shutdowns include triple faults that happen in SEAM root mode; VMX aborts during VM exit from SEAM non-root operation; VMX aborts during SEAMCALL; and VM entries to SEAM non-root operation with VMCS.ACTIVITY_STATE = 2 (shutdown) that do not inject an event.

APPENDIX A

TRUST DOMAIN EXTENSIONS

Intel® Trust Domain Extensions (TDX) is a platform feature built using SEAM and TME-MK architectures. For full information, see TDX specifications in <https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/documentation.html>.

A.1 TDX MODULES

The Intel TDX architecture supports trusted execution environment (TEE) called **Trust Domains (TDs)**, which are confidential and integrity-protected virtual machines.

TDs are supported by a SEAM module called a **TDX module**. A TDX module is hosted in the module sub-range of the SEAM range, executes in module-mode SEAM root operation, and implements SEAM calls by which the host VMM can build, import, run, manage, control, export and tear-down TDs.

TDX modules are installed by another SEAM module called a **Persistent SEAM Loader (P-SEAMLDR)**. P-SEAMLDR is hosted in the loader sub-range of the SEAM range, executes in loader-mode SEAM root operation, and implements SEAM calls by which the host VMM can load a TDX module and update it.

The P-SEAMLDR module is installed by an authenticated coded module (ACM) called a **Non-Persistent SEAM Loader (NP-SEAMLDR)**. NP-SEAMLDR is launched using GETSEC instruction with ENTERACCS leaf function (see Chapter “Safer Mode Extensions Reference” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D).

A.2 TDX CONFIGURATION AND ENABLING

To use TDX, privileged software should enumerate SEAM and TME-MK capabilities and perform various configuration and enabling actions, including but not limited to the following actions:

- Enumerate TME-MK and SEAM capabilities.
- Activate TME-MK on all packages with SEAM-private KeyIDs consistently.
- Activate TME-MK on all platform cores consistently
- Configure and lock SEAM and PRM range registers on all platform cores consistently.
- Program MCA banks on all logical processors such that all defined control bits are set to 1.
- Prepare TDX information.
- Install microcode update (using WRMSR 79H command) on all packages.
- Verify that PRM and SEAM ranges are valid.
- Launch SEAMLDR ACM to install a SEAMLDR module.
- Call SEAMLDR to install a TDX module.

Details are outside the scope of this document.

A.3 TDX SVNS

The security version (SE SVN) of the NP-SEAMLDR ACM can be retrieved in byte 7 of IA32_SGX_SVN_STATUS (MSR 500H).

The SE SVNs of the various components in TD’s Trust Compute Boundary (TCB) are reported in SEAMREPORTs as follows:

- **SE SVN of NP-SEAMLDR ACM** – reported in byte 7 of REPORTMACSTRUCT.CPUSVN.

- **SE SVN of TDX Module** – reported in byte 0 of SEAMREPORT.TEE_TCB_INFO.TEETCBVN (which reports SE SVNs at the time the TD was created) and in byte 0 of SEAMREPORT.TEE_TCB_INFO.TEETCBVN2 (which reports SE SVNs at the time the SEAMREPORT structure was filled).
- **Late SE SVN of MCU patch** – reported in byte 2 of SEAMREPORT.TEE_TCB_INFO.TEETCBVN (the SE SVN of the microcode update at the time the TD-creating TDX module was installed), and in byte 2 of SEAMREPORT.TEE_TCB_INFO.TEETCBVN2 (the SE SVN of the microcode update at the time the SEAMREPORT-generating TDX module was installed).
- **Late SE SVN of P-SEAMLDR** – reported in byte 3 of SEAMREPORT.TEE_TCB_INFO.TEETCBVN (the P-SEAMLDR that installed the TD-creating TDX module) and in byte 3 of SEAMREPORT.TEE_TCB_INFO.TEETCBVN2 (the P-SEAMLDR that installed the SEAMREPORT-generating TDX module). If value 0 is reported, the later SE SVN of P-SEAMLDR equals to the SE SVN of the NP-SEAMLDR ACM.

See Section 6.1, “Data Structures,” for details about SEAMREPORT fields.

APPENDIX B

GLOSSARY

Term	Description
CPUSVN	A data structure containing CPU security version numbers used for report generation, verification, and key derivation.
Guest-Physical Address (GPA)	An address used by software in VMX or SEAM non root operation that is translated by EPT.
Guest-Physical Address Width (GPAW)	A SEAM specific control that defines the width of guest physical addresses and the position of the SHARED bit.
Loader Mode	A SEAM root sub mode in which the SEAMLDR module executed. Allows execution on at most one logical processor at a time.
Module Mode	A SEAM root sub mode in which the TDX module executes. Allows concurrent execution on multiple logical processors.
Secure Arbitration Mode (SEAM)	A VMX extension that enables secure execution environments with SEAM specific instructions, memory protections, and controls.
SEAMCALL	A privileged instruction that invokes a SEAM Call.
SEAMRET	A privileged instruction that invokes a SEAM Return.
SEAM Call	A processor transition from VMX root operation to SEAM root operation.
SEAM Guest	Software running in SEAM non root operation.
SEAM Loader	A SEAM module that executes in loader mode to install or manage other SEAM modules.
SEAM Non-Root Operation	The SEAM extension of VMX non root operation.
SEAM Range	A protected physical memory range reserved exclusively for SEAM modules.
SEAM Root Operation	The SEAM extension of VMX root operation in which SEAM modules execute.
SEAM Return	A processor transition from SEAM root operation back to VMX root operation.
SEAM Transfer VMCS	A VMCS used to control transitions into and out of SEAM root operation.
SEAM-Private KeyID	A TME MK KeyID reserved for memory that is accessible only in SEAM operation.
SEAM-Private Address	A SEAM GPA with the SHARED bit cleared, indicating private memory.
SEAM-Shared Address	A SEAM GPA with the SHARED bit set, indicating shared memory.
SEAMOPS	A privileged instruction that invokes SEAM specific operations available only in SEAM root operation.
SEAMREPORT	A cryptographically protected report containing measurement and TCB information for a SEAM entity.
SHARED Bit	The bit of a SEAM guest physical address that distinguishes private and shared memory.
TDSCALL	A privileged instruction that allows a SEAM guest to request services from a SEAM module.
TDSTATE	A SEAM defined data structure used to save and restore partial CPU state for Trust Domains.
TEE_TCB_INFO	A report structure containing measurement, configuration, and security version information for a TEE.
TEE_TCB_SVN	A data structure containing security version numbers of a SEAM module.
TME-MK	Total Memory Encryption-Multi-Key, which SEAM uses to cryptographically protect private memory.
VM Entry	A processor transition into VMX or SEAM non root operation.
VM Exit	A processor transition from VMX or SEAM non root operation to VMX or SEAM root operation, respectively.

