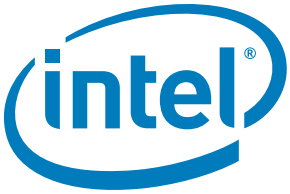

Intel® Trust Domain CPU Architectural Extensions

343754-002US
MAY 2021

Revision History

Revision	Description	Date
343754-001	Initial release.	September 2020
343754-002	Intel Persistent SEAMLDR module support and other updates.	May 2021

REVISION HISTORY

CHAPTER 1
SECURE ARBITRATION MODE (SEAM)

CHAPTER 2
VMX INSTRUCTION SET EXTENSIONS

CHAPTER 3
INTEL® SGX INSTRUCTION SET EXTENSIONS

TABLES

PAGE

FIGURES

PAGE

CHAPTER 1
SECURE ARBITRATION MODE (SEAM)

1.1 OVERVIEW

Secure Arbitration Mode (SEAM) is an extension to the Virtual Machines Extension (VMX) architecture to define a new, VMX root operation called SEAM VMX root and a new VMX non-root operation called SEAM VMX non-root. Collectively, the SEAM VMX root and SEAM VMX non-root execution modes are called operation in SEAM.

SEAM VMX root operation is designed to host a CPU-attested, software module called the Intel® Trust Domain Extensions (Intel® TDX) module to manage virtual machine (VM) guests called **Trust Domains (TD)**. The Intel TDX module implements the functions to build, tear down, and start execution of TD VMs. The VMM provides memory resources to build the TD, and the Virtual Machine Monitor (VMM) helps schedule the TD executions using the API provided by the Intel TDX module.

SEAM VMX root operation is designed to additionally host a CPU-attested, software module called the Intel Persistent SEAMLDR (Intel P-SEAMLDR) module to load and update the Intel TDX module.

Virtual machines launched/resumed from SEAM VMX root operation are TDs, and VMs launched/resumed from legacy VMX root operation are legacy VMs. Intel TDX modules use the SEAM instruction set extensions to help protect the confidentiality and integrity of TD memory contents and CPU state from all other software, including the hosting VMM, unless explicitly shared by the TD itself.

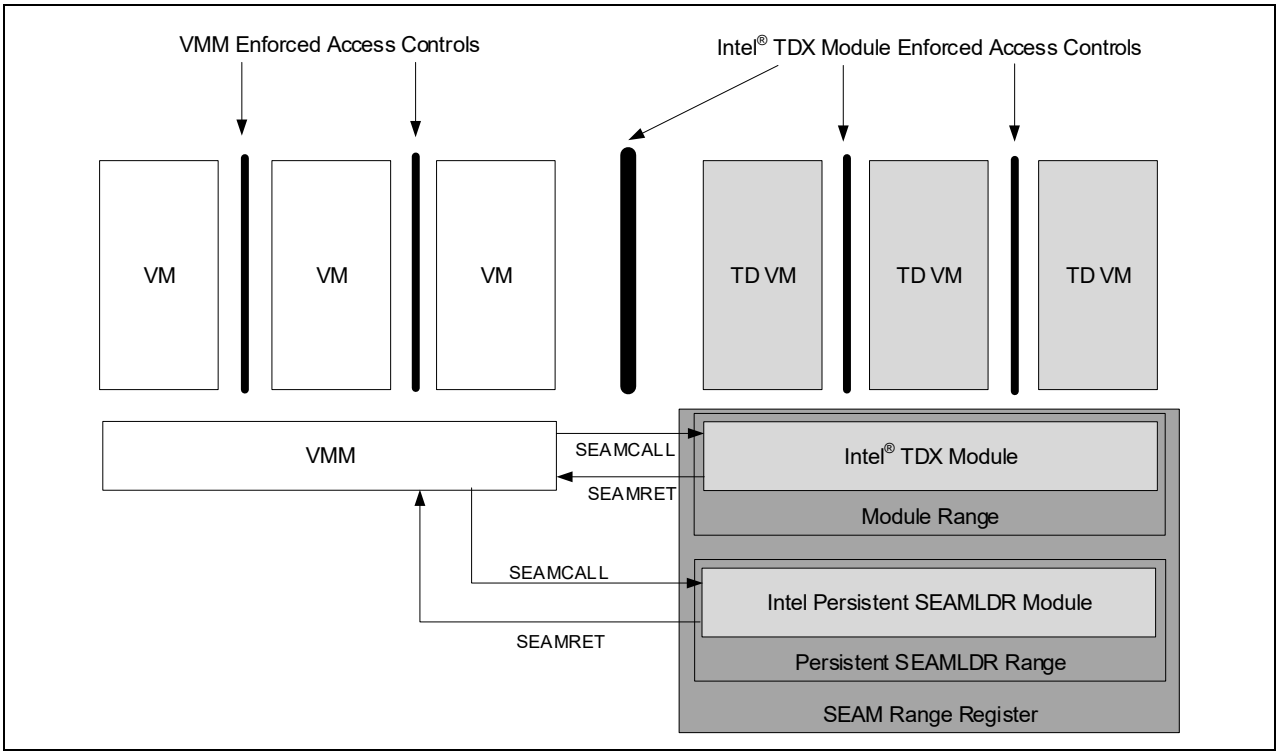

Figure 1-1. Intel® Trust Domain Extensions Components

The Intel TDX module and the Intel P-SEAMLDR module, that execute in SEAM VMX-root operation, execute out of memory defined by the SEAM range register (SEAMRR). The reserved range of memory specified using the SEAM range register (SEAMRR) is configured by the platform owner and programmed by the BIOS.

The SEAMRR range is partitioned into two sub-ranges by the processor:

- MODULE_RANGE
- P_SEAMLDR_RANGE

The MODULE_RANGE is used to install an Intel TDX-module that provides functions to build and manage TD VMs. The P_SEAMLDR_RANGE is used to install an Intel Persistent SEAM loader (P-SEAMLDR) module that is used to measure, verify, and install the Intel TDX-module into the MODULE_RANGE. By design, access to the P-SEAMLDR range is restricted to the P-SEAMLDR module.

Intel provides a non-persistent SEAMLDR (NP-SEAMLDR) ACM to install an Intel P-SEAMLDR module, in the P_SEAMLDR_RANGE of SEAMRR.

The Intel TDX-module in the MODULE_RANGE is loaded by the P-SEAMLDR.

The SEAM range register details are shown in Figure 1-2.

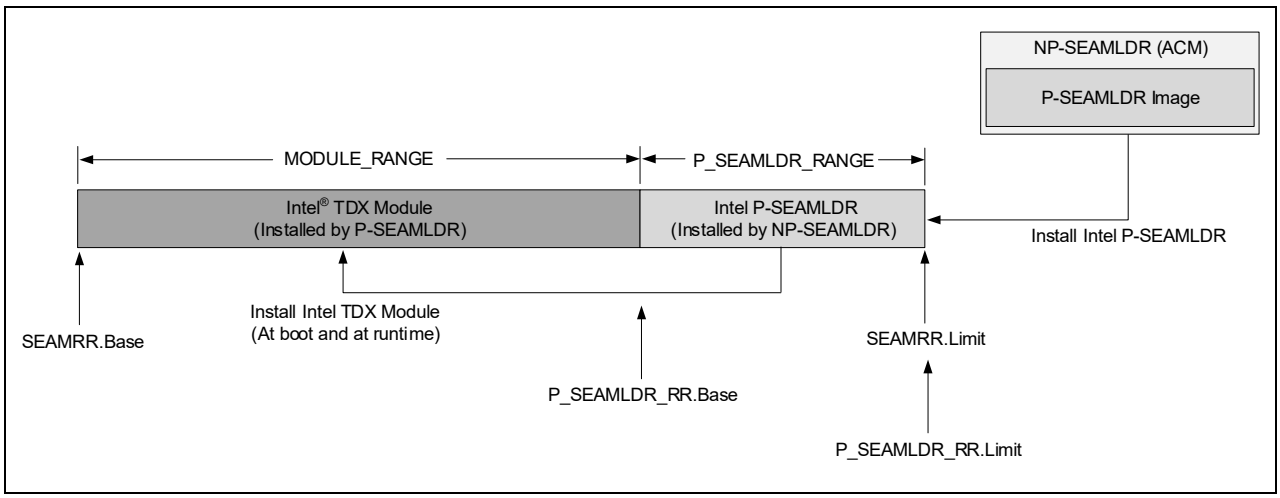

Figure 1-2. SEAM Range Register Details

The processor transitions from legacy VMX-root operation to SEAM VMX-root operation in response to a SEAMCALL instruction invoked by the VMM. The processor transitions out of SEAM VMX-root operation to legacy VMX-root operation in response to a SEAMRET instruction.

The RAX register is an input parameter to the SEAMCALL instruction. The instruction uses bit 63 of the RAX register to determine if the SEAMCALL should transition to software executing in the MODULE_RANGE (if bit 63 is 0) or the P_SEAMLDR_RANGE (if bit 63 is 1). Only one logical processor in the platform can execute from the P_SEAMLDR_RANGE at a time, and this is enforced by the SEAMCALL instruction. A SEAMCALL instruction that performs a transfer to the P_SEAMLDR enables access to the P_SEAMLDR_RANGE on that logical processor. A SEAMRET instruction invoked by the P_SEAMLDR closes access to the P_SEAMLDR_RANGE and transitions to the legacy VMX root operation.

The VMRESUME/VMLAUNCH instructions are used to help transition to the SEAM VMX non-root operation from SEAM VMX root operation. VM exits are designed to then exit from the SEAM VMX non-root transition to SEAM VMX root operation. The transitions are shown in Figure 1-3.

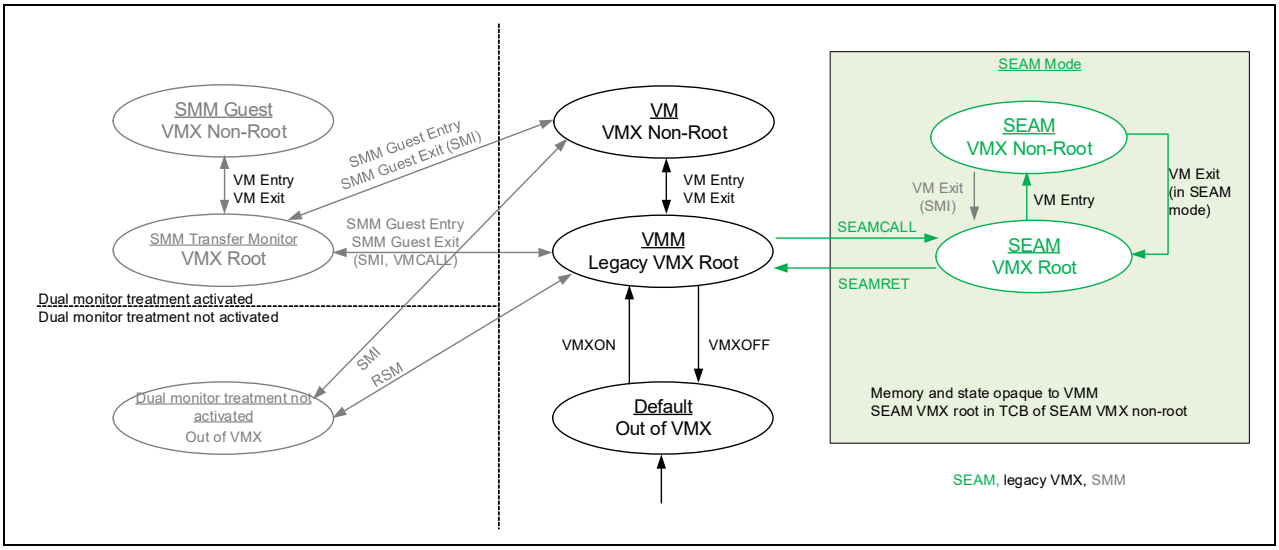

Figure 1-3. VMX and SEAM Transitions

1.2 INTEL® TDX MODULE AND INTEL P-SEAMLDR MODULE

The Intel TDX module and the Intel P-SEAMLDR module execute out of a range of memory defined using a SEAM range register (SEAMRR) interface. By design, access to this range is restricted to SEAM VMX root operation. Code fetches outside of SEAMRR when in SEAM VMX root operation are meant to be disallowed and lead to an unbreakable shutdown.

If the SEAMRR bit (bit 15) of the IA32_MTRRCAP MSR is set, the processor supports the SEAMRR interface to restrict access to a specified memory address range used by the Intel TDX module when executing in SEAM VMX root operation. The SEAM range registers consist of a pair of MSRs. The IA32_SEAMRR_PHYS_BASE MSR is intended to define the 32-MB aligned base address for the SEAM memory range. The IA32_SEAMRR_PHYS_MASK MSR contains a mask that helps determine the address range protected by the SEAMRR interface. The smallest address range that should be specified by the SEAM range registers is 32 MB. The intention is that the MSRs may be written only by boot BIOS and before the lock bit has been set; an attempt to write them outside of boot BIOS or after the lock bit has been set would cause a general-protection exception.

Table 1-1. IA32_SEAMRR_PHYS_BASE MSR and IA32_SEAMRR_PHYS_MASK MSR Layout

Register Address		Register Name / Bit Fields	Scope	Bit Description	Comment
Hex	Dec				
1400H	5120	IA32_SEAMRR_PHYS_BASE	Core	Secure Arbitration Mode Range Register - Physical Base Control Register	If IA32_MTRRCAP. SEAMRR [15] = 1.
		2:0		Reserved	
		3		When set, indicates the range is configured.	
		24:4		Reserved	
		(MAXPHYADDR-1):25		SEAMRR base address.	
		63: MAXPHYADDR		Reserved	

Table 1-1. IA32_SEAMRR_PHYS_BASE MSR and IA32_SEAMRR_PHYS_MASK MSR Layout (Continued)

Register Address		Register Name / Bit Fields	Scope	Bit Description	Comment
Hex	Dec				
1401H	5121	IA32_SEAMRR_PHYS_MASK	Core	Secure Arbitration Mode Range Register - Physical Mask Control Register	If IA32_MTRRCAP. SEAMRR [15] = 1.
		9:0		Reserved	
		10		Lock bit for the SEAMRR.	
		11		Enable bit for the SEAMRR.	
		(MAXPHYADDR-1):25		SEAMRR mask bits.	
		63: MAXPHYADDR		Reserved	

Starting at offset 4K from the SEAMRR base, a 4K page per addressable logical processor ID is used to host a SEAM transfer VMCS structure used by the SEAMCALL instruction invoked with RAX[63] set to 0 to aid the transfer from legacy VMX-root operation to SEAM VMX-root operation as a VM exit to the Intel TDX module. The SEAMRET instruction uses this VMCS to aid transfer from the SEAM VMX- root operation back to legacy VMX-root operation as a VM entry. The address of the SEAM transfer VMCS for a given, logical processor is IA32_SEAMRR_PHYS_BASE + 4096 + CPUID.B.0.EDX [31:0] * 4096.

In the P_SEAMLDR_RANGE, a 4K page is used to host a P-SEAMLDR transfer VMCS used by a SEAMCALL instruction invoked with RAX[63] set to 1 to aid the transfer from legacy VMX-root operation to SEAM VMX-root operation as a VM exit to the Intel P-SEAMLDR module. SEAMCALL with RAX[63] set to 1 is designed to allow only one logical processor to VM exit to the Intel P-SEAMLDR module. The Intel P-SEAMLDR module uses the SEAMRET instruction to transition to legacy VMX-root operation as a VM entry using the P-SEAMLDR transfer VMCS.

Intel provides a non-persistent Secure Arbitration Mode Loader (NP-SEAMLDR) ACM to help initialize the SEAM range, set up the P-SEAMLDR transfer VMCS structure, and load the Intel P-SEAMLDR module into the P_SEAMLDR_RANGE range of memory. The OS can launch the NP-SEAMLDR ACM using the GETSEC[ENTERACCS] instruction if the SEAMRR range enable bit (bit 11) of IA32_SEAMRR_PHYS_MASK MSR is 1.

The P-SEAMLDR module may be invoked by the VMM to load/update the Intel TDX module in the MODULE_RANGE of SEAMRR. The P-SEAMLDR module helps to initialize the SEAM transfer VMCS structures used for transfer to the Intel TDX module and load/update the Intel TDX module in the MODULE_RANGE of the SEAMRR. The P-SEAMLDR aims to measure and verify the Intel TDX module against its signature structure and record its security version number (SVN), measurements, and identity into CPU registers that are accessible only to the P-SEAMLDR module.

The SEAMCALL instruction invoked with RAX[63] set to 0 is meant to return VMFailInvalid if invoked when the Intel TDX module is not ready for execution. This may be due to i) either the P-SEAMLDR not having been invoked to successfully initialize and load the Intel TDX module, ii) the Intel TDX module is being updated, or iii) the Intel TDX module is not ready due to entering a shutdown when in SEAM.

The SEAMCALL instruction invoked with RAX[63] set to 1 is meant to return VMFailInvalid if invoked when the Intel P-SEAMLDR module is not ready for execution. This may be due to i) either the NP-SEAMLDR not having been invoked to successfully initialize and load the Intel P-SEAMLDR module, ii) P-SEAMLDR is itself being updated, or iii) the Intel P-SEAMLDR module is not ready due to entering a shutdown when in SEAM.

The SVN of the NP-SEAMLDR ACM itself is reported in the IA32_SGX_SVN_STATUS MSR. OS/VMMs that launch an ACM such as SINIT or SEAMLDR are expected to read the IA32_SGX_SVN_STATUS MSR to determine whether the ACM can be launched or if a new ACM is needed.

- If either the Intel® Software-Guard-Extensions (Intel® SGX) SVN of the ACM value in the ACM's header is greater than the value reported by IA32_SGX_SVN_STATUS or the lock bit in the IA32_SGX_SVN_STATUS is not set, then the OS/VMM can launch the ACM.
- If the Intel SGX SVN value reported in the corresponding component of the IA32_SGX_SVN_STATUS MSR is greater than the Intel SGX SVN value in the ACM's header, and if bit 0 of the IA32_SGX_SVN_STATUS MSR is 1, then the OS/VMM would not launch that version of the ACM. It would obtain an updated version of the ACM either from the BIOS or from an external resource.

However, OSVs/VMMs are strongly advised to update their version of the ACM any time they detect that the Intel SGX SVN of the ACM carried by the OS/VMM is lower than that reported by the IA32_SGX_SVN_STATUS MSR, irrespective of the setting of the lock bit.

Table 1-2. IA32_SGX_SVN_STATUS MSR Layout

Register Address		Register Name / Bit Fields	Scope	Bit Description	Comment
Hex	Dec				
400H	1280	IA32_SGX_SVN_STATUS	Core	Status and SVN Threshold of Intel SGX Support for ACM (RO)	If CPUID.(EAX=07H, ECX=0H): EBX[2] = 1.
		0		Lock	If 1, indicates that a non-faulting Intel SGX instruction or SEAMOPS instruction has been executed. Consequently, launching a properly signed ACM but with Intel SGX SVN value less than the BIOS specified Intel SGX SVN threshold would lead to an Intel TXT shutdown. If 0, indicates that the processor will allow a properly signed ACM to launch irrespective of the Intel SGX SVN value of the ACM.
		15:1		Reserved	
		23:16		SGX_SVN_SINIT	If CPUID. (EAX=01H, ECX=0H): ECX[SMX] =1, this field reflects the expected threshold of Intel SGX SVN for the SINIT ACM. If CPUID. (EAX=01H, ECX=0H): ECX[SMX] =0, this field is reserved (0).
		63:56		SGX_SVN_SEAMLDR	This field reflects the expected threshold of Intel SGX SVN for the NP-SEAMLDR ACM. On parts that do not support SEAM operation, this field is reserved (0).

1.3 SEAM VMX ROOT OPERATION

The Intel TDX module uses the SEAM VMX root operation to help arbitrate resource allocation to the TD VMs and transition to SEAM VMX non-root operations to execute the TD VMs. The Intel TDX module uses Multi-Key, Total-Memory-Encryption (MKTME) technology to aid protection of the confidentiality and integrity of the TD VMs' private memory. The Intel® Architecture Memory Encryption Technologies Specification is specified at: https://software.intel.com/content/dam/develop/external/us/en/documents/multi-key-total-memory-encryption-spec-753926.pdf.

Leaving VMX operation using VMXOFF or invoking SMM-monitor using VMCALL instructions is not supported and would result in a General-Protection fault (#GP(0)) in SEAM VMX root operation. This section discusses the SEAM VMX root operation differences compared to the legacy VMX root operation.

If a logical processor enters a shutdown state when in SEAM VMX root or non-root operation, the processor is designed to mark the Intel TDX module and Intel P-SEAMLDR module states as not loaded prior to entering shutdown, such that all subsequent SEAMCALL invocations on any logical processor return VMFailInvalid.

1.3.1 TDX Private KeyID

The TD VM private memory contents are meant to be encrypted and integrity protected using the MKTME technology by associating each TD VM with a different MKTME key identifier (KeyID). Protecting the confidentiality and integrity of the TD memory can require that the key associated with a KeyID assigned to a TD cannot be tampered by untrusted software. Likewise, untrusted software should not be allowed to read or write memory using the KeyID assigned to a TD.

Intel Trust Domain Extensions enable reservation of a configurable number of KeyIDs as private KeyIDs that can be used only in SEAM.

MKTME on an SOC that supports SEAM might support an integrity protected, memory encryption mode. When using keys with integrity enabled, the MKTME associates a message authentication code (MAC) with each cache line. By design, when reading a cache line using a KeyID with integrity enabled, if the MAC stored in the metadata does not match the MAC regenerated by the MKTME, then the cache line is marked poisoned to prevent the data from being consumed. Integrity protected memory must be initialized before being read, and such initialization must be performed using 64-bytes direct-store with 64-byte write atomicity using the MOVDIR64B instruction.

The maximum number of KeyIDs supported by the MKTME and the maximum number of upper physical address bits that can be configured to be used as KeyIDs are enumerated by the IA32_TME_CAPABILITY MSR. As part of activating the MKTME, BIOS is meant to use the IA32_TME_ACTIVATE MSR to configure the number of upper physical address bits to be used as KeyIDs and thereby activate the number of KeyIDs that are usable on that platform. The partitioning of this activated number of KeyIDs into shared and TDX private KeyIDs can also be performed by the boot BIOS using the IA32_TME_ACTIVATE MSR. The IA32_MKTME_KEYID_PARTITIONING MSR enables reading the KeyID partitioning done by the BIOS. The MKTME programming interface extensions are described in Table 1-3.

Table 1-3. MKTME Programming Interface Extensions

Register Address		Register Name / Bit Fields	Scope	Bit Description	Comment
Hex	Dec				
87H	135	IA32_MKTME_KEYID_PARTITIONING	Package	Multi-Key Total Memory Encryption (MKTME) KeyID Partitioning Enumeration MSR	
		31:0		NUM_MKTME_KIDS Total Number of activated MKTME KeyIDs. If the LOCK field in IA32_TME_ACTIVATE is 0, this field reports 0. MKTME KeyIDs span the KID range 1 through NUM_MKTME_KIDS.	
		63:32		NUM_TDX_PRIV_KIDS - Number of TDX Private KeyIDs If the LOCK field in IA32_TME_ACTIVATE is 0, this field reports 0. TDX private KeyIDs span the range (NUM_MKTME_KIDS + 1) through (NUM_MKTME_KIDS + NUM_TDX_PRIV_KIDS).	
981H	2433	IA32_TME_CAPABILITY	Package	Total Memory Encryption (TME) / Multi-Key Total Memory Encryption (MKTME) Capability Reporting MSR	
		0		When set to 1, AES-XTS 128-bit encryption algorithm supported.	
		1		When set to 1, AES-XTS 128-bit encryption with at least 28-bits of integrity algorithm supported.	

Table 1-3. MKTME Programming Interface Extensions (Continued)

Register Address		Register Name / Bit Fields	Scope	Bit Description	Comment
Hex	Dec				
		63:2		See Intel® Architecture Memory Encryption Technologies Specification.	
982H	2434	IA32_TME_ACTIVATE	Package	Total Memory Encryption (TME) / Multi-Key Total Memory Encryption (MKTME) Activation MSR	
		32:0		See Intel® Architecture Memory Encryption Technologies Specification.	
		35:32		MK_TME_KEYID_BITS The number of KeyID bits to allocate to MKTME usage. Like enumeration, this is an encoded value. Writing a value greater than MK_TME_MAX_KEYID_BITS will result in #GP(0). Writing a non-zero value to this field will #GP(0) if bit 1 of EAX (TME Enable) is not also set to 1, as TME must be enabled to use MKTME.	
		39:36		TDX_RESERVED_KEYID_BITS The number of physical address bits starting at (MAXPHYADDR - 1) that are reserved for Intel TDX use to encode TDX private KeyID.	The value in this field cannot exceed the value in bits 35:32 of this MSR.

The keys for the MKTME KeyID are programmed using the PCONFIG instruction. The PCONFIG instruction is designed to allow key programming for TDX private KeyID only in SEAM.

When MKTME is activated, the plan intends the upper bits of platform physical address (starting with the highest order bit available as enumerated by the CPUID MAXPA info) be repurposed for usage as a KeyID. When the KeyID space is partitioned, the TDX_RESERVED_KEYID_BITS number of bits, starting with the highest order bit of the physical address, is meant to be used to encode TDX private KeyID, and, if any of these bits are set, then the KeyID specified would be a TDX private KeyID.

Table 1-4. Encoding KeyID in Physical Address

PA Bits	63: MAXPHYADDR	(MAXPHYADDR - 1): (MAXPHYADDR - L)[1]	(MAXPHYADDR - L - 1): (MAXPHYADDR - N)[2]	(MAXPHYADDR - N - 1): 0
400H	1280	IA32_SGX_SVN_STATUS	Status and SVN Threshold of Intel SGX Support for ACM (RO)	If CPUID.(EAX=07H, ECX=0H): EBX[2] = 1.

NOTES:

1. L represents TDX_RESERVED_KEYID_BITS.
2. N represents MK_TME_KEYID_BITS.

MKTME usage for TD memory protection might require the Intel TDX module to write-back and invalidate caches for certain, maintenance operations. For some maintenance operations, the Intel TDX module might need to write-back the caches but not require invalidation of the cache contents. The WBINVD/WBNOINVD instructions may be used by the Intel TDX module to help perform these operations.

The processor can provide an alternate, cache management interface to SEAM VMX root operation through MSR_WBINVDP and MSR_WBNOINVDP. WRMSR to MSR_WBINVDP and MSR_WBNOINVDP can do write-back of one cache sub-block specified by software in EDX: EAX. The WRMSR to MSR_WBINVDP can additionally invalidate the specified, cache sub-block. The maximum of cache sub-blocks (NUM_CACHE_SUB_BLOCKS) is meant to be read by software using RDMSR to MSR_WBINVDP or MSR_WBNOINVDP. Software should invoke MSR_WBINVDP/MSR_WBNOINVDP with EDX: EAX at least once with a sub-block number between 0 and (MAX_CACHE_SUB_BLOCKS - 1) to operate on the entire cache. Specifying an invalid cache sub-block number, i.e., a number greater than or equal to MAX_CACHE_SUB_BLOCKS, can cause a General-Protection fault (#GP(0)). The operation of these WRMSR and RDMSR to MSR_WBINVDP and MSR_WBNOINVDP is designed as follows:

```
IF RDMSR
    IF inSEAM==0 THEN #GP(0);
    IF invoked from VMX load/store list THEN #GP(0);
    EDX:EAX = MAX_CACHE_SUB_BLOCKS;
ENDIF

IF WRMSR
    IF inSEAM==0 THEN #GP(0);
    IF invoked from VMX load/store list THEN #GP(0);
    IF EDX:EAX >= MAX_CACHE_SUB_BLOCKS THEN #GP(0);
    Flush cache sub-block indexed by EDX:EAX
    IF ECX == MSR_WBINVDP THEN Invalidate cache sub-block indexed by EDX:EAX;
ENDIF
```

1.3.2 Memory Typing

In SEAM VMX root operation, the determination of memory type for an access is designed as follows:

- If CR0.CD is 1, then the effective memory type is UC.
- If CR0.CD is 0, then the effective memory type depends on the physical address that is accessed and the KeyID used for the access.
  - — If the access is to the SEAMRR, then the MTRRs do not contribute to the memory typing, and the effective, memory type for the access is determined by the PAT alone.
  - — If the access is to physical memory outside SEAMRR and the access uses a TDX private KeyID, then the MTRRs do not contribute to the memory typing. The memory type is determined by the PAT alone.
  - — If the access is to physical memory outside SEAMRR and the access does not use a TDX private KeyID, then the memory type for the access is determine based on the MTRR matching the physical address and the PAT.

Special operations that explicitly force a memory type (e.g., fast strings, MOVDIR64, etc.) continue to operate with their special memory type as defined by those instructions.

1.3.3 Caching Translation Information

The address translation caching architecture is augmented with an in-SEAM state to support SEAM. When in SEAM VMX root operation, in-SEAM is 1. In SEAM VMX root operation, a logical processor might cache and use cached mappings for linear addresses derived from the paging structure referenced (directly or indirectly) by the current value of CR3 and associate them with:

- Current VPID.
- Current PCID (non-global translations) or any PCID (global translations).

- Current In-SEAM.

1.3.4 Event Handling

On transition to SEAM VMX root operation, the processor can inhibit NMI and SMI. While inhibited, if these events occur, then they are tailored to stay pending and be delivered when the inhibit state is removed. NMI and external interrupts can be uninhibited in SEAM VMX-root operation. In SEAM VMX-root operation, MSR_INTR_PENDING can be read to help determine status of any pending events.

On transition to SEAM VMX non-root operation using a VM entry, NMI and INTR inhibit states are, by design, updated based on the configuration of the TD VMCS used to perform the VM entry.

SMIs that are incident to SEAM VMX non-root operation or were pending from prior to transition to SEAM VMX non-root operation can cause a VM to exit to the SEAM root-mode operation with the exit reason set to "IO SMI" or "Other SMI". The exit qualification bit 0 is meant to be set to 1, if the SMI is a machine check initiated SMI (MSMI). The SMI would then remain pending following the VM exit.

On transition to legacy VMX root operation using SEAMRET, the NMI and SMI inhibit state can be restored to the inhibit state at the time of the previous SEAMCALL and any pending NMI/SMI would be delivered if not inhibited.

The layout of the MSR_INTR_PENDING is described in Table 1-5.

Table 1-5. MSR_INTR_PENDING MSR Layout

Register Address		Register Name / Bit Fields	Scope	Bit Description	Comment
Hex	Dec				
981H	2433	MSR_INTR_PENDING	Thread	Determine if there are pending interrupts or events. (RO)	Not allowed in VMX MSR store list. Not readable outside SEAM VMX-root operation.
		0		INTR pending.	
		1		NMI pending.	
		2		SMI pending.	
		3		Other/unspecified event pending.	
		4		Other/unspecified event pending.	
		63:5		Reserved	

The reporting of these pending events in MSR_INTR_PENDING is designed not to be affected by:

- EFLAGS.IF.
- NMI blocking.
- SEAM blocking of SMI, or NMI.
- SMI inhibited by SENTER.
- MOV-SS/POP-SS blocking.
- STI blocking.

1.4 SEAM VMX NON-ROOT OPERATION

The TD VMs execute in SEAM VMX non-root operation. This section discusses the SEAM VMX non-root operation differences compared to the legacy VMX non-root operation.

1.4.1 SEAM VMX Non-Root Execution Controls

Each TD VM is associated with a VMCS that manages transitions into and out of SEAM VMX non-root operation (VM entries and VM exits) as well as processor behavior in SEAM VMX non-root operation.

When in SEAM non-root operation, the processor uses the following, additional controls to help accomplish its aim:

1. Shared EPT Pointer (Shared-EPTP): A 64-bit execution control field (encoding pair 203CH/203DH) to specify the Shared-EPT pointer. In SEAM VMX non-root operation, the plan dictates two EPTs be active: the private EPT specified using the EPTP field of the VMCS and a Shared-EPT specified using the Shared-EPTP field of the VMCS.
   a. Bits 11:0 are reserved.
   b. Bits (MAXPHYADDR - 1):12 contain bits (MAXPHYADDR - 1):12 of the physical address of the 4-Kbyte aligned EPT PML4/PML5 table.
   c. Bits 63:MAXPHYADDR are reserved.
2. TD Key identifier (TD-KeyID) (encoding 4026H): A 32-bit execution control field to help specify the TD assigned, MKTME KeyID. The processor is meant to use this TD-KeyID to access EPT paging structures referenced by EPTP as the physical addresses obtained as a result of translation through the EPT referenced by the EPTP.
3. Guest Physical Address Width (GPAW): GPAW execution control (bit position 5) in the tertiary, processor-based execution controls (encoding 2034H) that, along with the EPT walk levels (4 or 5 level), is used to assist in determining the GPA width and thus a SHARED bit position in the GPA. The GPA space of a TD is partitioned into two halves: a shared GPA range and a private GPA range. The GPA range where a SHARED bit is 0 is TD private memory, and the EPT determined by the EPTP is used to help translate the GPA to a physical address. GPA loaded into CR3, PDPTRs, and HLAT root pointer are always translated using the EPTP. The processor uses the TD-KeyID to aid gaining access to the physical address as a result of translation of private GPA. The GPA range where a SHARED bit is 1 is TD shared memory, and the EPT determined by Shared-EPTP is used to help translate the GPA to a physical address. The processor uses the KeyID determined from the Shared EPT to aid gaining access to the physical address as a result of translation of shared GPA.
   a. When 4-level EPT is active, the SHARED bit position would always be bit 47.
   b. When 5-level EPT is active, the SHARED bit position would be bit 47 if GPAW is 0. Otherwise, else it would be bit 51.

The tertiary, processor-based execution control is designed to be activated by bit 17 of the primary, processor-based execution control, and the IA32_VMX_PROCBASED_CTLS3 MSR (index 492H) would then indicate the bits that can be set to 1 in the tertiary, processor-based execution control field in the VMCS. Bit position 5 of IA32_VMX_PROCBASED_CTLS3 MSR aims to report if the GPAW execution control in tertiary, processor-based execution controls field of the VMCS can be set to 1. When the GPAW execution control can be set to 1, the processor also supports programming the Shared-EPTP and TD-KeyID execution controls in the VMCS.

These VMCS fields would be ignored by VM entry when not in SEAM VMX root operation.

1.4.2 Guest Physical Address Translation

Transition to SEAM VMX non-root operation is formatted to require Extended Page Tables (EPT) to be enabled. In SEAM VMX non-root operation, there should be two EPTs active: the private EPT specified using the EPTP field of the VMCS and a shared EPT specified using the Shared-EPTP field of the VMCS.

When translating a GPA using the shared EPT, an EPT misconfiguration can occur if the entry is present and the physical address bits in the range (MAXPHYADDR-1) to (MAXPHYADDR-TDX_RESERVED_KEYID_BITS) are set, i.e., if configured with a TDX private KeyID.

If the CPU's maximum physical-address width (MAXPA) is 52 and the guest physical address width is configured to be 48, accesses with GPA bits 51:48 not all being 0 can cause an EPT-violation, where such EPT-violations are not mutated to #VE, even if the "EPT-violations #VE" execution control is 1.

If the CPU's physical-address width (MAXPA) is less than 48 and the SHARED bit is configured to be in bit position 47, GPA bit 47 would be reserved, and GPA bits 46:MAXPA would be reserved. On such CPUs, setting bits 51:48 or bits 46:MAXPA in any paging structure can cause a reserved bit page fault on access.

1.4.3 Linear Address Translation

In SEAM VMX non-root operation, all paging structures used to translate linear addresses to GPA should be in private GPA space. Setting a SHARED bit to 1 in CR3 or paging structures that reference another paging structure can cause a reserved bit page fault exception.

In SEAM VMX non-root operation, attempting to execute out of a page in TD shared memory, i.e., page mapped with a SHARED bit set to 1, can cause a page fault exception.

1.4.4 Memory Typing

For translated, guest physical accesses, the memory type should normally be determined based on CR0.CD, PAT memory type, and EPT memory type. The EPT memory type provided by the EPT (shared or private) helps to perform the translation.

Special operations that explicitly force a memory type (e.g., fast strings, MOVDIR64, etc.) are intended to continue to operate with their respective, special memory type.

The memory type for accessing VMCS linked data structures using a TDX private KeyID (MSR bitmaps, VAPIC page, etc.) would be UC if CR0.CD is 1. Otherwise, it would be WB.

1.4.5 Caching Translations Information

When in SEAM VMX non-root operations, EPT is intended to always be in use. Address translation caching architecture is augmented with an additional in-SEAM state to support SEAM. When in SEAM VMX root or VMX non-root operation, in-SEAM would be 1 (otherwise, it would be 0) and augment the caching architecture as follows:

- For accesses using linear addresses, the processor can create combined mappings. Combined mappings would be derived from the EPT paging structures referenced (directly or indirectly) by the current EP4TA (when 4-level EPT is active) or current EP5TA (when 5-level EPT is active). The EPT4TA/EP5TA are from the EPTP field of the VMCS. If CR0.PG = 1, the combined mappings can also be derived from the paging structures referenced (directly or indirectly) by the current value of CR3. No combined, paging-structure-cache entries would be created if CR0.PG = 0. The combined mappings are associated with:
  - Current VPID.
  - Current PCID (non-global translations) or any PCID (global translations).
  - Current EP4TA (when 4-level EPT is active) or current EP5TA (when 5-level EPT is active) from the EPTP field of the VMCS.
  - Current In-SEAM.
- For guest-physical address accesses, a processor can use guest-physical mappings derived from the EPT paging structures referenced (directly or indirectly) by bits 51:12 of the current EPTP or current Shared-EPTP, depending on the EPT used to perform the translation. Newly created, guest-physical mappings are associated with current EP4TA or current EP5TA from the EPTP field of the VMCS.

Combined and guest-physical mappings obtained through EPT referenced by Shared EPTP are also associated with the EP4TA/EP5TA from the EPTP field of the VMCS. To invalidate guest-physical or combined mappings created in SEAM, software (including the VMM executing outside SEAM operation) would use the EPTP (not Shared-EPTP) with the KeyID fields of the address set to 0.

INVVPID or INVPCID are designed to invalidate combined mappings created in SEAM only when invoked in SEAM.

1.4.6 Virtual Interrupt Delivery

Virtual interrupt delivery in SEAM VMX non-root operation aims to ignore 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). These vectors are formatted not to be delivered as virtual interrupts through virtual interrupt delivery.

1.5 OPERATION OUTSIDE SEAM

The physical address bits reserved for encoding TDX private KeyID are meant to be treated as reserved bits when not in SEAM operation.

When translating a linear address outside SEAM, if any paging structure entry has bits reserved for TDX private KeyID encoding in the physical address set, then the processor helps generate a reserved bit page fault exception.

When translating a guest physical address outside SEAM, if any EPT structure entry has bits reserved for TDX private KeyID encoding in the physical address set, then the processor helps generate an EPT misconfiguration.

Instructions that accept physical address operands when invoked with physical addresses that set bits reserved for encoding TDX private KeyID are designed to generate a failure or exception as described in Table 1-6.

Table 1-6. Encoding KeyID in Physical Address

Instruction	Behavior
VMXON	VMFailInvalid: the physical address sets reserved bits.
VMPTRLD	VMFail with reason "VMPTRLD with invalid physical address".
VMCLEAR	VMFail with reason "VMCLEAR with invalid physical address".
VMLAUNCH VMRESUME	Treat any VMCS linked physical address with bits reserved for encoding TDX private KeyID set, i.e., specifying a TDX private KeyID as invalid. The treatment is like programming these fields with a physical address with bits set beyond the maximum physical address width of the processor. Examples of such VMCS fields are MSR bitmap pointer, PML buffer, VAPIC page, etc.
INVEPT	VMFail with reason "Invalid operand to INVEPT/INVVPID".
MOV to CR3	General Protection Fault (#GP(0)).
WRMSR	Programming control registers and MSRs outside SEAM with physical addresses that have bits reserved for encoding TDX private KeyID set to 1 are treated as errors and cause a #GP(0) exception. Examples of such MSRs and CRs include: ▪ IA32_APIC_BASE. ▪ MTRR, SMRR, and PRMRR base and mask registers. ▪ IA32_HW_FEEDBACK_PTR. ▪ IA32_TME_EXCLUDE_BASE.

Enabling Intel® Processor Trace, by setting TraceEn to 1 in the IA32_RTIT_CTL MSR, causes an operational ToPA error if the IA32_RTIT_OUTPUT_BASE MSR was a) programmed with a physical address with TDX private KeyID or b) was programmed with a GPA whose translation results in a physical address with TDX private KeyID.

PCONFIG invoked outside SEAM can fail with error code INVALID_KEYID (encoding 3; KeyID not valid) if the KeyID operand is a private KeyID.

CHAPTER 2
VMX INSTRUCTION SET EXTENSIONS

Instructions described in this document follow the general documentation convention established in *Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A*. Additionally, some instructions use notation conventions as described below.

2.1 CONVENTIONS

The operation sections for the VMX instructions in Section 2.3 use the pseudo-function VMexit, which indicates that the logical processor performs a VM exit.

The operation sections also use the pseudo-functions VMsucceed, VMfail, VMfailInvalid, and VMfailValid. These pseudo-functions can signal instruction success or failure by setting or clearing bits in RFLAGS and, in some cases, by writing the VM-instruction error field. The following pseudocode fragments detail these functions:

```
VMsucceed:
        CF := 0;
        PF := 0;
        AF := 0;
        ZF := 0;
        SF := 0;
        OF := 0;

VMfail(ErrorNumber):
        IF VMCS pointer is valid
            THEN VMfailValid(ErrorNumber);
            ELSE VMfailInvalid;
        FI;

VMfailInvalid:
        CF := 1;
        PF := 0;
        AF := 0;
        ZF := 0;
        SF := 0;
        OF := 0;

VMfailValid(ErrorNumber):// executed only if there is a current VMCS
        CF := 0;
        PF := 0;
        AF := 0;
        ZF := 1;
        SF := 0;
        OF := 0;
        Set the VM-instruction error field to ErrorNumber;
```

The different VM-instruction error numbers are enumerated in Section 2.2, "VM Instruction Error Numbers".

2.2 VM INSTRUCTION ERROR NUMBERS

For certain error conditions, the VM-instruction error field is loaded with an error number to indicate the source of the error. Table 2-1 lists VM instruction error numbers.

Table 2-1. VM Instruction Error Numbers

Error Number	Description
1	VMCALL executed in VMX root operation.
2	VMCLEAR with invalid physical address.
3	VMCLEAR with VMXON pointer.
4	VMLAUNCH with non-clear VMCS.
5	VMRESUME with non-launched VMCS.
6	VMRESUME after VMXOFF (VMXOFF and VMXON between VMLAUNCH and VMRESUME).
7	VM entry with invalid control field(s)[1,2].
8	VM entry with invalid host-state field(s)[1].
9	VMPTRLD with invalid physical address.
10	VMPTRLD with VMXON pointer.
11	VMPTRLD with incorrect VMCS revision identifier.
12	VMREAD/VMWRITE from/to unsupported VMCS component.
13	VMWRITE to read-only VMCS component.
15	VMXON executed in VMX root operation.
16	VM entry with invalid executive-VMCS pointer[1].
17	VM entry with non-launched executive VMCS[1].
18	VM entry with executive-VMCS pointer not VMXON pointer (when attempting to deactivate the dual-monitor treatment of SMIs and SMM)[1].
19	VMCALL with non-clear VMCS (when attempting to activate the dual-monitor treatment of SMIs and SMM).
20	VMCALL with invalid VM-exit control fields.
22	VMCALL with incorrect MSEG revision identifier (when attempting to activate the dual-monitor treatment of SMIs and SMM).
23	VMXOFF under dual-monitor treatment of SMIs and SMM.
24	VMCALL with invalid SMM-monitor features (when attempting to activate the dual-monitor treatment of SMIs and SMM).
25	VM entry with invalid VM-execution control fields in executive VMCS (when attempting to return from SMM)[1,2].
26	VM entry with events blocked by MOV SS.
28	Invalid operand to INVEPT/INVVPID.

NOTES:

1. VM-entry checks on control fields and host-state fields can be performed in any order. Thus, an indication by error number of one cause does not imply that there are not also other errors. Different processors might give different error numbers for the same VMCS.
2. Error number 7 is not designed to be used for VM entries that return from SMM that fail due to invalid VM-execution control fields in the executive VMCS. Error number 25 is intended to be used for these cases.

2.3 INSTRUCTION SET REFERENCE

SEAMCALL — Call to SEAM VMX Root Operation

Opcode/ Instruction	Op/ En	Description
66 0F 01 CF SEAMCALL	ZO	Call to SEAM VMX root operation.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
ZO	NA	NA	NA	NA

Description

The SEAMCALL instruction uses RAX register bit 63 to determine if the request is to transition to the Intel P-SEAMLDR module (if bit 63 is 1) or the Intel TDX module (if bit 63 is 0).

When invoked in legacy VMX-root operation with RAX bit 63 set to 0, this instruction can cause a VM exit (exit reason 4CH - SEAMCALL) to the Intel TDX module in SEAM VMX root operation using the SEAM transfer VMCS of the logical processor on which the instruction was invoked. These SEAM transfer VMCS are in the module range of the SEAMRR. Bit 29 of the exit reason (VM exit from VMX root operation) would be set to 1 on such VM exits.

When invoked in legacy VMX-root operation with RAX bit 63 set to 1, this instruction can cause a VM exit (exit reason 4CH - SEAMCALL) to the Intel P-SEAMLDR module in SEAM VMX root operation using the SEAM transfer VMCS in the P-SEAMLDR range. Only one logical processor in the system, at a time, can exit to the Intel P-SEAMLDR module and the SEAMCALL instruction enforces this property through a mutex called P_SEAMLDR_MUTEX.

When invoked in SEAM VMX non-root operation or legacy VMX non-root operation, this instruction can cause a VM exit (exit reason 4CH - SEAMCALL). Bit 29 of the exit reason (VM exit from VMX root operation) will be set to 0 on such VM exits.

SEAMCALL invokes an indirect branch control mechanism that establishes a barrier, preventing software that executed before the SEAMCALL from controlling the predicted targets of indirect branches executed after SEAMCALL on the same logical processor.

Operation

```
IF not in VMX operation or inSMM or inSEAM or ((IA32_EFER.LMA & CS.L) == 0)
    THEN #UD;
ELSIF in VMX non-root operation
    THEN VMexit("basic reason" = SEAMCALL,
            "VM exit from VMX root operation" (bit 29) = 0);
ELSIF CPL > 0 or IA32_SEAMRR_MASK.VALID == 0 or "events blocking by MOV-SS"
    THEN #GP(0);
SEAM_CVP = (SEAMRR.BASE + 4K) + CPUID.B.0.EDX[31:0] * 4K
// Certain events/conditions that could affect security of SEAM could disable SEAM execution
IF
If RAX[63] == 0 AND "Intel TDX module not loaded or disabled"
    THEN VMfailInvalid
If RAX[63] == 1
   THEN
      Acquire P_SEAMLDR_MUTEX
      IF P_SEAMLDR_MUTEX acquisition failed
         THEN VMFailInvalid
      IF "Intel Persistent SEAMLDR module not loaded/or disabled"
         THEN
           Release P_SEAMLDR_MUTEX
           VMFailInvalid
```

```
        FI;
      inP_SEAMLDR = 1
      SEAM_CVP = P_SEAMLDR_CVP
   FI;

    RFLAGS.{CF, OF, SF, PF, AF, ZF} = 0

// Exiting from legacy VMX root operation
inSEAM = 1
SEAM_CVP.VMCS_link_pointer = current-VMCS
current-VMCS = SEAM_CVP
Save VMM state in current-VMCS based on its VM-exit controls
Save event inhibits in VMM interruptibility status - SMI inhibit, NMI inhibit
Load/force SEAM state based on VM-exit controls and host-state area
// Further details of the operation of the VM-exit appear in Chapter 27 "VM Exits" of Intel® 64 and IA-32 Architectures Software
// Developer's Manual Volume 3, System Programming Guide
Inhibit SMI and NMI
current-VMCS.exit_reason.basic_reason = SEAMCALL
current-VMCS.exit_reason."VM exit from VMX root operation" (bit 29) = 1
current-VMCS.exit_qualification = 0
```

Flags Affected

The operation section uses the pseudo-functions VMsucceed and VMfailInvalid. See the operation section and Section 2.1.

Protected Mode Exceptions

#UD	The SEAMCALL instruction is not designed to be recognized in protected mode.

Real-Address Mode Exceptions

#UD	The SEAMCALL instruction is not designed to be recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	The SEAMCALL instruction is not designed to be recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The SEAMCALL instruction is not designed to be recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If CPL > 0.
	If IA32_SEAMRR_MASK.VALID is 0.
	If events are being blocked by MOV SS.
#UD	If executed outside VMX operation.
	If executed in SEAM VMX root operation.
	If logical processor is in SMM.
	If IA32_VMX_PROCBASED_CTLS3[5] is 0.

SEAMOPS — Invoke SEAM Operations

Opcode/ Instruction	Op/ En	Description
66 0F 01 CE SEAMOPS	ZO	Invoke SEAM specific operations.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
ZO	NA	NA	NA	NA

Description

The SEAMOPS instruction is used to help execute SEAM-specific operations in SEAM VMX root operation.

The SEAMOPS instruction is designed to only be executed by privileged software running in SEAM VMX root operation and invoke leaf functions for performing the requested functionality. Software would select the leaf function by setting the appropriate value in register RAX. Other registers might have leaf-specific purposes. The instruction would be used only in 64-bit mode.

Two leaf functions are currently defined:

- CAPABILITIES (RAX == 0): Returns a bitmap of the supported SEAMOPS leaves.
- SEAMREPORT (RAX == 1): Generate SEAMREPORT structure.

The instruction is formatted to return the outcome of execution of leaf-specific function in RAX register. Leaf specific functions might return information in additional GPR.

SEAMOPS is intended to lock CRPL_CPUSVN and BIOS_SE_SVN, if CRPL_CPUSVN is not already locked, if the SEAMREPORT leaf is enabled. Once locked, the CPU helps prevent any further modifications to these registers. Thus, CRPL_CPUSVN might be locked by either the first, non-faulting Intel SGX instruction or the first, SEAMOPS instruction that executes with SEAMREPORT_ENABLED, i.e., the CAPABILITIES leaf reports support for the SEAMREPORT leaf.

Operation

```
IF inSEAM==0 or ((IA32_EFER.LMA & CS.L) == 0) or in VMX non-root operation or lock-prefix-used
    THEN #UD;
ELSIF CPL > 0
    THEN #GP(0);
IF SEAMREPORT_ENABLED == 1 and CRPL_CPUSVN not locked
    Lock CRPL_CPUSVN and BIOS_SE_SVN
ENDIF
SWITCH RAX
   CASE 0: RAX = SEAMREPORT_ENABLED ? 0x3 : 0x1;break; // SEAMREPORT (leaf 1) is supported if
                                                       // only if SEAMREPORT enabled
   DEFAULT: #GP(0); break;
ENDSWITCH
```

Flags Affected

None.

Protected Mode Exceptions

#UD	The SEAMOPS instruction is not designed to be recognized in protected mode.

Real-Address Mode Exceptions

#UD	The SEAMOPS instruction is not designed to be recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	The SEAMOPS instruction is not designed to be recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The SEAMOPS instruction is not designed to be recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0):	If CPL > 0. Illegal leaf function specified in RAX. Leaf specific exceptions. If any leaf-specific memory operand is in CS, DS, ES, FS, or GS segment and memory address is non-canonical.
#SS(0):	If any leaf-specific memory operand is in SS segment and memory address is non-canonical.
#UD:	If not in SEAM root operation. If LOCK prefix is used. If IA32_VMX_PROCBASED_CTLS3[5] is 0.
#PF(PFEC):	If a page fault occurs in accessing a leaf-specific memory operand.

SEAMOPS Data Structures

Table 2-2. TEE_TCB_SVN

Name	Offset (Bytes)	Size (Bytes)	Description
SEAM	0	2	Intel TDX module SVN.
RESERVED	2	14	Must be zero.

Table 2-3. TEE_TCB_INFO Structure

Name	Offset (Bytes)	Size (Bytes)	Description
VALID	0	8	Indicates TEE_TCB_INFO fields which are valid. ▪ 1 in the i-th significant bit reflects that the 8 bytes starting at offset (8 * i) are valid. ▪ 0 in the i-th significant bit reflects that either 8 bytes starting at offset (8 * i) is not populated or reserved, and is set to zero.
TEE_TCB_SVN	8	16	TEE_TCB_SVN array.
MRSEAM	24	48	Measurement of the Intel TDX module.
MRSIGNERSEAM	72	48	Measurement of TDX module signer if valid.
ATTRIBUTES	120	8	Additional configuration ATTRIBUTES if valid.
RESERVED	128	111	Must be zero.

Table 2-4. REPORTTYPE

Name	Offset (Bytes)	Size (Bytes)	Description
TYPE	0	1	TEE Type 0x00: SGX 0x7F - 0x01: RESERVED 0xFF - 0x80: SEAM Defined Note: Bit 7 defines either hardware implementation (0) or Intel TDX module implementation (1).
SUBTYPE	1	1	TYPE-specific subtype.
VERSION	2	1	TYPE-specific version. Must be 0 for Intel SGX.
RESERVED	3	1	Must be zero.

Table 2-5. REPORTMACSTRUCT

Name	Offset (Bytes)	Size (Bytes)	Description	MAC
REPORTTYPE	0	4	Type Header Structure	Y
RESERVED	4	12	Must be zero.	Y
CPUSVN	16	16	CPUSVN	Y
TEE_TCB_INFO_HASH	32	48	SHA384 of TEE_TCB_INFO	Y
TEE_INFO_HASH	80	48	SHA384 of TEE_INFO (or 0x00 if no TEE is represented)	Y
REPORTDATA	128	64	A set of data used for communication between the caller and the target.	Y
RESERVED	192	32	Must be zero.	Y
MAC	224	32	The MAC over the REPORTMACSTRUCT with report-type specific MAC key.	N

Table 2-6. SEAMREPORT

Name	Offset (Bytes)	Size (Bytes)	Description
REPORTMACSTRUCT	0	256	REPORTMAC Structure for REPORT.
TEE_TCB_INFO	256	239	TEE_TCB_INFO whose HASH is found in the REPORTMACSTRUCT.

SEAMOPS Leaf Functions

SEAMREPORT Leaf

Helps create a SEAMREPORT structure that contains the measurements/configuration information of the SEAM and, when invoked on behalf of a TEE, it includes the TEE measurements. The SEAMREPORT operands are described in Table 2-7.

Table 2-7. SEAMREPORT Operands

Operand	In / Out	Description
RAX	In	1 (SEAMOPS instruction leaf number).
	Out	Leaf-specific return code.
RCX	In	1024B-aligned linear address, of newly created SEAMREPORT structures.
RDX	In	Report Type Header in the low order 32 bits. Upper 32 bits must be zero.
R8	In	64B-aligned linear address of REPORTDATA to be signed.
R9	In	64B-aligned linear address of TEE_INFO_HASH to be signed.

SEAMREPORT Leaf Description

This instruction is used by the Intel TDX module and helps create a report, SEAMREPORT, of the Intel TDX module or a TEE (e.g., TD) hosted by the Intel TDX module. The SEAMREPORT structure can contain the measurements/configuration information of the TEE, the Intel TDX module, and any additional components in the TEE TCB. The SEAMREPORT has a REPORTMACSTRUCT that is designed to be integrity protected with a MAC that covers the TEE_TCB_INFO_HASH, the TEE_INFO_HASH (when provided), and the REPORTDATA. The TEE_TCB_INFO_HASH is meant as the SHA384 hash of the TEE_TCB_INFO structure, which reflects the measurement/configuration of the SEAM and other elements in the TCB of all instances of TEE hosted by the SEAM. The VALID field contains an array of flags that can indicate which fields have been populated, as not all fields apply to all reports. When used by the Intel TDX module to create a TEE report, the TEE_INFO_HASH is expected to be a SHA384 hash of measurement and configuration of the TEE being reported. When used to report just the Intel TDX module, TEE_INFO_HASH might be a string of zeros.

The Intel TDX module can select the report type values, however, all valid values for REPORTTYPE.TYPE should have the high order bit set to 1. Failure to set this bit will result in a SEAM_INVALID_REPORT_TYPE error. The Intel TDX module additionally helps provide the REPORTDATA, a 64-byte value to be included in the SEAMREPORT and, eventually, the quote. This aids the caller of the instruction in associating data with the Intel TDX module or the TEE it reports.

To assist verification of the integrity of the SEAMREPORT, software should help verify any hashes present in the report. The REPORTMACSTRUCT.TEE_TCB_INFO_HASH should match the hash of SEAMREPORT.TEE_TCB_INFO. If the REPORTMACSTRUCT.TEE_INFO_HASH is non-zero, it should match the hash of the separate, TEE_INFO structure. By design, if either value does not match, the report has been tampered. Software uses ENCLU[EVERIFYREPORT2] to help verify the integrity of the REPORTMACSTRUCT itself.

Table 2-8. SEAMREPORT Memory Operands

Explicit / Implicit	Linear Address	Name	Access Permissions	Alignment	Concurrency Restrictions
Explicit	RCX	SEAMREPORT	RW	1024B	None
Explicit	R8	REPORTDATA	R	64B	None
Explicit	R9	TEE_INFO_HASH	R	64B	None

SEAMREPORT Leaf Operation

```
// Temporary Variables setup with input register operands
SEAMREPORT          *tmp_seamreport_la = RCX;      // Linear address of SEAMREPORT
uint64_t             tmp_report_type = RDX;        // Report type
void                *tmp_reportdata_la = R8;       // Linear address of REPORTDATA
SHA384_HASH_t       *tmp_tee_info_hash_la = R9;    // Linear address of TEE_INFO_HASH
KEY256_t             tmp_report_key;               // 256b report key

RAX = SEAM_SUCCESS;
RFLAGS.{ZF, CF, PF, AF, OF, SF} = 0;

// Ensure SEAMREPORT pointer is 1024B aligned and read/write accessible
IF ( tmp_seamreport_la is not 1024B aligned) #GP(0);
<< tmp_seamreport_la must be read/write accessible>>

// Ensure TEE_INFO_HASH pointer is 64B aligned and read accessible
IF ( tmp_tee_info_hash_la is not 64B aligned) #GP(0);
<< tmp_tee_info_hash_la must be read accessible>>

// Ensure REPORTDATA pointer is 64B aligned and read accessible
IF ( tmp_reportdata_la is not 64B aligned) #GP(0);
<< tmp_reportdata_la must be read accessible>>

// Check reserved bit in REPORT TYPE and that TYPE reflects SEAM implementation
IF ((tmp_report_type & 0xFFFFFFFFFF000000) != 0) || (tmp_report_type & 0x0000000000000080) == 0))
{
      RAX = SEAM_INVALID_REPORT_TYPE;
      RFLAGS.ZF = 1;
      GOTO END;
}

// Create SEAMREPORT in a temporary buffer
tmp_seamreport = 0;
tmp_seamreport.REPORTMACSTRUCT.REPORTTYPE = tmp_report_type[31:0];
tmp_seamreport.REPORTMACSTRUCT.RESERVED = 0x00;
tmp_seamreport.REPORTMACSTRUCT.CPUSVN = CRPL_CPUSVN;

// Populate TEE_TCB_INFO depending on the Intel TDX module type
tmp_seamreport.TEE_TCB_INFO.TEE_TCB_SVN = CRPL_TEE_TCB_INFO.TEE_TCB_SVN;
tmp_seamreport.TEE_TCB_INFO.MRSEAM = CRPL_TEE_TCB_INFO.MRSEAM;
IF (<Intel SEAM>)
{
      tmp_seamreport.TEE_TCB_INFO.VALID = 111111111b;
}
ELSE
{
      tmp_seamreport.TEE_TCB_INFO.VALID = 1111111111111111b;
      tmp_seamreport.TEE_TCB_INFO.MRSIGNERSEAM = CRPL_TEE_TCB_INFO.MRSIGNERSEAM;
      tmp_seamreport.TEE_TCB_INFO.ATTRIBUTES = CRPL_TEE_TCB_INFO.ATTRIBUTES;
}
tmp_seamreport.REPORTMACSTRUCT.TEE_TCB_INFO_HASH = SHA384(tmp_seamreport.TEE_TCB_INFO);
```

```
    // Copy TEE_INFO_HASH and REPORTDATA from memory
    tmp_seamreport.REPORTMACSTRUCT.TEE_INFO_HASH = *tmp_tee_info_hash_la;
    tmp_seamreport.REPORTMACSTRUCT.REPORTDATA = *tmp_reportdata_la;

    // Compute MAC on the first 224B of REPORTMACSTRUCT
    tmp_report_key = CR_REPORT_KEY2;
    tmp_seamreport.REPORTMACSTRUCT.MAC = HMAC-SHA256(tmp_report_key, &tmp_seamreport.REPORTMACSTRUCT, 224);

    // Copy out report to memory
    *tmp_seamreport_la = tmp_seamreport;
END:
```

Flags Affected

ZF: Set if failed due to an error (see error codes below); otherwise, it is cleared to zero.

CF, PF, AF, OF, SF: Cleared to zero.

Error Codes

- SEAM_SUCCESS (0x00): SEAMREPORT is successful.
- SEAM_INVALID_REPORT_TYPE (0x01): Invalid report type.

64-Bit Mode Exceptions

#GP(0):	If a memory address is non-canonical form.
	If a memory operand is not properly aligned.
#PF(fault-code):	If a page fault occurs in accessing memory operands.

SEAMRET — Return to Legacy VMX Root Operation

Opcode/ Instruction	Op/ En	Description
66 0F 01 CD SEAMRET	ZO	Return to legacy VMX root operation from SEAM VMX root operation.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
ZO	NA	NA	NA	NA

Description

This instruction helps the SEAM VMX root software exit from SEAM VMX root operation and resume execution of the VMM software in legacy VMX root operation.

SEAMRET from the P-SEAMLDR clears the current VMCS structure pointed to by the current-VMCS pointer. A VMM that invokes the P-SEAMLDR using SEAMCALL must reload the current-VMCS, if required, using the VMPTRLD instruction.

Operation

```
IF inSEAM==0 or ((IA32_EFER.LMA & CS.L) == 0) or in VMX non-root operation
    THEN #UD;
ELSIF CPL > 0
    THEN #GP(0);
ELSIF current-VMCS pointer is not valid
    THEN VMfailInvalid;
Check settings of VMX controls and host-state area;
IF invalid settings
   THEN
      VMfailValid(VM entry with invalid VMX-control field(s)) or
      VMfailValid(VM entry with invalid host-state field(s)) as appropriate;
   ELSE
      Attempt to load guest state including event inhibits (NMI, SMI, etc.)
      and PDPTRs as appropriate based on entry controls;
      Clear address-range monitoring;
      IF failure in checking guest state or PDPTRs
         THEN VM entry fails      // see Section 26.8 of the Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3C
         ELSE
            Attempt to load MSRs from VM-entry MSR-load area;
            IF failure
               THEN VM entry fails   // see Section 26.8 of the Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3C
               ELSE
                  current-VMCS = current-VMCS.VMCS-link-pointer
                  IF inP_SEAMLDR == 1
                        THEN
                              If current-VMCS != FFFFFFFF_FFFFFFFFH
                                    THEN
                                          Ensure data for VMCS referenced by current-VMC is in memory
                                          Initialize implementation-specific data in all VMCS referenced by current-VMCS
                                          Set launch state of VMCS referenced by current-VMCS to "clear"
                                          current-VMCS = FFFFFFFF_FFFFFFFFH
                                    FI;
                              inP_SEAMLDR = 0
```

```
                    Release P_SEAMLDR_MUTEX
                FI;
            inSEAM = 0 // Closes SEAMRR access
            VM entry succeeds;
        FI;
    FI;
FI;
```

// Further details of the operation of the VM-entry appear in Chapter 26 of the Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3C.

Flags Affected

The operation section uses the pseudo-functions VMsucceed, VMfail(ErrorNumber), VMfailInvalid, and VMfailValid(ErrorNumber). See the operation section and Section 2.1. See Section 2.2 for error numbers. The following error numbers can be reported by SEAMRET: 7, 8, and 26.

Protected Mode Exceptions

#UD	The SEAMRET instruction is not designed to be recognized in protected mode.

Real-Address Mode Exceptions

#UD	The SEAMRET instruction is not designed to be recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	The SEAMRET instruction is not designed to be recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The SEAMRET instruction is not designed to be recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If CPL > 0.
#UD	If logical processor is not in SEAM VMX root operation. If IA32_VMX_PROCBASED_CTLS3[5] is 0.

TDCALL — Call to VM Monitor from TD Guest

Opcode/ Instruction	Op/ En	Description
66 0F 01 CC TDCALL	ZO	Call to VM monitor by causing a VM exit.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
ZO	NA	NA	NA	NA

Description

This instruction is designed to allow guest privileged software to make a call for service into a VM monitor. The details of the programming interface for such calls are Intel TDX module-specific; this instruction can cause a VM exit, registering the appropriate, exit reason. Use of this instruction outside VMX non-root operation can cause #UD fault.

Operation

IF not in VMX non-root operation THEN #UD;
ELSIF CPL > 0 THEN #GP(0);
ELSE VM exit with exit reason TDCALL (4DH) and zero exit qualification

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If CPL > 0.
#UD	If logical processor is not in VMX non-root operation.

Real-Address Mode Exceptions

#GP(0)	If CPL > 0.
#UD	If logical processor is not in VMX non-root operation.

Virtual-8086 Mode Exceptions

#GP(0)	If CPL > 0.
#UD	If logical processor is not in VMX non-root operation.

Compatibility Mode Exceptions

#GP(0)	If CPL > 0.
#UD	If logical processor is not in VMX non-root operation.

64-Bit Mode Exceptions

#GP(0)	If CPL > 0.
#UD	If logical processor is not in VMX non-root operation. If IA32_VMX_PROCBASED_CTLS3[5] is 0.

VMRESUME/VMLAUNCH — Resume/Launch Virtual Machine

The following additional actions can be performed by VMRESUME/VMLAUNCH in SEAM VMX root operation:

1. If the "enable EPT" VM-execution control is 0, then cause a VM entry failure due to invalid control field.
2. Shared-EPTP execution control must not set any reserved bits.

CHAPTER 3
INTEL® SGX INSTRUCTION SET EXTENSIONS

The ENCLU instruction defines a new leaf, EVERIFYREPORT2, that can be used to help verify the MAC on reports generated using SEAMOPS[SEAMREPORT]. CPUID leaf 12, sub-leaf 0, EAX bit 7 is set to 1 to help enumerate support for the EVERIFYREPORT2 leaf of ENCLU. The EVERIFYREPORT2 instruction layout is described in Table 3-1.

Table 3-1. EVERIFYREPORT2 Instruction Layout

Instruction	EAX	RBX	RCX	RDX
EVERIFYREPORT2	08H	REPORTMACSTRUCT effective address.		

3.1 EVERIFYREPORT2 LEAF DESCRIPTION

Opcode / Instruction	Description
EAX = 08H ENCLU[EVERIFYREPORT2]	Verifies a cryptographic report of the TD. RBX holds the address of a REPORTMACSTRUCT.

Description

This enclave mode instruction enables verification of a cryptographic REPORTMACSTRUCT that describes the contents of a TD, and the REPORTMACSTRUCT can be used to determine that the TEE described in the TDREPORT was running on the same platform.

Table 3-2. EVERIFYREPORT2 Memory Parameter Information

Memory Parameter	Permissions	Semantics
[RBX]REPORTMACSTRUCT	R	Enclave Access

The instruction flow is designed as follows:

1. Verify alignment requirements of the operands.
2. Validate the operand (RBX) is inside the enclave.
3. Compute a MAC over REPORTMACSTRUCT structure.
4. Determine if the computed MAC matches the MAC attached to the REPORTMACSTRUCT structure.

Operation

Table 3-3. Temp Variables in EVERIFYREPORT2 Operational Flow

Variable Name	Type	Size	Description
TMP_CURRENTSECS	Effective Address	32/64 Bytes	The address of the SECS for the currently executing enclave.
TMP_REPORTMACSTRUCT	REPORTMACSTRUCT	256 Bytes	Cryptographic Report of the TEE.
TMP_MAC	MAC	32 Bytes	MAC over REPORT calculated by instruction.

```
// check alignment of REPORTMACSTRUCT
IF (DS:RBX is not 256 Byte aligned) #GP(0);

// check to see if REPORTMACSTRUCT is inside the current enclave
IF (DS:RBX is not within CR_ELRANGE) #GP(0);

// make sure DS:RBX is read accessible
<< DS:RBX should be read accessible >>

// read EPCM VALID, PENDING, MODIFIED, BLOCKED, PT, R, W, X, and ENCLAVESECS
// fields atomically check that DS:RBX is a valid and accessible EPC page
IF ((DS:RBX does not resolve to an EPC address) OR
   (EPCM(DS:RBX).VALID = 0) OR
   (EPCM(DS:RBX).PENDING = 1) OR
   (EPCM(DS:RBX).MODIFIED = 1) OR
   (EPCM(DS:RBX).BLOCKED = 1) OR
   (EPCM(DS:RBX).R is 0) OR
   (EPCM(DS:RBX).PT != PT_REG) OR
   (EPCM(DS:RBX).ENCLAVESECS != CR_ACTIVE_SECS) OR
   (EPCM(DS:RBX).ENCLAVEADDRESS != (DS:RBX & ~0xFFF)))
{
   #PF(DS:RBX);
}

// Create local/protected copy of REPORTMACSTRUCT
TMP_REPORTMACSTRUCT[255:0B] = DS:RBX[255:0B];

// Verify REPORTMACSTRUCT header
IF ((TMP_REPORTMACSTRUCT.TYPE != 0x81) OR   // TDX implemented by Intel TDX module
   (TMP_REPORTMACSTRUCT.REPORTTYPE.SUBTYPE != 0x00) OR
   (TMP_REPORTMACSTRUCT.REPORTTYPE.VERSION != 0x00) OR
   (TMP_REPORTMACSTRUCT.RESERVED != 0))
{
      RFLAGS.ZF = 1;
      RAX = SGX_INVALID_REPORTMACSTRUCT;
      goto EXIT;
}

// Verify CPUSVN is a valid value
IF (TMP_REPORTMACSTRUCT.CPUSVN is unsupported by the CPU) {
     RFLAGS.ZF = 1;
     RAX = SGX_INVALID_CPUSVN;
     goto EXIT
}

// Verify MAC on REPORTMACSTRUCT
TMP_MAC = MAC(CR_REPORT_KEY2, TMP_REPORTMACSTRUCT[223:0B], 224);
IF (TMP_MAC != TMP_REPORTMACSTRUCT.MAC)
{
   RFLAGS.ZF = 1;
   RAX = SGX_INVALID_REPORTMACSTRUCT;
   goto EXIT
}
RAX=0;
```

```
RFLAGS.ZF=0;

EXIT:
RFLAGS.CF=0;
RFLAGS.PF=0;
RFLAGS.AF=0;
RFLAGS.OF=0;
RFLAGS.SF=0;
```

Flags Affected

None.

Error Codes

- SGX_INVALID_CPUSVN (32): If REPORTMACSTRUCT.CPUSVN is an unsupported value.
- SGX_INVALID_REPORTMACSTRUCT (28): REPORTMACSTRUCT included illegal values or MAC verification failed.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If the DS segment is unusable.
	If a memory operand is not properly aligned.
#PF(fault-code)	If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

#GP(0)	If a memory address is in a non-canonical form.
	If a memory operand is not properly aligned.
#PF(fault-code)	If a page fault occurs in accessing memory operands.