Intel® Architecture Memory Encryption Technologies

Specification

April 2025

Revision 1.6

Document Number: 336907-006US

Notices & Disclaimers

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Your costs and results may vary.

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 which includes subject matter disclosed herein.

All product plans and roadmaps are subject to change without notice.

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

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Code names are used by Intel to identify products, technologies, or services that are in development and not publicly available. These are not "commercial" names and not intended to function as trademarks.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document, with the sole exception that you may publish an unmodified copy. You may create software implementations based on this document and in compliance with the foregoing that are intended to execute on the Intel product(s) referenced in this document. No rights are granted to create modifications or derivatives of this document.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

Contents

Figures

Tables

Revision History

Revision Number	Description	Date
1.0	• Initial release of the document.	December 2017
1.2	• Additional details added.	April 2019
1.3	• Added support for 256b keys and KeyID0/TME encryption bypass.	April 2021
1.4	• Naming change: MKTME is now known as TME-MK.	August 2022
1.5	• PCONFIG changes to faulting behaviors, and the MTRR clarification • Added Integrity bit information to Capability MSR, Activate MSR and PCONFIG • Added TDX info	October 2024
1.6	• Clarified behavior of TME policy bits in IA32_TME_ACTIVATE MSR with respect to algorithms with integrity.	April 2025

Terminology

TME (Total Memory Encryption): This is a baseline capability for memory encryption with a single ephemeral key.

TME-MK (Total Memory Encryption-Multi-Key): Add support to use multiple keys for page granular memory encryption with additional support for software provisioned keys.

1 Introduction

This document describes the memory encryption support available beginning with the 3rd generation Intel® Xeon® Scalable Processor Family. Note that Intel platforms support many different types of memory and not all SoCs will support this capability for all types of memory. Initial implementation is focused on traditional DRAM.

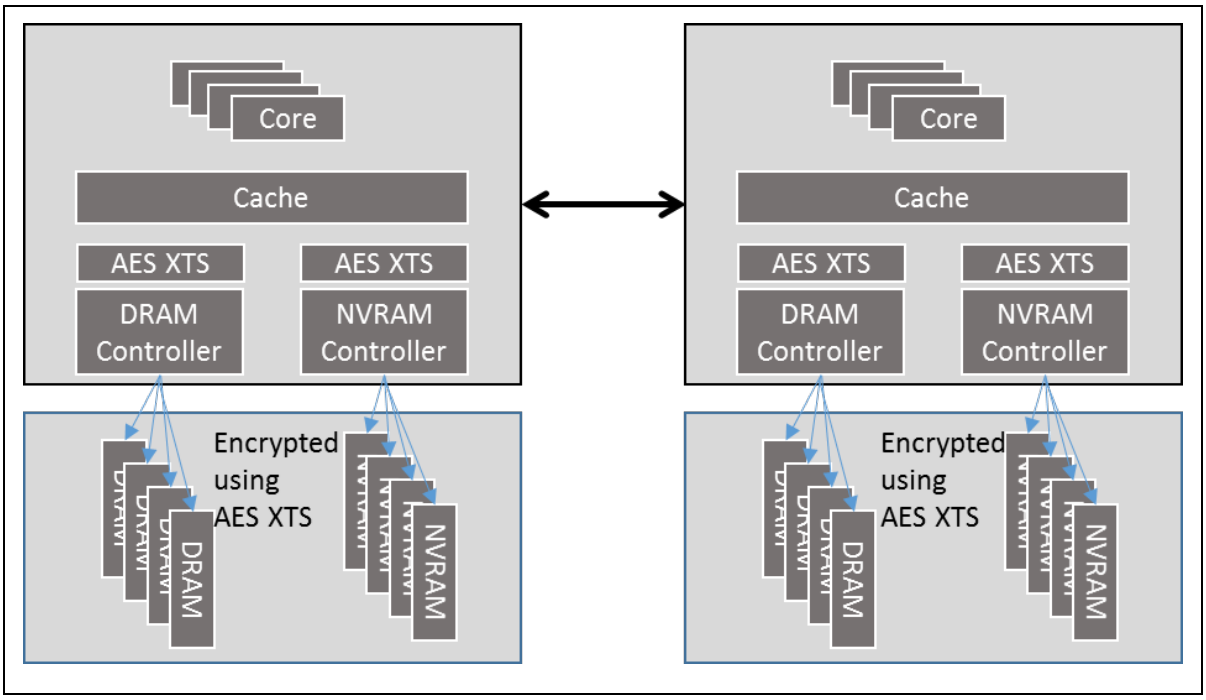
Total Memory Encryption (TME) – the capability to encrypt the entirety of physical memory of a system. This capability is typically enabled in the very early stages of the boot process with a small change to BIOS and once configured and locked, will encrypt all the data on external memory buses of an SoC using the NIST standard AES-XTS algorithm with 128-bit keys or 256-bit keys depending on the algorithm availability and selection. The encryption key used for TME uses a hardware random number generator implemented in the Intel SoC, and the keys are not accessible by software or using external interfaces to the Intel SoC. TME capability is intended to provide protections of AES-XTS to external memory buses and DIMMs. The architecture is flexible and will support additional memory protection schemes in the future. This capability, when enabled, is intended to support (unmodified) existing system and application software. Overall performance impact of this capability is likely to be relatively small and is highly dependent on workload.

Total Memory Encryption-Multi-Key (TME-MK) builds on TME and adds support for multiple encryption keys. The SoC implementation supports a fixed number of encryption keys, and software can configure the SoC to use a subset of available keys. Software manages the use of keys and can use each of the available keys for encrypting any page of the memory. Thus, TME-MK allows page granular encryption of memory. By default, TME-MK uses the TME encryption key unless explicitly specified by software. In addition to supporting a CPU generated ephemeral key (not accessible by software or using external interfaces to the SoC), TME-MK also supports software provided keys. Software provided keys are particularly useful when used with non-volatile memory or when combined with attestation mechanisms and/or used with key provisioning services. In a virtualization scenario, we anticipate the VMM or hypervisor managing the use of keys to transparently support legacy operating systems without any changes (thus, TME-MK can also be viewed as TME virtualization in such a deployment scenario). An OS may be enabled to take additional advantage of the TME-MK capability both in native and in a virtualized environment. When properly enabled, TME-MK is available to each guest OS in a virtualized environment, and the guest OS can take advantage of TME-MK in the same way as a native OS.

2 Introduction to Total Memory Encryption (TME)

The diagram below gives an overview of total memory encryption in a two-socket configuration. Actual implementation may vary.

Figure 2-1. Two-Socket Configuration of TME



The AES XTS encryption engine is in the direct data path to external memory buses and therefore, all the memory data entering and/or leaving the SoC on memory buses is encrypted using AES XTS. The data inside the SoC (in caches, etc.) remains plain text and supports all the existing software and I/O models.

In a typical deployment, the encryption key is generated by the CPU and therefore is not visible to the software. When the system is configured with NVRAM, if the NVRAM is to be treated as DRAM, then it can also use CPU generated keys. However, if NVRAM is to be treated as non-volatile memory, there is an option to have the same key generated/reused across platform power cycles/reboots.

3 Introduction to Total Memory Encryption-Multi-Key (TME-MK)

3.1 High-level Architecture

The high-level architecture of TME-MK is shown in the figure below.

Figure 3-1. High-level Architecture of TME-MK

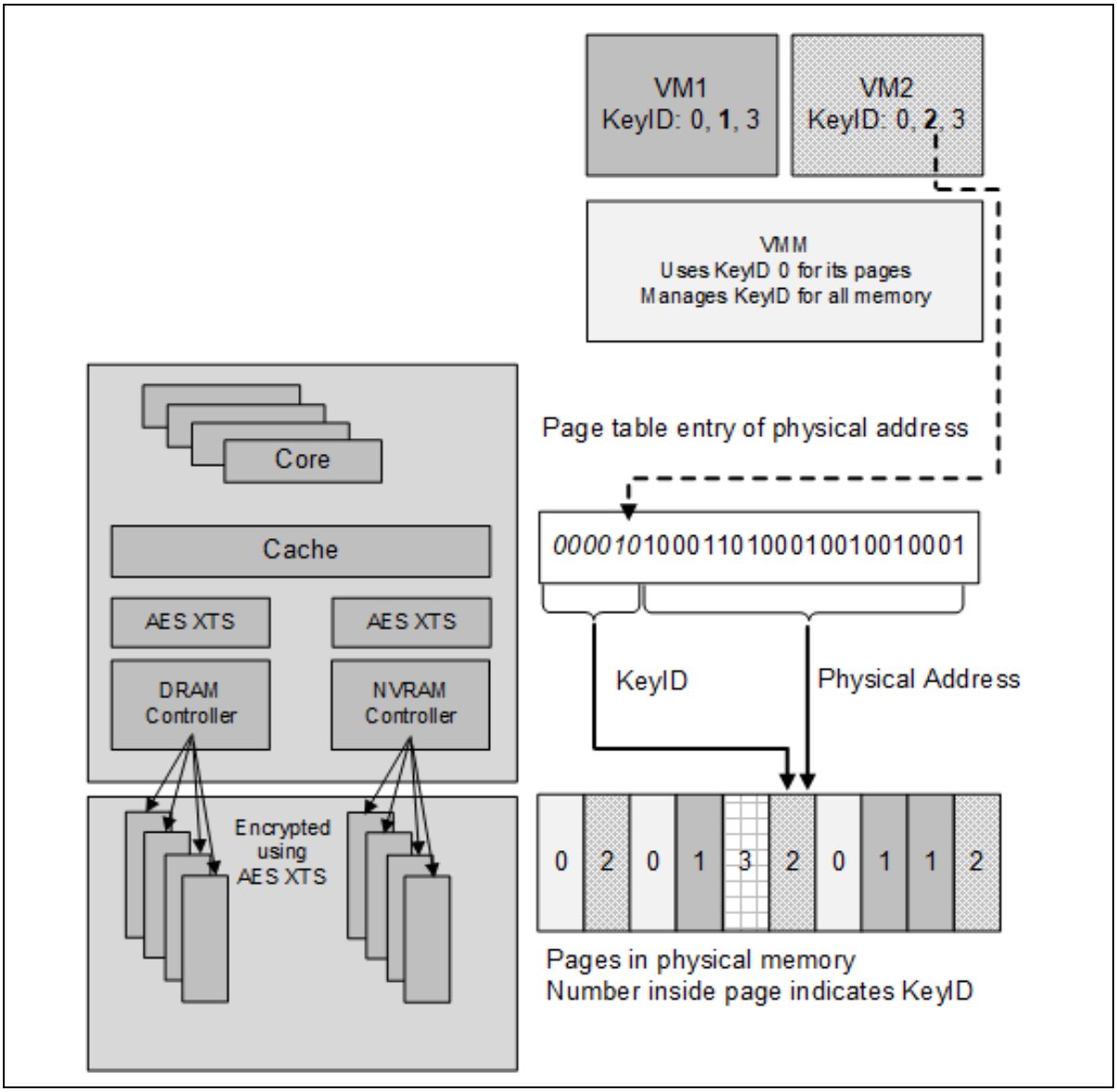


Figure 3-1 shows the basic architecture of TME-MK which shares basic hardware architecture with TME, with the exception that AES XTS now supports multiple keys. The right side of the figure shows the use of TME-MK in a virtualized environment, though architecture supports use of TME-MK in a native OS deployment scenario as well. In this example we show one hypervisor/VMM and two VMs. By default, a hypervisor uses KeyID 0 (same as TME), though it can use a different KeyID for its own memory as well. VM1 uses KeyID1 for its own private pages, and VM2 is using KeyID 2 for its own private pages. Additionally, VM1 can always use KeyID 0 (TME KeyID) for any page and is also opting to use KeyID 3 for shared memory between itself and VM2. The KeyID is included in the Page Table Entry as upper bits of the physical address field. As in this example, KeyID 2 is shown. The remainder of the bits in the physical address field are used to actually address bits in the memory. The figure shows one possible page assignment along with the KeyID for illustration purposes, though in this case the hypervisor has full freedom to use any KeyID with any pages for itself or any of its guest VMs. Note that the idea of oversubscribing physical address bits in the page table extends to other page tables as well, including IA page tables and IOMMU page tables. The KeyID remains part of the physical address bits everywhere in the SoC, with the exception of a tweak for AES XTS and on external memory buses. The KeyID is not used outside of the SoC or in the tweak for AES XTS.

3.2 TDX Enhancements

The TME-MK hardware is also used by Intel® Trust Domain Extensions (TDX) as a way to protect the memory of a Trust Domain (TD). To enable this usage, there are several extensions made to the hardware interface to allow for the TDX Module to maintain a separate pool of KeyID's dedicated for TD usage.

Since the same TME-MK encryption engine and KeyIDs are used for legacy TME-MK operation and for TDX operation, there's a need to enumerate and configure the allocation of activated KeyID space between the two technologies.

4 TME & TME-MK: Enumeration and Control Registers

This information is applicable only to CPUs that enumerate TME and/or TME-MK capabilities.

4.1 Enumeration

TME and TME-MK capability is exposed to the BIOS/Software via the MSR described in this section. The maximum number of keys available/supported in the processor for TME-MK are enumerated. BIOS will need to activate this capability via an MSR (described later) and it must select the number of keys to be supported/used for TME-MK, as well as TDX during the early boot process. Upon activation, all memory (except memory in the TME Exclusion range) attached to the CPU/SoC is encrypted using AES-XTS with a 128-bit or 256-bit ephemeral key (platform key) that is generated by the CPU on every boot. Note that this behavior is applicable only when TME encryption is not bypassed (using bit 31 in the IA32_TME_ACTIVATE MSR). If TME encryption is bypassed, all accesses with KeyID0 will bypass encryption/decryption.

Intel processors support external memory controllers. These memory controllers may be attached to the processor via coherent buses such as the Intel® Ultra Path Interconnect (Intel® UPI) or Compute Express Link (CXL). TME-MK enumeration can be used to discover the capabilities of the Intel processor and some of the memory attached to the integrated memory controller, but does not necessarily represent external memory controller features, or some types of memory attached to the integrated controller. The memory regions that are capable of being protected by CPU cryptographic capabilities are communicated to the system software via a new UEFI memory attribute, EFI_MEMORY_CPU_CRYPTO, introduced in UEFI 2.8. If this flag is set, the memory region is capable of being protected with the CPU's memory cryptographic capabilities. If this flag is cleared, the memory region is not capable of being protected with the CPU's memory cryptographic capabilities or the CPU does not support CPU memory cryptographic capabilities. System software must consult the attribute to determine the ranges that can be encrypted using TME-MK.

4.1.1 TME

CPUID.TME (CPUID.(EAX=07H, ECX=0H): ECX[13]) enumerates the existence of these five architectural MSRs and their MSR addresses:

- IA32_TME_CAPABILITY – Address 981H
- IA32_TME_ACTIVATE – Address 982H
- IA32_TME_EXCLUDE_MASK – Address 983H
- IA32_TME_EXCLUDE_BASE – Address 984H
- IA32_MKTME_KEYID_PARTITIONING – Address 0x87

4.1.2 TME-Multi-key

The CPUID.TME bit indicates the presence of the TME_CAPABILITY MSR, and that MSR will further enumerate the TME characteristics as well as the TME-MK availability and characteristics. TME-MK is enabled/configured by BIOS using the IA32_TME_ACTIVATE MSR. TME-MK requires TME and therefore cannot be enabled without enabling TME.

4.1.3 Memory Encryption Capability MSR (IA32_TME_CAPABILITY)

Table 4-1. IA32_TME_CAPABILITY MSR – Address 981H

Register Address	Architectural MSR Name and Bit Fields	MSR/Bit Description	Comment
981H	IA32_TME_CAPABILITY MSR	Memory Encryption Capability MSR	One MSR for TME and TME-MK.
	0	Support for AES-XTS 128-bit encryption algorithm.	NIST standard.
	1	AES-XTS 128-bit encryption algorithm with at least 29b SHA-3 based integrity.	
	2	Support for AES-XTS 256-bit encryption algorithm.	NIST standard.
	3	AES-XTS 256-bit encryption algorithm with at least 29b SHA-3 based integrity.	
	30:4	Reserved	
	31	TME encryption bypass supported.	
	35:32	MK_TME_MAX_KEYID_BITS Number of bits which can be allocated for usage as key identifiers for multi-key memory encryption. Zero if TME-MK is not supported.	4 bits allow for a max value of 15, which can address 32K keys.
	50:36	MK_TME_MAX_KEYS Indicates the maximum number of keys which are available for usage. This value may not be a power of 2. Zero if TME-MK is not supported.	KeyID 0 is specially reserved and is not accounted for in this field. Max value is 32K-1 keys.
	63:51	Reserved	

4.1.4 CPUID Reporting of MAX_PA_WIDTH

CPUID enumeration of MAX_PA_WIDTH (leaf 80000008.EAX) is unaffected by TME-MK activation and will continue to report the maximum number of physical address bits available for software to use, irrespective of the number of KeyID bits.

4.2 Memory Encryption Configuration and Status Registers

4.2.1 Activation MSR (IA32_TME_ACTIVATE)

This MSR is used to lock the following MSRs. Any write to the following MSRs will be ignored after they are locked. The lock is reset when CPU is reset.

- IA32_TME_ACTIVATE
- IA32_TME_EXCLUDE_MASK
- IA32_TME_EXCLUDE_BASE

Note: IA32_TME_EXCLUDE_MASK and IA32_TME_EXCLUDE_BASE MSRs are expected to be configured before the IA32_TME_ACTIVATE MSR.

To enable TME-MK, the Hardware Encryption Enable bit in the IA32_TME_ACTIVATE MSR must be set, and bits 35:32 must have a non-zero value (which will specify the number of KeyID bits configured for TME-MK).

Table 4-2. IA32_TME_ACTIVATE MSR – Address 982H

Register Address	Architectural MSR Name and Bit Fields	MSR/Bit Description	Comment
982H	IA32_TME_ACTIVATE MSR	Memory Encryption Activation MSR	
	0	Lock RO – Will be set upon successful WRMSR (or first SMI); written value ignored.	
	1	Hardware Encryption Enable (TME Enabled depending on TME Encryption Bypass Enable (bit 31))	This bit also enables TME-MK; TME-MK cannot be enabled without enabling encryption hardware.
	2	Key Select: 0 – Create a new TME key (expected cold/warm boot). 1 – Restore the TME key from storage (expected when resume from standby).	
	3	Save TME key for Standby: Save key into storage to be used when resume from standby.	May not be supported in all CPUs.

Register Address	Architectural MSR Name and Bit Fields	MSR/Bit Description	Comment
	7:4	TME Policy/Encryption Algorithm: Only algorithms enumerated in the IA32_TME_CAPABILITY MSR are allowed, any other values are invalid and will result in #GP. Additionally, any algorithm that supports integrity checking is not allowed to be used for TME even if it is listed as allowed in the IA32_TME_CAPABILITY MSR and will result in #GP.	TME Encryption algorithm to be used.
	30:8	Reserved	
	31	TME Encryption Bypass Enable When encryption hardware is enabled: • Total Memory Encryption is enabled using CPU generated ephemeral key based on hardware random number generator when this bit is set to 0. • Total Memory Encryption is bypassed (no encryption/decryption for KeyID0) when this bit is set to 1. On some processors, bypassing TME encryption can provide performance benefits to accesses made with KeyID 0 by avoiding the latency of decryption or encryption and decryption. Software must inspect the Hardware Encryption Enable (bit 1) and TME Encryption Bypass Enable (bit 31) to determine if TME encryption is enabled.	
	35:32	Reserved if TME-MK is not enumerated.	
		MK_TME_KEYID_BITS The number of key identifier bits to allocate to TME-MK usage. Writing a value greater than MK_TME_MAX_KEYID_BITS will result in #GP. Writing a non-zero value to this field will #GP if bit 1 of EAX (Hardware Encryption Enable) is not also set to ‘1, as encryption hardware must be enabled to use TME-MK.	

Register Address	Architectural MSR Name and Bit Fields	MSR/Bit Description	Comment
		Example: To support 255 keys, this field would be set to a value of 8.	
	39:36	TDX_RESERVED_KEYID_BITS The number of key identifier bits to allocate to TDX usage, which are allocated from the most significant bit downward. Writing a value greater than MK_TME_KEYID_BITS will result in a #GP.	Note: these bits are a subset of the overall KeyID bits which are declared by MK_TME_MAX_KEYID_BITS.
	47:40	Reserved	
	63:48	MK_TME_CRYPTO_ALGS Bit 48: AES-XTS 128 Bit 49: AES-XTS-128 with at least 29b integrity Bit 50: AES-XTS-256 Bit 51: AES-XTS-256 with at least 29b integrity Bit 63:52: Reserved (#GP) Bitmask for BIOS to set which encryption algorithms are allowed for TME-MK, will be later enforced by the key loading ISA (‘1 = allowed).	

4.2.2 IA32_TME_ACTIVATE WRMSR Response and Error Handling

Table 4-3. IA32_TME_ACTIVATE WRMSR Response and Error Handling

Conditions	Response
WRMSR when not enumerated.	#GP(0)
WRMSR while lock status = 1.	#GP(0)
WRMSR with 63:8 (reserved) ≠ 0.	#GP(0)
WRMSR with Unsupported policy value (IA32_TME_CAPABILITY[IA32_TME_ACTIVATE[7:4]]= 0).	#GP(0)
WRMSR with enabled=0.	TME disabled, MSR locked subsequent RDMSR returns x..x01b.
WRMSR with enabled=1 and key select=0 (new key); RNG success.	TME enabled and MSR locked subsequent RDMSR returns x..x011b.
WRMSR with enabled=1 and key select=0; RNG fail	Not enabled subsequent RDMSR returns x..x000b.
WRMSR with enabled=1 and key select=1; Non-zero key restored from CPU.	TME enabled and MSR locked subsequent RDMSR returns x..x111b.
WRMSR with enabled=1 and key select=1; Fail - Zero key restored from CPU.	Not enabled subsequent RDMSR returns x..x100b.

WRMSR with any other legal values.	Subsequent RDMSR returns written values + lock status=1.
If MK_TME_KEYID_BITS > MK_TME_MAX_KEYID_BITS	#GP(0)
If MK_TME_KEYID_BITS > 0 && (TME) Enable == 0 (TME must be enabled at the same point as MK-TME).	#GP(0)
If MK_TME_KEYID_BITS > 0 and TME is not successfully activated (lock is not set).	Write not committed.
If MK_TME_CRYPTO_ALGS reserved bits are set.	#GP(0)
If TDX_RESERVED_KEYID_BITS > MK_TME_KEYID_BITS	#GP(0)

4.2.3 Core Address Masking MSR (MK_TME_CORE_ACTIVATE)

This is a BIOS only MSR.

After successful activation using the IA32_TME_ACTIVATE MSR, this register should be written on each physical core with a value of 0 in EDX:EAX; failure to do so may result in unpredictable behavior. Accesses to this MSR will #GP if TME-MK is not supported.

BIOS is expected to write to this MSR on each core after doing TME-MK activation. The first SMI on each core will also cause this value to be synchronized with the package MSR value.

Table 4-4. MK_TME_CORE_ACTIVATE MSR – Address 9FFH

Register Address	MSR Name and Bit Fields	MSR/Bit Description	Comment
9FFH	MK_TME_CORE_ACTIVATE MSR	This MSR will #GP if TME-MK is not supported.	
	31:0	Reserved	
	35:32	MK_TME_KEYID_BITS (read only) The number of key identifier bits allocated to TME-MK usage. This is a read-only field. #GP on a non-zero write.	Will be shadowed from the package MSR value on write.
	39:36	TDX_RESERVED_KEYID_BITS (read only) The number of key identifier bits allocated to TDX usage. This is a read-only field. #GP on a non-zero write.	Will be shadowed from the package MSR value on write.
	63:40	Reserved	

4.2.4 IA32_MKTME_KEYID_PARTITIONING MSR

This is a read-only MSR.

After successful activation using the IA32_TME_ACTIVATE MSR, this register should be consulted by software when trying to determine the number of KeyIDs which are

available for TME-MK or TDX. It is important for platform software to use this MSR to determine the number of KeyIDs as there may be cases where KeyIDs are required to be reserved for internal usage and thus not be available for general usage.

Table 4-5. IA32_MKTME_KEYID_PARTITIONING MSR – Address 87H

Register Address	Architectural MSR Name and Bit Fields	MSR/Bit Description	Comment
87H	IA32_MKTME_KEYID _PARTITIONING MSR	TME-MK KeyID Partitioning MSR	
	31:0	NUM_MKTME_KEYIDS	Number of activated KeyIDs available for TME-MK use. Note: KeyID 0 is reserved for TME and will not be included.
	63:32	NUM_TDX_KEYIDS	Number of activated KeyIDs available for TDX use. Note: KeyID 0 is reserved for TME and will not be included.

4.2.5 Exclusion Range MSRs

TME and TME-MK (for KeyID=0 only) support one exclusion range to be used for special cases. (Note: For all KeyIDs other than 0, the TME Exclusion Range does not apply to TME-MK.) The range of physical addresses specified in this MSR does not apply memory encryption described in this document. This range is primarily intended to be used for memory not available to the OS and typically configured by BIOS. However, TME/TME-MK (for KeyID=0) architecture does not place any restrictions on the use of the exclusion range. The software is able to determine this range by reading the MSR. The definition of this range follows the definition of many range registers implemented in Intel processors.

Table 4-6. IA32_TME_EXCLUDE_MASK MSR – Address 983H

Register Address	MSR Name and Bit Fields	MSR/Bit Description	Comment
983H	IA32_TME_EXCLUDE_ MASK MSR		
	10:0	Reserved	
	11	Enable - When set to '1', then IA32_TME_EXCLUDE_BASE and IA32_TME_EXCLUDE_MASK MSRs are used to define an exclusion region for TME/TME-MK (for KeyID=0).	
	MAXPHYSADDR-1:12	TMEEMASK - This field indicates the bits that must match TMEEBASE in order to qualify as a TME/TME-MK (for KeyID=0) exclusion memory range access.	
	63:MAXPHYSADDR	Reserved; must be zero.	

Table 4-7. IA32_TME_EXCLUDE_BASE MSR – Address 984H

Register Address	MSR Name and Bit Fields	MSR/Bit Description	Comment
984H	IA32_TME_EXCLUDE_ BASE MSR		
	11:0	Reserved	
	MAXPHYSADDR-1:12	TMEEBASE - Base physical address to be excluded for TME/TME-MK (for KeyID=0) encryption.	
	63:MAXPHYSADDR	Reserved; must be zero.	

Note: Writing '1' into bits above the max supported physical size will result in #GP.

The IA32_TME_EXCLUDE_MASK MSR must define a contiguous region. WRMSR will #GP if the TMEEMASK field does not specify a contiguous region.

These MSRs are locked by the IA32_TME_ACTIVATE MSR. If lock=1, then WRMSR to IA32_TME_EXCLUDE_MASK/IA32_TME_EXCLUDE_BASE MSRs will result in #GP.

5 Runtime Behavior of TME-MK

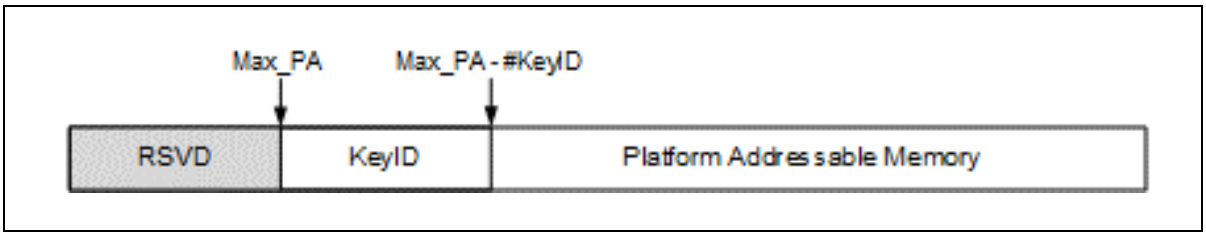
After TME-MK is activated by the BIOS, there are a number of changes to the runtime behavior of the processor which are described in this section.

5.1 Changes to Specification of Physical Address

The most significant change for TME-MK is the repurposing of physical address bits to communicate the KeyID to the encryption engine(s) in the memory controller(s). This change necessitates a number of other hardware and software changes in order to maintain proper behavior.

When TME-MK is activated, the upper bits of the platform physical address (starting with the highest order bit available as enumerated by the CPUID MAX_PA info) are repurposed for usage as a KeyID as shown below.

Figure 5-1. KeyID Usage



Additionally, when TDX KeyIDs have also been enabled, the KeyID space is further partitioned to accommodate KeyIDs which can only be used for TDX. KeyIDs are divided into up to 3 ranges:

- A single key, with KeyID value 0, is the legacy TME key, used as a platform shared memory encryption key (or cleartext if in TME bypass mode).
- A range of keys, with KeyID values 1 to NUM_MKTME_KEYIDS, used as legacy TME-MK keys. Note that a configuration may allocate all non-zero KeyIDs for TDX usage, in which case this range will be empty.
- A range of keys, with KeyID values NUM_MKTME_KEYIDS+1 to NUM_MKTME_KEYIDS + NUM_TDX_KEYIDS is used as TDX keys. Note that a configuration may allocate all non-zero KeyIDs for TME-MK usage, in which case this range will be empty.

Additionally, outside of Secure Arbitration Mode (SEAM), physical address bits which are associated with TDX-specific KeyIDs are treated as reserved bits and cannot be used by software (ex: on a CPU supporting 52b of PA, if there are 4 bits for TME-MK and 3 of those bits are for TDX, then bits 51:49 would be treated as reserved bits outside of SEAM). This ensures that only the SEAM module can create valid address references using TDX KeyIDs.

5.1.1 Note that when IA Paging

When IA paging is being used without EPT, the upper bits starting with MAX_PA for each level of the IA page table are repurposed for usage as KeyID bits. Similarly, the upper bits of the physical address in CR3 will be treated in the same manner.

Note that when EPT is active, IA paging does not generate/use platform physical addresses, instead it produces/uses guest physical addresses. Guest physical addresses are not modified by TME-MK and will continue to index into EPT page table walks as they did prior to enabling TME-MK.

5.1.2 EPT Paging

When EPT is enabled during VMX non-root operation, the upper bits for each level of EPT page walk are repurposed for usage as KeyID bits. Similarly, the upper bits of the physical address in EPTP will be treated in the same manner. Note that a guest OS may also use a KeyID in an IA page address, and full guest PA (including KeyID) is used by EPT.

5.1.3 Other Physical Addresses

Other physically addressed structures such as VMCS pointers, physically addressed bitmaps, etc., will receive similar treatment with the upper bits of the address starting with MAX PA being repurposed as KeyID bits. Note that any reserved bit checking remains unchanged, which means that the checking of these addresses will only be based upon the CPUID MAX_PA value.

5.1.4 Range Register Considerations

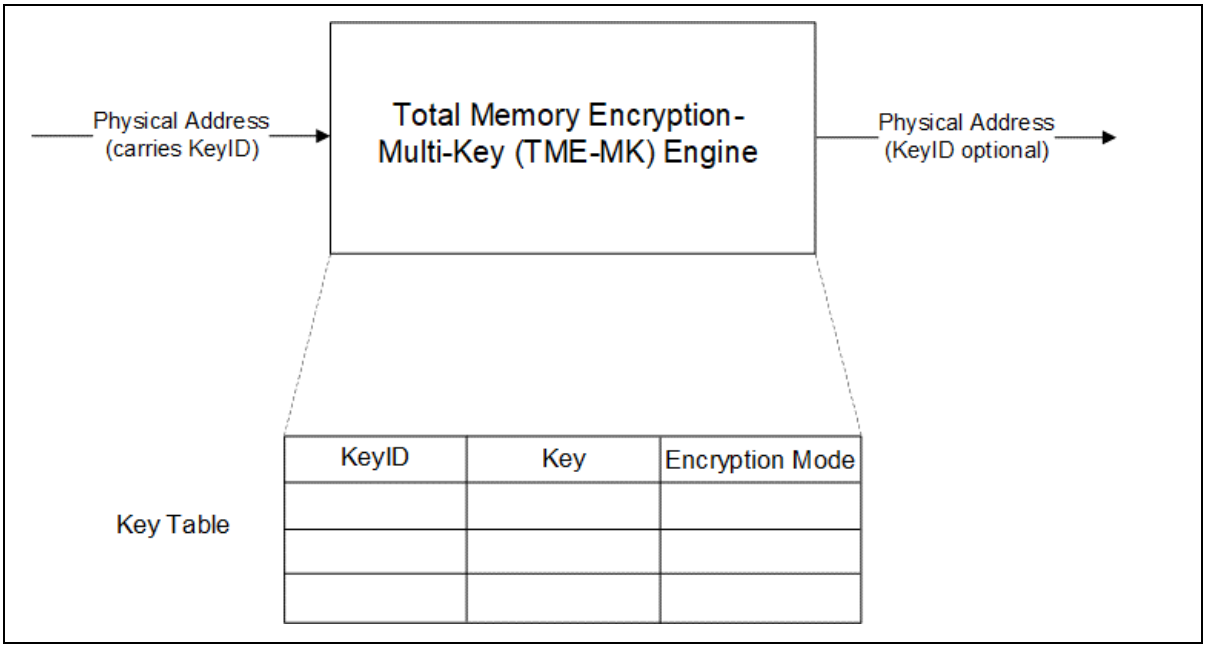
Range registers, such as the Memory Type Range Register (MTRRs), use a combination of a physical address base and mask register to check for matches and apply memory attribute behaviors. When programming these registers, it is important that the BIOS or system software understand the implications of KeyIDs in relation to these registers to ensure proper behavior. As an example, if an MTRR mask register is programmed with all of the KeyID bits set, the memory type for this range would only be applied for a single KeyID (likely KeyID 0) upon usage. This creates potential issues if this memory region is used with a non-zero KeyID, as it will not match on the MTRR and can result in receiving the default memory type (likely UC) which may not be desirable. In order to prevent this, KeyID bits should be cleared in the MTRR MASK register, which will allow it to be applied to an actual memory region regardless of KeyID.

6 TME-MK Key Programming

6.1 Overview

Figure 6-1 shows a high-level overview of the TME-MK engine meant to introduce the terminology that is used for the rest of the document and does not imply implementation.

Figure 6-1. TME-MK Engine Overview



The TME-MK engine maintains an internal key table not accessible by software to store the information (key and encryption mode) associated with each KeyID. Each KeyID may be associated with three encryption modes: Encryption using key specified, do not encrypt at all (memory will be plain text), or encrypt using TME Key. Future implementation may support additional encryption modes. PCONFIG is a new instruction that is used to program KeyID attributes for TME-MK. While initial implementation may only use PCONFIG for TME-MK, it may be extended in the future to support additional usages. Therefore, PCONFIG is enumerated separately from TME-MK.

6.2 PCONFIG Instruction

The PCONFIG instruction details are available in the latest Intel® 64 and IA-32 Architectures Software Developer's Manual.

7 Software Life Cycle: Managing Pages with KeyID

7.1 Overview

As mentioned earlier in the document, the KeyID is an integral part of the physical address, meaning it is not only present in page tables but is also present in the TLB, caches, etc. Therefore, software needs be aware of this and must take appropriate steps to maintain correctness of operations and security.

Note that while this section focuses on virtualization scenarios, the TME and TME-MK architecture is applicable to both native OS and virtualized environments, and for DRAM and NVRAM types of memory.

7.2 Restrictions and Cache Management

The hardware/CPU does not enforce coherency between mappings of the same physical page with different KeyIDs or encryption keys. System software is responsible for carefully managing the caches in regard to usage of key identifiers (KeyIDs) and maintaining cache coherency when the KeyID or a key associated with a physical page is changed by the software. Specifically, the CPU will treat two physical addresses that are identical except for the KeyID bits as two different physical addresses though these two addresses reference the same location in memory. Software must take necessary steps to ensure that this does not result in unpredictable or incorrect behavior or violate security properties desired. TME-MK retains the existing behavior of the caches and TLB for the entire physical address including the KeyID portion of the physical address and expects software to properly flush the caches and/or perform TLB shootdowns.

The sections below are intended to give examples of algorithms that shouldn't be used by software to ensure correctness and security. Please check the final version of this specification for any updated algorithms or requirements in this area.

7.3 General Software Guidance for Dealing with Aliased Address Mappings

The following list details some general guidelines for OS/VMM software vendors to consider when using TME-MK with more than the default single KeyID.

1. Software should avoid mapping the same physical address with multiple KeyIDs.
2. If software must map the same physical address with multiple KeyIDs, it should mark those pages as read-only, except for one KeyID.
3. If software must map the same physical address with multiple KeyIDs as read-write, then software must ensure that all writes are done with a single KeyID (this includes locked and non-locked writes that do not modify the data).

7.4 AddPage: Associating a KeyID to a Page

The following algorithm should be used by the OS/VMM when assigning a new KeyID to a physical page.

1. Program a new key for the KeyID, if not already programmed (using the PCONFIG instruction).
2. Map the physical page to the VMM's address space (with the new KeyID) by updating its paging structure entries (IA-PT), if not already mapped.
3. Ensure that the step 1 has successfully completed.
4. Zero-page contents via the new mapping (with new KeyID) to avoid data leakage between KeyID domains.
5. Make the page available to a new VM with the new KeyID set in the EPT page-table entry.

This will ensure against data leakage between KeyID domains, such as VMs with KeyIDs, when the KeyID is changed for a physical page (but the data is in clear in the CPU caches). The assumption is that before using this algorithm to assign a new KeyID, the software/VMM makes sure that the page was evicted correctly from the previous KeyID (using the algorithm defined in the next section).

Note: Guidance for usage of the PCONFIG instruction: PCONFIG is package scope and hence software is expected to execute PCONFIG on one LP on each package/socket. Software can use CPUID Leaf 0BH to determine the topology of the system which will indicate the physical packages present on the system.

7.5 EvictPage: Disassociating a KeyID from a Page

The following algorithm should be used by the OS/VMM when changing the KeyID of a physical page so that the current KeyID is no longer used with the page.

1. Steps to be completed before changing the KeyID:

   a) Make the physical page not accessible to the VM (by updating the EPT page-table entry).
   b) Invalidate all page mappings/aliases (the INVEPT instruction and IOMMU (VT-d) invalidation if page was mapped as device accessible) from the TLB (across the logical processors, with the old KeyID).
   c) Map the page to VMM address space (with the old KeyID) by updating its paging structure entries (IA-PT) if not already mapped.
   d) OS/VMM flushes dirty cache lines (for page using old KeyID) to prevent aliasing overwrite/data corruption.
      - Options: CLFLUSH, CLWB+fence, CLFLUSHOPT+fence or WBINVD.
      - Software can optionally avoid doing these flushes if it tracks page modification using EPT page-modification logging or accessed and dirty flags for EPT (optimization).

2. The page is now ready to be used with a new KeyID (example, using steps in the previous section).

This will ensure that no cache lines aliased by physical address exist in the CPU caches when the KeyID of the physical page is changed.

Note: Guidance for usage of the WBINVD instruction: The WBINVD instruction should be run on each socket if those invalidate all coherent caches on the sockets.

7.6 Paging by OS/VMM Example

Below is an example of a software sequence where the OS/VMM is reallocating a page from VM2 to VM3. VM2 memory uses KeyID2, and VM3 memory uses KeyID3.

1. Evict a page with KeyID2 from VM2 using the EvictPage algorithm described in section 7.5.
2. The OS/VMM reads the evicted page with KeyID2, encrypts the page contents with the Full Disk Encryption key (optional), and writes the page to disk/stores on a swap file (or in OS/VMM memory, using the VMM KeyID=0).
3. Add the evicted page to VM3 KeyID3 using the AddPage algorithm in section 7.4.

7.7 OS/VMM Access to Guest Memory

The OS/VMM can access guest memory (in clear) for emulation purposes (MMIO) by setting the guest KeyID bits in its paging structure entries (IA-PT). Note: OS/VMM should not program KeyID for MMIO pages (for TME-MK usages).

7.8 I/O Interactions

The OS/VMM can use the TME key (KeyID=0) to set up shared memory between the Guest VM and the VMM as needed for I/O purposes. For directed I/O (e.g., SR-IOV), the OS/VMM should program the KeyID as part of the physical addresses in IOMMU (VT-d) page tables corresponding to the KeyID as part of the physical addresses in EPT (for the Guest VM). This will allow DMAs to be able to access memory in clear without requiring changes to I/O devices and/or I/O drivers in the guest VM or OS/VMM.