Intel® Architecture Memory Protections for Confidential Computing

Technical Paper

November 2025

Revision 1.0

Document Number: 869103-0010

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.

Table of Contents

1 Introduction 6

2 Memory Encryption 7

2.1 Total Memory Encryption 7
2.2 Total Memory Encryption – Multi-Key (TME-MK) 8
2.3 TME-MK with Integrity 10

3 Confidential Computing Enhancements 12

3.1 SEAM vs TDX 12
3.2 SEAM Specific Keys & Addressing 12
3.3 SGX Memory Encryption Key 13
3.4 Memory Configuration Aliasing Checking 13
3.5 Memory & System Configuration Locking 13
3.6 KeyID Address Aliasing 13
3.7 TEE Ownership Indication 13
3.7.1 Access Control Checks 14
3.7.2 Memory Writes 16

4 Memory Protection Security Properties 17

4.1 Memory Protection Model 17
4.2 Addressing Dictionary Based Attacks 18
4.3 Addressing Cache Based Timing Side Channels 18

Revision History

Revision Number	Description	Date
1.0	• Initial release of the document.	November 2025

Terminology

SGX Intel® Software Guard Extensions

TDX Intel® Trust Domain Extensions

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.

TMEi-MK Total Memory Encryption-Multi-Key with Integrity: TME-MK with a 29b integrity value.

Li Logical Integrity Mode: A mode of TDX when used with only TME-MK.

Ci Cryptographic Integrity Mode: A mode of TDX when used with TME-MKi

1 Introduction

Memory protection protects data in use and is fundamental to Confidential Computing. This document describes the memory protection schemes supported by Intel products to provide additional protection to confidential computing technologies such as Intel® Software Guard Extensions and Intel® Trust Domain Extensions available beginning with the 3rd generation Intel® Xeon® Scalable Processor Family. There are three encryption schemes available:

1) Total Memory Encryption (TME) which encrypts the entirety of physical memory in a system

2) Total Memory Encryption-Multi-Key (TME-MK) supports encryption of memory using selectable encryption keys

3) Total Memory Encryption-Multi-Key with Integrity (TMEi-MK) allows the TME-MK sub-system to create/store/check a 28b integrity value for each cache line

Note: Intel platforms support many different types of memory protections and not all SoCs will support this capability for all types of memory.

A description of each of these modes is provided in section 2.

To support confidential computing, it is necessary to extend these basic encryption schemes to support additional mechanisms to provide enhanced protections and access controls. See section 3.

Section 4 outlines the expected security model that can be derived from the combination of these memory encryption technologies and additional protections.

2 Memory Encryption

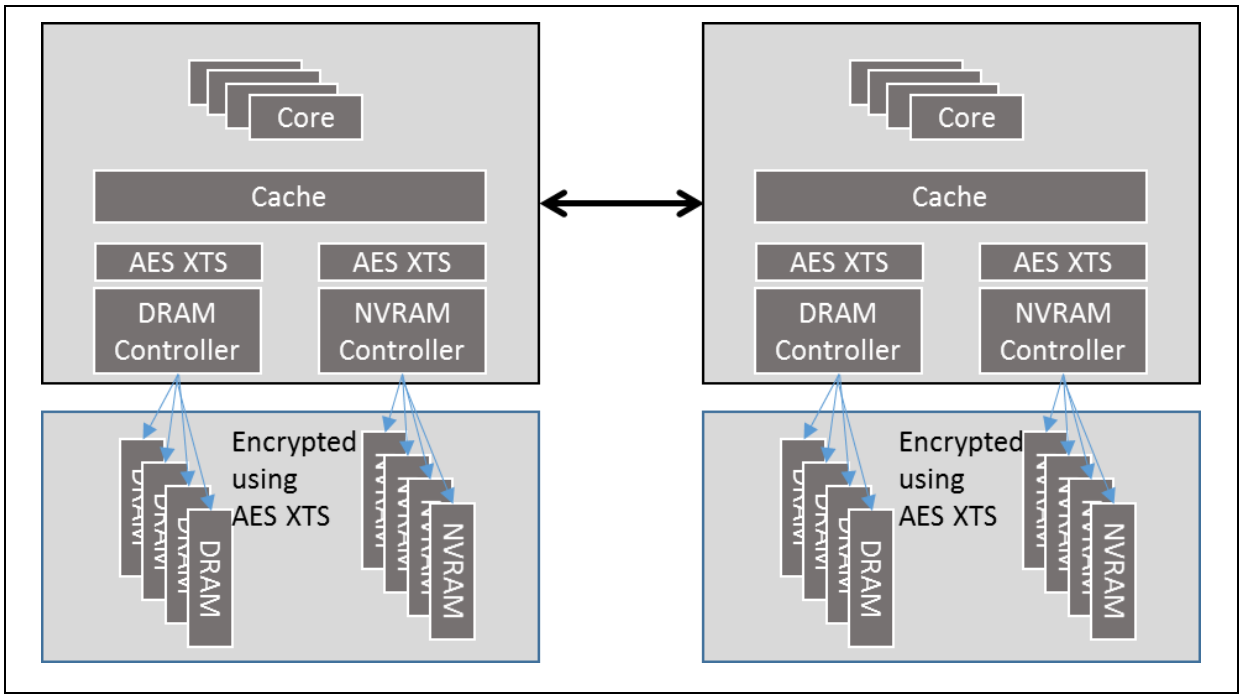
Memory protection schemes address different security and operational needs depending on the CSP's environment and workload. SGX/TDX can be built upon TME-Multikey or TME-Multikey with Integrity, since both are built upon TME it is described first for completeness.

2.1 Total Memory Encryption

Total Memory Encryption (TME) is 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, it 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 is generated by mixing two different values - a per part random value that is fused into the HW at manufacturing time and an ephemeral value that can either be randomly generated by Intel SoC or configured by BIOS prior to enabling 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

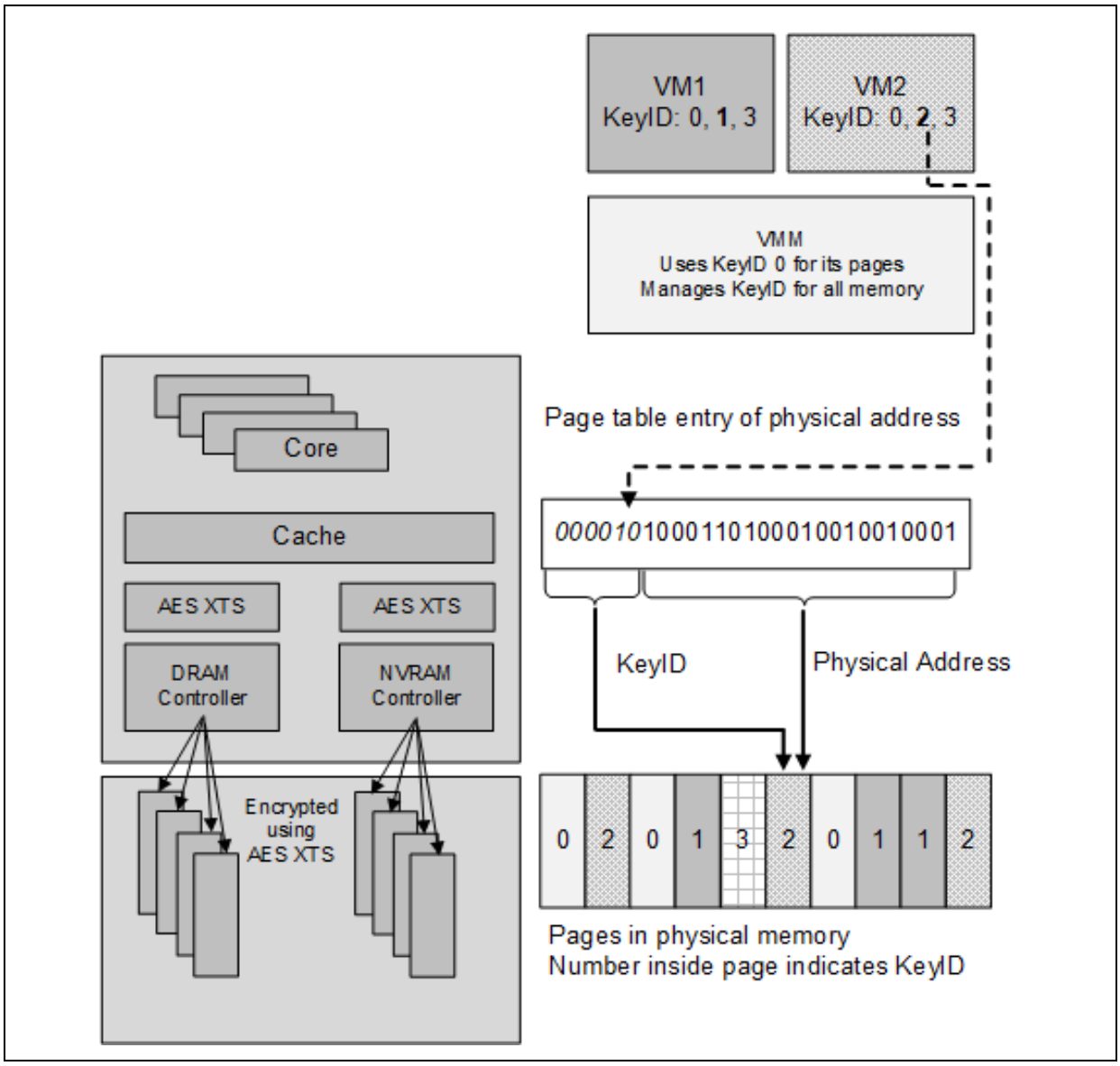


The AES XTS encryption engine is in the direct data path to external memory buses and therefore, all the memory data entering and/or leaving the SoC on memory buses is encrypted using AES XTS. The data inside the SoC (in caches, etc.) remains plain text and supports all the existing software and I/O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

In a typical deployment, the encryption key is generated by the CPU and therefore is not visible to the software. In multi-package platforms each CPU has its own key value for each KeyID.

2.2 Total Memory Encryption – Multi-Key (TME-MK)

Figure 2-2. High-level Architecture of TME-MK



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 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, software can either directly specify a given TME-MK key or instruct CPU to generate an ephemeral key. If the ephemeral key is specified, it is not accessible by software or on external interfaces of CPU.

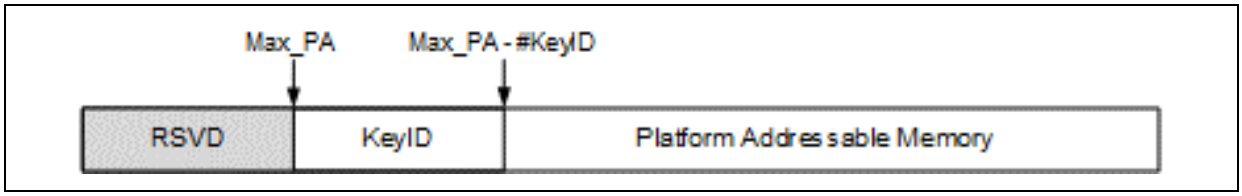
Figure 2-2 shows the basic architecture of TME-MK which shares basic hardware architecture with TME, while supporting multiple keys. The right side of the figure shows the use of TME-MK in a virtualized environment, though architecture supports use of TME-MK in a native OS deployment scenario as well.

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 the instruction that is used to program KeyID attributes for TME-MK. PCONFIG is enumerated separately from TME-MK. The PCONFIG instruction details are available in the latest Intel® Software Developers Manual.

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

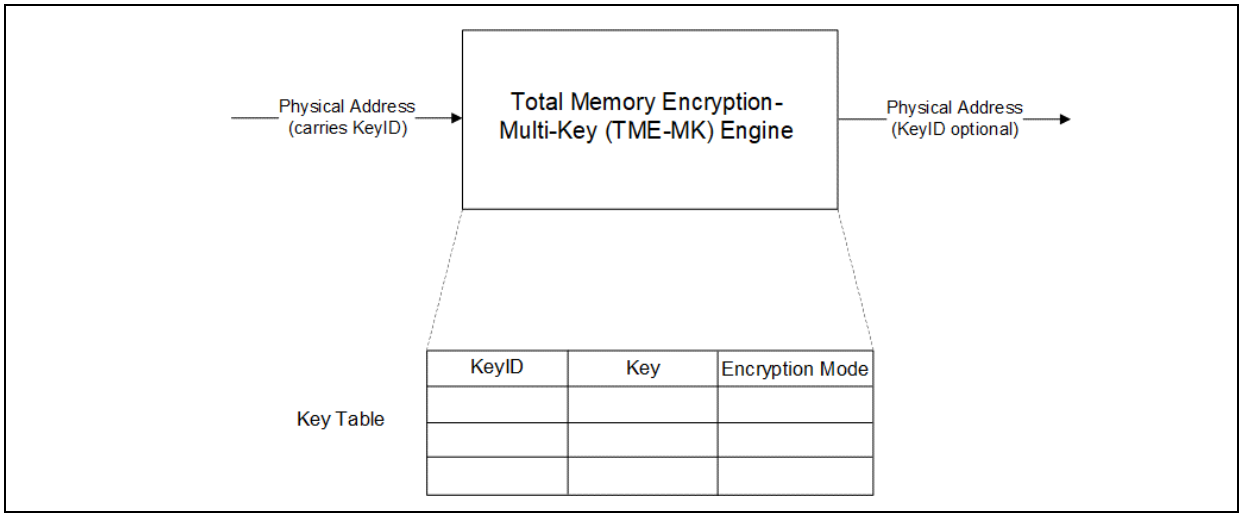
When TME-MK is activated, the upper bits of the platform physical address (starting with the highest order bit available as enumerated by the CPUID MAX_PA info) are repurposed for usage as a KeyID as shown below.

Figure 2-3. KeyID Usage



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. 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 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.

Figure 2-4. TME-MK Engine Overview



2.3 TME-MK with Integrity

Total Memory Encryption-Multi-key with Integrity (TMEi-MK) adds the ability for the TME-MK sub-system to create/store/check a 28b integrity value with each cacheline.

On some SoC's the memory protections can also be configured to support encryption with integrity. To provide integrity, a cryptographic Message Authentication Code (MAC) is associated with each cacheline in memory. The MAC is generated when a line is written to memory and is verified when the line is read from memory. If a data line is modified while resident in memory (or when moving over the memory bus), the MAC verification fails on a subsequent read of the data line. MAC verification failure is signaled back to the core and handled appropriately.

For these implementations of memory integrity, a 29b SHA-3 based keyed MAC is stored for each cacheline in memory in the ECC memory. This avoids sequestering memory and results in no additional bandwidth overheads when providing integrity. However, it does result in reduced space for ECC codes and thus impacts the overall reliability metrics supportable when integrity is enabled.

Keccak-p[1600] is used as the underlying function for MAC generation. The variant used for the MAC construction has a capacity of 512b and rate of 1088b (1600b – 512b (capacity)). In other words, 1088b can be used by data/metadata that needs to be integrity protected.

The MAC includes the following components:

- Ciphertext data: This is the primary asset to be protected. Including the ciphertext in the MAC provides detection against modification attacks.
- Encryption tweak: Including the encryption tweak serves two purposes. One, since each KeyID has a unique tweak key, the encryption tweak provides the domain uniqueness to the MAC to detect cross-domain attacks. Secondly, since the encryption tweak is generated by encrypting the physical address with the tweak key, it also provides detection against splicing/relocation attacks.

- Metadata bits: The metadata bits include the TEE ownership indication, and poison bit. The TEE ownership indication is used to control access to TEE ciphertext from non-TEE software respectively
- MAC key: 128b/256b MAC key (depending on the mode of encryption being supported) as required by SHA-3 based MAC construction

3 Confidential Computing Enhancements

For confidential computing memory encryption by itself is insufficient. There are other access controls and protections that must be in place for the TEE architecture to work successfully. Many of these are implemented in the memory sub-system of the SoC.

3.1 SEAM vs TDX

SEAM is the CPU mode of operation that is used to provide protection from the VMM and other privileged software running on the platform. More details about the architectural aspects of SEAM can be found in Volume 3C of the Intel® Software Developers Manual. Intel® Trust Domain Extensions is an architecture built up top of the SEAM and its hardware enhancements. When operating in SEAM additional protections are enforced by the memory subsystem, how these mechanisms are implemented may vary from platform to platform.

When TDX is used with TME-MK this is referred to as Logical Integrity Mode, and when used with TME-MK with Integrity this is referred to as Cryptographic Integrity Mode.

3.2 SEAM Specific Keys & Addressing

When SEAM is enabled the TME-MK KeyID space is further partitioned to accommodate KeyIDs which can only be used for SEAM based operations. 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 private KeyIDs.

3.3 SGX Memory Encryption Key

Platforms that support SGX with TME-MK based encryption uses an additional private KeyID (that is not part of TME-MK or Private SEAM allocation) for SGX memory. This key is programmed when SGX is enabled and is used to encrypt memory in the Processor Reserved Memory Range (PRMRR).

3.4 Memory Configuration Aliasing Checking

Memory configurations on some types of platforms can be very complicated. These serve the ability for the platform to deliver the highest bandwidth to applications. They allow for interleaving of memory across a variety of channels, memory controllers (when there is more than one in a CPU socket) and across different CPU sockets. We are concerned about whether system address could be mapped to the same physical address, one system address being secure and the other being insecure. Before confidential computing technologies can be enabled platform firmware checks for aliases across a set of known-good configurations.

3.5 Memory & System Configuration Locking

Also, before enabling confidential computing technologies platform firmware also ensures that configuration mechanisms are locked which would otherwise allow platform software to create conditions to manipulate memory configurations or other system configurations that would give an attacker an advantage.

3.6 KeyID Address Aliasing

Under the original TME-MK hardware two addresses with the different KeyID's but matching lower bits resulted in two separate entries in the caching coherency infrastructure, even though they map to the same physical address with under different keys.

In the general TME-MK case system software is responsible for managing the coherency of references to addresses under different KeyID's. In the case of SEAM, the memory coherency hardware is further improved to only allow one address alias at a time to be resident in the cache.

3.7 TEE Ownership Indication

When operation in SEAM, i.e. TDX is enabled, the memory encryption subsystem supports the notion of an Ownership indication for every cacheline. The TEE ownership indication is set to 1 on memory writes when the cacheline was written with a SEAM assigned KeyID (known in TDX terms as private KeyID) and cleared to 0 when the cacheline was written with a non-SEAM assigned KeyID (known in TDX terms as a shared KeyID).

How the Ownership indication is stored is model-specific. In some platforms a single TEE Ownership indication can be stored in the ECC metadata associated with every cacheline. In others where there is no additional metadata storage a separate table in memory is

required. In some cases, the TEE indication ownership indication may appear as two separate indications Non-secure or NS indication for SGX-TEM, Trust Domain or TD indication for TDX.

3.7.1 Access Control Checks

On server platforms checks on memory reads depend on whether Logical Integrity is being used or whether Cryptographic Integrity (Ci) is enabled on the platform. This is shown in the tables below.

- When the memory read transaction uses a private KeyID, TEE Ownership bit mismatch and/or integrity check failure (for Ci) result in a return of fixed data pattern along with a poison indication.
- Any reads of TD private data (TEE Ownership is 1) done outside SEAM mode (i.e., with a shared KeyID) return fixed data pattern. This is intended to prevent the host VMM from getting access to cipher text, since the VMM will deterministically see only a fixed data pattern in the cache for speculative accesses. No poison indication is returned due to TEE mismatch. A previous poison indication that has been stored in memory or a poison generated due to uncorrectable ECC will be returned.

Table 3.1: Checks on Memory Reads in Li Mode

KeyID Type	Integrity Enabled for KeyID	TEE Owner Indication	Integrity Check	Returned Data	New Poison	Comments
Private	No	0	N/A	Fixed Pattern	Poison	TEE ownership intent mismatch failure may be triggered if the memory was previously written using a shared KeyID.
		1	N/A	Decrypted data	None	If the memory line has been previously poisoned, the read transaction may return a poison.
Shared	No	0	N/A	Decrypted data	None	If the memory line has been previously poisoned, the read transaction may return a poison.
		1	N/A	Fixed Pattern	None	If the memory line has been previously poisoned, the read transaction may return a poison.

Table 3.2: Checks on Memory Reads in Ci Mode

KeyID Type	Integrity Enabled for KeyID	TEE Owner Indication	Integrity Check	Returned Data	New Poison	Comments
Private	Yes	0	N/A	Fixed Pattern	Poison	TEE ownership intent mismatch failure may be triggered if the memory was previously written using a shared KeyID.
		1	Pass	Decrypted data	None	If the memory line has been previously poisoned, the read transaction may return a poison.
		1	Fail	Fixed Pattern	Poison	Integrity check failure may be triggered if the memory was previously written using a different encryption key.
Shared	Yes	0	Pass	Decrypted data	None	If the memory line has been previously poisoned, the read transaction may return a poison.
		0	Fail	Fixed Pattern	Poison	Integrity check failure may be triggered if the memory was previously written using a different encryption key.
		1	N/A	Fixed Pattern	Poison	TEE ownership intent mismatch failure may be triggered if the memory was previously written using a private KeyID.
Shared	No	0	N/A	Decrypted data	None	If the memory line has been previously poisoned, the read transaction may return a poison.
		1	N/A	Fixed Pattern	None	If the memory line has been previously poisoned, the read transaction may return a poison.

The value of the fixed pattern returned is machine dependent.

How poisoned memory lines are treated when operating in non-root mode of SEAM is specific to the software running in SEAM – for TDX this is the TDX Module. However, when running in SEAM root mode the CPU enters SEAM shutdown on the consumption of poison, see Volume 3C of the Software Developers Manual for further details on SEAM.

3.7.2 Memory Writes

On server platforms the TEE Ownership indication is not checked on full cache line memory writes. It is the responsibility of the host VMM to prevent writing to memory that has been assigned as TD private memory. Failing to do so will result in memory corruption; such corruption will be detected when the guest TD or the TDX Module attempts to read that memory, as described above.

> ***Note:*** The host VMM should always initialize memory that has been used with a private KeyID (i.e., TD private memory and TDX control structures), using a full line write with a shared KeyID prior to allocating it for shared usages. The correct way to do so is by using the MOVDIR64B instruction. This helps ensure that the TEE Ownership bit is cleared.

3.7.2.1 Partial Memory Writes

In cases where partial cache lines can be written to memory, the value of the TEE ownership indication is taken into consideration when determining the behavior of the SoC.

Table 3.3: Partial Write Treatment

KeyID Type	Previous TEE Owner Indication	Data Written	New TEE Owner Indication	Poison	Comments
Private	0	Zero data, merge partial data	1	Poison	TEE ownership intent mismatch considered a potential attack. Only place partial data in cacheline.
	1	Merge existing and partial data	1	None	Normal case like full write
Shared	0	Merge existing and partial data	0	None	Normal case like full write
	1	Zero data, merge partial data	0	None	Clears prior data, sets TEE to zero, no poison required.

4 Memory Protection Security Properties

While products cannot provide protection against all possible security threats, in this section we outline some of the specific security properties that the memory encryption and the additional access control enhancements for confidential computing provides.

4.1 Memory Protection Model

The table below sets out the protections expected when Logical Integrity Mode and Cryptographic Integrity Mode are enabled with Intel® SGX or Intel® TDX. For SW the table may assume other access control mechanisms required for preventing SW accesses to memory.

Table 4-1. Memory Protection Security Properties

Protect Memory from	Attack Vector	Logical Integrity Mode	Cryptographic Integrity Mode
Loss of Confidentiality	SW	Yes	Yes
Loss of Integrity		Yes	Yes[1]
Anti-Replay		Yes	Yes
Loss of Confidentiality	Physical Attacks	Partial[2]	Partial[2]
Loss of Integrity		No	Partial[3]
Anti-Replay		No	No

Notes:

1) Cryptographic Integrity (CI) Mode, provides enhanced protection from software-based fault inducing attacks on memory such as RowHammer.

2) The use of AES-XTS encryption for Logical Integrity Mode and Cryptographic Mode, only provides confidentiality protection from adversaries that can only see ciphertext once and not while data is changing (this includes, but is not limited to, low entropy dictionary attacks).

3) Cryptographic Integrity (CI) Mode can provide enhanced protection against cross-domain attacks that do not involve replay.

4.2 Addressing Dictionary Based Attacks

As the table implies there are limitations to the mode of encryption selected for TME and TME-MK that work best when physical attacker can only see the ciphertext once, as in a cold boot or platform reset based attack. When a physical attacker can see the ciphertext multiple times they can build dictionaries per cacheline as the method of encryption is a codebook – meaning that the same plaintext will always result in the same ciphertext (at the same address) for a given key.

A 64byte cacheline is covered by 4 separate AES-XTS blocks (bytes 0-15, 16-31, 32-48 & 48-63), each one forming its own 16byte dictionary. In cases of low entropy in the plaintext of a block in the cacheline SW can choose to add random values direction, either by placing entropy into spare space in the block or XORing with the original plaintext. Adding randomness once does not remove the dictionary nature of AES-XTS, it merely increases the size of the dictionary. Varying the plaintext on each write will improve the protection from physical attackers that can read the ciphertext being written, but it comes at the cost of additional storage.

4.3 Addressing Cache Based Timing Side Channels

Cache-based side-channels such as 'Prime + Probe' and 'Flush + Reload' can be used to determine a control channel over the use of the data. The affect the anti-aliasing coherency mechanisms added to prevent untrusted software from using the coherency mechanism to attack TEE memory can be used to create a more precise side-channel down to the cache-line level. To issue can be mitigated by data/secret dependent memory accesses. More information can be found in the best practice guidance Intel has issued.