



# **Intel® Trust Domain Extensions (Intel® TDX) Module Architecture Application Binary Interface (ABI) Reference Specification**

DRAFT

348551-002US

January 2023

## Notices and Disclaimers

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

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

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

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

No product or component can be absolutely secure.

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

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

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

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

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

DRAFT

## Table of Contents

	<b>1. About this Document</b>	<b>7</b>
	1.1. Scope of this Document	7
	1.2. Glossary	7
5	1.3. Notation	7
	1.4. References	7
	<b>2. CPU Virtualization Tables</b>	<b>8</b>
	2.1. MSR Virtualization	8
	2.2. <b>UPDATED:</b> CPUID Virtualization	12
10	<b>3. Data Types</b>	<b>22</b>
	3.1. <b>UPDATED:</b> Interface Function Completion Status	22
	3.1.1. <b>UPDATED:</b> Function Completion Status Structure	22
	3.1.2. Function Completion Status Code Classes (Bits 47:40)	23
	3.1.3. Function Completion Status Codes	23
15	3.1.4. Function Completion Status Operand IDs	31
	3.2. Basic Crypto Types	34
	3.3. <b>UPDATED:</b> TDX Module Configuration, Enumeration and Initialization Types	34
	3.3.1. CPUID_CONFIG	34
	3.3.2. <b>NEW:</b> Global-Scope (TDX Module) Metadata	34
20	3.3.2.1. TDX Features Enumeration	35
	3.3.2.2. How to Read the Global Fields Table	36
	3.3.2.3. Global Metadata Fields	36
	3.3.3. <b>UPDATED:</b> CMR_INFO	38
	3.3.4. <b>UPDATED:</b> TDSYSINFO_STRUCT	39
25	3.3.5. <b>UPDATED:</b> TDMR_INFO	41
	3.4. TD Parameter Types	42
	3.4.1. <b>UPDATED:</b> ATTRIBUTES	42
	3.4.2. XFAM	43
	3.4.3. CPUID_VALUES	43
30	3.4.4. <b>UPDATED:</b> TD_PARAMS	44
	3.5. Physical Memory Management Types	46
	3.5.1. Physical Page Size	46
	3.6. <b>UPDATED:</b> TD Private Memory Management Data Types: Secure EPT	46
	3.6.1. Secure EPT Levels	46
35	3.6.2. Secure EPT Entry Information as Returned by TDX Module Functions	47
	3.6.2.1. Returned Secure EPT Entry Content	47
	3.6.2.2. Additional Returned Secure EPT Information	47
	3.6.3. <b>NEW:</b> GPA_ATTR: GPA Attributes	49
	3.6.3.1. GPA Attributes Rules	49
40	3.6.4. Page GLA List	49
	3.6.4.1. PAGE_GLA_LIST_ENTRY	50
	3.6.4.2. PAGE_GLA_LIST_INFO: Page GLA List GPA and Additional Information	50
	3.7. TD Entry and Exit Types	50
	3.7.1. Extended Exit Qualification	50
45	3.8. L2 VM Transition Types	52
	3.8.1. <b>NEW:</b> L2_ENTER_GUEST_STATE	52
	3.9. Measurement and Attestation Types	53
	3.9.1. CPUSVN	53
	3.9.2. TDREPORT_STRUCT	53

	3.9.3.	REPORTMACSTRUCT (Reference) .....	53
	3.9.4.	REPORTTYPE .....	54
	3.9.5.	<b>UPDATED:</b> TDINFO_STRUCT .....	55
	3.10.	<b>UPDATED:</b> Metadata Access Types .....	56
5	3.10.1.	MD_FIELD_ID: Metadata Field Identifier / Sequence Header .....	56
	3.10.2.	Meaning of Field Codes .....	57
	3.10.3.	Class Codes .....	59
	3.10.3.1.	<b>NEW:</b> TDX Module Global Scope Field Class Codes .....	59
	3.10.3.2.	<b>UPDATED:</b> TD-Scope (TDR and TDCS) Field Class Codes .....	59
10	3.10.3.3.	VCPU-Scope (TDVPS) Field Class Codes .....	60
	3.10.4.	Order of Field Identifiers .....	60
	3.10.5.	MD_LIST_HEADER: Metadata List Header .....	60
	3.10.6.	Private Page List .....	61
	3.10.7.	HPA_AND_SIZE: HPA and Size of a Buffer .....	61
15	3.10.8.	HPA_AND_LAST: HPA and Last Byte Index of a Page-Aligned Buffer .....	61
	3.11.	<b>NEW:</b> Service TD Types .....	62
	3.11.1.	SERVTD_BINDING_TABLE: Service TD Binding Table .....	62
	3.11.2.	SERVTD_BINDING_STATE: Service TD Binding State .....	62
	3.11.3.	SERVTD_TYPE: Service TD Binding Type .....	62
20	3.11.4.	SERVTD_ATTR: Service TD Binding Attributes .....	62
	3.12.	<b>Migration Types</b> .....	63
	3.12.1.	MBMD: Migration Bundle Metadata .....	63
	3.12.1.1.	Generic MBMD Structure .....	64
25	3.12.1.2.	TD-Scope Immutable Non-Memory State MBMD Fields .....	64
	3.12.1.3.	TD-Scope Mutable Non-Memory State MBMD Fields .....	65
	3.12.1.4.	VCPU-Scope Mutable Non-Memory State MBMD Fields .....	65
	3.12.1.5.	TD Private Memory MBMD Fields .....	65
	3.12.1.6.	Epoch Token MBMD Fields .....	66
	3.12.1.7.	Abort Token MBMD Fields .....	66
30	3.12.1.8.	TD Migration Protocol Version Compatibility .....	66
	3.12.2.	GPA List .....	66
	3.12.2.1.	GPA_LIST_INFO: HPA, First and Last Entries of a GPA List .....	66
	3.12.2.2.	GPA List Entry .....	67
	3.12.2.3.	GPA List Entry Details .....	67
35	3.12.2.4.	TD Migration Protocol Version Compatibility .....	69
	3.12.3.	Memory Migration Buffers List .....	70
	3.12.3.1.	Migration Buffers List Entry .....	70
	3.12.4.	Page Attributes List .....	70
	3.12.5.	Memory Migration Page MAC List .....	70
40	3.12.6.	Non-Memory State Migration Buffers List .....	70
	3.12.6.1.	PAGE_LIST_INFO: HPA and Attributes of a Page List .....	70
	4.	<b>UPDATED:</b> TD Metadata (Non-Memory State) .....	72
	4.1.	<b>UPDATED:</b> TD-Scope Metadata .....	72
45	4.1.1.	<b>UPDATED:</b> How to Read the TDR and TDCS Tables .....	72
	4.1.2.	<b>UPDATED:</b> TDR .....	72
	4.1.3.	<b>UPDATED:</b> TDCS .....	73
	4.2.	<b>UPDATED:</b> TDVPS: VCPU-Scope Metadata .....	77
	4.2.1.	<b>UPDATED:</b> Overview .....	77
	4.2.2.	How to Read the TDVPS (including TD VMCS) Tables .....	78
50	4.2.2.1.	<b>UPDATED:</b> Field Access .....	78
	4.2.3.	TDVPS (excluding TD VMCS) .....	78
	4.2.4.	TD VMCS .....	82
	4.2.4.1.	TD VMCS Guest State Area .....	83
	4.2.4.2.	TD VMCS Host State Area .....	85
55	4.2.4.3.	TD VMCS VM-Execution Control Fields .....	85
	4.2.4.4.	TD VMCS VM-Exit Control Fields .....	92

	4.2.4.5.	TD VMCS VM-Entry Control Fields .....	93
	4.2.4.6.	TD VMCS VM-Exit Information Fields.....	94
	4.2.5.	<b>NEW: L2 VMCS</b> .....	95
	4.2.5.1.	L2 VMCS Guest State Area .....	95
5	4.2.5.2.	L2 VMCS Host State Area .....	97
	4.2.5.3.	L2 VMCS VM-Execution Control Fields.....	97
	4.2.5.4.	L2 VMCS VM-Exit Control Fields .....	104
	4.2.5.5.	L2 VMCS VM-Entry Control Fields.....	105
	4.2.5.6.	L2 VMCS VM-Exit Information Fields .....	106
10	<b>5.</b>	<b>UPDATED: Interface Functions .....</b>	<b>108</b>
	5.1.	<i>How to Read the Interface Function Definitions .....</i>	<i>108</i>
	5.2.	<b>NEW: Common Algorithms Used by Multiple Interface Functions.....</b>	<b>108</b>
	5.2.1.	VCPU Association with an LP .....	108
	5.2.2.	Metadata Access.....	109
15	5.2.2.1.	Single Metadata Field Read .....	109
	5.2.2.2.	Single Metadata Field Write .....	109
	5.2.2.3.	Multiple Metadata Fields Write based on a Metadata List .....	109
	5.3.	<b>UPDATED: Host-Side (SEAMCALL) Interface Functions .....</b>	<b>111</b>
	5.3.1.	<b>UPDATED: SEAMCALL Instruction (Common) .....</b>	<b>111</b>
20	5.3.2.	<b>NEW: TDH.EXPORT.ABORT Leaf .....</b>	<b>114</b>
	5.3.3.	<b>NEW: TDH.EXPORT.BLOCKW Leaf .....</b>	<b>117</b>
	5.3.4.	<b>NEW: TDH.EXPORT.MEM Leaf .....</b>	<b>120</b>
	5.3.5.	<b>NEW: TDH.EXPORT.PAUSE Leaf .....</b>	<b>125</b>
	5.3.6.	<b>NEW: TDH.EXPORT.RESTORE Leaf .....</b>	<b>127</b>
25	5.3.7.	<b>NEW: TDH.EXPORT.STATE.IMMUTABLE Leaf .....</b>	<b>130</b>
	5.3.8.	<b>NEW: TDH.EXPORT.STATE.TD Leaf .....</b>	<b>134</b>
	5.3.9.	<b>NEW: TDH.EXPORT.STATE.VP Leaf .....</b>	<b>137</b>
	5.3.10.	<b>NEW: TDH.EXPORT.TRACK Leaf .....</b>	<b>140</b>
	5.3.11.	<b>NEW: TDH.EXPORT.UNBLOCKW Leaf .....</b>	<b>143</b>
30	5.3.12.	<b>NEW: TDH.IMPORT.ABORT Leaf .....</b>	<b>146</b>
	5.3.13.	<b>NEW: TDH.IMPORT.COMMIT Leaf .....</b>	<b>149</b>
	5.3.14.	<b>NEW: TDH.IMPORT.END Leaf .....</b>	<b>151</b>
	5.3.15.	<b>NEW: TDH.IMPORT.MEM Leaf .....</b>	<b>153</b>
	5.3.16.	<b>NEW: TDH.IMPORT.STATE.IMMUTABLE Leaf .....</b>	<b>159</b>
35	5.3.17.	<b>NEW: TDH.IMPORT.STATE.TD Leaf .....</b>	<b>163</b>
	5.3.18.	<b>NEW: TDH.IMPORT.STATE.VP Leaf .....</b>	<b>167</b>
	5.3.19.	<b>NEW: TDH.IMPORT.TRACK Leaf .....</b>	<b>171</b>
	5.3.20.	<b>UPDATED: TDH.MEM.PAGE.ADD Leaf .....</b>	<b>174</b>
	5.3.21.	<b>UPDATED: TDH.MEM.PAGE.AUG Leaf .....</b>	<b>177</b>
40	5.3.22.	<b>UPDATED: TDH.MEM.PAGE.DEMOTE Leaf .....</b>	<b>180</b>
	5.3.23.	<b>UPDATED: TDH.MEM.PAGE.PROMOTE Leaf .....</b>	<b>186</b>
	5.3.24.	<b>UPDATED: TDH.MEM.PAGE.RELOCATE Leaf .....</b>	<b>191</b>
	5.3.25.	<b>UPDATED: TDH.MEM.PAGE.REMOVE Leaf .....</b>	<b>194</b>
	5.3.26.	<b>UPDATED: TDH.MEM.RANGE.BLOCK Leaf .....</b>	<b>197</b>
45	5.3.27.	<b>UPDATED: TDH.MEM.RANGE.UNBLOCK Leaf .....</b>	<b>200</b>
	5.3.28.	TDH.MEM.RD Leaf .....	203
	5.3.29.	<b>UPDATED: TDH.MEM.SEPT.ADD Leaf .....</b>	<b>205</b>
	5.3.30.	<b>UPDATED: TDH.MEM.SEPT.RD Leaf .....</b>	<b>211</b>
	5.3.31.	<b>UPDATED: TDH.MEM.SEPT.REMOVE Leaf .....</b>	<b>215</b>
50	5.3.32.	<b>UPDATED: TDH.MEM.TRACK Leaf .....</b>	<b>219</b>
	5.3.33.	TDH.MEM.WR Leaf .....	221
	5.3.34.	<b>NEW: TDH.MIG.STREAM.CREATE Leaf .....</b>	<b>224</b>
	5.3.35.	<b>UPDATED: TDH.MNG.ADDCX Leaf .....</b>	<b>225</b>
	5.3.36.	TDH.MNG.CREATE Leaf .....	228
55	5.3.37.	<b>UPDATED: TDH.MNG.INIT Leaf .....</b>	<b>230</b>
	5.3.38.	TDH.MNG.KEY.CONFIG Leaf .....	233
	5.3.39.	TDH.MNG.KEY.FREEID Leaf .....	235

5.3.40.	TDH.MNG.KEY.RECLAIMID Leaf (Deprecated)	237
5.3.41.	UPDATED: TDH.MNG.RD Leaf	238
5.3.42.	TDH.MNG.VPFLUSHDONE Leaf	240
5.3.43.	UPDATED: TDH.MNG.WR Leaf	242
5.3.44.	UPDATED: TDH.MR.EXTEND Leaf	244
5.3.45.	UPDATED: TDH.MR.FINALIZE Leaf	247
5.3.46.	TDH.PHYMEM.CACHE.WB Leaf	249
5.3.47.	TDH.PHYMEM.PAGE.RDMD Leaf	252
5.3.48.	TDH.PHYMEM.PAGE.RECLAIM Leaf	254
5.3.49.	TDH.PHYMEM.PAGE.WBINVD Leaf	257
5.3.50.	NEW: TDH.SERVTD.BIND Leaf	259
5.3.51.	NEW: TDH.SERVTD.PREBIND Leaf	262
5.3.52.	TDH.SYS.CONFIG Leaf	264
5.3.53.	UPDATED: TDH.SYS.INFO Leaf	267
5.3.54.	TDH.SYS.INIT Leaf	269
5.3.55.	TDH.SYS.KEY.CONFIG Leaf	272
5.3.56.	TDH.SYS.LP.INIT Leaf	274
5.3.57.	UPDATED: TDH.SYS.LP.SHUTDOWN Leaf (Deprecated)	277
5.3.58.	NEW: TDH.SYS.RD Leaf	278
5.3.59.	NEW: TDH.SYS.RDALL Leaf	280
5.3.60.	NEW: TDH.SYS.SHUTDOWN Leaf	282
5.3.61.	TDH.SYS.TDMR.INIT Leaf	284
5.3.62.	NEW: TDH.SYS.UPDATE Leaf	286
5.3.63.	UPDATED: TDH.VP.ADDCX Leaf	288
5.3.64.	UPDATED: TDH.VP.CREATE Leaf	290
5.3.65.	UPDATED: TDH.VP.ENTER Leaf	292
5.3.66.	UPDATED: TDH.VP.FLUSH Leaf	302
5.3.67.	UPDATED: TDH.VP.INIT Leaf	304
5.3.68.	UPDATED: TDH.VP.RD Leaf	307
5.3.69.	UPDATED: TDH.VP.WR Leaf	310
5.4.	UPDATED: Guest-Side (TDCALL) Interface Functions	313
5.4.1.	TDCALL Instruction (Common)	313
5.4.2.	TDG.MEM.PAGE.ACCEPT Leaf	315
5.4.3.	NEW: TDG.MEM.PAGE.ATTR.RD Leaf	318
5.4.4.	NEW: TDG.MEM.PAGE.ATTR.WR Leaf	320
5.4.5.	TDG.MR.REPORT Leaf	324
5.4.6.	TDG.MR.RTM.R.EXTEND Leaf	326
5.4.7.	TDG.MR.VERIFYREPORT	328
5.4.8.	NEW: TDG.SERVTD.RD Leaf	330
5.4.9.	NEW: TDG.SERVTD.WR Leaf	334
5.4.10.	NEW: TDG.SYS.RD Leaf	338
5.4.11.	NEW: TDG.SYS.RDALL Leaf	340
5.4.12.	UPDATED: TDG.VM.RD Leaf	342
5.4.13.	UPDATED: TDG.VM.WR Leaf	344
5.4.14.	UPDATED: TDG.VP.CPUIDVE.SET Leaf	346
5.4.15.	NEW: TDG.VP.ENTER Leaf	348
5.4.16.	UPDATED: TDG.VP.INFO Leaf	353
5.4.17.	NEW: TDG.VP.INVEPT Leaf	355
5.4.18.	NEW: TDG.VP.INVVPID Leaf	357
5.4.19.	NEW: TDG.VP.RD Leaf	359
5.4.20.	TDG.VP.VEINFO.GET Leaf	361
5.4.21.	TDG.VP.VMCALL Leaf	363
5.4.22.	NEW: TDG.VP.WR Leaf	366

## 1. About this Document

### 1.1. Scope of this Document

This document describes the Application Binary Interface (ABI) of the Intel® Trust Domain Extensions (Intel® TDX) module, implemented using the Intel TDX Instruction Set Architecture (ISA) extensions, for confidential execution of Trust Domains in an untrusted hosted cloud environment.

This document is part of the **TDX Module Architecture Specification Set**, which includes the following documents:

**Table 1.1: TDX Module Architecture Specification Set**

Document Name	Reference	Description
<b>TDX Module Base Architecture Specification</b>	[TDX Module Base Spec]	Base TDX module architecture overview and specification, covering key management, TD lifecycle management, memory management, virtualization, measurement and attestation, service TDs, debug aspects etc.
<b>TDX Module TD Migration Architecture Specification</b>	[TD Migration Spec]	Architecture overview and specification for TD migration
<b>TDX Module TD Partitioning Architecture Specification</b>	[TD Partitioning Spec]	Architecture overview and specification for TD Partitioning
<b>TDX Module ABI Reference Specification</b>	[TDX Module ABI Spec]	Detailed TDX module Application Binary Interface (ABI) reference specification, covering the entire TDX module architecture
<b>TDX Module ABI Incompatibilities between TDX 1.0 and TDX 1.4/1.5</b>	[TDX Module ABI Incompatibilities]	Description of the incompatibilities between TDX 1.0 and TDX 1.4/1.5 that may impact the host VMM and/or guest TDs

This document is a work in progress and is subject to change based on customer feedback and internal analysis. This document does not imply any product commitment from Intel to anything in terms of features and/or behaviors.

**Note:** The contents of this document are accurate to the best of Intel's knowledge as of the date of publication, though Intel does not represent that such information will remain as described indefinitely in light of future research and design implementations. Intel does not commit to update this document in real time when such changes occur.

### 1.2. Glossary

See the [TDX Module Base Spec].

### 1.3. Notation

See the [TDX Module Base Spec].

### 1.4. References

See the [TDX Module Base Spec].

## 2. CPU Virtualization Tables

### 2.1. MSR Virtualization

Table 2.2 below describes how the Intel TDX module virtualizes MSRs to guest TDs. The table uses a notation that is described in Table 2.1 below.

**Table 2.1: MSR Virtualization Notation Definition**

Text	Virtualization
<b>Native</b>	Direct read or write from/to CPU
<b>#GP(0)</b>	Inject a #GP(0) exception
<b>#VE</b>	<b>For L1:</b> Inject a #VE exception <b>For L2:</b> TDX Module injects a #VE or does an L2->L1 exit: if (TDVPS.L2_CTL.S.ENABLE_EXTENDED_VE) Inject a #VE exception else Do an L2->L1 exit
<b>Inject_GP(condition)</b>	TDX Module injects a #GP(0) if condition is true, else reads from CPU or write to CPU: if (condition) #GP(0) else Native
<b>Inject_GP_or_VE(condition)</b>	TDX Module injects a #GP(0) if condition is true, else it injects a #VE: if (condition) #GP(0) else #VE

For MSRs that are not listed in the table, the Intel TDX module injects a #VE on both RDMSR and WRMSR by the guest TD.

**Note:** The table below provides a high-level overview of MSR virtualization. Implementation details may differ.

**Table 2.2: MSR Virtualization**

MSR Index Range (Hex)			MSR Architectural Name	MSR Virtualization	
First (Hex)	Last (Hex)	Size (Hex)		On RDMSR	On WRMSR
0x0010	0x0010	0x1	IA32_TIME_STAMP_COUNTER	Native	#VE
0x0048	0x0048	0x1	IA32_SPEC_CTRL	Native	Native
0x0049	0x0049	0x1	IA32_PRED_CMD	Native	Native
0x0087	0x0087	0x1	IA32_MKTIME_PARTITIONING	Inject_GP_or_VE (~virt. CPUID(7,0).EDX[18])	Inject_GP_or_VE (~virt. CPUID(7,0).EDX[18])
0x008C	0x008F	0x4	IA32_SGXLEPUBKEYHASHx	#GP(0)	#GP(0)
0x0098	0x0098	0x1	MSR_WBINVDP	#GP(0)	#GP(0)
0x0099	0x0099	0x1	MSR_WBNOINVDP	#GP(0)	#GP(0)
0x009A	0x009A	0x1	MSR_INTR_PENDING	#GP(0)	#GP(0)
0x009B	0x009B	0x1	IA32_SMM_MONITOR_CTL	#GP(0)	#GP(0)
0x009E	0x009E	0x1	IA32_SMBASE	#GP(0)	#GP(0)
0x00BC	0x00BC	0x1	IA32_MISC_PACKAGE_CTL.S	Native	#VE
0x00C1	0x00C8	0x8	IA32_PMCx	Inject_GP(~PERFMON)	Inject_GP(~PERFMON)
0x00E1	0x00E1	0x1	IA32_UMWAIT_CONTROL	Inject_GP(~virt. CPUID(7,0).ECX[5])	Inject_GP(~virt. CPUID(7,0).ECX[5])
0x010A	0x010A	0x1	IA32_ARCH_CAPABILITIES	Bit 4 (SSB_NO): ALLOW_DIRECT Bit 7 (TSX_CTRL): If virt. TSX is enabled, native. Else, 0 Bit 9 (reserved) = 0 Bit 10 (MISC_PACKAGE_CTRL.S) = 0 Bit 11 (ENERGY_FILTERING_CTL) = 0 Bit 16 (reserved) = 0 Bit 17 (FB_CLEAR) = 0 Bit 18 (FB_CLEAR_CTRL) = 0	Native



MSR Index Range (Hex)			MSR Architectural Name	MSR Virtualization	
First (Hex)	Last (Hex)	Size (Hex)		On RDMSR	On WRMSR
				Bit 19 (RRSBA): FORCE_DIRECT Bit 20 (BHI_NO): ALLOW_DIRECT Bit 21 (XAPIC_DISABLE_STATUS) = 0 Bit 22 (reserved) = 0 Bit 23 (OVERCLOCKING_STATUS) = 0 Bit 24 (PBRBSB_NO): ALLOW_DIRECT Bits 63:25 (reserved) = 0 Other bits = FIXED_AS_VERIFIED	
0x010B	0x010B	0x1	IA32_FLUSH_CMD	Native	Native
0x0122	0x0122	0x1	IA32_TSX_CTRL	Inject_GP(~(virt. TSX enabled))	Inject_GP(~(virt. TSX enabled))
0x0174	0x0174	0x1	IA32_SYSENTER_CS	Native	Native
0x0175	0x0175	0x1	IA32_SYSENTER_ESP	Native	Native
0x0176	0x0176	0x1	IA32_SYSENTER_EIP	Native	Native
0x0186	0x018D	0x8	IA32_PERFEVTSELx	Inject_GP(~PERFMON)	Inject_GP(~PERFMON)
0x01A0	0x01A0	0x1	IA32_MISC_ENABLE	if ~PERFMON RDMSR current value Indicate Perfmon and PEBS are unavailable: Bit 7 = 0 Bit 12 = 1 else Native	#VE
0x01A6	0x01A7	0x2	MSR_OFFCORE_RSPx	Inject_GP(~PERFMON)	Inject_GP(~PERFMON)
0x01C4	0x01C4	0x1	IA32_XFD	Inject_GP(~(virt. CPUID(0xD,0x1).EAX[4]))	Inject_GP(~(virt. CPUID(0xD,0x1).EAX[4]))
0x01C5	0x01C5	0x1	IA32_XFD_ERR	Inject_GP(~(virt. CPUID(0xD,0x1).EAX[4]))	Inject_GP(~(virt. CPUID(0xD,0x1).EAX[4]))
0x01D9	0x01D9	0x1	IA32_DEBUGCTL	Clear ENABLE_UNCORE_PMI (bit 13)	#GP if invalid, #VE if value is not supported for TD
0x01F8	0x01F8	0x1	IA32_PLATFORM_DCA_CAP	Inject_GP_or_VE(~virt. CPUID(0x1).ECX[18])	Inject_GP_or_VE(~virt. CPUID(0x1).ECX[18])
0x01F9	0x01F9	0x1	IA32_CPU_DCA_CAP	Inject_GP_or_VE(~virt. CPUID(0x1).ECX[18])	Inject_GP_or_VE(~virt. CPUID(0x1).ECX[18])
0x01FA	0x01FA	0x1	IA32_DCA_0_CAP	Inject_GP_or_VE(~virt. CPUID(0x1).ECX[18])	Inject_GP_or_VE(~virt. CPUID(0x1).ECX[18])
0x0276	0x0276	0x1	MSR_SLAM_ENABLE	#GP(0)	#GP(0)
0x0277	0x0277	0x1	IA32_PAT	Native	Native
0x0309	0x0310	0x8	IA32_FIXED_CTRx	Inject_GP(~PERFMON)	Inject_GP(~PERFMON)
0x0329	0x0329	0x1	IA32_PERF_METRICS	Inject_GP(~PERFMON)	Inject_GP(~PERFMON)
0x0345	0x0345	0x1	IA32_PERF_CAPABILITIES	if ~PERFMON return 0 else if ~XFAM[8] clear bit 16 else Native	Inject_GP(~PERFMON)
0x038D	0x038D	0x1	IA32_FIXED_CTR_CTRL	Inject_GP(~PERFMON)	Inject_GP(~PERFMON)
0x038E	0x038E	0x1	IA32_PERF_GLOBAL_STATUS	Inject_GP(~PERFMON)	Inject_GP(~PERFMON)
0x038F	0x038F	0x1	IA32_PERF_GLOBAL_CTRL	Inject_GP(~PERFMON)	Inject_GP(~PERFMON)
0x0390	0x0390	0x1	IA32_PERF_GLOBAL_STATUS_RESET	Inject_GP(~PERFMON)	Inject_GP(~PERFMON)
0x0391	0x0391	0x1	IA32_PERF_GLOBAL_STATUS_SET	Inject_GP(~PERFMON)	Inject_GP(~PERFMON)
0x0392	0x0392	0x1	IA32_PERF_GLOBAL_INUSE	Inject_GP(~PERFMON)	Inject_GP(~PERFMON)
0x03F1	0x03F1	0x1	IA32_PEBB_ENABLE	Inject_GP(~PERFMON)	Inject_GP(~PERFMON)
0x03F2	0x03F2	0x1	MSR_PEBB_DATA_CFG	Inject_GP(~PERFMON)	Inject_GP(~PERFMON)
0x03F6	0x03F6	0x1	MSR_PEBB_LD_LAT	Inject_GP(~PERFMON)	Inject_GP(~PERFMON)
0x03F7	0x03F7	0x1	MSR_PEBB_FRONTEND	Inject_GP(~PERFMON)	Inject_GP(~PERFMON)
0x0480	0x0480	0x1	IA32_VMX_BASIC	For L1: Bit 54 (VM exit info on INS/OUTS) = 1 Bit 55 (true VMX controls) = 1 Bit 56 (VOE w/o err code) = 1 Other bits = 0 For L2: #VE	#GP(0)
0x0481	0x0481	0x1	IA32_VMX_PINBASED_CTLS	#VE	#GP(0)
0x0482	0x0482	0x1	IA32_VMX_PROCBASED_CTLS	#VE	#GP(0)
0x0483	0x0483	0x1	IA32_VMX_EXIT_CTLS	#VE	#GP(0)
0x0484	0x0484	0x1	IA32_VMX_ENTRY_CTLS	#VE	#GP(0)
0x0485	0x0485	0x1	IA32_VMX_MISC	For L1: Bit 5 (unrestricted guest) = 1 Bit 6 (HLT activity state) = 1 Bit 7 (shutdown activity state) = 1 Bit 14 (PT in VMX) = 1	#GP(0)

MSR Index Range (Hex)			MSR Architectural Name	MSR Virtualization	
First (Hex)	Last (Hex)	Size (Hex)		On RDMSR	On WRMSR
				Bits 24:16 (CR3 target count) = 4 Bit 29 (VMWRITE any field) = 1 Bit 30 (VOE with 0 inst length) = 1 Other bits = 0 For L2: #VE	
0x0486	0x0486	0x1	IA32_VMX_CR0_FIXED0	For L1: See L2 VMCS guest CR0 field description For L2: #VE	#GP(0)
0x0487	0x0487	0x1	IA32_VMX_CR0_FIXED1	For L1: See L2 VMCS guest CR0 field description For L2: #VE	#GP(0)
0x0488	0x0488	0x1	IA32_VMX_CR4_FIXED0	For L1: See L2 VMCS guest CR0 field description For L2: #VE	#GP(0)
0x0489	0x0489	0x1	IA32_VMX_CR4_FIXED1	For L1: See L2 VMCS guest CR0 field description For L2: #VE	#GP(0)
0x048A	0x048A	0x1	IA32_VMX_VMCS_ENUM	#VE	#GP(0)
0x048B	0x048B	0x1	IA32_VMX_PROCBASED_CTL2	For L1: See L2 VMCS guest CR0 field description For L2: #VE	#GP(0)
0x048C	0x048C	0x1	IA32_VMX_EPT_VPID_CAP	For L1: Execute-only (bit 0) = 1 2MB pages (bit 16) = 1 1GB pages (bit 17) = 1 A/D (bit 21) = 0 EPT violation info (bit 22) = 1 SSS (bit 23) = XFAM.CET_S (bit 12) HLAT prefix size (bits 53:48) taken from real MSR Other bits = 0 For L2: #VE	#GP(0)
0x048D	0x048D	0x1	IA32_VMX_TRUE_PINBASED_CTL2	For L1: See L2 VMCS guest CR0 field description For L2: #VE	#GP(0)
0x048E	0x048E	0x1	IA32_VMX_TRUE_PROCBASED_CTL2	For L1: See L2 VMCS guest CR0 field description For L2: #VE	#GP(0)
0x048F	0x048F	0x1	IA32_VMX_TRUE_EXIT_CTL2	For L1: See L2 VMCS guest CR0 field description For L2: #VE	#GP(0)
0x0490	0x0490	0x1	IA32_VMX_TRUE_ENTRY_CTL2	For L1: See L2 VMCS guest CR0 field description For L2: #VE	#GP(0)
0x0491	0x0491	0x1	IA32_VMX_VMFUNC	For L1: All-0 For L2: #VE	#GP(0)
0x0492	0x0492	0x1	IA32_VMX_PROCBASED_CTL3	For L1: See L2 VMCS guest CR0 field description For L2: #VE	#GP(0)
0x04C1	0x04C8	0x8	IA32_A_PMCx	Inject_GP(~PERFMON)	Inject_GP(~PERFMON)
0x0500	0x0500	0x1	IA32_SGX_SVN_STATUS	#GP(0)	#GP(0)
0x0560	0x0560	0x1	IA32_RTIT_OUTPUT_BASE	Inject_GP(~XFAM[8])	Inject_GP(~XFAM[8])
0x0561	0x0561	0x1	IA32_RTIT_OUTPUT_MASK_PTRS	Inject_GP(~XFAM[8])	Inject_GP(~XFAM[8])
0x0570	0x0570	0x1	IA32_RTIT_CTL	Inject_GP(~XFAM[8])	Inject_GP(~XFAM[8])
0x0571	0x0571	0x1	IA32_RTIT_STATUS	Inject_GP(~XFAM[8])	Inject_GP(~XFAM[8])
0x0572	0x0572	0x1	IA32_RTIT_CR3_MATCH	Inject_GP(~XFAM[8])	Inject_GP(~XFAM[8])
0x0580	0x0580	0x1	IA32_RTIT_ADDR0_A	Inject_GP(~XFAM[8])	Inject_GP(~XFAM[8])
0x0581	0x0581	0x1	IA32_RTIT_ADDR0_B	Inject_GP(~XFAM[8])	Inject_GP(~XFAM[8])
0x0582	0x0582	0x1	IA32_RTIT_ADDR1_A	Inject_GP(~XFAM[8])	Inject_GP(~XFAM[8])
0x0583	0x0583	0x1	IA32_RTIT_ADDR1_B	Inject_GP(~XFAM[8])	Inject_GP(~XFAM[8])
0x0584	0x0584	0x1	IA32_RTIT_ADDR2_A	Inject_GP(~XFAM[8])	Inject_GP(~XFAM[8])
0x0585	0x0585	0x1	IA32_RTIT_ADDR2_B	Inject_GP(~XFAM[8])	Inject_GP(~XFAM[8])
0x0586	0x0586	0x1	IA32_RTIT_ADDR3_A	Inject_GP(~XFAM[8])	Inject_GP(~XFAM[8])
0x0587	0x0587	0x1	IA32_RTIT_ADDR3_B	Inject_GP(~XFAM[8])	Inject_GP(~XFAM[8])
0x0600	0x0600	0x1	IA32_DS_AREA	Native	Native
0x06A0	0x06A0	0x1	IA32_U_CET	Inject_GP(~(XFAM[11]   XFAM[12]))	Inject_GP(~(XFAM[11]   XFAM[12]))
0x06A2	0x06A2	0x1	IA32_S_CET	Inject_GP(~(XFAM[11]   XFAM[12]))	Inject_GP(~(XFAM[11]   XFAM[12]))
0x06A4	0x06A4	0x1	IA32_PL0_SSP	Inject_GP(~(XFAM[11]   XFAM[12]))	Inject_GP(~(XFAM[11]   XFAM[12]))
0x06A5	0x06A5	0x1	IA32_PL1_SSP	Inject_GP(~(XFAM[11]   XFAM[12]))	Inject_GP(~(XFAM[11]   XFAM[12]))
0x06A6	0x06A6	0x1	IA32_PL2_SSP	Inject_GP(~(XFAM[11]   XFAM[12]))	Inject_GP(~(XFAM[11]   XFAM[12]))

MSR Index Range (Hex)			MSR Architectural Name	MSR Virtualization	
First (Hex)	Last (Hex)	Size (Hex)		On RDMSR	On WRMSR
0x06A7	0x06A7	0x1	IA32_PL3_SSP	Inject_GP(~(XFAM[11]   XFAM[12]))	Inject_GP(~(XFAM[11]   XFAM[12]))
0x06A8	0x06A8	0x1	IA32_INTERRUPT_SSP_TABLE_ADDR	Inject_GP(~(XFAM[11]   XFAM[12]))	Inject_GP(~(XFAM[11]   XFAM[12]))
0x06E0	0x06E0	0x1	IA32_TSC_DEADLINE	Inject_GP_or_VE(~virt. CPUID(0x1).ECX[24])	Inject_GP_or_VE(~virt. CPUID(0x1).ECX[24])
0x06E1	0x06E1	0x1	IA32_PKRS	Inject_GP(~PKS)	Inject_GP(~PKS)
0x0800	0x0801	0x2	Reserved for xAPIC MSRs	#GP(0)	#GP(0)
0x0804	0x0807	0x4	Reserved for xAPIC MSRs	#GP(0)	#GP(0)
0x0808	0x0808	0x1	IA32_X2APIC_TPR	Native	Native
0x0809	0x0809	0x1	Reserved for xAPIC MSRs	Native	Native
0x080A	0x080A	0x1	IA32_X2APIC_PPR	Native	Native
0x080B	0x080B	0x1	IA32_X2APIC_EOI	Native	Native
0x080C	0x080C	0x1	Reserved for xAPIC MSRs	Native	Native
0x080E	0x080E	0x1	Reserved for xAPIC MSRs	Native	Native
0x0810	0x0817	0x8	IA32_X2APIC_ISR <sub>x</sub>	Native	Native
0x0818	0x081F	0x8	IA32_X2APIC_TMR <sub>x</sub>	Native	Native
0x0820	0x0827	0x8	IA32_X2APIC_IRR <sub>x</sub>	Native	Native
0x0829	0x082E	0x6	Reserved for xAPIC MSRs	#GP(0)	#GP(0)
0x0831	0x0831	0x1	Reserved for xAPIC MSRs	#GP(0)	#GP(0)
0x083F	0x083F	0x1	IA32_X2APIC_SELF_IPI	Native	Native
0x0840	0x087F	0x40	Reserved for xAPIC MSRs	#GP(0)	#GP(0)
0x0880	0x08BF	0x40	Reserved for xAPIC MSRs	#GP(0)	#GP(0)
0x08C0	0x08FF	0x40	Reserved for xAPIC MSRs	#GP(0)	#GP(0)
0x0981	0x0981	0x1	IA32_TME_CAPABILITY	Inject_GP_or_VE(~virt. CPUID(7,0).ECX[13])	Inject_GP_or_VE(~virt. CPUID(7,0).ECX[13])
0x0982	0x0982	0x1	IA32_TME_ACTIVATE	Inject_GP_or_VE(~virt. CPUID(7,0).ECX[13])	Inject_GP_or_VE(~virt. CPUID(7,0).ECX[13])
0x0983	0x0983	0x1	IA32_TME_EXCLUDE_MASK	Inject_GP_or_VE(~virt. CPUID(7,0).ECX[13])	Inject_GP_or_VE(~virt. CPUID(7,0).ECX[13])
0x0984	0x0984	0x1	IA32_TME_EXCLUDE_BASE	Inject_GP_or_VE(~virt. CPUID(7,0).ECX[13])	Inject_GP_or_VE(~virt. CPUID(7,0).ECX[13])
0x0985	0x0985	0x1	IA32_UINTR_RR	Inject_GP(~XFAM[14])	Inject_GP(~XFAM[14])
0x0986	0x0986	0x1	IA32_UINTR_HANDLER	Inject_GP(~XFAM[14])	Inject_GP(~XFAM[14])
0x0987	0x0987	0x1	IA32_UINTR_STACKADJUST	Inject_GP(~XFAM[14])	Inject_GP(~XFAM[14])
0x0988	0x0988	0x1	IA32_UINTR_MISC	Inject_GP(~XFAM[14])	Inject_GP(~XFAM[14])
0x0989	0x0989	0x1	IA32_UINTR_PD	Inject_GP(~XFAM[14])	Inject_GP(~XFAM[14])
0x098A	0x098A	0x1	IA32_UINTR_TT	Inject_GP(~XFAM[14])	Inject_GP(~XFAM[14])
0x0C80	0x0C80	0x1	IA32_DEBUG_INTERFACE	Native	#VE
0x0D90	0x0D90	0x1	IA32_BNDCFGS	#GP(0)	#GP(0)
0x0D93	0x0D93	0x1	IA32_PASID	#GP(0)	#GP(0)
0x0DA0	0x0DA0	0x1	IA32_XSS	Native	if invalid or does not match XFAM #GP(0) else Write to CPU
0x1200	0x12FF	0x100	IA32_LBR_INFO	Inject_GP(~XFAM[15])	Inject_GP(~XFAM[15])
0x14CE	0x14CE	0x1	IA32_LBR_CTL	Inject_GP(~XFAM[15])	Inject_GP(~XFAM[15])
0x14CF	0x14CF	0x1	IA32_LBR_DEPTH	Inject_GP(~XFAM[15])	Inject_GP(~XFAM[15])
0x1500	0x15FF	0x100	IA32_LBR_x_FROM_IP	Inject_GP(~XFAM[15])	Inject_GP(~XFAM[15])
0x1600	0x16FF	0x100	IA32_LBR_x_TO_IP	Inject_GP(~XFAM[15])	Inject_GP(~XFAM[15])
0x1B01	0x1B01	0x1	IA32_UARCH_MISC_CTL	Native	Native
0xC0000080	0xC0000080	0x1	IA32_EFER	Native	If TD Partitioning is supported: - Ignore read-only bit LMA (10) - Allow update of bit SCE (0) - For L2, allow update of bits LME (8) and NXE (11) - #VE on any other change Else: - #VE
0xC0000081	0xC0000081	0x1	IA32_STAR	Native	Native
0xC0000082	0xC0000082	0x1	IA32_LSTAR	Native	Native
0xC0000084	0xC0000084	0x1	IA32_FMASK	Native	Native
0xC0000100	0xC0000100	0x1	IA32_FSBASE	Native	Native
0xC0000101	0xC0000101	0x1	IA32_GSBASE	Native	Native
0xC0000102	0xC0000102	0x1	IA32_KERNEL_GS_BASE	Native	Native
0xC0000103	0xC0000103	0x1	IA32_TSC_AUX	Native	Native

## 2.2. **UPDATED:** CPUID Virtualization

Table 2.4 below describes how the Intel TDX module virtualizes CPUID to guest TDs. Note the following:

- The “Configuration by TDH.MNG.INIT” column details which section of the TD\_PARAMS structure is used for configuring how each CPUID bit field is virtualized.
- The “Virtualization” column uses a notation defined in Table 2.3 below.
- If the guest TD executes CPUID with a valid leaf / sub-leaf number combination that is not listed in the table, the Intel TDX module injects a #VE.
- The host VMM should always consult the list of CPUID leaves and sub-leaves configured by TD\_PARAMS.CPUID\_CONFIG, as enumerated by TDH.SYS.RD/RDALL or TDH.SYS.INFO.

**Table 2.3: CPUID Virtualization Notation Definition**

CPUID Bit or Field Virtualization	Meaning	Virtualization Details
<b>Configured</b>	Virtual bit field value reflects the host VMM configuration in either TD_PARAMS.CPUID_CONFIG or by another TD_PARAMS fields.	
<b>Configured &amp; Native</b>	The virtual bit value is a 1 if and only if the bit configuration in TD_PARAMS.CPUID_CONFIG is 1, and the native bit value returned by executing CPUID is 1.	
<b>Attribute &amp; Native</b>	The virtual bit value is a 1 if and only if the configuration in TD_PARAMS.ATTRIBUTES<attribute> is 1, and the native bit value returned by executing CPUID is 1.	
<b>XFAM &amp; Native</b>	The virtual bit value is a 1 if and only if the configuration in TD_PARAMS.XFAM<bits> is set, and the native bit value returned by executing CPUID is 1.	
<b>XFAM &amp; Configured &amp; Native</b>	The virtual bit value is a 1 if and only if the configuration in TD_PARAMS.XFAM<bits> is set, the bit configuration in TD_PARAMS.CPUID_CONFIG is 1, and the native bit value returned by executing CPUID is 1.	
<b>Calculated</b>	Bit field is calculated by the Intel TDX module.	Calculation method
<b>Fixed</b>	The virtual bit field value is fixed.	Bit field value
<b>Native</b>	The virtual bit field value reflects the native value returned by executing CPUID.	

**Note:** The table below provides a high-level overview of CPUID virtualization. Implementation details may differ.

**Table 2.4: CPUID Virtualization Overview**

CPUID Field					Configuration	Virtualization	
Reg.	MSB	LSB	Field Size	Field Name	Configuration Details	Bit or Field Virtualization Type	Virtualization Details
<b>Leaf 0x0</b>							
EAX	31	0	32	MaxIndex		Fixed	0x23
EBX	31	0	32	Genu		Fixed	0x756e6547
ECX	31	0	32	ntel		Fixed	0x6c65746e
EDX	31	0	32	inel		Fixed	0x49656e69
<b>Leaf 0x1</b>							
EAX	3	0	4	Stepping ID	Minimum stepping value of all packages	Calculated	Min. of all packages
EAX	7	4	4	Model ID		Native	
EAX	11	8	4	Family ID		Native	
EAX	13	12	2	Processor Type		Native	
EAX	15	14	2	Reserved		Fixed	0x0
EAX	19	16	4	Extended Model ID		Native	
EAX	27	20	8	Extended Family ID		Native	
EAX	31	28	4	Reserved		Fixed	0x0
EBX	7	0	8	Brand Index		Fixed	0x0
EBX	15	8	8	CLFLUSH Line Size		Fixed	0x8
EBX	23	16	8	Maximum Addressable IDs	TD_PARAMS.CPUID_CONFIG	Configured	

CPUID Field					Configuration	Virtualization	
Reg.	MSB	LSB	Field Size	Field Name	Configuration Details	Bit or Field Virtualization Type	Virtualization Details
EBX	31	24	8	Initial APIC ID	Bits 7:0 of TDVPS.VCPU_INDEX	Calculated	TDVPS.VCPU_INDEX[7:0]
ECX	0	0	1	SSE3		Fixed	0x1
ECX	1	1	1	PCLMULQDQ		Fixed	0x1
ECX	2	2	1	DTES64		Fixed	0x1
ECX	3	3	1	MONITOR	TD_PARAMS.CPUID_CONFIG	Configured & Native	
ECX	4	4	1	DS-CPL		Fixed	0x1
ECX	5	5	1	VMX		Fixed	0x0
ECX	6	6	1	SMX		Fixed	0x0
ECX	7	7	1	EST	TD_PARAMS.CPUID_CONFIG	Configured & Native	
ECX	8	8	1	TM2	TD_PARAMS.CPUID_CONFIG	Configured & Native	
ECX	9	9	1	SSSE3		Fixed	0x1
ECX	10	10	1	CNXT-ID	TD_PARAMS.CPUID_CONFIG	Configured & Native	
ECX	11	11	1	SDBG	TD_PARAMS.CPUID_CONFIG	Configured & Native	
ECX	12	12	1	FMA	TD_PARAMS.XFAM	XFAM & Native	XFAM[2]
ECX	13	13	1	CMPXCHG16B		Fixed	0x1
ECX	14	14	1	xTPR Update Control	TD_PARAMS.CPUID_CONFIG	Configured & Native	
ECX	15	15	1	PDCM		Fixed	0x1
ECX	16	16	1	Reserved		Fixed	0x0
ECX	17	17	1	PCID		Fixed	0x1
ECX	18	18	1	DCA	TD_PARAMS.CPUID_CONFIG	Configured & Native	
ECX	19	19	1	SSE4_1		Fixed	0x1
ECX	20	20	1	SSE4_2		Fixed	0x1
ECX	21	21	1	x2APIC		Fixed	0x1
ECX	22	22	1	MOVBE		Fixed	0x1
ECX	23	23	1	POPCNT		Fixed	0x1
ECX	24	24	1	TSC-Deadline	TD_PARAMS.CPUID_CONFIG	Configured & Native	
ECX	25	25	1	AESNI		Fixed	0x1
ECX	26	26	1	XSAVE		Fixed	0x1
ECX	27	27	1	OSXSAVE	TD_VMCS.GUEST_CR4.OSXSAVE	Calculated	CR4.OSXSAVE
ECX	28	28	1	AVX	TD_PARAMS.XFAM, TD_PARAMS.CPUID_CONFIG	XFAM & Configured & Native	XFAM[2]
ECX	29	29	1	F16C	TD_PARAMS.XFAM, TD_PARAMS.CPUID_CONFIG	XFAM & Configured & Native	XFAM[2]
ECX	30	30	1	RDRAND		Fixed	0x1
ECX	31	31	1	Hypervisor		Fixed	0x1
EDX	0	0	1	FPU		Fixed	0x1
EDX	1	1	1	VME		Fixed	0x1
EDX	2	2	1	DE		Fixed	0x1
EDX	3	3	1	PSE		Fixed	0x1
EDX	4	4	1	TSC		Fixed	0x1
EDX	5	5	1	MSR		Fixed	0x1
EDX	6	6	1	PAE		Fixed	0x1
EDX	7	7	1	MCE		Fixed	0x1
EDX	8	8	1	CX8		Fixed	0x1
EDX	9	9	1	APIC		Fixed	0x1
EDX	10	10	1	Reserved		Fixed	0x0
EDX	11	11	1	SEP		Fixed	0x1
EDX	12	12	1	MTRR		Fixed	0x1
EDX	13	13	1	PGE		Fixed	0x1
EDX	14	14	1	MCA		Fixed	0x1
EDX	15	15	1	CMOV		Fixed	0x1
EDX	16	16	1	PAT		Fixed	0x1
EDX	17	17	1	PSE-36		Fixed	0x0
EDX	18	18	1	PSN	TD_PARAMS.CPUID_CONFIG	Configured & Native	
EDX	19	19	1	CLFSH		Fixed	0x1
EDX	20	20	1	Reserved		Fixed	0x0
EDX	21	21	1	DS		Fixed	0x1
EDX	22	22	1	ACPI	TD_PARAMS.CPUID_CONFIG	Configured & Native	
EDX	23	23	1	MMX		Fixed	0x1
EDX	24	24	1	FXSR		Fixed	0x1
EDX	25	25	1	SSE		Fixed	0x1
EDX	26	26	1	SSE2		Fixed	0x1
EDX	27	27	1	SS		Configured & Native	
EDX	28	28	1	HTT	TD_PARAMS.CPUID_CONFIG	Configured & Native	
EDX	29	29	1	TM	TD_PARAMS.CPUID_CONFIG	Configured & Native	
EDX	30	30	1	Reserved		Fixed	0x0
EDX	31	31	1	PBE	TD_PARAMS.CPUID_CONFIG	Configured & Native	
Leaf 0x3							

CPUID Field					Configuration	Virtualization	
Reg.	MSB	LSB	Field Size	Field Name	Configuration Details	Bit or Field Virtualization Type	Virtualization Details
EAX	31	0	32	Reserved		Fixed	0x0
EBX	31	0	32	Reserved		Fixed	0x0
ECX	31	0	32	Reserved		Fixed	0x0
EDX	31	0	32	Reserved		Fixed	0x0
Leaf 0x4 / Sub-Leaf 0x0							
EAX	4	0	5	Type	TD_PARAMS.CPUID_CONFIG	Configured	
EAX	7	5	3	Level	TD_PARAMS.CPUID_CONFIG	Configured	
EAX	8	8	1	Self Initializing	TD_PARAMS.CPUID_CONFIG	Configured	
EAX	9	9	1	Fully Associative	TD_PARAMS.CPUID_CONFIG	Configured	
EAX	13	10	4	Reserved	TD_PARAMS.CPUID_CONFIG	Fixed	0x0
EAX	25	14	12	Addressable IDs Sharing this Cache	TD_PARAMS.CPUID_CONFIG	Configured	
EAX	31	26	6	Addressable IDs for Cores in Package	TD_PARAMS.CPUID_CONFIG	Configured	
EBX	11	0	12	L		Fixed	0x3F
EBX	21	12	10	P	TD_PARAMS.CPUID_CONFIG	Configured	
EBX	31	22	10	W	TD_PARAMS.CPUID_CONFIG	Configured	
ECX	31	0	32	Number of Sets	TD_PARAMS.CPUID_CONFIG	Configured	
EDX	0	0	1	WBINVD	TD_PARAMS.CPUID_CONFIG	Configured	
EDX	1	1	1	Cache Inclusiveness	TD_PARAMS.CPUID_CONFIG	Configured	
EDX	2	2	1	Reserved	TD_PARAMS.CPUID_CONFIG	Fixed	0x0
EDX	31	3	29		TD_PARAMS.CPUID_CONFIG	Configured	
Leaf 0x4 / Sub-Leaf 0x1							
EAX	4	0	5	Type	TD_PARAMS.CPUID_CONFIG	Configured	
EAX	7	5	3	Level	TD_PARAMS.CPUID_CONFIG	Configured	
EAX	8	8	1	Self Initializing	TD_PARAMS.CPUID_CONFIG	Configured	
EAX	9	9	1	Fully Associative	TD_PARAMS.CPUID_CONFIG	Configured	
EAX	13	10	4	Reserved	TD_PARAMS.CPUID_CONFIG	Fixed	0x0
EAX	25	14	12	Addressable IDs Sharing this Cache	TD_PARAMS.CPUID_CONFIG	Configured	
EAX	31	26	6	Addressable IDs for Cores in Package	TD_PARAMS.CPUID_CONFIG	Configured	
EBX	11	0	12	L		Fixed	0x3F
EBX	21	12	10	P	TD_PARAMS.CPUID_CONFIG	Configured	
EBX	31	22	10	W	TD_PARAMS.CPUID_CONFIG	Configured	
ECX	31	0	32	Number of Sets	TD_PARAMS.CPUID_CONFIG	Configured	
EDX	0	0	1	WBINVD	TD_PARAMS.CPUID_CONFIG	Configured	
EDX	1	1	1	Cache Inclusiveness	TD_PARAMS.CPUID_CONFIG	Configured	
EDX	2	2	1	Reserved	TD_PARAMS.CPUID_CONFIG	Fixed	0x0
EDX	31	3	29		TD_PARAMS.CPUID_CONFIG	Configured	
Leaf 0x4 / Sub-Leaf 0x2							
EAX	4	0	5	Type	TD_PARAMS.CPUID_CONFIG	Configured	
EAX	7	5	3	Level	TD_PARAMS.CPUID_CONFIG	Configured	
EAX	8	8	1	Self Initializing	TD_PARAMS.CPUID_CONFIG	Configured	
EAX	9	9	1	Fully Associative	TD_PARAMS.CPUID_CONFIG	Configured	
EAX	13	10	4	Reserved	TD_PARAMS.CPUID_CONFIG	Fixed	0x0
EAX	25	14	12	Addressable IDs Sharing this Cache	TD_PARAMS.CPUID_CONFIG	Configured	
EAX	31	26	6	Addressable IDs for Cores in Package	TD_PARAMS.CPUID_CONFIG	Configured	
EBX	11	0	12	L		Fixed	0x3F
EBX	21	12	10	P	TD_PARAMS.CPUID_CONFIG	Configured	
EBX	31	22	10	W	TD_PARAMS.CPUID_CONFIG	Configured	
ECX	31	0	32	Number of Sets	TD_PARAMS.CPUID_CONFIG	Configured	
EDX	0	0	1	WBINVD	TD_PARAMS.CPUID_CONFIG	Configured	
EDX	1	1	1	Cache Inclusiveness	TD_PARAMS.CPUID_CONFIG	Configured	
EDX	2	2	1	Reserved	TD_PARAMS.CPUID_CONFIG	Fixed	0x0
EDX	31	3	29		TD_PARAMS.CPUID_CONFIG	Configured	
Leaf 0x4 / Sub-Leaf 0x3							
EAX	4	0	5	Type	TD_PARAMS.CPUID_CONFIG	Configured	
EAX	7	5	3	Level	TD_PARAMS.CPUID_CONFIG	Configured	
EAX	8	8	1	Self Initializing	TD_PARAMS.CPUID_CONFIG	Configured	
EAX	9	9	1	Fully Associative	TD_PARAMS.CPUID_CONFIG	Configured	
EAX	13	10	4	Reserved	TD_PARAMS.CPUID_CONFIG	Fixed	0x0
EAX	25	14	12	Addressable IDs Sharing this Cache	TD_PARAMS.CPUID_CONFIG	Configured	
EAX	31	26	6	Addressable IDs for Cores in Package	TD_PARAMS.CPUID_CONFIG	Configured	
EBX	11	0	12	L		Fixed	0x3F
EBX	21	12	10	P	TD_PARAMS.CPUID_CONFIG	Configured	
EBX	31	22	10	W	TD_PARAMS.CPUID_CONFIG	Configured	
ECX	31	0	32	Number of Sets	TD_PARAMS.CPUID_CONFIG	Configured	
EDX	0	0	1	WBINVD	TD_PARAMS.CPUID_CONFIG	Configured	
EDX	1	1	1	Cache Inclusiveness	TD_PARAMS.CPUID_CONFIG	Configured	
EDX	2	2	1	Complex cache indexing	TD_PARAMS.CPUID_CONFIG	Configured	
EDX	31	3	29	Reserved	TD_PARAMS.CPUID_CONFIG	Fixed	0x0



CPUID Field					Configuration	Virtualization	
Reg.	MSB	LSB	Field Size	Field Name	Configuration Details	Bit or Field Virtualization Type	Virtualization Details
Leaf 0x4 / Sub-Leaf 0x4							
EAX	4	0	5	Type		Fixed	0x0
EAX	7	5	3	Level		Fixed	0x0
EAX	8	8	1	Self Initializing		Fixed	0x0
EAX	9	9	1	Fully Associative		Fixed	0x0
EAX	13	10	4	Reserved		Fixed	0x0
EAX	25	14	12	Addressable IDs Sharing this Cache		Fixed	0x0
EAX	31	26	6	Addressable IDs for Cores in Package		Fixed	0x0
EBX	11	0	12	L		Fixed	0x0
EBX	21	12	10	P		Fixed	0x0
EBX	31	22	10	W		Fixed	0x0
ECX	31	0	32	Number of Sets		Fixed	0x0
EDX	0	0	1	WBINVD		Fixed	0x0
EDX	1	1	1	Cache Inclusiveness		Fixed	0x0
EDX	2	2	1	Complex Cache Indexing		Fixed	0x0
EDX	31	3	29	Reserved		Fixed	0x0
Leaf 0x7 / Sub-Leaf 0x0							
EAX	31	0	32	Max Sub-Leaves		Fixed	0x2
EBX	0	0	1	FSGSBASE		Fixed	0x1
EBX	1	1	1	IA32_TSC_ADJUST		Fixed	0x0
EBX	2	2	1	SGX		Fixed	0x0
EBX	3	3	1	BMI1	TD_PARAMS.CPUID_CONFIG	Configured & Native	
EBX	4	4	1	HLE	TD_PARAMS.CPUID_CONFIG	Configured & Native	
EBX	5	5	1	AVX2	TD_PARAMS.XFAM	XFAM & Native	XFAM[2]
EBX	6	6	1	FDP_EXCPTN_ONLY		Fixed	0x1
EBX	7	7	1	SMEP		Fixed	0x1
EBX	8	8	1	BMI2	TD_PARAMS.CPUID_CONFIG	Configured & Native	
EBX	9	9	1	Enhanced REP MOVSB/STOSB	TD_PARAMS.CPUID_CONFIG	Configured & Native	
EBX	10	10	1	INVPICID		Fixed	0x1
EBX	11	11	1	RTM	TD_PARAMS.CPUID_CONFIG	Configured & Native	
EBX	12	12	1	PQM	TD_PARAMS.CPUID_CONFIG	Configured & Native	
EBX	13	13	1	FCS/FDS Deprecation		Fixed	0x1
EBX	14	14	1	MPX		Fixed	0x0
EBX	15	15	1	Cache QoS Enforcement	TD_PARAMS.CPUID_CONFIG	Configured & Native	
EBX	16	16	1	AVX512F	TD_PARAMS.XFAM, TD_PARAMS.CPUID_CONFIG	XFAM & Configured & Native	XFAM[7:5]
EBX	17	17	1	AVX512DQ	TD_PARAMS.XFAM, TD_PARAMS.CPUID_CONFIG	XFAM & Configured & Native	XFAM[7:5]
EBX	18	18	1	RDSEED		Fixed	0x1
EBX	19	19	1	ADCX/ADOX	TD_PARAMS.XFAM	Configured & Native	
EBX	20	20	1	SMAP/CLAC/STAC		Fixed	0x1
EBX	21	21	1	AVX512_IFMA	TD_PARAMS.XFAM, TD_PARAMS.CPUID_CONFIG	XFAM & Configured & Native	XFAM[7:5]
EBX	22	22	1	PCOMMIT		Fixed	0x0
EBX	23	23	1	CLFLUSHOPT		Fixed	0x1
EBX	24	24	1	CLWB		Fixed	0x1
EBX	25	25	1	RTIT	TD_PARAMS.XFAM	XFAM & Native	XFAM[8]
EBX	26	26	1	AVX512PF	TD_PARAMS.XFAM, TD_PARAMS.CPUID_CONFIG	XFAM & Configured & Native	XFAM[7:5]
EBX	27	27	1	AVX512ER	TD_PARAMS.XFAM, TD_PARAMS.CPUID_CONFIG	XFAM & Configured & Native	XFAM[7:5]
EBX	28	28	1	AVX512CD	TD_PARAMS.XFAM, TD_PARAMS.CPUID_CONFIG	XFAM & Configured & Native	XFAM[7:5]
EBX	29	29	1	SHA		Fixed	0x1
EBX	30	30	1	AVX512BW	TD_PARAMS.XFAM, TD_PARAMS.CPUID_CONFIG	XFAM & Configured & Native	XFAM[7:5]
EBX	31	31	1	AVX512VL	TD_PARAMS.XFAM, TD_PARAMS.CPUID_CONFIG	XFAM & Configured & Native	XFAM[7:5]
ECX	0	0	1	PREFETCHWT1	TD_PARAMS.CPUID_CONFIG	Configured & Native	
ECX	1	1	1	AVX512VBMI	TD_PARAMS.XFAM	XFAM & Native	XFAM[7:5]
ECX	2	2	1	UMIP	TD_PARAMS.CPUID_CONFIG	Configured & Native	
ECX	3	3	1	PKU	TD_PARAMS.XFAM	XFAM & Native	XFAM[9]
ECX	4	4	1	OSPKE		Calculated	CR4.PKE
ECX	5	5	1	MONITORX/MWAITX		Configured & Native	
ECX	6	6	1	AVX512_VBMI2	TD_PARAMS.XFAM, TD_PARAMS.CPUID_CONFIG	XFAM & Configured & Native	XFAM[7:5]
ECX	7	7	1	CET Shadow Stack	TD_PARAMS.XFAM	XFAM & Native	XFAM[12:11]
ECX	8	8	1	GFNI	TD_PARAMS.CPUID_CONFIG	Configured & Native	

CPUID Field					Configuration	Virtualization	
Reg.	MSB	LSB	Field Size	Field Name	Configuration Details	Bit or Field Virtualization Type	Virtualization Details
ECX	9	9	1	VAES	TD_PARAMS.XFAM, TD_PARAMS.CPUID_CONFIG	XFAM & Configured & Native	XFAM[2]
ECX	10	10	1	VPCLMULQDQ	TD_PARAMS.XFAM, TD_PARAMS.CPUID_CONFIG	XFAM & Configured & Native	XFAM[2]
ECX	11	11	1	AVX512_VNNI	TD_PARAMS.XFAM, TD_PARAMS.CPUID_CONFIG	XFAM & Configured & Native	XFAM[7:5]
ECX	12	12	1	AVX512_BITALG	TD_PARAMS.XFAM, TD_PARAMS.CPUID_CONFIG	XFAM & Configured & Native	XFAM[7:5]
ECX	13	13	1	TME		Configured & Native	
ECX	14	14	1	AVX512_VPOPCNTDQ	TD_PARAMS.XFAM, TD_PARAMS.CPUID_CONFIG	XFAM & Configured & Native	XFAM[7:5]
ECX	15	15	1	Reserved		Fixed	0x0
ECX	16	16	1	57 bit Address Support	TD_PARAMS.CPUID_CONFIG	Configured	
ECX	21	17	5	MAWAU for MPX		Fixed	0x0
ECX	22	22	1	RDPID	TD_PARAMS.CPUID_CONFIG	Configured & Native	
ECX	23	23	1	KL_ENABLED		Attributes & Native	KL
ECX	24	24	1	BUSLOCK		Fixed	0x1
ECX	25	25	1	CLDEMOTTE	TD_PARAMS.CPUID_CONFIG	Configured & Native	
ECX	26	26	1	Reserved		Fixed	0x0
ECX	27	27	1	MOVDIRI		Fixed	0x1
ECX	28	28	1	MOVDIR64B		Fixed	0x1
ECX	29	29	1	ENQCMD		Fixed	0x0
ECX	30	30	1	SGX_LC		Fixed	0x0
ECX	31	31	1	PKS		Attributes & Native	PKS
EDX	0	0	1	Reserved		Fixed	0x0
EDX	1	1	1	Reserved		Fixed	0x0
EDX	2	2	1	AVX512_4VNNIW	TD_PARAMS.XFAM, TD_PARAMS.CPUID_CONFIG	XFAM & Configured & Native	XFAM[7:5]
EDX	3	3	1	AVX512_4FMAPS	TD_PARAMS.XFAM, TD_PARAMS.CPUID_CONFIG	XFAM & Configured & Native	XFAM[7:5]
EDX	4	4	1	Fast Short REP MOV	TD_PARAMS.CPUID_CONFIG	Configured & Native	
EDX	5	5	1	ULI	TD_PARAMS.XFAM	XFAM & Native	XFAM[14]
EDX	6	6	1	Reserved		Fixed	0x0
EDX	7	7	1	Reserved		Fixed	0x0
EDX	8	8	1	AVX512_VP2INTERSECT	TD_PARAMS.XFAM, TD_PARAMS.CPUID_CONFIG	XFAM & Configured & Native	XFAM[7:5]
EDX	9	9	1	MCU_OPT supported		Fixed	0x0
EDX	10	10	1	MD_CLEAR supported		Fixed	0x1
EDX	11	11	1	Reserved		Fixed	0x0
EDX	12	12	1	Reserved		Fixed	0x0
EDX	13	13	1	RTM_FORCE_ABORT_SUPPORT		Fixed	0x0
EDX	14	14	1	SERIALIZE Inst	TD_PARAMS.CPUID_CONFIG	Configured & Native	
EDX	15	15	1	Hybrid Part	TD_PARAMS.CPUID_CONFIG	Configured & Native	
EDX	16	16	1	TSXLDTRK	TD_PARAMS.CPUID_CONFIG	Configured & Native	
EDX	17	17	1	Reserved		Fixed	0x0
EDX	18	18	1	PCONFIG		Configured & Native	
EDX	19	19	1	Architectural LBR support	TD_PARAMS.XFAM	XFAM & Native	XFAM[15]
EDX	20	20	1	CET	TD_PARAMS.XFAM	XFAM & Native	XFAM[12:11]
EDX	21	21	1	Reserved		Fixed	0x0
EDX	22	22	1	TMUL_AMX-BF16	TD_PARAMS.XFAM	XFAM & Native	XFAM[18:17]
EDX	23	23	1	FP16		XFAM & Native	XFAM[7:5]
EDX	24	24	1	TMUL_AMX-TILE	TD_PARAMS.XFAM	XFAM & Native	XFAM[18:17]
EDX	25	25	1	TMUL_AMX-INT8	TD_PARAMS.XFAM	XFAM & Native	XFAM[18:17]
EDX	26	26	1	IBRS (indirect branch restricted speculation)		Fixed	0x1
EDX	27	27	1	STIBP (single thread indirect branch predictors)		Fixed	0x1
EDX	28	28	1	L1D_FLUSH. IA32_FLUSH_CMD support.	TD_PARAMS.CPUID_CONFIG	Fixed	0x1
EDX	29	29	1	IA32_ARCH_CAPABILITIES Support		Fixed	0x1
EDX	30	30	1	IA32_CORE_CAPABILITIES Present		Fixed	0x1
EDX	31	31	1	SSBD (Speculative Store Bypass Disable)		Fixed	0x1
Leaf 0x7 / Sub-Leaf 0x1							
EAX	0	0	1	Reserved		Fixed	0x0
EAX	1	1	1	Reserved		Fixed	0x0
EAX	2	2	1	Reserved		Fixed	0x0
EAX	3	3	1	Reserved		Fixed	0x0
EAX	4	4	1	VEX VNNI	TD_PARAMS.XFAM, TD_PARAMS.CPUID_CONFIG	XFAM & Configured & Native	XFAM[7:5]



CPUID Field				Configuration		Virtualization	
Reg.	MSB	LSB	Field Size	Field Name	Configuration Details	Bit or Field Virtualization Type	Virtualization Details
EAX	5	5	1	AVX512_BF16	TD_PARAMS.XFAM, TD_PARAMS.CPUID_CONFIG	XFAM & Configured & Native	XFAM[7:5]
EAX	6	6	1	LASS		Configured & Native	
EAX	7	7	1	Reserved		Fixed	0x0
EAX	8	8	1	Arch Perfmon Extended Leaf Supported	TD_PARAMS.ATTRIBUTES.PERFMON, TD_PARAMS.CPUID_CONFIG	Attributes & Configured & Native	PERFMON
EAX	9	9	1	Reserved		Fixed	0x0
EAX	10	10	1	Fast Zero-Length MOVSB	TD_PARAMS.CPUID_CONFIG	Configured & Native	
EAX	11	11	1	Fast Short STOSB	TD_PARAMS.CPUID_CONFIG	Configured & Native	
EAX	12	12	1	Fast short CMPSB/SCASB	TD_PARAMS.CPUID_CONFIG	Configured & Native	
EAX	16	13	4	Reserved		Fixed	0x0
EAX	17	17	1	Reserved		Fixed	0x0
EAX	18	18	1	Reserved		Fixed	0x0
EAX	21	19	3	Reserved		Fixed	0x0
EAX	22	22	1	HRESET		Fixed	0x0
EAX	23	23	1	Reserved		Fixed	0x0
EAX	24	24	1	Reserved		Fixed	0x0
EAX	25	25	1	Reserved		Fixed	0x0
EAX	26	26	1	LAM	TD_PARAMS.CPUID_CONFIG	Configured & Native	
EAX	31	27	5	Reserved		Fixed	0x0
EBX	29	0	30	Reserved		Fixed	0x0
EBX	30	30	1	Reserved		Fixed	0x0
EBX	31	31	1	Reserved		Fixed	0x0
ECX	31	0	32	Reserved		Fixed	0x0
EDX	0	0	1	Reserved		Fixed	0x0
EDX	1	1	1	Reserved		Fixed	0x0
EDX	2	2	1	Reserved		Fixed	0x0
EDX	3	3	1	Reserved		Fixed	0x0
EDX	4	4	1	Reserved		Fixed	0x0
EDX	5	5	1	Reserved		Fixed	0x0
EDX	31	6	26	Reserved		Fixed	0x0
Leaf 0x7							
EAX	31	0	32	Reserved		Fixed	0x0
EBX	31	0	32	Reserved		Fixed	0x0
ECX	31	0	32	Reserved		Fixed	0x0
EDX	0	0	1	PSFD		Fixed	0x1
EDX	1	1	1	IPRED_CTRL		Fixed	0x1
EDX	2	2	1	RRSBA_CTRL		Fixed	0x1
EDX	3	3	1	DDPD		Configured & Native	
EDX	4	4	1	BHI_CTRL		Fixed	0x1
EDX	5	5	1	MCDT_NO		Configured & Native	
EDX	31	6	26	Reserved		Fixed	0x0
Leaf 0x8							
EAX	31	0	32	Reserved		Fixed	0x0
EBX	31	0	32	Reserved		Fixed	0x0
ECX	31	0	32	Reserved		Fixed	0x0
EDX	31	0	32	Reserved		Fixed	0x0
Leaf 0xa							
EAX	7	0	8	Version	TD_PARAMS.ATTRIBUTES.PERFMON	Attributes & Native	PERFMON
EAX	15	8	8	Number of GP Counters	TD_PARAMS.ATTRIBUTES.PERFMON	Attributes & Native	PERFMON
EAX	23	16	8	Width of GP Counters	TD_PARAMS.ATTRIBUTES.PERFMON	Attributes & Native	PERFMON
EAX	31	24	8	Length of EBX Vector	TD_PARAMS.ATTRIBUTES.PERFMON	Attributes & Native	PERFMON
EBX	0	0	1	Core Cycles Not Available	TD_PARAMS.ATTRIBUTES.PERFMON	Attributes & Native	PERFMON
EBX	1	1	1	Instructions Retired Not Available	TD_PARAMS.ATTRIBUTES.PERFMON	Attributes & Native	PERFMON
EBX	2	2	1	Reference Cycles Not Available	TD_PARAMS.ATTRIBUTES.PERFMON	Attributes & Native	PERFMON
EBX	3	3	1	Last-Level Cache References Not Available	TD_PARAMS.ATTRIBUTES.PERFMON	Attributes & Native	PERFMON
EBX	4	4	1	Last-Level Cache Misses Not Available	TD_PARAMS.ATTRIBUTES.PERFMON	Attributes & Native	PERFMON
EBX	5	5	1	Branch Instruction Retired Not Available	TD_PARAMS.ATTRIBUTES.PERFMON	Attributes & Native	PERFMON
EBX	6	6	1	Branch Mispredict Retired Not Available	TD_PARAMS.ATTRIBUTES.PERFMON	Attributes & Native	PERFMON
EBX	7	7	1	Top-down slots event not available	TD_PARAMS.ATTRIBUTES.PERFMON	Attributes & Native	PERFMON
EBX	31	8	24	Other events not available bitmap	TD_PARAMS.ATTRIBUTES.PERFMON	Attributes & Native	PERFMON
ECX	3	0	4	Fixed Counter Support Bitmap [3:0]	TD_PARAMS.ATTRIBUTES.PERFMON, TD_PARAMS.CPUID_CONFIG	Attributes & Configured & Native	PERFMON
ECX	31	4	28	Fixed Counter Support Bitmap [31:4]		Fixed	0
EDX	4	0	5	Number of Fixed-Function Counters	TD_PARAMS.ATTRIBUTES.PERFMON	Attributes & Native	PERFMON
EDX	12	5	8	Width of Fixed-Function Counters	TD_PARAMS.ATTRIBUTES.PERFMON	Attributes & Native	PERFMON
EDX	13	13	1	Reserved	TD_PARAMS.ATTRIBUTES.PERFMON	Fixed	0x0
EDX	14	14	1	Reserved	TD_PARAMS.ATTRIBUTES.PERFMON	Fixed	0x0

CPUID Field				Configuration		Virtualization	
Reg.	MSB	LSB	Field Size	Field Name	Configuration Details	Bit or Field Virtualization Type	Virtualization Details
EDX	15	15	1	AnyThread Deprecation		Fixed	0x1
EDX	31	16	16	Reserved	TD_PARAMS.ATTRIBUTES.PERFMON	Fixed	0x0
Leaf 0xd / Sub-Leaf 0x0							
EAX	0	0	1	X87		Fixed	0x1
EAX	1	1	1	SSE		Fixed	0x1
EAX	2	2	1	AVX256	TD_PARAMS.XFAM	XFAM & Native	XFAM[2]
EAX	3	3	1	PL_BNDREGS		Fixed	0x0
EAX	4	4	1	PL_BNDCFS		Fixed	0x0
EAX	5	5	1	KMASK	TD_PARAMS.XFAM	XFAM & Native	XFAM[7:5]
EAX	6	6	1	AVX3 ZMM 15:0	TD_PARAMS.XFAM	XFAM & Native	XFAM[7:5]
EAX	7	7	1	AVX3 ZMM 31:18	TD_PARAMS.XFAM	XFAM & Native	XFAM[7:5]
EAX	8	8	1	Reserved		Fixed	0x0
EAX	9	9	1	PKRU	TD_PARAMS.XFAM	XFAM & Native	XFAM[9]
EAX	16	10	7	Reserved		Fixed	0x0
EAX	17	17	1	AMX - XTILECFG	TD_PARAMS.XFAM	XFAM & Native	XFAM[18:17]
EAX	18	18	1	AMX - XTILEDATA	TD_PARAMS.XFAM	XFAM & Native	XFAM[18:17]
EAX	31	19	13	Reserved		Fixed	0x0
EBX	31	0	32	Max Bytes for Enabled Features		Calculated	Native
ECX	31	0	32	Max Bytes for Supported Features	Computed by TDINIT from TD_PARAMS.XFAM (only for bits in XCR0)	XFAM	
EDX	31	0	32	Reserved		Fixed	0x0
Leaf 0xd / Sub-Leaf 0x1							
EAX	0	0	1	Supports XSAVEOPT		Fixed	0x1
EAX	1	1	1	Supports XSAVEC and compacted XRSTOR		Fixed	0x1
EAX	2	2	1	Supports XGETBV with ECX = 1		Fixed	0x1
EAX	3	3	1	Supports XSAVES/XRSTORS and IA32_XSS		Fixed	0x1
EAX	4	4	1	XFD support	Computed by TDINIT	XFAM	
EAX	31	5	27	Reserved		Fixed	0x0
EBX	31	0	32	Max Bytes for Enabled Features		Calculated	Native
ECX	7	0	8	Reserved		Fixed	0x0
ECX	8	8	1	XSS_RTIT	TD_PARAMS.XFAM	XFAM & Native	XFAM[8]
ECX	9	9	1	Reserved		Fixed	0x0
ECX	10	10	1	PASID		Fixed	0x0
ECX	11	11	1	U_CET	TD_PARAMS.XFAM	XFAM & Native	XFAM[12:11]
ECX	12	12	1	S_CET	TD_PARAMS.XFAM	XFAM & Native	XFAM[12:11]
ECX	13	13	1	HDC		Fixed	0x0
ECX	14	14	1	ULI/UNIT	TD_PARAMS.XFAM	XFAM & Native	XFAM[14]
ECX	15	15	1	XSS_ARCH_LBRS	TD_PARAMS.XFAM	XFAM & Native	XFAM[15]
ECX	16	16	1	HWP Request		Fixed	0x0
ECX	31	17	15	Reserved		Fixed	0x0
EDX	31	0	32	Reserved		Fixed	0x0
Leaf 0xd / Sub-Leaves 0x2-0x12							
EAX	31	0	32	Size	TD_PARAMS.XFAM	XFAM & Native	XFAM[n]
EBX	31	0	32	Offset	TD_PARAMS.XFAM	XFAM & Native	XFAM[n]
ECX	0	0	1	IA32_XSS	TD_PARAMS.XFAM	XFAM & Native	XFAM[n]
ECX	1	1	1		TD_PARAMS.XFAM	XFAM & Native	XFAM[n]
ECX	31	2	30		TD_PARAMS.XFAM	XFAM & Native	XFAM[n]
EDX	31	0	32	Reserved	TD_PARAMS.XFAM	Fixed	0x0
Leaf 0xe							
EAX	31	0	32	Reserved		Fixed	0x0
EBX	31	0	32	Reserved		Fixed	0x0
ECX	31	0	32	Reserved		Fixed	0x0
EDX	31	0	32	Reserved		Fixed	0x0
Leaf 0x11 / Sub-Leaf N/A							
EAX	31	0	32	Reserved		Fixed	0x0
EBX	31	0	32	Reserved		Fixed	0x0
ECX	31	0	32	Reserved		Fixed	0x0
EDX	31	0	32	Reserved		Fixed	0x0
Leaf 0x12							
EAX	31	0	32	Reserved		Fixed	0x0
EBX	31	0	32	Reserved		Fixed	0x0
ECX	31	0	32	Reserved		Fixed	0x0
EDX	31	0	32	Reserved		Fixed	0x0
Leaf 0x13							
EAX	31	0	32	Reserved		Fixed	0x0
EBX	31	0	32	Reserved		Fixed	0x0
ECX	31	0	32	Reserved		Fixed	0x0

CPUID Field				Configuration		Virtualization	
Reg.	MSB	LSB	Field Size	Field Name	Configuration Details	Bit or Field Virtualization Type	Virtualization Details
EDX	31	0	32	Reserved		Fixed	0x0
Leaf 0x14 / Sub-Leaf 0x0							
EAX	31	0	32	Max Valid Subleaf	TD_PARAMS.XFAM	XFAM & Native	XFAM[8]
EBX	0	0	1	CR3 Filtering	TD_PARAMS.XFAM	XFAM & Native	XFAM[8]
EBX	1	1	1	Cycle Accurate Mode	TD_PARAMS.XFAM	XFAM & Native	XFAM[8]
EBX	2	2	1	IP Filtering	TD_PARAMS.XFAM	XFAM & Native	XFAM[8]
EBX	3	3	1	MSRs Preserved Across Warm Reset	TD_PARAMS.XFAM	XFAM & Native	XFAM[8]
EBX	4	4	1	PTWRITE Support	TD_PARAMS.XFAM	XFAM & Native	XFAM[8]
EBX	5	5	1	Power Event Trace Support	TD_PARAMS.XFAM	XFAM & Native	XFAM[8]
EBX	6	6	1	PSB/PMI Injection Support	TD_PARAMS.XFAM	XFAM & Native	XFAM[8]
EBX	7	7	1	PT Event Trace Support	TD_PARAMS.XFAM	XFAM & Native	XFAM[8]
EBX	31	8	24	Reserved	TD_PARAMS.XFAM	XFAM & Native	XFAM[8]
ECX	0	0	1	ToPA Output Supported	TD_PARAMS.XFAM	XFAM & Native	XFAM[8]
ECX	1	1	1	ToPA Tables Support Multiple Regions	TD_PARAMS.XFAM	XFAM & Native	XFAM[8]
ECX	2	2	1	Single-Range Output Supported	TD_PARAMS.XFAM	XFAM & Native	XFAM[8]
ECX	30	3	28	Reserved	TD_PARAMS.XFAM	XFAM & Native	XFAM[8]
ECX	31	31	1	IP Payload Contains LIP	TD_PARAMS.XFAM	XFAM & Native	XFAM[8]
EDX	31	0	32	Reserved	TD_PARAMS.XFAM	XFAM & Native	XFAM[8]
Leaf 0x14 / Sub-Leaf 0x1							
EAX	1	0	2	MTC Period Options	TD_PARAMS.XFAM	XFAM & Native	XFAM[8]
EAX	15	2	14	Reserved	TD_PARAMS.XFAM	XFAM & Native	XFAM[8]
EAX	31	16	16	Number of Address Ranges Supported	TD_PARAMS.XFAM	XFAM & Native	XFAM[8]
EBX	15	0	16	Cycle Thresholds	TD_PARAMS.XFAM	XFAM & Native	XFAM[8]
EBX	31	16	16	PSB Frequencies	TD_PARAMS.XFAM	XFAM & Native	XFAM[8]
ECX	31	0	32	Reserved	TD_PARAMS.XFAM	XFAM & Native	XFAM[8]
EDX	31	0	32	Reserved	TD_PARAMS.XFAM	XFAM & Native	XFAM[8]
Leaf 0x15							
EAX	31	0	32	Denominator		Fixed	0x1
EBX	31	0	32	Numerator	TD_PARAMS.TSC_FREQUENCY	Configured	
ECX	31	0	32	Nominal ART Frequency		Fixed	0x017D7840
EDX	31	0	32	Reserved		Fixed	0x0
Leaf 0x19							
EAX	0	0	1	CPL0 Restriction		Attributes & Native	KL
EAX	1	1	1	Decrypt only Restriction		Attributes & Native	KL
EAX	2	2	1	Encrypt only Restriction		Attributes & Native	KL
EAX	3	3	1	Process Restriction		Attributes & Native	KL
EAX	31	4	28	Reserved_31_4		Attributes & Native	KL
EBX	0	0	1	AES KL Enabled		Attributes & Native	KL
EBX	1	1	1	Reserved		Attributes & Native	KL
EBX	2	2	1	AES wide KL Support		Attributes & Native	KL
EBX	3	3	1	Reserved		Attributes & Native	KL
EBX	4	4	1	IW Key Backup Support		Attributes & Native	KL
EBX	31	5	27	Reserved		Attributes & Native	KL
ECX	0	0	1	LOADIWKEY No Backup parameter Support		Attributes & Native	KL
ECX	1	1	1	Random IWKey Support		Fixed	0x0
ECX	31	2	30	Reserved		Fixed	0x0
EDX	31	0	32	Reserved		Fixed	0x0
Leaf 0x1c							
EAX	7	0	8	Supported LBR depth values	TD_PARAMS.XFAM, TD_PARAMS.CPUID_CONFIG	XFAM & Configured & Native	XFAM[15]
EAX	29	8	22	Reserved_29_8	TD_PARAMS.XFAM	XFAM & Native	XFAM[15]
EAX	30	30	1	Deep C-state May Reset	TD_PARAMS.XFAM	XFAM & Native	XFAM[15]
EAX	31	31	1	IP values contain LIP	TD_PARAMS.XFAM	XFAM & Native	XFAM[15]
EBX	0	0	1	CPL Filtering Supported	TD_PARAMS.XFAM	XFAM & Native	XFAM[15]
EBX	1	1	1	Branch Filtering Supported	TD_PARAMS.XFAM	XFAM & Native	XFAM[15]
EBX	2	2	1	Call-stack Mode Supported	TD_PARAMS.XFAM	XFAM & Native	XFAM[15]
EBX	31	3	29	Reserved_31_3	TD_PARAMS.XFAM	XFAM & Native	XFAM[15]
ECX	0	0	1	Mispredict Bit Supported	TD_PARAMS.XFAM	XFAM & Native	XFAM[15]
ECX	1	1	1	Timed LBRs Supported	TD_PARAMS.XFAM	XFAM & Native	XFAM[15]
ECX	2	2	1	Branch Type Field Supported	TD_PARAMS.XFAM	XFAM & Native	XFAM[15]
ECX	31	3	29	Reserved_31_3	TD_PARAMS.XFAM	XFAM & Native	XFAM[15]
EDX	31	0	32	Reserved	TD_PARAMS.XFAM	XFAM & Native	XFAM[15]
Leaf 0x1d / Sub-Leaf 0x0							
EAX	0	0	1	TILE support	TD_PARAMS.XFAM	XFAM & Native	XFAM[18:17]
EAX	31	1	31	Reserved_31_1	TD_PARAMS.XFAM	XFAM & Native	XFAM[18:17]
EBX	31	0	32	Reserved	TD_PARAMS.XFAM	XFAM & Native	XFAM[18:17]
ECX	31	0	32	Reserved	TD_PARAMS.XFAM	XFAM & Native	XFAM[18:17]

CPUID Field				Configuration	Virtualization		
Reg.	MSB	LSB	Field Size	Field Name	Configuration Details	Bit or Field Virtualization Type	Virtualization Details
EDX	31	0	32	Reserved	TD_PARAMS.XFAM	XFAM & Native	XFAM[18:17]
Leaf 0x1d / Sub-Leaf 0x1							
EAX	15	0	16	total_tile_bytes	TD_PARAMS.XFAM	XFAM & Native	XFAM[18:17]
EAX	31	16	16	bytes_per_tile	TD_PARAMS.XFAM	XFAM & Native	XFAM[18:17]
EBX	15	0	16	bytes_per_row	TD_PARAMS.XFAM	XFAM & Native	XFAM[18:17]
EBX	31	16	16	max_names	TD_PARAMS.XFAM	XFAM & Native	XFAM[18:17]
ECX	15	0	16	max_rows	TD_PARAMS.XFAM	XFAM & Native	XFAM[18:17]
ECX	31	16	16	Reserved_31_16	TD_PARAMS.XFAM	XFAM & Native	XFAM[18:17]
EDX	31	0	32	Reserved	TD_PARAMS.XFAM	XFAM & Native	XFAM[18:17]
Leaf 0x1e / Sub-Leaf N/A							
EAX	31	0	32	Reserved	TD_PARAMS.XFAM	XFAM & Native	XFAM[18:17]
EBX	7	0	8	impl.tmul_maxk (rows or cols)	TD_PARAMS.XFAM	XFAM & Native	XFAM[18:17]
EBX	23	8	16	impl.tmul_maxn (column bytes)	TD_PARAMS.XFAM	XFAM & Native	XFAM[18:17]
EBX	31	24	8	Reserved_31_24	TD_PARAMS.XFAM	XFAM & Native	XFAM[18:17]
ECX	31	0	32	Reserved	TD_PARAMS.XFAM	XFAM & Native	XFAM[18:17]
EDX	31	0	32	Reserved	TD_PARAMS.XFAM	XFAM & Native	XFAM[18:17]
Leaf 0x20							
EAX	31	0	32	Reserved		Fixed	0x0
EBX	31	0	32	Reserved		Fixed	0x0
ECX	31	0	32	Reserved		Fixed	0x0
EDX	31	0	32	Reserved		Fixed	0x0
Leaf 0x21 / Sub-Leaf 0x0							
EAX	31	0	32	Maximum sub-leaf		Fixed	0x00000000
EBX	31	0	32	"Inte"		Fixed	0x65746E49
ECX	31	0	32	" "		Fixed	0x20202020
EDX	31	0	32	"ITDX"		Fixed	0x5844546C
Leaf 0x22							
EAX	31	0	32	Reserved		Fixed	0x0
EBX	31	0	32	Reserved		Fixed	0x0
ECX	31	0	32	Reserved		Fixed	0x0
EDX	31	0	32	Reserved		Fixed	0x0
Leaf 0x23 / Sub-Leaf 0x0							
EAX	31	0	32	Valid sub-leaf bitmap	TD_PARAMS.ATTRIBUTES.PERFMON, TD_PARAMS.CPUID_CONFIG	Attributes & Configured & Native	PERFMON
EBX	31	0	32	Perfmon feature bits	TD_PARAMS.ATTRIBUTES.PERFMON, TD_PARAMS.CPUID_CONFIG	Attributes & Configured & Native	PERFMON
ECX	31	0	32	Reserved		Fixed	0x0
EDX	31	0	32	Reserved		Fixed	0x0
Leaf 0x23 / Sub-Leaf 0x1							
EAX	7	0	8	PMC counter bitmap	TD_PARAMS.ATTRIBUTES.PERFMON, TD_PARAMS.CPUID_CONFIG	Attributes & Configured & Native	PERFMON
EAX	31	8	24	PMC counter bitmap	TD_PARAMS.ATTRIBUTES.PERFMON, TD_PARAMS.CPUID_CONFIG	Attributes & Configured & Native	PERFMON
EBX	3	0	4	Fixed counter bitmap		Fixed	0x0
EBX	31	4	28	Fixed counter bitmap		Fixed	0x0
ECX	31	0	32	Reserved		Fixed	0x0
EDX	31	0	32	Reserved		Fixed	0x0
Leaf 0x23 / Sub-Leaf 0x2							
EAX	31	0	32	Reserved	TD_PARAMS.ATTRIBUTES.PERFMON, TD_PARAMS.CPUID_CONFIG	Fixed	0x0
EBX	31	0	32	Reserved	TD_PARAMS.ATTRIBUTES.PERFMON, TD_PARAMS.CPUID_CONFIG	Fixed	0x0
ECX	31	0	32	Reserved		Fixed	0x0
EDX	31	0	32	Reserved		Fixed	0x0
Leaf 0x23 / Sub-Leaf 0x3							
EAX	31	0	32	Perfmon events bitmap	TD_PARAMS.ATTRIBUTES.PERFMON, TD_PARAMS.CPUID_CONFIG	Attributes & Configured & Native	PERFMON
EBX	31	0	32	Reserved	TD_PARAMS.ATTRIBUTES.PERFMON	Fixed	0x0
ECX	31	0	32	Reserved		Fixed	0x0
EDX	31	0	32	Reserved		Fixed	0x0
Leaf 0x80000000							
EAX	31	0	32	MaxIndex		Fixed	0x80000008
EBX	31	0	32	Reserved		Fixed	0x0
ECX	31	0	32	Reserved		Fixed	0x0
EDX	31	0	32	Reserved		Fixed	0x0
Leaf 0x80000001							
EAX	31	0	32	Reserved		Fixed	0x0
EBX	31	0	32	Reserved		Fixed	0x0

CPUID Field					Configuration	Virtualization	
Reg.	MSB	LSB	Field Size	Field Name	Configuration Details	Bit or Field Virtualization Type	Virtualization Details
ECX	0	0	1	LAHF/SAHF in 64-bit Mode		Fixed	0x1
ECX	4	1	4	Reserved		Fixed	0x0
ECX	5	5	1	LZCNT		Fixed	0x1
ECX	7	6	2	Reserved		Fixed	0x0
ECX	8	8	1	PREFETCHW		Fixed	0x1
ECX	31	9	23	Reserved		Fixed	0x0
EDX	10	0	11	Reserved		Fixed	0x0
EDX	11	11	1	SYSCALL/SYSRET in 64-bit Mode		Calculated	1 in 64-bit mode, 0 in other modes
EDX	19	12	8	Reserved		Fixed	0x0
EDX	20	20	1	Execute Disable Bit		Fixed	0x1
EDX	25	21	5	Reserved		Fixed	0x0
EDX	26	26	1	1GB Pages		Fixed	0x1
EDX	27	27	1	RDTSCP and IA32_TSC_AUX		Fixed	0x1
EDX	28	28	1	Reserved_28		Fixed	0x0
EDX	29	29	1	Intel 64		Fixed	0x1
EDX	31	30	2	Reserved_31_30		Fixed	0x0
Leaf 0x80000007							
EAX	31	0	32	Reserved		Fixed	0x0
EBX	31	0	32	Reserved		Fixed	0x0
ECX	31	0	32	Reserved		Fixed	0x0
EDX	7	0	8	Reserved_7_0		Fixed	0x0
EDX	8	8	1	Invariant TSC		Fixed	0x1
EDX	31	9	23	Reserved_31_9		Fixed	0x0
Leaf 0x80000008							
EAX	7	0	8	Number of Physical Address Bits		Fixed	0x34
EAX	15	8	8	Number of Linear Address Bits		Fixed	0x39
EAX	31	16	16	Reserved		Fixed	0x0
EBX	8	0	9	Reserved_8_0		Fixed	0x0
EBX	9	9	1	WBNOINVD support	TD_PARAMS.CPUID_CONFIG	Configured & Native	
EBX	31	10	22	Reserved_31_10		Fixed	0x0
ECX	31	0	32	Reserved		Fixed	0x0
EDX	31	0	32	Reserved		Fixed	0x0

DRAFT

### 3. Data Types

This section describes data types that are designed to be used by the Intel TDX module.

#### 3.1. **UPDATED:** Interface Function Completion Status

**Note:** This section provides a high-level overview of function completion status, as defined. Implementation details may differ.

A high-level definition of the interface functions completion status is provided in the [TDX Module Base Spec].

##### 3.1.1. **UPDATED:** Function Completion Status Structure

**Table 3.1: Intel TDX Interface Functions Completion Status (Returned in RAX) Definition**

Bits	Name	Description
63	ERROR	Interface function aborted due to error. 0: Indicates that the function completed successfully – possibly with some warnings. 1: Indicates that the function aborted due to some error.
62	NON_RECOVERABLE	Recoverability hint – applicable only when ERROR is 1. 0: Indicates that the function may possibly be retried after some conditions have been corrected. 1: Indicates that the error is probably not recoverable.
61	FATAL	Fatality hint – applicable only for SEAMCALL. 0: Indicates that the TD can continue its normal lifecycle. 1: Indicates that the TD entered a state where it can only be torn down. E.g., when an import has failed and the TD's OP_STATE is FAILED_IMPORT.
60	HOST_RECOVERABILITY_HINT	As a TDH.VP.ENTER output, indicates a TDCALL that resulted in a trap-like TD exit for which the host VMM needs to provide a recoverability hint in the following TD entry. On the following TDH.VP.ENTER, the host VMM provides a hint to the guest TD, which is the output of the TDCALL: 0: The host VMM hints that the guest-side function may possibly be retried (e.g., the host may have corrected some conditions). 1: The host VMM hints that the error is probably not recoverable.
59:48	RESERVED	Reserved – set to 0
47:40	CLASS	Class of the function completion status
39:32	DETAILS_L1	Details of the function completion status
31:0	DETAILS_L2	Additional details of the function completion status – e.g., includes: <ul style="list-style-type: none"> <li>• Implicit or explicit operand identifier</li> <li>• CPUID leaf or sub-leaf</li> <li>• MSR index</li> <li>• VMCS field code</li> <li>• VM exit reason</li> <li>• CMR index</li> <li>• TDMR index</li> </ul>

### 3.1.2. Function Completion Status Code Classes (Bits 47:40)

**Table 3.2: Function Completion Status Code Classes (Bits 47:40) Definition**

Class ID	Class Name	Description
0	General	General function completion status
1	Invalid Operand	An invalid operand value has been provided, e.g., HKID is out of range, HPA overlaps SEAMRR, GPA is not private, etc.
2	Resource Busy	Resource is busy, there is a concurrency conflict.
3	Page Metadata	Page metadata (in PAMT) are incorrect, e.g., page type is wrong.
4	Dependent Resources	The state of dependent resources is incorrect, e.g., there are TD pages while trying to reclaim a TDR page.
5	Intel TDX Module State	The Intel TDX module state is incorrect.
6	TD State	The state of the TD is incorrect, e.g., it has not been initialized yet.
7	TD VCPU State	The state of the TD VCPU is incorrect, e.g., it is corrupted.
8	Key Management	The status code is related to key management, e.g., keys are not configured.
9	Platform	The status code is related to platform configuration or state.
10	Physical Memory	The status code is related to physical memory.
11	Guest TD Memory	The status code is related to guest TD memory.
12	Metadata	The status code is related to metadata (global scope, TD scope or VCPU scope)
13	Service TD	The status code is related to a service TD
14	Migration	The status code is related to TD migration
15	TDX I/O	The status code is related to TDX I/O
16	Measurement	The status code is related to TDX measurement
17	TD Partitioning	The status code is related to TD partitioning
255	Reserved	Reserved for use by host VMM or guest TD software This value is never used by the TDX module.

### 5 3.1.3. Function Completion Status Codes

**Table 3.3: Function Completion Status Codes Definition**

Flags, Class and Name (Bits 63:32)			Details L2 (Bits 31:0)	Description
Value (Hex)	Status	Name		
0x00000000	Success	TDX_SUCCESS	For TDH.VP.ENTER: Exit Reason For TDG.VP.ENTER: Exit Reason For list operations: Number of problematic sub-operations (e.g., a page in a list not processed due to an incorrect state).	Function completed successfully.
0x40000001	Non-Recover.	TDX_NON_RECOVERABLE_VCPU	For TDH.VP.ENTER: Exit Reason	TD exit due to a non-recoverable VCPU state (e.g., triple fault) – VCPU is disabled
0x60000002	Fatal	TDX_NON_RECOVERABLE_TD	For TDH.VP.ENTER: Exit Reason	TD exit due to a non-recoverable TD state – TD is disabled



Flags, Class and Name (Bits 63:32)			Details L2 (Bits 31:0)	Description
Value (Hex)	Status	Name		
0x80000003	Recover. Error	TDX_INTERRUPTED_RESUMABLE	For list operations: Number of problematic sub-operations (e.g., a page in a list not processed due to an incorrect state).	Function operation has been interrupted by an external event, and it may be resumed from the point it was interrupted by calling it again.
0x80000004	Recover. Error	TDX_INTERRUPTED_RESTARTABLE	For list operations: Number of problematic sub-operations (e.g., a page in a list not processed due to an incorrect state).	Function operation has been interrupted by an external event, and it may be restarted (from its beginning) by calling it again.
0x60000005	Fatal	TDX_NON_RECOVERABLE_TD_NON_ACCESSIBLE	For TDH.VP.ENTER: Exit Reason	TD exit due to a fatal TD state (e.g., machine check caused by a memory integrity check error) – TD is disabled and its private memory can't be accessed.
0xC0000006	Error	TDX_INVALID_RESUMPTION	0	Resumed function in invalid, e.g., its operands are different than the last interrupted function.
0xEE000007	Fatal	TDX_NON_RECOVERABLE_TD_WRONG_APIC_MODE	TDH.VP.ENTER: Exit Reason	TD is disabled due to host VMM running with wrong local APIC mode, e.g.: - Local APIC is disabled. - Local APIC is mode is xAPIC and there are more than 255 logical processors in the platform.
0x80000008	Recover. Error	TDX_CROSS_TD_FAULT	0	Fault-like TD exit due to a cross-TD error, i.e., the current TD encountered an error that is related to some other TD.
0x90000009	Host Recover. Error	TDX_CROSS_TD_TRAP	0	Trap-like TD exit due to a cross-TD error, i.e., the current TD encountered an error that is related to some other TD.
0x6000000A	Fatal	TDX_NON_RECOVERABLE_TD_CORRUPTED_MD	0	TD exit due to a non-recoverable corrupted TD metadata – TD is disabled
0xC0000100	Error	TDX_OPERAND_INVALID	Operand ID	Operand is invalid.
0xC0000101	Error	TDX_OPERAND_ADDR_RANGE_ERROR	Operand ID	Operand address is out of range (e.g., not in a TDMR).
0x80000200	Recover. Error	TDX_OPERAND_BUSY	Operand ID	The operand is busy (e.g., it is locked in Exclusive mode).
0x80000201	Recover. Error	TDX_PREVIOUS_TLB_EPOCH_BUSY	0	TDH.MEM.TRACK failed because one or more of the TD's VCPUs are running, and their VCPU epoch is the previous TD epoch.
0x80000202	Recover. Error	TDX_SYS_BUSY	0	The Intel TDX module (as a whole) is busy.
0x80000203	Recover. Error	TDX_RND_NO_ENTROPY	0	Random number generation (e.g., RDRAND or RDSEED) failed because the hardware random number generator did not have enough entropy. The caller should retry the operation.
0x80000204	Recover. Error	TDX_OPERAND_BUSY_HOST_PRIORITY	Operand ID	For host-side functions: The operand is busy; HOST_PRIORITY has been set - the host VMM should retry the operation until successful to avoid guest being stuck on host priority. For guest-side functions: The operand is busy (e.g., it is locked in Exclusive mode) due to host priority.
0x90000205	Host Recover. Error	TDX_HOST_PRIORITY_BUSY_TIMEOUT	0	Guest TD encountered a resource that has been busy due to host priority for more than the configured timeout. This is typically the result of the host VMM failing to acquire access to some resource and not retrying the operation until successful.
0xC0000300	Error	TDX_PAGE_METADATA_INCORRECT	Operand ID	Physical page metadata (in PAMT) are incorrect for the requested operation.
0x00000301	Success	TDX_PAGE_ALREADY_FREE	Operand ID	Physical page is already marked as PT_FREE.
0xC0000302	Error	TDX_PAGE_NOT_OWNED_BY_TD	Operand ID	Physical page PAMT entry's OWNER field does not point to the TD's TDR page



Flags, Class and Name (Bits 63:32)			Details L2 (Bits 31:0)	Description
Value (Hex)	Status	Name		
0xC0000303	Error	TDX_PAGE_NOT_FREE	Operand ID	Physical page is not free
0xC0000400	Error	TDX_TD_ASSOCIATED_PAGES_EXIST	0	Physical pages associated with the TD exist in memory.
0xC0000500	Error	TDX_SYS_INIT_NOT_PENDING	0	Attempting TDH.SYS.INIT when not expected.
0xC0000502	Error	TDX_SYS_LP_INIT_NOT_DONE	0	Attempting non-TDH.SYS.LP.INIT SEAMCALL leaf before TDH.SYS.LP.INIT was done on this LP.
0xC0000503	Error	TDX_SYS_LP_INIT_DONE	0	Attempting TDH.SYS.LP.INIT when already done on this LP.
0xC0000505	Error	TDX_SYS_NOT_READY	0	Attempting to execute a non-initialization SEAMCALL function before initialization sequence completed.
0xC0000506	Error	TDX_SYS_SHUTDOWN	0	Attempting to execute SEAMCALL when the Intel TDX module is being shut down.
0xC0000507	Error	TDX_SYS_KEY_CONFIG_NOT_PENDING	0	Attempting TDH.SYS.KEY.CONFIG when it is not pending.
0xC0000508	Error	TDX_SYS_STATE_INCORRECT	0	TDX module state is incorrect for the requested operation.
0xC0000509	Error	TDX_SYS_INVALID_HANDOFF	0	Handoff data is incorrect for TD-preserving TDX module update.
0xC000050A	Error	TDX_SYS_INCOMPATIBLE_SIGSTRUCT	0	TDX module SIGSTRUCT is incompatible with the current TDX module.
0xC000050B	Error	TDX_SYS_LP_INIT_NOT_PENDING	0	Attempting TDH.LP.INIT when it is not pending.
0xC000050C	Error	TDX_SYS_CONFIG_NOT_PENDING	0	Attempting TDH.SYS.CONFIG when it is not pending.
0xC0000600	Error	TDX_TD_NOT_INITIALIZED	Operand ID (0 if default)	TD has not been initialized (by TDH.MNG.INIT).
0xC0000601	Error	TDX_TD_INITIALIZED	Operand ID (0 if default)	TD has been initialized (by TDH.MNG.INIT).
0xC0000602	Error	TDX_TD_NOT_FINALIZED	Operand ID (0 if default)	TD measurement has not been finalized (by TDH.MR.FINALIZE).
0xC0000603	Error	TDX_TD_FINALIZED	Operand ID (0 if default)	TD measurement has been finalized (by TDH.MR.FINALIZE).
0xE0000604	Fatal	TDX_TD_FATAL	Operand ID (0 if default)	TD is in a FATAL error state.
0xC0000605	Error	TDX_TD_NON_DEBUG	Operand ID (0 if default)	TD's ATTRIBUTES.DEBUG bit is 0.
0xC0000606	Error	TDX_TDCS_NOT_ALLOCATED	Operand ID (0 if default)	TDCS pages have not been allocated
0xC0000607	Error	TDX_LIFECYCLE_STATE_INCORRECT	Operand ID (0 if default)	The TD's LIFECYCLE_STATE is incorrect for the required operation.
0xC0000608	Error	TDX_OP_STATE_INCORRECT	Operand ID (0 if default)	The TD's OP_STATE is incorrect for the required operation.
0xC0000610	Error	TDX_TDCX_NUM_INCORRECT	Operand ID (0 if default)	The number of TDCX pages is incorrect.
0xC0000700	Error	TDX_VCPU_STATE_INCORRECT	0	The VCPU state is incorrect for the requested operation.
0x80000701	Recover. Error	TDX_VCPU_ASSOCIATED	0	The VCPU is already associated with another LP.
0x80000702	Recover. Error	TDX_VCPU_NOT_ASSOCIATED	0	The VCPU is not associated with the current LP.
0xC0000704	Error	TDX_NO_VALID_VE_INFO	0	There is no valid #VE information.

Flags, Class and Name (Bits 63:32)			Details L2 (Bits 31:0)	Description
Value (Hex)	Status	Name		
0xC0000705	Error	TDX_MAX_VCPUS_EXCEEDED	0	TD's maximum number of VCPUs has been exceeded.
0xC0000706	Error	TDX_TSC_ROLLBACK	0	Time Stamp Counter value is lower than on last TD exit.
0xC0000730	Error	TDX_TD_VMCS_FIELD_NOT_INITIALIZED	Bits 31:0: VMCS field code	The TD VMCS field has not been initialized.
0xC0000731	Error	TD_VMCS_FIELD_ERROR	Bits 31:0: VMCS field code	The TD VMCS field read or write failed.
0x80000800	Recover. Error	TDX_KEY_GENERATION_FAILED	0	Failed to generate a random key. This is typically caused by an entropy error of the CPU's random number generator, and may be impacted by RDSEED, RDRAND or PCONFIG executing on other LPs. The operation should be retried.
0x80000810	Recover. Error	TDX_TD_KEYS_NOT_CONFIGURED	0	TD keys have not been configured on the hardware.
0xC0000811	Error	TDX_KEY_STATE_INCORRECT	0	KOT entry state is incorrect for the required operation.
0x00000815	Success	TDX_KEY_CONFIGURED	0	The key is already configured on the current package.
0x80000817	Recover. Error	TDX_WBCACHE_NOT_COMPLETE	0	Attempting to execute TDH.MNG.KEY.FREEID when TDH.PHYMEM.CACHE.WB has not completed its operation.
0xC0000820	Error	TDX_HKID_NOT_FREE	0	A provided HKID cannot be assigned because it is not free.
0x00000821	Success	TDX_NO_HKID_READY_TO_WBCACHE	0	No private HKID is in the HKID_FLUSHED state, ready for TDH.PHYMEM.CACHE.WB.
0xC0000823	Error	TDX_WBCACHE_RESUME_ERROR	0	Resume of a previously interrupted function has been aborted due to wrong HKID.
0x80000824	Recover. Error	TDX_FLUSHVP_NOT_DONE	0	TDH.VP.FLUSH was not done on all required VCPUs; some VCPUs are still associated with LPs.
0xC0000825	Error	TDX_NUM_ACTIVATED_HKIDS_NOT_SUPPORTED	Bits 31:0: Maximum supported HKIDs	The number of activated key IDs on the platform is not supported.
0xC0000900	Error	TDX_INCORRECT_CPUID_VALUE	0	A CPUID value is incorrect.
0xC0000901	Error	TDX_BOOT_NT4_SET	0	MSR IA32_MISC_ENABLES bit 22 (Boot NT4) is set.
0xC0000902	Error	TDX_INCONSISTENT_CPUID_FIELD	0	A field returned by CPUID is inconsistent between LPs.
0xC0000903	Error	TDX_CPUID_MAX_SUBLEAVES_UNRECOGNIZED	CPUID leaf	The maximum number of sub-leaves for this CPUID leaf is not recognized.
0xC0000904	Error	TDX_CPUID_LEAF_1F_FORMAT_UNRECOGNIZED	0	CPUID leaf 1F format is not recognized or sub-leaves are not in order.
0xC0000905	Error	TDX_INVALID_WBINVD_SCOPE	0	WBINVD scope is not supported.
0xC0000906	Error	TDX_INVALID_PKG_ID	Package ID	Package ID is larger than the maximum supported.
0xC0000907	Error	TDX_ENABLE_MONITOR_FSM_NOT_SET	0	MSR IA32_MISC_ENABLES bit 3 (Enable Monitor FSM) is not set.
0xC0000908	Error	TDX_CPUID_LEAF_NOT_SUPPORTED	CPUID leaf	
0xC0000910	Error	TDX_SMRR_NOT_LOCKED	0: SMRR, 1: SMRR2	SMRR* is not locked.
0xC0000911	Error	TDX_INVALID_SMRR_CONFIGURATION	0: SMRR, 1: SMRR2	SMRR* configuration is invalid.
0xC0000912	Error	TDX_SMRR_OVERLAPS_CMR	Bits 7:0: 0: SMRR, 1: SMRR2 Bits 15:8: Overlapping CMR index	SMRR* overlaps a CMR.

Flags, Class and Name (Bits 63:32)			Details L2 (Bits 31:0)	Description
Value (Hex)	Status	Name		
0xC0000913	Error	TDX_SMRR_LOCK_NOT_SUPPORTED	0	Platform does not support SMRR locking.
0xC0000914	Error	TDX_SMRR_NOT_SUPPORTED	0	Platform does not support SMRR.
0xC0000920	Error	TDX_INCONSISTENT_MSR	Bits 31:0: MSR index	MSR configuration is inconsistent between LPs.
0xC0000921	Error	TDX_INCORRECT_MSR_VALUE	Bits 31:0: MSR index	MSR value is incorrect.
0xC0000930	Error	TDX_SEAMREPORT_NOT_AVAILABLE	0	SEAMOPS(SEAMREPORT) instruction leaf is not available.
0xC0000931	Error	TDX_SEAMDB_GETREF_NOT_AVAILABLE	0	SEAMOPS(SEAMDB_GETREF) instruction leaf is not available.
0xC0000932	Error	TDX_SEAMDB_REPORT_NOT_AVAILABLE	0	SEAMOPS(SEAMDB_REPORT) instruction leaf is not available.
0xC0000933	Error	TDX_SEAMVERIFYREPORT_NOT_AVAILABLE	0	SEAMOPS(SEAMVERIFYREPORT) instruction leaf is not available.
0xC0000A00	Error	TDX_INVALID_TDMR	Bits 7:0: TDMR index	TDMR base address is not aligned on 1GB, its HKID bits are not 0, TDMR size is not specified with 1GB granularity or TDMR is outside the platform's maximum PA.
0xC0000A01	Error	TDX_NON_ORDERED_TDMR	Bits 7:0: TDMR index	TDMR is not specified in an ascending, non-overlapping order.
0xC0000A02	Error	TDX_TDMR_OUTSIDE_CMRS	Bits 7:0: TDMR index	TDMR non-reserved parts are not fully contained in CMRs.
0x00000A03	Success	TDX_TDMR_ALREADY_INITIALIZED	0	TDMR is already fully initialized.
0xC0000A10	Error	TDX_INVALID_PAMT	Bits 7:0: TDMR index Bits 15:8: PAMT level (2: 1GB, 1: 2MB, 0: 4KB)	PAMT region base address is not aligned on 4KB, its HKID bits are not 0, PAMT region size is not specified with 4KB granularity, it is not large enough for the TDMR size or PAMT region is outside the platform's maximum PA.
0xC0000A11	Error	TDX_PAMT_OUTSIDE_CMRS	Bits 7:0: TDMR index Bits 15:8: PAMT level (2: 1GB, 1: 2MB, 0: 4KB)	PAMT is not fully contained in CMRs.
0xC0000A12	Error	TDX_PAMT_OVERLAP	Bits 7:0: TDMR index Bits 15:8: PAMT level (2: 1GB, 1: 2MB, 0: 4KB) Bits 23:16: Overlapping TDMR index	PAMT overlaps with TDMR non-reserved parts or with another PAMT.
0xC0000A20	Error	TDX_INVALID_RESERVED_IN_TDMR	Bits 7:0: TDMR index Bits 15:8: Reserved area index	Reserved area in TDMR's base offset is not aligned on 4KB, its size is not specified with 4KB granularity or it is not fully contained within the TDMR.
0xC0000A21	Error	TDX_NON_ORDERED_RESERVED_IN_TDMR	Bits 7:0: TDMR index Bits 15:8: Reserved area index	Reserved area in TDMR is not specified in an ascending, non-overlapping order.
0xC0000A22	Error	TDX_CMRS_LIST_INVALID	0	CMR list provided to the TDX module is invalid
0xC0000B00	Error	TDX_EPT_WALK_FAILED	Operand ID	EPT walk failed
0xC0000B01	Error	TDX_EPT_ENTRY_FREE	Operand ID	EPT entry is free
0xC0000B02	Error	TDX_EPT_ENTRY_NOT_FREE	Operand ID	EPT entry is not free
0xC0000B03	Error	TDX_EPT_ENTRY_NOT_PRESENT	Operand ID	EPT entry is not present
0xC0000B04	Error	TDX_EPT_ENTRY_NOT_LEAF	Operand ID	EPT entry is not a leaf

Flags, Class and Name (Bits 63:32)			Details L2 (Bits 31:0)	Description
Value (Hex)	Status	Name		
0xC0000B05	Error	TDX_EPT_ENTRY_LEAF	Operand ID	EPT entry is a leaf
0xC0000B06	Error	TDX_GPA_RANGE_NOT_BLOCKED	Operand ID	GPA range is not blocked
0x00000B07	Success	TDX_GPA_RANGE_ALREADY_BLOCKED	Operand ID	GPA range is already blocked
0xC0000B08	Error	TDX_TLB_TRACKING_NOT_DONE	Operand ID	TLB tracking has not been done
0xC0000B09	Error	TDX_EPT_INVALID_PROMOTE_CONDITIONS	Operand ID	Conditions for GPA mapping promotions as invalid
0x00000B0A	Success	TDX_PAGE_ALREADY_ACCEPTED	Error EPT level	Page has already been accepted
0xC0000B0B	Error	TDX_PAGE_SIZE_MISMATCH	Error EPT level	Requested page size does not match the current GPA mapping size
0xC0000B0C	Error	TDX_GPA_RANGE_BLOCKED	Operand ID	GPA range is blocked
0xC0000B0D	Error	TDX_EPT_ENTRY_STATE_INCORRECT	Operand ID	EPT entry state is incorrect
0xC0000B0E	Error	TDX_EPT_PAGE_NOT_FREE	Operand ID	EPT page is not free
0xC0000B0F	Error	TDX_L2_SEPT_WALK_FAILED	Bits 15:0: Error EPT level Bits 31:16: VM index	L2 Secure EPT walk failed
0xC0000B10	Error	TDX_L2_SEPT_ENTRY_NOT_FREE	Bits 15:0: Error EPT level Bits 31:16: VM index	L2 Secure EPT entry is not free
0xC0000B11	Error	TDX_PAGE_ATTR_INVALID	Operand ID	The attributes combination requested for a page is invalid.
0xC0000B12	Error	TDX_L2_SEPT_PAGE_NOT_PROVIDED	Operand ID	On TDH.MEM.PAGE.DEMOTE, a new L2 SEPT page was not provided but a page alias exists.
0xC0000C00	Error	TDX_METADATA_FIELD_ID_INCORRECT	For a single field, set to 0. For a metadata list: Bits 15:0: Sequence number 0xFFFF indicates the list header. Bits 31:16: Field number in sequence 0xFFFF indicates the sequence header.	The provided FIELD_ID is incorrect.
0xC0000C01	Error	TDX_METADATA_FIELD_NOT_WRITABLE	For a single field, set to 0. For a metadata list: Bits 15:0: Sequence number 0xFFFF indicates the list header. Bits 31:16: Field number in sequence 0xFFFF indicates the sequence header.	Field code and write mask are for a read-only field.

Flags, Class and Name (Bits 63:32)			Details L2 (Bits 31:0)	Description
Value (Hex)	Status	Name		
0xC0000C02	Error	TDX_METADATA_FIELD_NOT_READABLE	For a single field, set to 0. For a metadata list: Bits 15:0: Sequence number 0xFFFF indicates the list header. Bits 31:16: Field number in sequence 0xFFFF indicates the sequence header.	Field code is for an unreadable field.
0xC0000C03	Error	TDX_METADATA_FIELD_VALUE_NOT_VALID	For a single field, set to 0. For a metadata list: Bits 15:0: Sequence number 0xFFFF indicates the list header. Bits 31:16: Field number in sequence 0xFFFF indicates the sequence header.	The provided field value is not valid.
0xC0000C04	Error	TDX_METADATA_LIST_OVERFLOW	For a single field, set to 0. For a metadata list: Bits 15:0: Sequence number 0xFFFF indicates the list header. Bits 31:16: Field number in sequence 0xFFFF indicates the sequence header.	A metadata list does not fit within the provided buffer
0xC0000C05	Error	TDX_INVALID_METADATA_LIST_HEADER	0	Metadata list header is invalid
0xC0000C06	Error	TDX_REQUIRED_METADATA_FIELD_MISSING	0	A required metadata field is missing
0xC0000C07	Error	TDX_METADATA_ELEMENT_SIZE_INCORRECT	For a single field, set to 0. For a metadata list: Bits 15:0: Sequence number 0xFFFF indicates the list header. Bits 31:16: Field number in sequence 0xFFFF indicates the sequence header.	A metadata field identifier specifies an incorrect ELEMENT_SIZE_CODE for the field
0xC0000C08	Error	TDX_METADATA_LAST_ELEMENT_INCORRECT	For a single field, set to 0. For a metadata list: Bits 15:0: Sequence number 0xFFFF indicates the list header. Bits 31:16: Field number in sequence 0xFFFF indicates the sequence header.	A metadata field identifier specifies an incorrect LAST_ELEMENT_IN_FIELD for the field
0xC0000C09	Error	TDX_METADATA_FIELD_CURRENTLY_NOT_WRITABLE	For a single field, set to 0. For a metadata list: Bits 15:0: Sequence number 0xFFFF indicates the list header. Bits 31:16: Field number in sequence 0xFFFF indicates the sequence header.	The metadata field is currently not writable, e.g., per some state of the TD

Flags, Class and Name (Bits 63:32)			Details L2 (Bits 31:0)		Description
Value (Hex)	Status	Name			
0xC0000C0A	Error	TDX_METADATA_WR_MASK_NOT_VALID	For a single field, set to 0. For a metadata list: Bits 15:0: Sequence number 0xFFFF indicates the list header. Bits 31:16: Field number in sequence 0xFFFF indicates the sequence header.		The write mask value is not valid for the metadata field
0x00000C0B	Success	TDX_METADATA_FIRST_FIELD_ID_IN_CONTEXT	0		Indicates that the first accessible field ID in context is returned
0x00000C0C	Success	TDX_METADATA_FIELD_SKIP	0		Indicates that the field being read is to be skipped Used internally as part of a sequence read
0xC0000D00	Error	TDX_SERVTD_ALREADY_BOUND_FOR_TYPE	0		A single service TD of this type is supported
0xC0000D01	Error	TDX_SERVTD_TYPE_MISMATCH	0		Service TD type does not match the currently bound type
0xC0000D02	Error	TDX_SERVTD_ATTR_MISMATCH	0		Service TD attributes do not match the currently bound attributes
0xC0000D03	Error	TDX_SERVTD_INFO_HASH_MISMATCH	0		Service TD hash of TDINFO_STRUCT does not match the currently bound hash
0xC0000D04	Error	TDX_SERVTD_UUID_MISMATCH	0		Service TD UUID does not match the currently bound UUID
0xC0000D05	Error	TDX_SERVTD_NOT_BOUND		Binding slot number	Service TD is not bound
0xC0000D06	Error	TDX_SERVTD_BOUND	0		Service TD is already bound
0xC0000D07	Error	TDX_TARGET_UUID_MISMATCH	0		Target TD UUID does not match the requested TD_UUID
0xC0000D08	Error	TDX_TARGET_UUID_UPDATED	0		Target TD UUID does not match the requested TD_UUID, but pre-migration target TD UUID does match it
0xC0000E00	Error	TDX_INVALID_MBMD	0		MBMD is invalid
0xC0000E01	Error	TDX_INCORRECT_MBMD_MAC	0		MBMD.MAC field value is incorrect
0xC0000E02	Error	TDX_NOT_WRITE_BLOCKED		Operand ID	Secure EPT entry is not blocked for writing
0x00000E03	Success	TDX_ALREADY_WRITE_BLOCKED		Operand ID	Secure EPT entry is already blocked for writing
0xC0000E04	Error	TDX_NOT_EXPORTED		Operand ID	Secure EPT entry is not marked as exported
0xC0000E05	Error	TDX_MIGRATION_STREAM_STATE_INCORRECT	0		Migration stream has not been initialized or is not enabled
0xC0000E06	Error	TDX_MAX_MIGS_NUM_EXCEEDED		MAX_MIGS	The maximum number of supported migration streams has been exceeded
0xC0000E07	Error	TDX_EXPORTED_DIRTY_PAGES_REMAIN	0		There are some pages that have been exported, but need to be re-exported because their contents have changed
0xC0000E08	Error	TDX_MIGRATION_SESSION_KEY_NOT_SET	0		A new migration session key has not been set before a migration session start is attempted
0xC0000E09	Error	TDX_TD_NOT_MIGRATABLE	0		The TD's ATTRIBUTES.MIGRATABLE bit is not set
0xC0000E0A	Error	TDX_PREVIOUS_EXPORT_CLEANUP_INCOMPLETE	0		A previous aborted export session cleanup (using TDH.EXPORT.CANCEL) has not been completed
0xC0000E0B	Error	TDX_NUM_MIGS_HIGHER_THAN_CREATED	0		The number of migration streams used by the session is higher than the number of created migration streams

Flags, Class and Name (Bits 63:32)			Details L2 (Bits 31:0)	Description
Value (Hex)	Status	Name		
0xC0000E0C	Error	TDX_IMPORT_MISMATCH	0	A re-import or an import cancellation does not match the existing Secure EPT entry
0xC0000E0D	Error	TDX_MIGRATION_EPOCH_OVERFLOW	0	Migration epoch has exceeded its maximum value
0xC0000E0E	Error	TDX_MAX_EXPORTS_EXCEEDED	0	Maximum number of TD export attempts (2 <sup>31</sup> ) has been exceeded
0xC0000E0F	Error	TDX_INVALID_PAGE_MAC	Operand ID	Imported page MAC is invalid
0xC0000E10	Error	TDX_MIGRATED_IN_CURRENT_EPOCH	Operand ID	Page already migrated in the current epoch
0xC0000E11	Error	TDX_DISALLOWED_IMPORT_OVER_REMOVED	Operand ID	Disallowed age import over a previously-removed page
0xC0000E12	Error	TDX_SOME_VCPUS_NOT_MIGRATED	0	Some VCPUs have not been migrated
0xC0000E13	Error	TDX_ALL_VCPUS_IMPORTED	0	Attempting to import a VCPU when all VCPUs have already been imported
0xC0000E14	Error	TDX_MIN_MIGS_NOT_CREATED	0	The minimum number of migration streams (2) to support migration has not been created.
0xC0001000	Error	TDX_INVALID_CPUSVN	0	REPORTMACSTRUCT.CPUSVN is invalid
0xC0001001	Error	TDX_INVALID_REPORTMACSTRUCT	0	REPORTMACSTRUCT is invalid
0x00001100	Success	TDX_L2_EXIT_HOST_ROUTED_ASYNC	For TDG.VP.ENTER: Exit Reason	L1-to-L2 entry succeeded. Later, following an asynchronous TD exit from L2, the host VMM requested resumption of L1.
0x00001101	Success	TDX_L2_EXIT_HOST_ROUTED_TDVMCALL	For TDG.VP.ENTER: Exit Reason	L1-to-L2 entry succeeded. Later, following a TDG.VP.VMCALL TD exit from L2, the host VMM requested resumption of L1.
0x00001102	Success	TDX_L2_EXIT_PENDING_INTERRUPT	For TDG.VP.ENTER: Exit Reason	L1-to-L2 entry succeeded, and later L2-to-L1 exit happened due to an interrupt that was posted to L1 and is pending.
0x00001120	Success	TDX_PENDING_INTERRUPT	0	L1-to-L2 entry was aborted because an interrupt is pending for the L1 VMM. This indication is returned even if the L1 VMMs cleared RFLAGS.IF.
0x00001140	Success	TDX_TD_EXIT_BEFORE_L2_ENTRY	0	A debuggable TD's L1-to-L2 entry was mutated to a TD exit
0xC0001160	Error	TDX_GLA_NOT_CANONICAL	0	Guest linear address is not canonical

### 3.1.4. Function Completion Status Operand IDs

Table 3.4: Function Completion Operand IDs Definition

Operand ID	Explicit/Implicit	Class	Operand	Description
0	Explicit	GPR	RAX	Explicit input operand RAX
1	Explicit	GPR	RCX	Explicit input operand RCX
2	Explicit	GPR	RDX	Explicit input operand RDX
3	Explicit	GPR	RBX	Explicit input operand RBX
4	Explicit	GPR	Reserved_RSP	Reserved

Operand ID	Explicit/Implicit	Class	Operand	Description
5	Explicit	GPR	RBP	Explicit input operand RBP
6	Explicit	GPR	RSI	Explicit input operand RSI
7	Explicit	GPR	RDI	Explicit input operand RDI
8	Explicit	GPR	R8	Explicit input operand R8
9	Explicit	GPR	R9	Explicit input operand R9
10	Explicit	GPR	R10	Explicit input operand R10
11	Explicit	GPR	R11	Explicit input operand R11
12	Explicit	GPR	R12	Explicit input operand R12
13	Explicit	GPR	R13	Explicit input operand R13
14	Explicit	GPR	R14	Explicit input operand R14
15	Explicit	GPR	R15	Explicit input operand R15
63-16	Explicit	Reserved	Reserved	Reserved for additional explicit operands (e.g., XMM).
64	Explicit	Component of explicit input	ATTRIBUTES	TD_PARAMS.ATTRIBUTES
65	Explicit	Component of explicit input	XFAM	TD_PARAMS.XFAM
66	Explicit	Component of explicit input	EXEC_CONTROLS	TD_PARAMS.EXEC_CONTROLS
67	Explicit	Component of explicit input	EPTP_CONTROLS	TD_PARAMS.EPTP_CONTROLS
68	Explicit	Component of explicit input	MAX_VCPUS	TD_PARAMS.MAX_VCPUS
69	Explicit	Component of explicit input	CPUID_CONFIG	TD_PARAMS.CPUID_CONFIG
70	Explicit	Component of explicit input	TSC_FREQUENCY	TD_PARAMS.TSC_FREQUENCY
71	Explicit	Component of explicit input	NUM_L2_VMS	TD_PARAMS.NUM_L2_VMS
72	Explicit	Component of explicit input	IA32_ARCH_CAPABILITIES_CONFIG	TD_PARAMS.IA32_ARCH_CAPABILITIES_CONFIG
94-73	Explicit	Component of explicit input	Reserved	Reserved for additional components.
95	Explicit	Component of explicit input	PAGE	PAGE PA array entry
96	Explicit	Component of explicit input	TDMR_INFO_PA	TDMR_INFO_PA array entry
97	Explicit	Component of explicit input	GPA_LIST_ENTRY	GPA_LIST array entry
98	Explicit	Component of explicit input	MIG_BUFF_LIST_ENTRY	Migration buffer list entry
99	Explicit	Component of explicit input	NEW_PAGE_LIST_ENTRY	New page list entry
127-100	Explicit	Component of explicit input	Reserved	Reserved for additional components.
128	Implicit	Physical Page	TDR	TDR Page
129	Implicit	Physical Page	TDCX	TDCX Page
130	Implicit	Physical Page	TDVPR	TDVPR Page
131	Implicit	Physical Page	Reserved	Reserved
132	Implicit	Physical Page	REG_PAGE	PT_REG private page
143-133	Implicit	Physical Page	Reserved	Reserved for additional implicit physical page types.



Operand ID	Explicit/Implicit	Class	Operand	Description
144	Implicit	TD logical control structure	TDCS	TDCS Logical Structure
145	Implicit	TD logical control structure	TDVPS	TDVPS Logical Structure
146	Implicit	TD logical control structure	SEPT_TREE	Secure EPT Tree
147	Implicit	TD logical control structure	SEPT_ENTRY	Secure EPT Entry
167-148	Implicit	TD logical control structure	Reserved	Reserved for additional implicit logical structure types.
168	Implicit	Component of logical control structure	RTMR	TDCS.RTMR
169	Implicit	Component of logical control structure	TD_EPOCH	TDCS.TD_EPOCH
170	Implicit	Component of logical control structure	L2_VAPIC_GPA	TDVPS.L2_VAPIC_GPA
171	Implicit	Component of logical control structure	MIGSC	TDCS.MIGSC_LINK and MIGSC page
172	Implicit	Component of logical control structure	OP_STATE	TDCS.OP_STATE
173	Implicit	Component of logical control structure	MIG	TDCS Migration Context
174	Implicit	Component of logical control structure	SERVTD_BINDINGS	Service TD bindings table
175	Implicit	Component of logical control structure	PASID_TRK	PASID tracking information
183-176	Implicit	Component of logical control structure	Reserved	Reserved for additional components.
184	Implicit	Abstract item	SYS	Intel TDX Module
185	Implicit	Abstract item	TDMR	TDMR
186	Implicit	Abstract item	KOT	KOT
187	Implicit	Abstract item	KET	KET
188	Implicit	Abstract item	WBCACHE	TDH.PHYMEM.CACHE.WB State
255-189	Implicit	Abstract item	Reserved	Reserved for additional abstract items
256	Implicit	TDX I/O abstract item	PASIDMT	PASID metadata table
257	Implicit	TDX I/O abstract item	MMIOMT	MMIO physical page metadata table
258	Implicit	TDX I/O abstract item	SPDMMT	SPDM metadata table
319-259	Implicit	TDX I/O abstract item	Reserved	Reserved for additional TDX I/O items
320	Implicit	Global control structure	DMAR	Trusted DMA remapping structure

### 3.2. Basic Crypto Types

Table 3.5: Basic Crypto Types

Name	Size (Bytes)	Description
SHA384_HASH	48	384-bit buffer containing the result of a SHA384 hash calculation
KEY128	16	128-bit key
KEY256	32	256-bit key

### 3.3. **UPDATED:** TDX Module Configuration, Enumeration and Initialization Types

- 5 **Note:** This section describes configuration, enumeration and initialization types, as defined. Implementation may differ.

#### 3.3.1. CPUID\_CONFIG

- 10 CPUID\_CONFIG is designed to enumerate how the host VMM may configure the virtualization done by the Intel TDX module for a single CPUID leaf and sub-leaf. An array of CPUID\_CONFIG entries is used for the Intel TDX module enumeration by TDH.SYS.INFO.

Table 3.6: CPUID\_CONFIG Definition

Field	Offset (Bytes)	Size (Bytes)	Description
LEAF	0	4	EAX input value to CPUID
SUB_LEAF	4	4	ECX input value to CPUID A value of -1 indicates a CPUID leaf with no sub-leaves.
EAX	8	4	Enumeration of the configurable virtualization of the value returned by CPUID in EAX: a value of 1 in any of the bits indicates that the host VMM is allowed to configure that bit
EBX	12	4	Enumeration of the configurable virtualization of the value returned by CPUID in EBX: a value of 1 in any of the bits indicates that the host VMM is allowed to configure that bit
ECX	16	4	Enumeration of the configurable virtualization of the value returned by CPUID in ECX: a value of 1 in any of the bits indicates that the host VMM is allowed to configure that bit
EDX	20	4	Enumeration of the configurable virtualization of the value returned by CPUID in EDX: a value of 1 in any of the bits indicates that the host VMM is allowed to configure that bit

#### 3.3.2. **NEW:** Global-Scope (TDX Module) Metadata

- 15 TDX module global scope fields provide enumeration information about the Intel TDX module. They are used with the TDH.SYS.RD, TDH.SYS.RDALL, TDG.SYS.RD and TDG.SYS.RDALL functions.

### 3.3.2.1. TDX Features Enumeration

The main enumeration of features supported by the TDX module is provided by the TDX\_FEATURES array of 64-bit metadata fields. The number of fields is enumerated by NUM\_TDX\_FEATURES.

The TDX module features of TDX 1.0 are considered a baseline. TDX\_FEATURES enumerate features beyond that baseline.

5 **Table 3.7: TDX\_FEATURES0 Definition**

Bit(s)	Name	Description
0	TD_MIGRATION	The TDX module supports TD migration. Further information is provided by the Migration fields.
1	TD_PRESERVING	The TDX module supports TD preserving updates. Further information is provided by the TDX Module Handoff metadata fields.
2	SERVICE_TD	The TDX module supports Service TDs. Further information is provided by the Service TD fields.
3	ENHANCED_METADATA	The TDX module supports enhanced metadata interface functions: <ul style="list-style-type: none"> <li>Version 1 of previously existing functions: TDH.MNG.RD/WR, TDH.VP.RD/WR and TDG.MNG.RD/WR.</li> <li>New functions: TDH.SYS.RD, TDH.SYS.RDALL, TDG.SYS.RD, TDG.SYS.RDALL, TDG.VP.RD/WR.</li> </ul>
4	RELAXED_MEM_MNG	The TDX module's memory management requirements are relaxed vs. TDX 1.0: <ul style="list-style-type: none"> <li>Many interface functions allow concurrent memory management operations by exclusively locking specific Secure EPT entries instead of the whole Secure EPT tree: <ul style="list-style-type: none"> <li>TDH.MEM.PAGE.AUG/DEMOTE/PROMOTE/RELOCATE/REMOVE</li> <li>TDH.MEM.SEPT.ADD</li> </ul> </li> <li>TLB tracking (e.g., TDH.MEM.RANGE.BLOCK followed by TDH.MEM.TRACK and IPIs) may be skipped if the TD's OP_STATE is such that the TD can't be running, i.e., in the following cases: <ul style="list-style-type: none"> <li>On normal TD build, the TD's measurement has not yet been finalized by TDH.MR.FINALIZED.</li> <li>The TD has been paused for export by TDH.EXPORT.PAUSE, and export has not been aborted by TDH.EXPOR.ABORT.</li> <li>On import, TD execution has been enabled by TDH.IMPORT.COMMIT or TDH.IMPORT.END.</li> </ul> </li> </ul>
5	CPUID_VIRT_GUEST_CTRL	Guest TD may request that on certain CPUID leaves/sub-leaves a #VE will always be injected, using the TDG.VP.RD/WR to access the TDVPS' CPUID_CONTROL fields.
6	TDX_IO	The TDX module and the CPU support TDX I/O.
7	TD_PARTITIONING	The TDX module supports TD partitioning: <ul style="list-style-type: none"> <li>New interface functions: TDG.MEM,PAGE.ATTR.RD/WR, TDG.VP.ENTER, TDG.VP.INVEPT, TDG.VP.INVVPID</li> <li>Updates to many interface functions, with new input and/or output operands</li> </ul>
8	LOCAL_ATTESTATION	The TDX module supports local attestation.
9	TD_ENTRY_ENHANCEMENTS	The TDX module supports the following TD entry enhancements:

Bit(s)	Name	Description
		<ul style="list-style-type: none"> <li>HOST_RECOVERABILITY_HINT: On trap-like asynchronous TD exit, bit 60 of the TDH.VP.ENTER completion status (returned in RAX) may be set to 1. In this case, the host VMM may set the following TD entry's input value of RCX' HOST_RECOVERABILITY_HINT bit; this bit is copied to the guest RAX bit 60, which the guest interprets as part of a TDCALL completion status.</li> </ul>
10	HOST_PRIORITY_LOCKS	The TDX module implements host-priority locks to avoid denial-of-service by guest TDs. This requires the host VMM to retry operations that fail with a TDX_OPERAND_BUSY_HOST_PRIORITY status.
11	CONFIG_IA32_ARCH_CAPABILITIES	The TDX module allows the host VMM to configure the virtualization of IA32_ARCH_CAPABILITIES MSR.
63:12	RESERVED	Set to 0

### 3.3.2.2. How to Read the Global Fields Table

The access columns describe whether this field is accessible to the host VMM and to guest TDs. Access is done using the TDX module metadata access functions, e.g., TDH.SYS.RD. Possible values are shown in the table below.

5

Table 3.8: Field Access Definition

Access	Meaning
None	No access to the field
RO	This field can only be read.

### 3.3.2.3. Global Metadata Fields

Table 3.9: Global Scope Metadata

Class	Field	VMM Access	Guest Access	Type	Field Size (Bytes)	Num Fields	Num Elem.	Elem. Size (Bytes)	Description	Base FIELD_ID (Hex)
Platform Info	NUM_PKGS	RO	None	Integer	4	1	1	4	Number of CPU packages in the system (1 - 8)	0x0000000200000000
Platform Info	PKG_FMS	RO	None	N/A	4	8	1	4	Array of version information (type, family, model, stepping) as returned by CPUID(1).EAX for each package. Unused entries (NUM_PKGS and above) are set to 0.	0x0000000200000001
TDX Module Version	VENDOR_ID	RO	RO	Integer	4	1	1	4	0x8086 for Intel	0x0800000200000000
TDX Module Version	BUILD_DATE	RO	None	BCD	4	1	1	4	Intel TDX module build data – in yyyymmdd BCD format (each digit occupies 4 bits)	0x8800000200000001
TDX Module Version	BUILD_NUM	RO	None	Integer	2	1	1	2	Build number of the Intel TDX module	0x8800000100000002
TDX Module Version	MINOR_VERSION	RO	RO	Integer	2	1	1	2	Minor version number of the Intel TDX module	0x0800000100000003
TDX Module Version	MAJOR_VERSION	RO	RO	Integer	2	1	1	2	Major version number of the Intel TDX module	0x0800000100000004
TDX Module Handoff	MODULE_HV	RO	None	16-bit integer	2	1	1	2	The native handoff version that this TDX module should support.	0x8900000100000000
TDX Module Handoff	MIN_UPDATE_HV	RO	None	16-bit integer	2	1	1	2	The minimum handoff version that this TDX module should support.	0x8900000100000001
TDX Module Handoff	NO_DOWNGRADE	RO	None	Boolean	1	1	1	1	A boolean flag that indicates whether this TDX module should disallow downgrades.	0x8900000000000002
TDX Module Handoff	NUM_HANDOFF_PAGES	RO	None	16-bit integer	2	1	1	2	The number of 4KB pages (minus 1) allocated at the beginning of the data region for handoff data.	0x8900000100000003
TDX Module Handoff	HANDOFF_DATA_VALID	RO	None	Boolean	1	1	1	1	A boolean flag that indicates whether the handoff data is valid. Handoff data is created by TDH.SYS.SHUTDOWN and consumed by TDH.SYS.UPDATE.	0x8900000000000004

Class	Field	VMM Access	Guest Access	Type	Field Size (Bytes)	Num Fields	Num Elem.	Elem. Size (Bytes)	Description	Base FIELD_ID (Hex)
TDX Module Handoff	HANDOFF_DATA_HV	RO	None	16-bit integer	2	1	1	2	The version of the handoff data. Handoff data is created by TDH.SYS.SHUTDOWN and consumed by TDH.SYS.UPDATE.	0x89000000100000005
TDX Module Handoff	HANDOFF_DATA_SIZE	RO	None	32-bit unsigned integer	4	1	1	4	Size of the HV-specific data, in bytes. Handoff data is created by TDH.SYS.SHUTDOWN and consumed by TDH.SYS.UPDATE.	0x89000000200000006
TDX Module Info	SYS_ATTRIBUTES	RO	RO	Bitmap	4	1	1	4	Module attributes Bits 30:0 Reserved – set to 0 Bit 31 0 indicates a production module. 1 indicates a debug module.	0x0A000000200000000
TDX Module Info	NUM_TDX_FEATURES	RO	RO	8-bit integer	1	1	1	1	Number of TDX_FEATURES fields	0x0A000000000000001
TDX Module Info	TDX_FEATURES0	RO	RO	64-bit bitmap	8	1	1	8	Enumerates TDX features: <b>Bit 0</b> TD_MIGRATION <b>Bit 1</b> TD_PRESERVING <b>Bit 2</b> SERVICE_TD <b>Bit 3</b> ENHANCED_METADATA <b>Bit 4</b> RELAXED_MEM_MNG <b>Bit 5</b> CPUID_VIRT_GUEST_CTRL <b>Bit 6</b> TDX_IO <b>Bit 7</b> TD_PARTITIONING <b>Bit 8</b> LOCAL_ATTESTATION <b>Bit 9</b> TD_ENTRY_ENHANCEMENTS <b>Bit 10</b> HOST_PRIORITY_LOCKS <b>Bit 11</b> CONFIG_IA32_ARCH_CAPABILITIES <b>Bits 63:12</b> Reserved, set to 0	0x0A000000300000008
CMR Info	NUM_CMRS	RO	None	Integer	2	1	1	2	Number of the following CMR entries	0x90000000100000000
CMR Info	CMR_BASE	RO	None	Physical Address	8	32	1	8	Array of CMR base addresses Since a CMR is aligned on 4KB, bits 11:0 are 0.	0x90000000300000080
CMR Info	CMR_SIZE	RO	None	Integer	8	32	1	8	Array of CMR sizes, in bytes Since a CMR is aligned on 4KB, bits 11:0 are 0. A value of 0 indicates a null entry.	0x90000000300000100
TDMR Info	MAX_TDMRS	RO	None	Integer	2	1	1	2	The maximum number of TDMRs supported	0x91000000100000008
TDMR Info	MAX_RESERVED_PER_TDMR	RO	None	Integer	2	1	1	2	The maximum number of reserved areas per TDMR	0x91000000100000009
TDMR Info	PAMT_4K_ENTRY_SIZE	RO	None	Integer	2	1	1	2	The size of a PAMT_4K (1 entry per 4KB of TDMR) entry, in bytes – determines the number of bytes that need to be reserved for the PAMT_4K area.	0x91000000100000010
TDMR Info	PAMT_2M_ENTRY_SIZE	RO	None	Integer	2	1	1	2	The size of a PAMT_2M (1 entry per 2MB of TDMR) entry, in bytes – determines the number of bytes that need to be reserved for the PAMT_2M area.	0x91000000100000011
TDMR Info	PAMT_1G_ENTRY_SIZE	RO	None	Integer	2	1	1	2	The size of a PAMT_1G (1 entry per 1GB of TDMR) entry, in bytes – determines the number of bytes that need to be reserved for the PAMT_1G area.	0x91000000100000012
TD Control Structures	TDR_BASE_SIZE	RO	None	Integer	2	1	1	2	Base value for the number of bytes required to hold TDR	0x98000000100000000
TD Control Structures	TDCS_BASE_SIZE	RO	None	Integer	2	1	1	2	Base value for the number of bytes required to hold TDCS	0x98000000100000100
TD Control Structures	TDCS_SIZE_PER_L2_VM	RO	None	Integer	2	1	1	2	Number of additional TDCS bytes per L2 VM	0x98000000100000101
TD Control Structures	TDVPS_BASE_SIZE	RO	None	Integer	2	1	1	2	Base value for the number of bytes required to hold TDVPS	0x98000000100000200
TD Control Structures	TDVPS_SIZE_PER_L2_VM	RO	None	Integer	2	1	1	2	Number of additional TDVPS bytes per L2 VM	0x98000000100000201
TD Configurability	ATTRIBUTES_FIXED0	RO	None	Bitmap	8	1	1	8	If any certain bit is 0 in ATTRIBUTES_FIXED0, it must be 0 in any TD's ATTRIBUTES. The value of this field reflects the Intel TDX module capabilities and configuration and CPU capabilities.	0x19000000300000000
TD Configurability	ATTRIBUTES_FIXED1	RO	None	Bitmap	8	1	1	8	If any certain bit is 1 in ATTRIBUTES_FIXED1, it must be 1 in any TD's ATTRIBUTES. The value of this field reflects the Intel TDX module capabilities and configuration and CPU capabilities.	0x19000000300000001

Class	Field	VMM Access	Guest Access	Type	Field Size (Bytes)	Num Fields	Num Elem.	Elem. Size (Bytes)	Description	Base FIELD_ID (Hex)
TD Configurability	XFAM_FIXED0	RO	None	Bitmap	8	1	1	8	If any certain bit is 0 in XFAM_FIXED0, it must be 0 in any TD's XFAM.	0x1900000300000002
TD Configurability	XFAM_FIXED1	RO	None	Bitmap	8	1	1	8	If any certain bit is 1 in XFAM_FIXED1, it must be 1 in any TD's XFAM.	0x1900000300000003
TD Configurability	NUM_CPUID_CONFIG	RO	None	Unsigned 16-bit integer	2	1	1	2	Number of the following CPUID_CONFIG entries	0x9900000100000004
TD Configurability	CPUID_CONFIG_LEAVES	RO	None		8	9	1	8	Array of CPUID leaf / sub-leaf numbers: Bits 31:0: Leaf number Bits 63:32 Sub-leaf number. A value of -1 indicates a CPUID leaf with no sub-leaves.	0x9900000300000400
TD Configurability	CPUID_CONFIG_VALUES	RO	None		16	9	2	8	Array of configurable virtualization of the value returned by CPUID: Element 0[31:0]: EAX Element 0[63:32]: EBX Element 1[31:0]: ECX Element 1[63:32]: EDX A bit value of 1 indicates that the host VMM can configure that bit	0x9900000300000500
TD Configurability	IA32_ARCH_CAPABILITIES_CONFIG_MASK	RO	None	64-bit bitmap	8	1	1	8	Bit mask of configurable virtualization of IA32_ARCH_CAPABILITIES MSR. A value of 1 indicates a configurable bit.	0x9A00000300000000
Migration	MIG_ATTRIBUTES	RO	RO	64-bit bitmap	8	1	1	8	Migration attributes (details are TBD)	0xA000000300000000
Migration	MIN_EXPORT_VERSION	RO	RO	16-bit integer	2	1	1	2	Minimum value of migration version supported for export	0x2000000100000001
Migration	MAX_EXPORT_VERSION	RO	RO	16-bit integer	2	1	1	2	Maximum value of migration version supported for export	0x2000000100000002
Migration	MIN_IMPORT_VERSION	RO	RO	16-bit integer	2	1	1	2	Minimum value of migration version supported for import	0x2000000100000003
Migration	MAX_IMPORT_VERSION	RO	RO	16-bit integer	2	1	1	2	Maximum value of migration version supported for import	0x2000000100000004
Migration	MAX_MIGS	RO	None	Unsigned integer	2	1	1	2	Maximum number of migration streams per TD	0xA000000100000010
Migration	NUM_IMMUTABLE_STATE_PAGES	RO	None	Integer	1	1	1	1	Number of pages required for exporting immutable state by TDH.EXPORT.STATE.IMMUTABLE	0xA000000000000020
Migration	NUM_TD_STATE_PAGES	RO	None	Integer	1	1	1	1	Number of pages required for exporting TD state by TDH.EXPORT.STATE.TD	0xA000000000000021
Migration	NUM_VP_STATE_PAGES	RO	None	Integer	1	1	1	1	Number of pages required for exporting VCPU state by TDH.EXPORT.STATE.VP	0xA000000000000022
Service TD	MAX_SERV_TDS	RO	None	Unsigned integer	2	1	1	2	Maximum number of service TDs per TD	0xA100000100000000
Service TD	SERVTD_ATTR_FIXED0	RO	None	64-bit bitmap	8	1	1	8	Fixed-0 bits of Service TD attributes. A bit value of 0 indicates corresponding SERVTD_ATTR bit must be 0	0xA100000300000001
Service TD	SERVTD_ATTR_FIXED1	RO	None	64-bit bitmap	8	1	1	8	Fixed-1 bits of Service TD attributes. A bit value of 1 indicates corresponding SERVTD_ATTR bit must be 1	0xA100000300000002
TD Partitioning	GUEST_L2_GPA_ATTR_MASK	RO	RO	GPA_ATTR	8	1	1	8	Bit mask of page attributes that the L1 VMM can configure for L2 VM GPA mapping. A bit value of 1 indicates the corresponding GPA_ATTR bit may be configured.	0xA200000300000000

### 3.3.3. UPDATED: CMR\_INFO

CMR\_INFO is designed to provide information about a Convertible Memory Range (CMR), as configured by BIOS and checked and stored securely by MCHECK.

- 5 **Note:** CMR\_INFO and TDH.SYS.INFO are provided for backward compatibility. TDH.SYS.RDALL is the recommended method to read Intel TDX module information. See also 3.3.2 above.

Table 3.10: CMR\_INFO Entry Definition

Name	Offset (Bytes)	Type	Size (Bytes)	Description
CMR_BASE	0	Physical Address	8	Base address of the CMR: since a CMR is aligned on 4KB, bits 11:0 are 0.
CMR_SIZE	8	Integer	8	Size of the CMR, in bytes: since a CMR is aligned on 4KB, bits 11:0 are 0. A value of 0 indicates a null entry.

TDH.SYS.INFO leaf function returns an array of CMR\_INFO entries. The CMRs are sorted from the lowest base address to the highest base address, and they are non-overlapping.

#### 5 3.3.4. UPDATED: TDSYSINFO\_STRUCT

TDSYSINFO\_STRUCT is designed to provide enumeration information about the Intel TDX module. It is an output of the TDH.SYS.INFO leaf function.

**Note:** TDSYSINFO\_STRUCT and TDH.SYS.INFO are provided for backward compatibility. TDH.SYS.RDALL is the recommended method to read Intel TDX module information. See also 3.3.2 above.

10 TDSYSINFO\_STRUCT's size is 1024B.

Table 3.11: TDSYSINFO\_STRUCT Definition

Section	Field Name	Offset (Bytes)	Type	Size (Bytes)	Description
Intel TDX Module Info	ATTRIBUTES	0	Bitmap	4	Module attributes <b>Bits 30:0</b> Reserved – set to 0 <b>Bit 31</b> 0 indicates a production module. 1 indicates a debug module.
	VENDOR_ID	4	Integer	4	0x8086 for Intel
	BUILD_DATE	8	BCD	4	Intel TDX module build data – in yyyyymmdd BCD format (each digit occupies 4 bits)
	BUILD_NUM	12	Integer	2	Build number of the Intel TDX module
	MINOR_VERSION	14	Integer	2	Minor version number of the Intel TDX module
	MAJOR_VERSION	16	Integer	2	Major version number of the Intel TDX module
	SYS_RD	18	Boolean	1	A non-0 value indicates that the information in this structure is incomplete. TDH.SYS.RD or TDH.SYS.RDALL should be used to obtain TDX module information.

Section	Field Name	Offset (Bytes)	Type	Size (Bytes)	Description
	<b>RESERVED</b>	19	N/A	13	This field is reserved for enumerating future Intel TDX module capabilities. Set to 0
<b>Memory Info</b>	<b>MAX_TDMRS</b>	32	Integer	2	The maximum number of TDMRs supported
	<b>MAX_RESERVED_PER_TDMR</b>	34	Integer	2	The maximum number of reserved areas per TDMR
	<b>PAMT_ENTRY_SIZE</b>	36	Integer	2	The size of a PAMT entry – determines the number of bytes that need to be reserved for the three PAMT areas: <ul style="list-style-type: none"> <li>PAMT_1G (1 entry per 1GB of TDMR)</li> <li>PAMT_2M (1 entry per 2MB of TDMR)</li> <li>PAMT_4K (1 entry per 4KB of TDMR)</li> </ul>
	<b>RESERVED</b>	38	N/A	10	Set to 0
<b>Control Struct Info</b>	<b>TDCS_BASE_SIZE</b>	48	Integer	2	Base value for the number of bytes required to hold TDCS
	<b>RESERVED</b>	50	N/A	2	Reserved for additional TDCS enumeration Set to 0
	<b>TDVPS_BASE_SIZE</b>	52	Integer	2	Base value for the number of bytes required to hold TDVPS
	<b>RESERVED</b>	54	N/A	10	Set to 0
<b>TD Capabilities</b>	<b>ATTRIBUTES_FIXED0</b>	64	Bitmap	8	If any certain bit is 0 in ATTRIBUTES_FIXED0, it must be 0 in any TD's ATTRIBUTES. The value of this field reflects the Intel TDX module capabilities and configuration and CPU capabilities.
	<b>ATTRIBUTES_FIXED1</b>	72	Bitmap	8	If any certain bit is 1 in ATTRIBUTES_FIXED1, it must be 1 in any TD's ATTRIBUTES. The value of this field reflects the Intel TDX module capabilities and configuration and CPU capabilities.



Section	Field Name	Offset (Bytes)	Type	Size (Bytes)	Description
	<b>XFAM_FIXED0</b>	80	Bitmap	8	If any certain bit is 0 in XFAM_FIXED0, it must be 0 in any TD's XFAM.
	<b>XFAM_FIXED1</b>	88	Bitmap	8	If any certain bit is 1 in XFAM_FIXED1, it must be 1 in any TD's XFAM.
	<b>RESERVED</b>	96	N/A	32	Set to 0
	<b>NUM_CPUID_CONFIG</b>	128	Integer	4	Number of the following CPUID_CONFIG entries
	<b>CPUID_CONFIG[0]</b>	132	CPUID_CONFIG	24	Enumeration of the CPUID leaves/sub-leaves that contain bit fields whose virtualization by the Intel TDX module is either: <ul style="list-style-type: none"> <li>Directly configurable (CONFIG_DIRECT) by the host VMM</li> <li>Bits that the host VMM may allow to be 1 (ALLOW_DIRECT) and their native value, as returned by the CPU, is 1.</li> </ul> See 3.3.1 for details. Note that the virtualization of many CPUID bit fields not enumerated in this list is configurable indirectly via the XFAM and ATTRIBUTES assigned to a TD by the host VMM.
	<b>CPUID_CONFIG[last]</b>		CPUID_CONFIG	24	
<b>Reserved</b>	<b>RESERVED</b>		N/A		Fills up to the structure size (1024B) – set to 0

### 3.3.5. UPDATED: TDMR\_INFO

TDMR\_INFO is designed to provide information about a single Trust Domain Memory Region (TDMR) and its associated PAMT. It is used as an input to TDH.SYS.CONFIG.

5 **Table 3.12: TDMR\_INFO Entry Definition**

Name	Offset (Bytes)	Type	Size (Bytes)	Description
<b>TDMR_BASE</b>	0	Physical Address	8	Base address of the TDMR (HKID bits must be 0): since a TDMR is aligned on 1GB, bits 29:0 are always 0.
<b>TDMR_SIZE</b>	8	Integer	8	Size of the TDMR, in bytes: must be greater than 0 and a whole multiple of 1GB (i.e., bits 29:0 are always 0).
<b>PAMT_1G_BASE</b>	16	Physical Address	8	Base address of the PAMT_1G range associated with the above TDMR (HKID bits must be 0): since a PAMT range is aligned on 4KB, bits 11:0 are always 0.

Name	Offset (Bytes)	Type	Size (Bytes)	Description
PAMT_1G_SIZE	24	Integer	8	Size of the PAMT_1G range associated with the above TDMR: since a PAMT range is aligned on 4KB, bits 11:0 are always 0.
PAMT_2M_BASE	32	Physical Address	8	Base address of the PAMT_2M range associated with the above TDMR (HKID bits must be 0): since a PAMT range is aligned on 4KB, bits 11:0 are always 0.
PAMT_2M_SIZE	40	Integer	8	Size of the PAMT_2M range associated with the above TDMR: since a PAMT range is aligned on 4KB, bits 11:0 are always 0.
PAMT_4K_BASE	48	Physical Address	8	Base address of the PAMT_4K range associated with the above TDMR (HKID bits must be 0): since a PAMT range is aligned on 4KB, bits 11:0 are always 0.
PAMT_4K_SIZE	56	Integer	8	Size of the PAMT_4K range associated with the above TDMR: since a PAMT range is aligned on 4KB, bits 11:0 are always 0.
RESERVED_OFFSET[0]	64	Integer	8	<ul style="list-style-type: none"> <li>Offset of reserved range 0 within the TDMR: since a reserved range is aligned on 4KB, bits 11:0 are always 0.</li> </ul>
RESERVED_SIZE[0]	72	Integer	8	Size of reserved range 0 within the TDMR: <ul style="list-style-type: none"> <li>A size of 0 indicates a null entry. All following reserved range entries must also be null.</li> <li>Since a reserved range is aligned on 4KB, bits 11:0 are always 0.</li> </ul>
RESERVED_OFFSET[N-1]	64 + 16*(N-1)	Integer	8	Offset of the last reserved range within the TDMR.
RESERVED_SIZE[N-1]	72 + 16*(N-1)	Integer	8	Size of the last reserved range within the TDMR.

**Notes:**

- The number of reserved areas within a TDMR is enumerated by TDX Module's MAX\_RESREVED\_PER\_TDMR metadata field, which can be read using TDH.SYS.RD, TDH.SYS.RDALL or TDH.SYS.RDM. For details, see 3.3.43.3.2.
- For backward compatibility, this value is also enumerated by TDSYSINFO\_STRUCT.MAX\_RESREVED\_PER\_TDMR (see 3.3.4).
- Within each TDMR entry, all reserved areas must be sorted from the lowest offset to the highest offset, and they must not overlap with each other.
- All TDMRs and PAMTs must be contained within CMRs.
- A PAMT area must not overlap with another PAMT area (associated with any TDMR), and it must not overlap with non-reserved areas of any TDMR. PAMT areas may reside within reserved areas of TDMRs.

**3.4. TD Parameter Types**

**Note:** This section describes TD parameter types, as defined. Implementation details may differ.

**3.4.1. UPDATED: ATTRIBUTES**

- ATTRIBUTES is defined as a 64b field that specifies various guest TD attributes. ATTRIBUTES is provided by the host VMM as a guest TD initialization parameter as part of TD\_PARAMS. It is reported to the guest TD by TDG.VP.INFO and as part

of TDREPORT\_STRUCT returned by TDG.MR.REPORT. **ATTRIBUTES** is migrated to a destination platform as part of the immutable TD state export by TDH.EXPORT.STATE.IMMUTABLE and import by TDH.IMPORT.STATE.IMMUTABLE.

The ATTRIBUTES bits are divided into three groups, as shown in the table below. If any bit in the TUD group is set to 1, the guest TD is under off-TD debug and is untrusted. The SEC group indicates features that may impact TD security but are not considered as impacting TD trust.

**Table 3.13: ATTRIBUTES Definition**

Bits	Group	Description	Bits	Field	Description
7:0	<b>TUD</b>	TD Under Debug If any of the bits in this group are set to 1, the guest TD is untrusted.	0	<b>DEBUG</b>	Guest TD runs in off-TD debug mode. Its VCPU state and private memory are accessible by the host VMM.
			7:1	<b>RESERVED</b>	Reserved for future TUD flags – must be 0
31:8	<b>SEC</b>	Attributes that may impact TD security	27:8	<b>RESERVED</b>	Reserved for future SEC flags – must be 0
			28	<b>SEPT_VE_DISABLE</b>	Disable EPT violation conversion to #VE on guest TD access of PENDING pages
			29	<b>MIGRATABLE</b>	TD is migratable (using a Migration TD)
			30	<b>PKS</b>	TD is allowed to use Supervisor Protection Keys.
			31	<b>KL</b>	TD is allowed to use Key Locker. Must be 0
63:32	<b>OTHER</b>	Attributes that do not impact TD security	61:32	<b>RESERVED</b>	Reserved for future OTHER flags – must be 0
			62	<b>TPA</b>	The TD is a TDX I/O Provisioning Agent.
			63	<b>PERFMON</b>	TD is allowed to use Perfmon and PERF_METRICS capabilities.

### 3.4.2. XFAM

Intel SDM, Vol. 1, 13 Managing State Using the XSAVE Feature Set  
Intel SDM, Vol. 3, 13 System Programming for Instruction Set Extensions and Processor Extended State

Intel TDX module extended state handling is described in the [TDX Module Base Spec].

XFAM (eXtended Features Available Mask) is defined as a 64b bitmap, which has the same format as XCR0 or IA32\_XSS MSR. XFAM determines the set of extended features available for use by the guest TD. XFAM is provided by the host VMM as a guest TD initialization parameter as part of TD\_PARAMS. It is reported to the guest TD by CPUID(0x0D, 0x01) and as part of TDREPORT\_STRUCT returned by TDG.MR.REPORT.

The Intel TDX module and the Intel® architecture impose some rules on how the bits of XFAM may be set. See the [TDX Module Base Spec] for details.

### 3.4.3. CPUID\_VALUES

CPUID\_VALUES is defined as a 128b structure composed of four 32b fields representing the values returned by CPUID in registers EAX, EBX, ECX and EDX. An array of CPUID\_RET is used during guest TD configuration by TDH.MNG.INIT.

Table 3.14: CPUID\_VALUES Definition

Field	Offset (Bytes)	Size (Bytes)	Description
EAX	0	4	Value returned by CPUID in EAX
EBX	4	4	Value returned by CPUID in EBX
ECX	8	4	Value returned by CPUID in ECX
EDX	12	4	Value returned by CPUID in EDX

#### 3.4.4. UPDATED: TD\_PARAMS

TD\_PARAMS is provided as an input to TDH.MNG.INIT, and some of its fields are included in the TD report. The format of this structure is valid for a specific MAJOR\_VERSION of the Intel TDX module, as reported by TDH.SYS.RD/RDALL or TDH.SYS.INFO.

TD\_PARAMS' size is 1024B.

Table 3.15: UPDATED: TD\_PARAMS Definition

Field	Offset (Bytes)	Type	Size (Bytes)	Description	Included in TDREPORT?
ATTRIBUTES	0	64b bitmap (see 3.4.1)	8	TD attributes: the value set in this field must comply with ATTRIBUTES_FIXED0 and ATTRIBUTES_FIXED1 enumerated by TDH.SYS.RD/RDALL or TDH.SYS.INFO.	Yes
XFAM	8	64b bitmap in XCRO format	8	Extended Features Available Mask: indicates the extended state features allowed for the TD. XFAM's format is the same as XCRO and IA32_XSS MSR. The value set in this field must satisfy the following conditions: <ul style="list-style-type: none"> <li>Natively valid value for XCRO and IA32_XSS (does not contain reserved bits, features not supported by the CPU, or invalid bit combinations)</li> <li>Complies with XFAM_FIXED0 and XFAM_FIXED1 as enumerated by TDH.SYS.RD/RDALL or TDH.SYS.INFO.</li> </ul>	Yes
MAX_VCPUS	16	Unsigned 16b Integer	2	Maximum number of VCPUs	No
NUM_L2_VMS	18	Unsigned 8b Integer	1	Number of L2 VMs May be between 0 and 3. A value of 0 indicates no TD Partitioning is supported.	No
MSR_CONFIG_CTLs	19	8b bitmap	1	MSR configuration controls:	No
				Bit	
				0	
				Other	
				Description	
				Indicates that TD configuration should use the IA32_ARCH_CAPABILITIES_CONFIG field below	
				Reserved, must be 0	

Field	Offset (Bytes)	Type	Size (Bytes)	Description	Included in TDREPORT?
RESERVED	20	N/A	4	Must be 0	No
EPTP_CONTROLS	24	EPTP	8	Control bits of EPTP – copied to each TD VMCS on TDH.VP.INIT: <b>Bits 2:0</b> Memory type – must be 110 (WB) <b>Bits 5:3</b> EPT level – 1 less than the EPT page-walk length. Must be either 3 or 4. <b>Bits 63:6</b> Reserved – must be 0	No
EXEC_CONTROLS	32	64b bitmap (see Table 3.16)	8	Non-measured TD-scope execution controls	No
TSC_FREQUENCY	40	16b unsigned integer	2	TD-scope virtual TSC frequency in units of 25MHz – must be between 4 and 400.	No
RESERVED	42	N/A	38	Must be 0	No
MRCONFIGID	80	SHA384_HASH	48	Software-defined ID for non-owner-defined configuration of the guest TD – e.g., run-time or OS configuration	Yes
MROWNER	128	SHA384_HASH	48	Software-defined ID for the guest TD's owner	Yes
MROWNERCONFIG	176	SHA384_HASH	48	Software-defined ID for owner-defined configuration of the guest TD – e.g., specific to the workload rather than the run-time or OS	Yes
IA32_ARCH_CAPABILITIES_CONFIG	224	64b bitmap	8	Configuration of IA32_ARCH_CAPABILITIES MSR virtualization (if enabled by MSR_CONFIG_CTLs above). Configuration capabilities are enumerated by the IA32_ARCH_CAPABILITIES_CONFIG_MASK, which can be read by TDH.SYS.RD/RDALL.	No
RESERVED	232	N/A	24	Must be 0	No
CPUID_CONFIG[0]	256	CPUID_VALUES	16	Direct configuration of CPUID leaves/sub-leaves virtualization: the number and order of entries must be equal to the number and order of directly configurable or allowable CPUID leaves/sub-leaves reported by TDH.SYS.RD/RDALL or TDH.SYS.INFO. Note that the leaf and sub-leaf numbers are implicit.  Only bits that have been reported as 1 by TDH.SYS.RD/RDALL or TDH.SYS.INFO may be set to 1.	No
CPUID_CONFIG[n-1]		CPUID_VALUES	16		
				Note that the virtualization of many CPUID bit fields not enumerated in this list is configurable indirectly, via the XFAM and ATTRIBUTES fields.	
RESERVED		N/A		Fills up to TD_PARAMS size (1024B) – must be 0	No

Table 3.16: TD\_PARAMS\_STRUCT.EXEC\_CONTROLS Definition

Bits	Name	Description
0	<b>GPAW</b>	TD-scope Guest Physical Address Width execution control: copied to each TD VMCS <b>GPAW</b> execution control on TDH.VP.INIT 0: GPA.SHARED bit is GPA[47] 1: GPA.SHARED bit is GPA[51]
63:1	<b>RESERVED</b>	Must be 0

### 3.5. Physical Memory Management Types

**Note:** This section describes physical memory types, as defined. Implementation may differ.

- 5 PAMT entry and PT (page type) are defined in the [TDX Module Base Spec].

#### 3.5.1. Physical Page Size

Three physical page size levels (4KB, 2MB and 1GB) are defined.

Table 3.17: Page Size Definition

Page Size	Associated Physical Page Size	Value
<b>PS_1G</b>	1GB	2
<b>PS_2M</b>	2MB	1
<b>PS_4K</b>	4KB	0

### 10 3.6. **UPDATED:** TD Private Memory Management Data Types: Secure EPT

Intel SDM, Vol. 3, 28.2.2 EPT Translation Mechanism

**Note:** This section describes private memory management types, as defined. Implementation may differ.

#### 3.6.1. Secure EPT Levels

- 15 Secure EPT level definition is identical to legacy VMX EPT level definition. As a rule, an EPT entry at level L maps a GPA range whose size is  $2^{12+9*L}$ .

Table 3.18: EPT Levels Definition

Level	0	1	2	3	4	5 (5-Level EPT Only)
<b>GPA Range Size</b>	4KB	2MB	1GB	512GB	256TB	16PB <sup>1</sup>
<b>Child Physical Page Size</b>	4KB	2MB	1GB	N/A	N/A	N/A
<b>EPT Page Type</b>	N/A	EPT	EPD	EPDPT	EPML4	EPML5
<b>Parent EPT Entry Type</b>	EPTE	EPDE	EPDPTE	EPML4E	EPML5E (5-level EPT) or VMCS.EPTP (4-level EPT)	VMCS.EPTP
<b>GPA Offset Bits</b>	20:12	29:21	38:30	47:39	51:48 (5-level EPT only)	N/A

<sup>1</sup> Only the lower half is available as TD private GPA space, because the SHARED bit must be 0

### 3.6.2. Secure EPT Entry Information as Returned by TDX Module Functions

Many Intel TDX module functions return Secure EPT entry information. This information is returned in the formats detailed below, which may be different than the actual Secure EPT format as maintained by the TDX module in memory.

**Note:** The returned Secure EPT information is subject to change with new versions of TDX.

#### 3.6.2.1. Returned Secure EPT Entry Content

The returned secure EPT entry format is detailed below. It may be different than the actual Secure EPT format as maintained by the TDX module in memory.

**Table 3.19: Secure EPT Entry Content as Returned by TDX Interface Functions**

Secure EPT Entry Field						Value Returned in RCX (per Entry State Returned in RDX)		
MSB	LSB	Size	Short Name	Full Name	Enabled	Non-FREE Leaf	Non-FREE Non-Leaf	FREE
0	0	1	R	Read	N/A	R	R	0
1	1	1	W	Write	N/A	W	W	0
2	2	1	Xs	Execute	N/A	X	X	0
5	3	3	MT	Memory Type	N/A	MT	0	0
6	6	1	IPAT	Ignore PAT	N/A	IPAT	0	0
7	7	1	PS	Leaf	N/A	1	0	0
8	8	1	A	Accessed	No	0	0	0
9	9	1	D	Dirty	No	0	0	0
10	10	1	Xu	Execute (User)	No	0	0	0
11	11	1	Ignored	Ignored	N/A	0	0	0
51	12	40	HPA[51:12]	Host Physical Address [51:12]	N/A	HPA[51:12]	HPA[51:12]	0
57	57	1	VGP	Verify Guest Paging	No	0	0	0
58	58	1	PWA	Paging-Write Access	No	0	0	0
59	59	1	Ignored	Ignored	N/A	0	0	0
60	60	1	SSS	Supervisor Shadow Stack	No	0	0	0
61	61	1	SPP	Check Sub-Page Permissions	No	0	0	0
62	62	1	Ignored	Ignored	N/A	0	0	0
63	63	1	SVE	Suppress #VE	Yes	SVE	0	1

The R, W and X access permission bits' values depend on the SEPT entry state:

- For leaf entries in the MAPPED and EPORTED\_DIRTY states, and non-leaf entries in the NL\_MAPPED state, RWX = 111.
- For leaf entries in the BLOCKED, PENDING\* and REMOVED states, non-leaf entries in the NL\_BLOCKED state and FREE entries, RWX = 000.
- For leaf entries in the \*BLOCKED\* states, RWX = 101.

#### 3.6.2.2. Additional Returned Secure EPT Information

Additional information for secure EPT entries is returned as defined below.



**Table 3.20: Additional Secure EPT Entry Information Returned by TDX Interface Functions**

Bits	Name	Description
2:0	Level	Level of the returned Secure EPT entry – see 3.6.1 above
7:3	Reserved	Set to 0
15:8	State	The TDX state of the Secure EPT entry – see Table 3.21 below
63:16	Reserved	Set to 0

**Table 3.21: Secure EPT Entry TDX State Returned by TDX Interface Functions**

State Name	Public State Number	
FREE	0	Secure EPT entry does not map a GPA range.
REMOVED	5	Secure EPT entry is of a removed page
NL_MAPPED	132	Secure EPT entry maps a private GPA range which is accessible by the guest TD.
NL_BLOCKED	129	Secure EPT entry maps a private GPA range, but new address translations to that range are blocked.
MAPPED	4	Secure EPT entry maps a private GPA page which is accessible by the guest TD.
BLOCKED	1	Secure EPT entry maps a private GPA page but new address translations to that range are blocked.
BLOCKEDW	8	Secure EPT entry maps a private GPA page, but new address translations for write operations to that range are blocked.
EXPORTED_BLOCKEDW	9	Secure EPT entry maps a private page that has been blocked for writing and exported.
EXPORTED_DIRTY	11	Secure EPT entry maps a private page that was exported, but is not blocked for writing and its content and/or attributes may have since been modified.
EXPORTED_DIRTY_BLOCKEDW	12	Secure EPT entry maps a private page that was previously exported, its content and/or attributes may have since been modified and then it was blocked for writing.
PENDING	2	Secure EPT entry maps a 4KB or a 2MB page that has been dynamically added to the guest TD using TDH.MEM.PAGE.AUG and is pending acceptance by the guest TD using TDG.MEM.PAGE.ACCEPT. This page is not yet accessible by the guest TD.
PENDING_BLOCKED	3	Secure EPT entry is both pending and blocked.
PENDING_BLOCKEDW	16	Secure EPT entry is both pending and blocked for writing.
PENDING_EXPORTED_BLOCKEDW	17	Secure EPT entry is both pending and exported.
PENDING_EXPORTED_DIRTY	19	Secure EPT entry is both pending and exported, and is not blocked for writing.
PENDING_EXPORTED_DIRTY_BLOCKEDW	20	Secure EPT entry is both pending and exported, and is blocked for writing.

### 3.6.3. NEW: GPA\_ATTR: GPA Attributes

GPA\_ATTR specifies the settable attributes of a page. It is used as an input of TDG.MEM.PAGE.ATTR.WR, as an output of TDG.PAGE.ATTR.WR, and for migration (TDH.EXPORT.MEM and TDH.IMPORT.MEM) and, if the TD is in debug mode, for returning of L2 attributes by TDH.MEM.SEPT.RD.

- 5 GPA\_ATTR is an array of four GPA\_ATTR\_SINGLE\_VM 16-bit entries:

**Table 3.22: GPA\_ATTR: GPA Attributes (all VMs) Definition**

Bits	VM Index	Bits	Description
15:0	0	15:0	GPA attributes for L1 (all-0 for migration)
31:16	1	31:24	GPA attributes for L2 VM #1
47:32	2	47:32	GPA attributes for L2 VM #2
63:48	3	63:48	GPA attributes for L2 VM #3

**Table 3.23: GPA\_ATTR\_SINGLE\_VM: GPA Attributes (Single VM) Definition**

Bit(s)	Size	Attribute Type	Name	Description	TDG.MEM.PAGE.ATTR.RD/WR Access		Used for Migration	
					L1	L2	L1	L2
0	1	Intel64	R	Read	None	RW	No	Yes
1	1	Intel64	W	Write	None	RW	No	Yes
2	1	Intel64	Xs	Execute (Supervisor)	None	RW	No	Yes
3	1	Intel64	Xu	Execute (User)	None	RW	No	Yes
4	1	Intel64	VGP	Verify Guest Paging	None	RW	No	Yes
5	1	Intel64	PWA	Paging-Write Access	None	RW	No	Yes
6	1	Intel64	SSS	Supervisor Shadow Stack	None	RW	No	Yes
7	1	Intel64	SVE	Suppress #VE	None	RW	No	Yes
14:8	6	N/A	RESERVED		None	None	No	No
15	1	TDX	VALID	Indicates that the other bits are valid. If its value is 0, other fields are reserved and must be 0.	RW	RW	No	Yes

#### 10 3.6.3.1. GPA Attributes Rules

The TDX module enforces the following rules to help ensure that GPA attributes will not cause an EPT Misconfiguration (see [Intel SDM, Vol. 3, 28.3.3.1]):

- If VALID is 0, all other bits must be 0
- Reserved bits must be 0.
- If bit W is 1, bit R must be 1
- If bit PWA is 1, bit R must be 1 (regardless of the VMCS “EPT paging-write control” VM-execution control.

The TDX module checks, on TDH.SYS.INIT, that the CPU supports setting Xs or Xu when R is 0.

### 3.6.4. Page GLA List

- 20 A page GLA list specifies a list of page guest linear addresses (GLAs) to be processed by TDG.VP.INVVPID. A page GPA list may have up to 512 entries, is contained in a single 4KB page and must be aligned on 4KB.

### 3.6.4.1. *PAGE\_GLA\_LIST\_ENTRY*

PAGE\_GLA\_LIST\_ENTRY species a range of consecutive guest linear addresses of 4KB pages.

**Table 3.24: PAGE\_GLA\_LIST\_ENTRY Definition**

Bits	Name	Description
11:0	<b>LAST_PAGE</b>	Index of the last 4KB page to be processed
63:12	<b>BASE_GLA</b>	Bits 63:12 of the guest linear address of the first 4KB page to be processed

### 5 3.6.4.2. *PAGE\_GLA\_LIST\_INFO: Page GLA List GPA and Additional Information*

PAGE\_GPA\_LIST\_INFO is a 64b structure used as a GPR input and output operand of TDG.VP.INVVPID. It provides the GPA of the page GLA list page in private memory, the index of the first entry and the number of entries to be processed.

**Table 3.25: PAGE\_GLA\_LIST\_INFO**

Bits	Name	Description
8:0	<b>FIRST_ENTRY</b>	Index of the first entry of the list to be processed
11:9	<b>RESERVED</b>	Reserved: must be 0
51:12	<b>LIST_GPA</b>	Bits 51:12 of the guest physical address of the GLA list page, which must be a private GPA
61:52	<b>NUM_ENTRIES</b>	Number of entries in the GLA list to be processed, must be between 0 through 512
63:62	<b>RESERVED</b>	Reserved: must be 0

## 10 3.7. *TD Entry and Exit Types*

### 3.7.1. *Extended Exit Qualification*

Extended Exit Qualification is a 64-bit field returned by TDH.VP.ENTER for asynchronous TD exits with an architectural VMX exit reasons. It contains additional non-VMX, TDX-specific information.

Table 3.26: Extended Exit Qualification

Bits	Name	Description		
3:0	TYPE	Extended exit qualification type		
		Value	Name	Description
		0	NONE	No extended exit qualification
		1	ACCEPT	Exit qualification for an EPT violation during TDG.MEM.PAGE.ACCEPT
		2	GPA_DETAILS	Exit qualification for an EPT violation caused by guest-side interface function failure of GPA→HPA translation
		3	TD_ENTRY_MSR_LOAD_FAILURE	Exit qualification for failures of TD entry due to loading guest MSR state
		4	TD_ENTRY_XSTATE_LOAD_FAILURE	Exit qualification for failures of TD entry due to loading guest extended state
		5	ATTR_WR	Exit qualification for an EPT violation during TDG.MEM.PAGE.ATTR.WR
		Other	Reserved	
31:4	Reserved	Set to 0		
63:32	INFO	TYPE-specific information		
		TYPE	Value	
		NONE	0	
		ACCEPT	See the table below	
		GPA_DETAILS	See the table below	
		TD_ENTRY_MSR_LOAD_FAILURE	MSR index	
		TD_ENTRY_XSTATE_LOAD_FAILURE	0	
		ATTR_WR	See the table below	
		Reserved	0	

**Table 3.27: UPDATED: Extended Exit Qualification INFO Field when TYPE is ACCEPT or ATTR\_WR**

Bits	Name	Description
34:32	REQ_SEPT_LEVEL	SEPT level requested as an input to TDG.MEM.PAGE.ACCEPT or TDG.MEM.PAGE.ATTR.WR
37:35	ERR_SEPT_LEVEL	SEPT level in which TDG.MEM.PAGE.ACCEPT or TDG.MEM.PAGE.ATTR.WR detected an error
45:38	ERR_SEPT_STATE	The TDX state of the Secure EPT entry where TDG.MEM.PAGE.ACCEPT or TDG.MEM.PAGE.ATTR.WR detected an error – see Table 3.21 above
46	ERR_SEPT_IS_LEAF	Indicates that the SEPT entry where TDG.MEM.PAGE.ACCEPT or TDG.MEM.PAGE.ATTR.WR detected an error is a leaf entry
63:47	Reserved	Set to 0

**Table 3.28: NEW: Extended Exit Qualification INFO Field when TYPE is GPA\_DETAILS**

Bits	Name	Description
34:32	Reserved	Set to 0
37:35	ERR_SEPT_LEVEL	Level where the Secure EPT walk error occurred
51:38	Reserved	Set to 0
53:52	VM_INDEX	Virtual machine index for which Secure EPT walk error occurred
63:54	Reserved	Set to 0

## 5 3.8. L2 VM Transition Types

### 3.8.1. NEW: L2\_ENTER\_GUEST\_STATE

L2\_ENTER\_GUEST\_STATE is used as input and output of TDG.VP.ENTER. It is an array of general-purpose (GPR) register values, organized according to their architectural number, with additional values of RFLAG, RIP and SSP.

**Table 3.29: L2\_ENTER\_GUEST\_STATE Definition**

Offset (Bytes)	Size (Bytes)	Name	Description
0	8	RAX	
8	8	RCX	
16	8	RDX	
24	8	RBX	
32	8	RSP	
40	8	RBP	
48	8	RSI	
56	8	RDI	
64	8	R8	
72	8	R9	
80	8	R10	

Offset (Bytes)	Size (Bytes)	Name	Description
88	8	R11	
96	8	R12	
104	8	R13	
112	8	R14	
120	8	R15	
128	8	RFLAGS	
136	8	RIP	
144	8	SSP	
152	2	GUEST_INTERRUPT_STATUS	Bits 7:0: RVI Bits 15:7: SVI

### 3.9. Measurement and Attestation Types

**Note:** This section describes measurement and attestation types, as defined. Implementation may differ.

#### 3.9.1. CPUSVN

5 CPUSVN is a 16B Security Version Number of the CPU.

- There is a single CPUSVN used for SGX and TDX.
- CPUSVN contents are considered micro-architectural. CPUSVN is composed of fields for PR\_RESET\_SVN, R\_LAST\_PATCH\_SVN, SINIT, BIOS ACM, Boot Guard ACM and BIOS Guard NP-PPPE module.

#### 3.9.2. TDREPORT\_STRUCT

10 TDREPORT\_STRUCT is the output of the TDG.MR.REPORT function. TDREPORT\_STRUCT is composed of a generic MAC structure (REPORTMACSTRUCT, see 3.9.3 below), a SEAMINFO structure and a TDX-specific TEE info structure (TDINFO\_STRUCT, see 3.9.5 below).

The size of TDREPORT\_STRUCT is 1024B.

**Table 3.30: TDREPORT\_STRUCT Definition**

Name	Offset (Bytes)	Type	Size (Bytes)	Description
REPORTMACSTRUCT	0	REPORTMACSTRUCT	256	REPORTMACSTRUCT for the TDG.MR.REPORT
TEE_TCB_INFO	256	TEE_TCB_INFO_STRUCT	239	Additional attestable elements in the TD's TCB are not reflected in the REPORTMACSTRUCT.CPUSVN – includes the Intel TDX module measurements.
RESERVED	495	N/A	17	Reserved – contains 0
TDINFO	512	TDINFO_STRUCT	512	TD's attestable properties

#### 3.9.3. REPORTMACSTRUCT (Reference)

REPORTMACSTRUCT is common to Intel's Trusted Execution Environments (TEEs) – e.g., SGX and TDX. REPORTMACSTRUCT is the first field in the TEE report structure. In the TDX architecture, that is TDREPORT\_STRUCT. REPORTMACSTRUCT is MAC-protected and contains hashes of the remainder of the report structure which includes the

TEE's measurements, and where applicable, the measurements of additional TCB elements not reflected in REPORTMACSTRUCT.CPUSVN – e.g., a SEAM's measurements.

Software verifying a TEE report structure (for TDX, this includes TEE\_TCB\_INFO\_STRUCT and TDINFO\_STRUCT) should first confirm that its REPORTMACSTRUCT.TEE\_TCB\_INFO\_HASH equals the hash of the TEE\_TCB\_INFO\_STRUCT (if applicable) and that REPORTMACSTRUCT.TEE\_INFO\_HASH equals the hash of the TDINFO\_STRUCT. Then, software uses ENCLU(EVERIFYREPORT) to help check the integrity of the REPORTMACSTRUCT. If all checks pass, the measurements in the structure describe a TEE on this platform.

The size of REPORTMACSTRUCT is 256B.

**Table 3.31: REPORTMACSTRUCT Definition**

Name	Offset (Bytes)	Type	Size (Bytes)	Description	MAC
<b>REPORTTYPE</b>	0	REPORTTYPE	4	Type Header Structure	Yes
<b>RESERVED</b>	4		12	Must be zero	Yes
<b>CPUSVN</b>	16	CPUSVN	16	CPU SVN	Yes
<b>TEE_TCB_INFO_HASH</b>	32	SHA384_HASH	48	SHA384 of TEE_TCB_INFO for TEEs implemented using Intel TDX	Yes
<b>TEE_INFO_HASH</b>	80	SHA384_HASH	48	SHA384 of TEE_INFO: a TEE-specific info structure (TDINFO_STRUCT or SGXINFO) or 0 if no TEE is represented	Yes
<b>REPORTDATA</b>	128		64	A set of data used for communication between the caller and the target.	Yes
<b>RESERVED</b>	192		32	Must be zero	Yes
<b>MAC</b>	224		32	The MAC over the REPORTMACSTRUCT with model-specific MAC	No

#### 3.9.4. REPORTTYPE

REPORTTYPE indicates the reported Trusted Execution Environment (TEE) type, sub-type and version.

The size of REPORTTYPE is 4B.



Table 3.32: TDX-Specific REPORTTYPE Definition

Name	Offset (Bytes)	Size (Bytes)	Description	Value
TYPE	0	1	Trusted Execution Environment (TEE) Type:	0x00: SGX 0x7F-0x01: Reserved (TEE implemented by CPU) 0x80: Reserved (TEE implemented by SEAM module) <b>0x81: TDX</b> 0xFF-0x82: Reserved (TEE implemented by SEAM module)
SUBTYPE	1	1	TYPE-specific subtype	0
VERSION	2	1	TYPE-specific version.	For TDX, VERSION may have the following values: 0: There are no bound nor pre-bound service TDs. TDINFO_STRUCT.SERVTD_HASH is not used (its value is 0). 1: There is one or more bound or pre-bound service TDs. TDINFO_STRUCT.SERVTD_HASH is used.
RESERVED	3	1	Must be zero	0

### 3.9.5. UPDATED: TDINFO\_STRUCT

TDINFO\_STRUCT is defined as the TDX-specific TEE\_INFO part of TDG.MR.REPORT. It contains the measurements and initial configuration of the TD that was locked at initialization and a set of measurement registers that are run-time extendable. These values are copied from the TDCS by the TDG.MR.REPORT function. Refer to the [TDX Module Base Spec] for additional details.

The size of TDINFO\_STRUCT is 512B.

Table 3.33: TDINFO\_STRUCT Definition

Name	Offset (Bytes)	Type	Size (Bytes)	Description
ATTRIBUTES	0		8	TD's ATTRIBUTES
XFAM	8		8	TD's XFAM
MRTD	16	SHA384_HASH	48	Measurement of the initial contents of the TD
MRCONFIGID	64	SHA384_HASH	48	Software-defined ID for non-owner-defined configuration of the guest TD – e.g., run-time or OS configuration
MROWNER	112	SHA384_HASH	48	Software-defined ID for the guest TD's owner
MROWNERCONFIG	160	SHA384_HASH	48	Software-defined ID for owner-defined configuration of the guest TD – e.g., specific to the workload rather than the run-time or OS
RTMR	208	SHA384_HASH	NUM_RTMR * 48	Array of NUM_RTMR (4) run-time extendable measurement registers

Name	Offset (Bytes)	Type	Size (Bytes)	Description
<b>SERVTD_HASH</b>	400	SHA384_HASH	48	If is one or more bound or pre-bound service TDs, SERVTD_HASH is the SHA384 hash of the TDINFO_STRUCTs of those service TDs bound. Else, SERVTD_HASH is 0.
<b>RESERVED</b>	448	N/A	64	Must be zero

### 3.10. **UPDATED:** Metadata Access Types

**Note:** This section describes control structure field access types, as defined. Implementation may differ.

Metadata access is described in the [TDX Module Base Spec].

#### 5 3.10.1. MD\_FIELD\_ID: Metadata Field Identifier / Sequence Header

MD\_FIELD\_ID is used for two purposes:

**Metadata Field Identifier:** Used for specifying a single element of a metadata field

**Metadata Sequence Header:** Used as the header of a metadata field sequence

10 Lists of metadata field identifiers for global-scope metadata, TD-scope metadata and VCPU-scope metadata are provided in Ch. 4. The metadata tables provide a **base identifier**. The table below specifies which components of MD\_FIELD\_ID are taken from the base identifier; other components need to be specified as required.

**Table 3.34: MD\_FIELD\_ID (Metadata Field Identifier / Sequence Header) Definition**

Bits	Size	Name	From Metadata Tables' Base Field ID	Single-Element or Sequence Header	Description
23:0	24	<b>FIELD_CODE</b>	Yes	Both	For a single-element identifier, identifies the element that is being accessed. For a metadata sequence header, identifies the first field that is being accessed in a sequence.
31:24	8	<b>RESERVED</b>	No	Both	Must be 0
33:32	2	<b>ELEMENT_SIZE_CODE</b>	Yes	Both	Size of a single element of a metadata field: <b>0:</b> 8 bits <b>1:</b> 16 bits <b>2:</b> 32 bits <b>3:</b> 64 bits For backward compatibility, TDH.MNG.RD, TDH.MNG.WR, TDH.VP.RD and TDH.VP.WR version 0 ignore this field and use a default value based on the field code.
37:34	4	<b>LAST_ELEMENT_IN_FIELD</b>	No	Sequence Header	Number of elements in a metadata field, minus 1 For a single-element identifier, the value is 0. This field is ignored when used as input to TD*.SYS.RDALL.
46:38	9	<b>LAST_FIELD_IN_SEQUENCE</b>	No	Sequence Header	Number of fields in a sequence, minus 1 For a single-element identifier, the value is 0. This field is ignored when used as input to TD*.SYS.RDALL.
49:47	3	<b>RESERVED</b>	Yes	Both	Must be 0

Bits	Size	Name	From Metadata Tables' Base Field ID	Single-Element or Sequence Header	Description
50	1	<b>INC_SIZE</b>	Yes	Sequence Header	For a single-element identifier, INC_SIZE is ignored. For a sequence header, INC_SIZE specifies how FIELD_CODE is incremented when accessing consecutive elements in a sequence: <b>0:</b> Increment FIELD_CODE by 1 for each element. <b>1:</b> Increment FIELD_CODE by 2 for each element. INC_SIZE is designed to support VMCS field encoding, where bit 0 (access type) is always 0 for full access.
51	1	<b>WRITE_MASK_VALID</b>	No	Both	Indicates that a write mask is provided together with the write value. For backward compatibility, single-element metadata write interface functions (e.g., TDH.MNG.WR, TDH.VP.WR etc.) and use an implicit value of 1. This field is ignored by metadata read interface functions.
54:52	3	<b>CONTEXT_CODE</b>	Yes	Both	Specifies the context of the field: <b>0:</b> Platform (whole Intel TDX module) <b>1:</b> TD <b>2:</b> TD VCPU <b>Other:</b> Reserved All metadata read and write interface functions (e.g., TDH.MNG.RD, TDH.MNG.WR, TDH.VP.RD, TDG.SYS.RDALL etc.) ignore this field when used as an input; they use an implicit value.
55	1	<b>RESERVED</b>	Yes	Both	Must be 0
61:56	6	<b>CLASS_CODE</b>	Yes	Both	Identifies the class of the fields being accessed Class codes are defined in 3.10.3.
62	1	<b>RESERVED</b>	Yes	Both	Must be 0
63	1	<b>NON_ARCH</b>	Yes	Both	Specifies forward compatibility, i.e., whether this field identifier will maintain their definition in a compatible way throughout Intel TDX module updates. <b>0:</b> Field identifier will maintain forward compatibility. <b>1:</b> Field identifier may not maintain forward compatibility.

### 3.10.2. Meaning of Field Codes

For some field classes, field codes have an architectural meaning, as shown below. For other classes, field codes are arbitrarily assigned.

**Table 3.35: Meaning of Field Codes**

Field Class	Field Code Meaning			Reference
<b>VMCS</b>	Field code is the architectural VMCS field code. The “HIGH” access type (for accessing the upper 32b of 64b fields) is not supported.			[Intel SDM, Vol. 3, 24.11.2 and App. B]
	<b>Bits</b>	<b>Name</b>	<b>Description</b>	
	23:16	RESERVED	Must be 0	

Field Class	Field Code Meaning			Reference
	15:0	VMCS_FIELD_CODE	Bits 15:0 of the architectural VMCS field code <b>Note:</b> Bits 32:16 of the VMCS field code are implicitly 0.	
MSR Bitmap	Offset (in 8B units) from the beginning of the architectural MSR bitmaps page			[Intel SDM, Vol. 3, 24.6.9]
Secure EPT Root	Offset (in 8B units) from the beginning of the page			
Virtual APIC Page	Offset (in 8B units) from the beginning of the architectural virtual APIC page structure			[Intel SDM, Vol. 3, 29.1]
CPUID Config	Each field contains two 64-bit element, with the values returned by CPUID for the leaf and sub-leaf, as follows: Element 0[31:0]: EAX      Element 0[63:32]: EBX Element 1[31:0]: ECX      Element 1[63:32]: EDX The field code is packed as shown below:			
	Bits	Name	Description	
	23:17	RESERVED	Must be 0	
	16	LEAF_31	Leaf number bit 31	
	15:9	LEAF_6_0	Leaf number bit 6:0 <b>Note:</b> Leaf bits 30:7 are implicitly 0.	
	8	SUBLEAF_NA	Sub-leaf not applicable flag	
	7:1	SUBLEAF_6_0	Sub-leaf number bits 6:0 If SUBLEAF_NA is 1, then SUBLEAF_6_0 is all-1. <b>Note:</b> Sub-leaf bits 31:7 are implicitly 0.	
	0	ELEMENT_I	Element index within field	
GPR State	Architectural GPR number			
MSR State	Architectural MSR index, packed as shown below:			
	Bits	Description		
	23:14	Reserved, must be 0		
	13	Bit 31 (equal to bit 30) of the architectural MSR index		
	12:0	Bits 12:0 of the architectural MSR index		
Extended State	Offset (in 8B units) from the beginning of the page extended state buffer			
Other	Arbitrary field identifiers			

### 3.10.3. Class Codes

#### 3.10.3.1. **NEW:** TDX Module Global Scope Field Class Codes

TDX Module global scope field classes are defined as follows:

**Table 3.36: TDX Module Global Scope Field Class Codes Definition**

Class Code	Field Class Name
0	Platform Info
8	TDX Module Version
9	TDX Module Handoff
10	TDX Module Info
16	CMR Info
17	TDMR Info
24	TD Control Structures
25	TD Configurability
32	Migration
33	Service TD
34	TD Partitioning

#### 3.10.3.2. **UPDATED:** TD-Scope (TDR and TDCS) Field Class Codes

TD-scope field classes are defined as follows:

**Table 3.37: TD Scope (TDR and TDCS) Field Class Codes Definition**

Class Code	Control Structure	Field Class Name
0	TDR	TD Management
1	TDR	Key Management
2	TDR	TD Preserving
3	TDR	TDX I/O
16	TDCS	TD Management
17	TDCS	Execution Controls
18	TDCS	TLB Epoch Tracking
19	TDCS	Measurement
20	TDCS	CPUID
21	TDCS	Zero Page
22	TDCS	Virt. MSR Values
24	TDCS	Migration
25	TDCS	Service TD
26	TDCS	MIGSC Links
27	TDCS	TDX I/O
32	TDCS	MSR Bitmaps

Class Code	Control Structure	Field Class Name
33	TDCS	Secure EPT Root
37	TDCS	L2 Secure EPT Root [1]
41	TDCS	L2 Secure EPT Root [2]
45	TDCS	L2 Secure EPT Root [3]

### 3.10.3.3. VCPU-Scope (TDVPS) Field Class Codes

TDVPS field classes are defined as follows:

**Table 3.38: TD VCPU Scope (TDVPS) Field Class Codes Definition**

Class Code	Field Class Name
0	TD VMCS
1	VAPIC
2	VE_INFO
16	Guest GPR State
17	Guest State
18	Guest Ext. State
19	Guest MSR State
32	Management
33	CPUID Control
34	EPT Violation Log
36	VMCS[1]
37	MSR Bitmaps[1]
38	MSR Bitmaps Shadow[1]
44	VMCS[2]
45	MSR Bitmaps[2]
46	MSR Bitmaps Shadow[2]
52	VMCS[3]

5

### 3.10.4. Order of Field Identifiers

For usages such as TD migration, there is a need to define strict ordering between field identifiers. For this purpose, we consider field identifiers to be orders by the following fields:

1. CONTEXT\_CODE
2. CLASS\_CODE
3. FIELD\_CODE

10

### 3.10.5. MD\_LIST\_HEADER: Metadata List Header

MD\_LIST\_HEADER is defined below. The size of MD\_LIST\_HEADER is 64 bits.

**Table 3.39: MD\_LIST\_HEADER Definition**

Bits	Name	Description
15:0	LIST_BUFF_SIZE	The size of memory buffer containing the list The buffer may be larger than the actual space occupied by the list; in this case the excess buffer space is ignored or read and may be overwritten on write.
31:16	NUM_SEQUENCES	The number of metadata field sequences in the list.
63:32	RESERVED	Reserved, set to 0

**3.10.6. Private Page List**

A private page list specifies a list of HPAs of 4KB pages that are, or will become, TD private pages. The list may have up to 512 64-bit entries, each containing a 4KB-aligned HPA (HKID bits must be 0) of a page. The list is contained in a single 4KB page and must be aligned on 4KB. The page list may contain null entries, indicated by the INVALID bit.

**Table 3.40: Private Page List Entry**

Bits	Name	Description
11:0	RESERVED	Reserved: must be 0 (unless bit 63 indicates an invalid entry)
51:12	HPA	Bits 51:12 of the host physical address (HKID bits must be 0) of the migration buffer page
62:52	RESERVED	Reserved: must be 0 (unless bit 63 indicates an invalid entry)
63	INVALID	A value of 1 indicates that this entry is invalid

**3.10.7. HPA\_AND\_SIZE: HPA and Size of a Buffer**

HPA\_AND\_SIZE is a 64-bit structure used to provide a buffer host physical address and size details.

**Table 3.41: HPA\_AND\_SIZE**

Bits	Name	Description
51:0	HPA	Bits 51:0 of the host physical address (including HKID bits) of the buffer
63:52	SIZE	Size of the buffer, in bytes

**3.10.8. HPA\_AND\_LAST: HPA and Last Byte Index of a Page-Aligned Buffer**

HPA\_AND\_LAST is a 64-bit structure used to provide a 4KB aligned buffer host physical address and size details.

**Table 3.42: HPA\_AND\_LAST**

Bits	Name	Description
11:0	LAST	Index of the last byte in the buffer
51:12	HPA	Bits 51:12 of the host physical address (including HKID bits) of the 4KB-aligned buffer
63:52	RESERVED	Reserved: must be 0



### 3.11. NEW: Service TD Types

#### 3.11.1. SERVTD\_BINDING\_TABLE: Service TD Binding Table

SERVTD\_BINDING\_TABLE is a table of service TD binding information, held in the TDCS. For details, see the [TDX Module Base Spec].

**Table 3.43: Service TD Binding Entry Definition**

Field	Type	Offset (Bytes)	Size (Bytes)	Description
STATE	SERVTD_BINDING_STATE	0	1	See below and [TDX Module Base Spec]
Reserved		1	1	Must be 0
TYPE	SERVTD_TYPE	2	2	See below and [TDX Module Base Spec]
Reserved		4	4	Must be 0
ATTR	SERVTD_TYPE	8	8	See below and [TDX Module Base Spec]
UUID	256-bit blob	16	32	See [TDX Module Base Spec]
INFO_HASH	SHA384_HASH	48	48	See [TDX Module Base Spec]
Reserved		96	32	Must be 0

#### 3.11.2. SERVTD\_BINDING\_STATE: Service TD Binding State

SERVTD\_BINDING\_STATE indicates the state of the service TD binding slot. For details, see the [TDX Module Base Spec].

**Table 3.44: SERVTD\_BINDING\_STATE Values**

Value	Name
0	NOT_BOUND
1	PRE_BOUND
2	BOUND

#### 3.11.3. SERVTD\_TYPE: Service TD Binding Type

SERVTD\_TYPE is a 16-bit field which specifies the binding type of a service TD. For details, see the [TDX Module Base Spec].

**Table 3.45: SERVTD\_TYPE Definition**

Value	Meaning	Multiple Bindings	Metadata Access
0	Migration TD	No	Migration session key
Other	Reserved	N/A	N/A

#### 3.11.4. SERVTD\_ATTR: Service TD Binding Attributes

SERVTD\_ATTR is a 64-bit field which specifies binding attributes of a service TD. For details, see the [TDX Module Base Spec].

Table 3.46: SERVTD\_ATTR Definition

Bit(s)	Name	Description
0	INSTANCE_BINDING	0: <b>Class Binding:</b> Rebinding can be done with any TD with the same SERVTD_INFO_HASH, SERVTD_TYPE and SERVTD_ATTR as the original binding. Those parameters are migrated. 1: <b>Instance Binding:</b> Rebinding can be done with the same TD instance (same SERVTD_UUID), using the same SERVTD_TYPE and SERVTD_ATTR as the original binding.
1	RESERVED	Must be 0
2	RESERVED	Must be 0
3	RESERVED	Must be 0
31:4	RESERVED	Must be 0
32	IGNORE_ATTRIBUTES	If set to 1, a value of 0 is used instead of the service TD's ATTRIBUTES field when calculating SERVTD_INFO_HASH
33	IGNORE_XFAM	If set to 1, a value of 0 is used instead of the service TD's XFAM field when calculating SERVTD_INFO_HASH
34	IGNORE_MRTRD	If set to 1, a value of 0 is used instead of the service TD's MRTRD field when calculating SERVTD_INFO_HASH
35	IGNORE_MRCONFIGID	If set to 1, a value of 0 is used instead of the service TD's MRCONFIGID field when calculating SERVTD_INFO_HASH
36	IGNORE_MROWNER	If set to 1, a value of 0 is used instead of the service TD's MROWNER field when calculating SERVTD_INFO_HASH
37	IGNORE_MROWNERCONFIG	If set to 1, a value of 0 is used instead of the service TD's MROWNER field when calculating SERVTD_INFO_HASH
38	IGNORE_RTMR0	If set to 1, a value of 0 is used instead of the service TD's RTMR0 field when calculating SERVTD_INFO_HASH
39	IGNORE_RTMR1	If set to 1, a value of 0 is used instead of the service TD's RTMR1 field when calculating SERVTD_INFO_HASH
40	IGNORE_RTMR2	If set to 1, a value of 0 is used instead of the service TD's RTMR2 field when calculating SERVTD_INFO_HASH
41	IGNORE_RTMR3	If set to 1, a value of 0 is used instead of the service TD's RTMR3 field when calculating SERVTD_INFO_HASH
42	IGNORE_SERVTD_HASH	If set to 1, a value of 0 is used instead of the service TD's SERVTD_HASH field when calculating SERVTD_INFO_HASH
63:43	RESERVED	Must be 0

### 3.12. Migration Types

#### 3.12.1. MBMD: Migration Bundle Metadata

- 5 MBMD is composed of a common header and a variable type-specific information.

## 3.12.1.1. Generic MBMD Structure

Table 3.47: Generic MBMD Structure Definition

Field	Offset (Bytes)	Size (Bytes)	Description	Included In MAC	Included in IV
SIZE	0	2	Overall size of the MBMD structure, in bytes	Yes	No
MIG_VERSION	2	2	Migration protocol version Changes in MBMD format, other migration bundle components format or migration protocol sequence require updating the protocol version. Migration protocol version is set by the MigTD before migration session starts.	Yes	No
MIGS_INDEX	4	2	Index of the migration stream used for migrating this migration bundle	As 0	Yes
MB_TYPE	6	1	The type of information being migrated: 0: TD-scope immutable non-memory state 1: TD-scope mutable non-memory state 2: VCPU-scope mutable non-memory state 3–15: Reserved 16: TD private memory 17–31: Reserved 32: Epoch token 33: Abort token Other: Reserved	Yes	No
RESERVED	7	1	Reserved, must be 0	Yes	No
MB_COUNTER	8	4	Per-stream migration bundle counter Starts from 0 on each migration epoch start, incremented by 1 on each MBMD export to the associated stream.	Yes	No
MIG_EPOCH	12	4	Migration epoch Starts from 0 on migration session start, incremented by 1 on each epoch token. A value of 0xFFFFFFFF indicates out-of-order phase.	Yes	No
IV_COUNTER	16	8	Monotonously incrementing counter, used as a component in the AES-GCM IV	As 0	Yes
Type-Specific Information	24	Variable	Variable-sized additional information for each specific type of MBMD	Yes	No
MAC	24+V	16	AES-256-GCM MAC over other MBMD fields and any associated migration data (all the migration pages)	No	No

## 3.12.1.2. TD-Scope Immutable Non-Memory State MBMD Fields

Table 3.48: TD-Scope Immutable Non-Memory State MBMD Fields Definition

Field	Offset (Bytes)	Size (Bytes)	Description	Included In MAC	Included in IV
NUM_F_MIGS	24	2	Maximum number of forward migration streams that will be used	Yes	No

Field	Offset (Bytes)	Size (Bytes)	Description	Included In MAC	Included in IV
RESERVED	26	2	Reserved, must be 0	Yes	No
NUM_SYS_MD_PAGES	28	1	Number of pages in the page list used for migrating TDX module metadata	Yes	No
RESERVED	29	3	Reserved, must be 0	Yes	No

### 3.12.1.3. TD-Scope Mutable Non-Memory State MBMD Fields

Table 3.49: TD-Scope Mutable Non-Memory State MBMD Fields Definition

Field	Offset (Bytes)	Size (Bytes)	Description	Included In MAC	Included in IV
RESERVED	24	8	Reserved, must be 0	Yes	No

### 5 3.12.1.4. VCPU-Scope Mutable Non-Memory State MBMD Fields

Table 3.50: VCPU-Scope Mutable Non-Memory State MBMD Fields Definition

Field	Offset (Bytes)	Size (Bytes)	Description	Included In MAC	Included in IV
VP_INDEX	24	2	Virtual CPU index	Yes	No
RESERVED	26	6	Reserved, must be 0	Yes	No

### 3.12.1.5. TD Private Memory MBMD Fields

Table 3.51: TD Private Memory MBMD Type-Specific Fields Definition

Field	Offset (Bytes)	Size (Bytes)	Description	Included In MAC	Included in IV
NUM_GPAS	24	2	Number of entries in the GPA list	Yes	No
GPA_LIST_ATTRIBUTES	26	1	Attributes of the GPA list, see Table 3.52 below	Yes	No
RESERVED	27	5	Reserved, must be 0	Yes	No

10

Table 3.52: GPA\_LIST\_ATTRIBUTES

Bits	Name	Description			
2:0	FORMAT	GPA list format			
		Value	Name	Description	
		0	GPA_ONLY	A GPA list page is provided	
		1	GPA_AND_ATTR	A GPA list page and a page attributes list page are provided	
		Other	RESERVED	Reserved	
7:3	RESERVED	Reserved: must be 0			

**3.12.1.6. Epoch Token MBMD Fields****Table 3.53: Epoch Token MBMD Fields Definition**

Field	Offset (Bytes)	Size (Bytes)	Description	Included In MAC	Included in IV
<b>TOTAL_MB</b>	24	8	The total number of migration bundles, including the current one, which have been exported since the beginning of the migration session	Yes	No

**3.12.1.7. Abort Token MBMD Fields****Table 3.54: Abort Token MBMD Fields Definition**

Field	Offset (Bytes)	Size (Bytes)	Description	Included In MAC	Included in IV
<b>RESERVED</b>	24	8	Reserved, must be 0	Yes	No

**3.12.1.8. TD Migration Protocol Version Compatibility**

The table below specifies the TD migration protocol versions for which the above MBMD definition is applicable.

**Table 3.55: MBMD Compatibility with TD Migration Versions**

TD Migration Version	Minimum	Maximum
Export version	0	0
Import version	0	0

**3.12.2. GPA List**

A GPA list specifies a list of GPAs to migrated by TDH.EXPORT.MEM and TDH.IMPORT.MEM, blocked for writing by TDH.EXPORT.BLOCKW or reset to their original SEPT entry state by TDH.EXPORT.RESTORE. GPA list may have up to 512 entries, is contained in a single 4KB page and must be aligned on 4KB. The GPA list may contain null entries, as indicated by OPERATION field's value set to 0 (NOP).

**3.12.2.1. GPA\_LIST\_INFO: HPA, First and Last Entries of a GPA List**

GPA\_LIST\_INFO is a 64b structure used as a GPR input and output operand of multiple migration interface functions, e.g., TDH.EXPORT.MEM. It provides the HPA of the GPA list page in shared memory, and the index of the first entry and last entries to be processed.

**Table 3.56: GPA\_LIST\_INFO**

Bits	Name	Description		
2:0	<b>FORMAT</b>	GPA list format		
		<b>Value</b>	<b>Name</b>	<b>Description</b>
		0	GPA_ONLY	A GPA list page is provided
		1	GPA_AND_L2_ATTR	A GPA list page and an L2 page attributes list page are provided. This format is only used by TDH.EXPORT.MEM and TDH.IMPORT.MEM.
		Other	RESERVED	Reserved

Bits	Name	Description
11:3	<b>FIRST_ENTRY</b>	Index of the first entry of the list to be processed
51:12	<b>HPA</b>	Bits 51:12 of the host physical address (including HKID) of the GPA list page, which must be a shared HPA
54:52	<b>RESERVED</b>	Reserved: must be 0
63:55	<b>LAST_ENTRY</b>	Index of the last entry in the GPA list

### 3.12.2.2. GPA List Entry

Table 3.57 below shows the format of a GPA list entry as used. Bits that are located in the same place as in SEPT entries are **highlighted**. The GPA list entry format is designed so that the output of TDH.EXPORT.BLOCKW can be used directly with TDH.EXPORT.MEM, and the output of TDH.EXPORT.MEM can be used directly with TDH.IMPORT.MEM.

**Table 3.57: GPA List Entry Definition**

Bit(s)	Size	Name	Description	TDH.EXPORT.BLOCKW		TDH.EXPORT.MEM		TDH.IMPORT.MEM		TDH.EXPORT.RESTORE	
				In	Out	In	Out	In	Out	In	Out
1:0	2	LEVEL	Mapping level (size)	Must be 0 (4KB)	Unmod.	Must be 0 (4KB)	Unmod.	Must be 0 (4KB)	Unmod.	Must be 0 (4KB)	Unmod.
2	1	PENDING	See below	Ignored	Unmod.	Ignored	Yes	Yes	Unmod.	Ignored	Unmod.
6:3	4	RESERVED	Reserved	Must be 0	Unmod.	Must be 0	Unmod.	Must be 0	Unmod.	Must be 0	Unmod.
9:7	3	L2_MAP	See below	Ignored	Unmod.	Ignored	Yes	Yes	Unmod.	Ignore	Unmod.
11:10	2	MIG_TYPE	See below	Yes	Unmod.	Yes	Unmod.	Yes	Unmod.	Yes	Unmod.
51:12	40	GPA	Guest Physical Address bits 51:12	Yes	Unmod.	Yes	Unmod.	Yes	Unmod.	Yes	Unmod.
53:52	2	OPERATION	See below	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
55:54	2	RESERVED	Reserved	Must be 0	Unmod.	Must be 0	Unmod.	Must be 0	Unmod.	Must be 0	Unmod.
60:56	5	STATUS	See below	Must be 0	Yes	Must be 0	Yes	Must be 0	Yes	Must be 0	Yes
63:61	3	RESERVED	Reserved	Must be 0	Unmod.	Must be 0	Unmod.	Must be 0	Unmod.	Must be 0	Unmod.

### 3.12.2.3. GPA List Entry Details

#### GPA List Details: LEVEL

- 10 Reserved for future support of page sizes other than 4KB.

#### GPA List Details: PENDING

**Table 3.58: PENDING Values Definition**

Value	Name	Description
0	MAPPED	SEPT entry is MAPPED
1	PENDING	SEPT entry is PENDING

#### GPA List Details: L2\_MAP

- 15 A bitmap with indicates whether the page is mapped in one or more L2 VM. This field is provided as part of the GPA list entry to enable the host VMM to prepare L2 SEPT pages before invoking TDH.IMPORT.MEM.

**GPA List Details: OPERATION**

The following tables describe the meaning of OPERATION, as used for each applicable interface function. Note that the OPERATION definitions for TDH.EXPORT.BLOCKW, TDH.EXPORT.MEM and TDH.IMPORT.MEM are designed to be compatible, so that the same GPA list can be used for all of them.

**Table 3.59: OPERATION Values Definition for TDH.EXPORT.BLOCKW**

Value	Input		Output	
	Name	Description	Name	Description
0	NOP	No operation	NOP	Not blocked for writing
1	BLOCKW	Block for writing	BLOCKW	Blocked for writing
2	NOP	No operation	NOP	Not blocked for writing
3	BLOCKW	Block for writing	BLOCKW	Blocked for writing

**Table 3.60: OPERATION Values Definition for TDH.EXPORT.MEM**

Value	Input		Output	
	Name	Description	Name	Description
0	NOP	No operation	NOP	Not exported
1	MIGRATE	Export	MIGRATE	Initial export during this migration session or following a CANCEL
2	CANCEL	Cancel previous export	CANCEL	Cancellation of a previous export
3	MIGRATE	Export	REMIGRATE	Re-export of updated content or attributes

**Table 3.61: OPERATION Values Definition for TDH.EXPORT.MEM**

Value	Input		Output	
	Name	Description	Name	Description
0	NOP	No operation	NOP	Not imported
1	MIGRATE	Initial import during this migration session or following a CANCEL	MIGRATE	Imported
2	CANCEL	Cancel previous import	CANCEL	Removed previous import
3	REMIGRATE	Re-import of updated page content or attributes	REMIGRATE	Imported

**Table 3.62: OPERATION Values Definition for TDH.EXPORT.RESTORE**

Value	Input		Output	
	Name	Description	Name	Description
0	NOP	No operation	NOP	Not restored
1	RESTORE	Restore SEPT entry to non-migration state	RESTORE	Restored
2	NOP	Reserved	NOP	Not restored
3	RESTORE	Restore SEPT entry to non-migration state	RESTORE	Restored



**GPA List Details: MIG\_TYPE****Table 3.63: MIG\_TYPE Values Definition**

Value	Name	Description
0	PAGE_4K	4KB private memory page
Other	RESERVED	Reserved for future types

**GPA List Details: STATUS****Table 3.64: STATUS Values Definition**

Value	Name	Description
0	SUCCESS	GPA list entry was processed successfully
1	SKIPPED	GPA list entry was skipped because NOP was requested
2	SEPT_WALK_FAILED	Secure EPT walk failed for the requested GPA
3	SEPT_ENTRY_BUSY_HOST_PRIORITY	Secure EPT entry was busy. The host VMM should retry the operation until successful.
4	SEPT_ENTRY_STATE_INCORRECT	Secure EPT entry state was incorrect for the requested operation and the TD's OP_STATE
5	TLB_TRACKING_NOT_DONE	TLB tracking was not done for the requested GPA
6	OP_STATE_INCORRECT	The TD's OP_STATE was incorrect for the requested operation and Secure EPT entry state
7	MIGRATED_IN_CURRENT_EPOCH	Requested GPA has already been migrated during the current migration epoch
8	MIG_BUFFER_NOT_AVAILABLE	Required migration buffer was not provided
9	NEW_PAGE_NOT_AVAILABLE	Required new TD page was not provided
10	INVALID_PAGE_MAC	Page MAC was invalid
11	DISALLOWED_IMPORT_OVER_REMOVED	Page import over a removed page is not allowed
12	TD_PAGE_BUSY_HOST_PRIORITY	TD page was busy. The host VMM should retry the operation until successful.
13	L2_SEPT_WALK_FAILED	L2 Secure EPT walk failed for the requested GPA
14	ATTR_LIST_ENTRY_INVALID	The L2 attributes list entry is invalid
15	GPA_LIST_ENTRY_INVALID	The GPA list entry is invalid
31-16	Reserved	Reserved

**3.12.2.4. TD Migration Protocol Version Compatibility**

The table below specifies the TD migration protocol versions for which the above GPA List definition is applicable.

**Table 3.65: GPA List Compatibility with TD Migration Versions**

TD Migration Version	Minimum	Maximum
Export version	0	0
Import version	0	0

### 3.12.3. Memory Migration Buffers List

A memory migration buffer list specifies a list of HPAs of 4KB pages in shared memory, to be used as output by TDH.EXPORT.MEM and as input by TDH.IMPORT.MEM. The list may have up to 512 64-bit entries, each containing a 4KB-aligned HPA (including HKID bits) of a page in shared memory. The list is contained in a single 4KB page and must be aligned on 4KB. The page list may contain null entries, indicated by the INVALID bit.

#### 3.12.3.1. Migration Buffers List Entry

Table 3.66: Migration Buffers List Entry

Bits	Name	Description
11:0	<b>RESERVED</b>	Reserved: must be 0 (unless bit 63 indicates an invalid entry)
51:12	<b>HPA</b>	Bits 51:12 of the host physical address (including HKID) of the migration buffer page, which must be a shared HPA
62:52	<b>RESERVED</b>	Reserved: must be 0 (unless bit 63 indicates an invalid entry)
63	<b>INVALID</b>	A value of 1 indicates that this entry is invalid

### 3.12.4. Page Attributes List

A page attributes list specifies a list of page aliases, to be used as output by TDH.EXPORT.MEM and as input by TDH.IMPORT.MEM. The list must contain an entry for each respective GPA list entry used with the same interface functions. The list may have up to 512 64-bit entries, in page L2 attributes format as defined in 3.6.3. The list is contained in a single 4KB page and must be aligned on 4KB.

#### 3.12.5. Memory Migration Page MAC List

A page MAC list specifies a list of MACs over 4KB migrated pages, their GPA list entries and (if applicable) page L2 attributes list entries, to be used as output by TDH.EXPORT.MEM and as input by TDH.IMPORT.MEM. The list must contain an entry for each respective GPA list entry used with the same interface functions. The list may have up to 256 128-bit entries, each containing a single AES-GMAC-256 of a migrated page. The list is contained in a single 4KB page and must be aligned on 4KB.

### 3.12.6. Non-Memory State Migration Buffers List

A non-memory state migration buffer list specifies a list of HPAs of 4KB pages in shared memory, to be used as output by TDH.EXPORT.STATE.\* and as input by TDH.IMPORT.STATE.\*. The list may have up to 512 64-bit entries, each containing a 4KB-aligned HPA (including HKID bits) of a page in shared memory. The list is contained in a single 4KB page and must be aligned on 4KB.

#### 3.12.6.1. PAGE\_LIST\_INFO: HPA and Attributes of a Page List

PAGE\_LIST\_INFO is a 64b structure used as a GPR input and output operand of multiple migration interface functions, e.g., TDH.EXPORT.STATE.TD. It provides the HPA of the migration buffers list page in shared memory, and the index of the last entry to be processed.

Table 3.67: PAGE\_LIST\_INFO

Bits	Name	Description
11:0	<b>RESERVED</b>	Reserved: must be 0
51:12	<b>HPA</b>	Bits 51:12 of the host physical address (including HKID) of the page list, which must be a shared HPA
54:52	<b>RESERVED</b>	Reserved: must be 0
63:55	<b>LAST_ENTRY</b>	Index of the last entry in the page list

DRAFT

## 4. UPDATED: TD Metadata (Non-Memory State)

This chapter describes the details of TD metadata, a.k.a. non-memory state or control state.

### 4.1. UPDATED: TD-Scope Metadata

TD-scope control structures are described the [TDX Module Base Spec].

#### 4.1.1. UPDATED: How to Read the TDR and TDCS Tables

The access columns describe whether this field is accessible to the host VMM in production mode (`ATTRIBUTES.DEBUG == 0`) and debug mode (`ATTRIBUTES.DEBUG == 1`), to the guest TD and to a Migration TD. Access is done using the TDX module metadata access functions, e.g., `TDH.MNG.RD`. Possible values are shown in the table below.

Table 4.1: Field Access Definition

Access	Meaning
None	No access to the field
RO	This field can only be read.
ROS	This field can only be read. The TDX module performs some special operation on read.
RW	This field can be read and written.
RWS	This field can be read and written. The TDX module performs some special operation on read and/or on write.

#### 4.1.2. UPDATED: TDR

**Note:** This section describes TDR, as defined. Implementation may differ.

TDR is the root control structure of a guest TD. TDR is encrypted using the Intel TDX global private HKID. It contains the minimal set of fields that allow TD management operation when the guest TD's private ephemeral HKID is not known yet or when the TD's key state is such that memory encrypted with the guest TD's private ephemeral key is not accessible.

TDR occupies a single 4KB naturally aligned page of memory. It is the first TD page to be allocated and the last to be removed. **None of the state in the TDR is migrated – it is locally initialized on the destination platform for a migrated TD.**

TRD fields are divided into the following classes:

Table 4.2: TDR Field Classes Definition

Field Class	Description
TD Management	These fields are used to manage the TDR page, its descendent TD private memory pages and control structure pages.
Key Management	These fields are used by the Intel TDX module to manage memory encryption keys. See the [TDX Module Base Spec] for details.
TD Preserving	These fields are used by the Intel TDX module to manage the TD across TD preserving updates.

**Note:** The table below lists only TDR fields that may be accessed by the host VMM in either production or debug mode.

Table 4.3: TDR Definition

Class	Field	VMM Prod. Access	VMM Debug Access	Guest Access	MigTD Access	Type	Field Size (Bytes)	Num Fields	Num Elem.	Elem. Size (Bytes)	Description	Base FIELD_ID (Hex)
TD Management	FATAL	RO	RO	None	None	Boolean	1	1	1	1	Indicates a fatal error, e.g., #MC during TD operation.	0x8010000000000001
TD Management	NUM_TDCX	RO	RO	None	None	32b Unsigned Integer	4	1	1	4	Number of TDCX pages that have been added by TDH.MNG.ADDCX	0x8010000200000002

Class	Field	VMM Prod. Access	VMM Debug Access	Guest Access	MigTD Access	Type	Field Size (Bytes)	Num Fields	Num Elem.	Elem. Size (Bytes)	Description	Base FIELD_ID (Hex)
TD Management	CHLDCNT	RO	RO	None	None	64b Unsigned Integer	8	1	1	8	The number of 4KB child pages (including opaque control structure pages) associated with this TDR	0x8010000300000004
TD Management	LIFECYCLE_STATE	RO	RO	None	None	LIFECYCLE_STATE	4	1	1	4	The life cycle state of this TD. LIFECYCLE_STATE values below are provided for debug only; they are subject to change in future TDX module versions: TD_HKID_ASSIGNED = 0x0 TD_KEYS_CONFIGURED = 0x1 TD_BLOCKED = 0x2 TD_TEARDOWN = 0x3	0x8010000200000005
TD Management	TDCX_PA	RO	RO	None	None	Array of Physical Address	8	9	1	8	Physical addresses of the TDCX pages	0x8010000300000010
TD Management	TD_UUID	RO	RO	RO	RO	256-bit blob	32	1	4	8	Universally Unique Identifier of the TD	0x8010000300000020
Key Management	HKID	RO	RO	None	None	16b Unsigned Integer	2	1	1	2	Private HKID	0x8110000100000001
Key Management	PKG_CONFIG_BITMAP	RO	RO	None	None	Bitmap	8	1	1	8	Bitmap that indicates on which package TDH.MNG.KEY.CONFIG was executed successfully using this private key entry	0x8110000300000002
TD Preserving	HANDOFF_VERSION	RO	RO	None	None	16b Unsigned Integer	2	1	1	2	The handoff version to which this TD is committed	0x8210000100000000
TD Preserving	SEAMDB_INDEX	RO	RO	None	None	64b Unsigned Integer	8	1	1	8	The index of the SEAMDB entry that holds the TDX module's TCB at TD creation time.	0x8210000300000001
TD Preserving	SEAMDB_NONCE	None	None	None	None	256-bit blob	32	1	4	8	The nonce of the SEAMDB entry that holds the TDX module's TCB at TD creation time.	0x8210000300000002
TD Management	FATAL	RO	RO	None	None	Boolean	1	1	1	1	Indicates a fatal error, e.g., #MC during TD operation.	0x8010000000000001
TD Management	NUM_TDCX	RO	RO	None	None	32b Unsigned Integer	4	1	1	4	Number of TDCX pages that have been added by TDH.MNG.ADDCX	0x8010000200000002
TD Management	CHLDCNT	RO	RO	None	None	64b Unsigned Integer	8	1	1	8	The number of 4KB child pages (including opaque control structure pages) associated with this TDR	0x8010000300000004
TD Management	LIFECYCLE_STATE	RO	RO	None	None	LIFECYCLE_STATE	4	1	1	4	The life cycle state of this TD. LIFECYCLE_STATE values below are provided for debug only; they are subject to change in future TDX module versions: TD_HKID_ASSIGNED = 0x0 TD_KEYS_CONFIGURED = 0x1 TD_BLOCKED = 0x2 TD_TEARDOWN = 0x3	0x8010000200000005
TD Management	TDCX_PA	RO	RO	None	None	Array of Physical Address	8	9	1	8	Physical addresses of the TDCX pages	0x8010000300000010
TD Management	TD_UUID	RO	RO	RO	RO	256-bit blob	32	1	4	8	Universally Unique Identifier of the TD	0x8010000300000020

#### 4.1.3. UPDATED: TDCS

**Note:** This section describes TDCS, as defined. Implementation may differ.

TDCS complements TDR as the logical control structure of a guest TD. TDCS is encrypted with the guest TS's ephemeral private key. It controls the guest TD operation and holds the state that is global to all the TD's VCPUs. **TDCS state fields are initialized either via TDH.MNG.INIT, or via TDH.IMPORT.STATE.IMMUTABLE – the latter when the TD is the target for migration.**

TDCS fields are divided into the following classes:

**Table 4.4: TDCS Field Classes Definition**

Field Class	Description
<b>TD Management</b>	These fields are used to manage the TDCS, its descendent TD private memory pages and control structure pages.
<b>TD Execution Control</b>	Control the execution of the guest TD: some TD execution control fields are provided as an input to TDH.MNG.INIT, and some of those are included in the TDG.MR.REPORT.

Field Class	Description
<b>TLB Epoch Tracking</b>	Track the TLB epoch of the guest TD – see the [TDX Module Base Spec] for details
<b>Measurement</b>	TD measurement registers and associated fields – see the [TDX Module Base Spec] for details
<b>Migration</b>	TDCS fields that control TD migration
<b>MIGSC Links</b>	Links to Migration Stream Context pages
<b>Service TD</b>	TDCS fields that control Service TD binding and operation
<b>MSR Bitmaps</b>	MSR bitmaps that control VM exit from the guest TD on RDMSR/WRMSR are common to all TD VCPUs and thus are stored as part of TDCS.
<b>Secure EPT Root Page</b>	The root page (PML5 or PML4) of the secure EPT
<b>L2 Secure EPT Root[3:1]</b>	The root pages (PML5 or PML4) of the secure EPTs associated with L2 VM 1, 2 and 3

**Note:** The table below lists only TDCS fields that may be accessed by the host VMM in either production or debug mode.

Table 4.5: TDCS Definition

Class	Field	VMM Prod. Access	VMM Debug Access	Guest Access	MigTD Access	Type	Field Size (Bytes)	Num Fields	Num Elem.	Elem. Size (Bytes)	Description	Base FIELD_ID (Hex)
TD Management	NUM_VCPUS	RO	RO	RO	None	32b Unsigned Integer	4	1	1	4	The number of VCPUs that are either in TDX non-root mode (TDVPS.VCPU_STATE == VCPU_ACTIVE) or are ready to run (TDVPS.VCPU_STATE == VCPU_READY); this includes VCPUs that have been successfully initialized (by TDH.VP.INIT) or imported (by TDH.IMPORT.STATE.VP) and have not since started teardown (due to a Triple Fault)	0x9010000200000001
TD Management	NUM_ASSOC_VCPUS	RO	RO	None	None	32b Unsigned Integer	4	1	1	4	The number of VCPUS associated with LPs – i.e., the LPs might hold TLB translations and/or cached TD VMCS	0x9010000200000002
TD Management	OP_STATE	RO	RO	None	None	OP_STATE	4	1	1	4	The operation state (sub-state of life cycle TD_KEYS_CONFIGURED state) of this TD. OP_STATE values below are provided for debug only; they are subject to change in future TDX module releases: UNINITIALIZED = 0 INITIALIZED = 1 RUNNABLE = 2 LIVE_EXPORT = 3 PAUSED_EXPORT = 4 COMMITTED_EXPORT = 6 MEMORY_IMPORT = 7 STATE_IMPORT = 8 POST_IMPORT = 9 LIVE_IMPORT = 10 FAILED_IMPORT = 11	0x9010000200000004
TD Management	NUM_L2_VMS	RO	RO	RO	None	16b Unsigned Integer	2	1	1	2	Number of L2 VM, may be 0 to 3	0x9010000100000005
Execution Controls	ATTRIBUTES	RO	RO	RO	None	ATTRIBUTES	8	1	1	8	TD attributes	0x1110000300000000
Execution Controls	XFAM	RO	RO	RO	None	XCRO	8	1	1	8	Extended Features Available Mask: indicates the extended user and system features which are available for the TD. Copied to each TDVPS on TDH.VP.INIT.	0x1110000300000001
Execution Controls	MAX_VCPUS	RO	RO	RO	None	32b Unsigned Integer	4	1	1	4	Maximum number of VCPUs	0x1110000200000002
Execution Controls	GPAW	RO	RO	RO	None	Boolean	1	1	1	1	This bit has the same meaning as the VMCS GPAW execution control: 0: GPA.SHARED bit is GPA[47] 1: GPA.SHARED bit is GPA[51]	0x1110000000000003
Execution Controls	EPTP	RO	RO	None	None	EPTP	8	1	1	8	TD-scope Secure EPT pointer: format is the same as the VMCS EPTP execution control; copied to each TD VMCS EPTP on TDH.VP.INIT	0x1110000300000004
Execution Controls	TSC_OFFSET	RO	RO	None	None	64b unsigned Integer	8	1	1	8	TD-scope TSC offset execution control: copied to each TD VMCS TSC-offset execution control on TDH.VP.INIT	0x111000030000000A

Class	Field	VMM Prod. Access	VMM Debug Access	Guest Access	MigTD Access	Type	Field Size (Bytes)	Num Fields	Num Elem.	Elem. Size (Bytes)	Description	Base FIELD_ID (Hex)
Execution Controls	TSC_MULTIPLIER	RO	RO	None	None	64b Unsigned Integer	8	1	1	8	TD-scope TSC multiplier execution control: copied to each TD VMCS TSC-multiplier execution control on TDH.VP.INIT	0x111000030000000B
Execution Controls	TSC_FREQUENCY	RO	RO	RO	None	16b Unsigned Integer	2	1	1	2	Virtual TSC frequency – in units of 25MHz	0x111000010000000C
Execution Controls	VIRTUAL_TSC	None	None	None	None	64b Unsigned Integer	8	1	1	8	Virtual TSC value. This value is only calculated and used at the end of migration (TDH.EXPORT.STATE.TD) and doesn't need to actually reside in TDCS.	0x111000030000000D
Execution Controls	NUM_CPUID_VALUES	ROS	ROS	None	None	16b Unsigned Integer	2	1	1	2	Number of valid fields in CPUID_VALUES	0x911000010000000E
Execution Controls	XBUFF_SIZE	RO	RO	None	None	Unsigned Integer	4	1	1	4	Actual size of the XSAVE buffer – calculated by TDH.MNG.INIT based on XFAM	0x911000020000000F
Execution Controls	NOTIFY_ENABLES	None	RW	RW	None	Bitmap	8	1	1	8	Enable guest notification of events: Bit 0: Notify when Zero Step attack is suspected Bits 63:1: Reserved, must be 0	0x9110000300000010
Execution Controls	HP_LOCK_TIMEOUT	RWS	RWS	None	None	Unsigned 32b integer	8	1	1	8	Host priority timeout value, in usec (internally, stored in TSC tick units)	0x9110000300000011
Execution Controls	VM_CTLs	RW	RW	None	None	Array of 64-bit bitmaps	8	4	1	8	An array of 4 per-VM controls: Bit 0: Applicable for L2 VMs only. If set, a TDCALL flow that encounters an SEPT walk error causes a TD exit. Else, it returns an error code to the guest. Bits 63:1: Reserved, must be 0	0x9110000300000012
Execution Controls	CPUID_VALID	None	RO	None	None	Array of boolean	1	80	1	1	Non-architectural - an array of boolean flag, indicating the validity of CPUID_VALUES. Indexed by the internal CPUID lookup table indexing.	0x9110000000000080
Execution Controls	XBUFF_OFFSETS	RO	RO	None	None	Unsigned Integer	4	19	1	4	XSAVE buffer components offsets – calculated by TDH.MNG.INIT based on XFAM	0x9110000200000800
TLB Epoch Tracking	TD_EPOCH	RO	RO	None	None	64b Integer	8	1	1	8	The TD epoch counter: incremented by the host VMM using the TDH.MEM.TRACK function	0x9210000300000000
TLB Epoch Tracking	REFCOUNT	RO	RO	None	None	16b Unsigned Integer	2	2	1	2	Each REFCOUNT counts the number of LPs which may have TLB entries created during a specific TD_EPOCH and are currently executing in TDX non-root mode.	0x9210000100000001
TLB Epoch Tracking	LATEST_BLOCK_EPOCH	RO	RO	None	None	64b integer	8	1	1	8	Records TD_EPOCH at time the last block or block-for-writing operation was done	0x9210000300000003
Measurement	MRTD	RO	RO	RO	None	SHA384_H ASH	48	1	6	8	Measurement of the initial contents of the TD	0x1310000300000000
Measurement	MIRCONFIGID	RO	RO	RO	None	SHA384_H ASH	48	1	6	8	Software-defined ID for non-owner-defined configuration of the guest TD – e.g., run-time or OS configuration	0x1310000300000010
Measurement	MROWNER	RO	RO	RO	None	SHA384_H ASH	48	1	6	8	Software-defined ID for the guest TD's owner	0x1310000300000018
Measurement	MROWNERCONFIG	RO	RO	RO	None	SHA384_H ASH	48	1	6	8	Software-defined ID for owner-defined configuration of the guest TD – e.g., specific to the workload rather than the run-time or OS	0x1310000300000020
Measurement	RTMR	None	RO	RO	None	Array of SHA384_H ASH	48	4	6	8	Array of NUM_RTMRs run-time extendable measurement registers	0x1310000300000040
Virt. MSR Values	VIRTUAL_IA32_VMX_B ASIC	None	RO	None	None	64-bit integer	8	1	1	8	Virtual value of MSR	0x9610000300000480
Virt. MSR Values	VIRTUAL_IA32_VMX_M ISC	None	RO	None	None	64-bit integer	8	1	1	8	Virtual value of MSR	0x9610000300000485
Virt. MSR Values	VIRTUAL_IA32_VMX_CR 0_FIXED0	None	RO	None	None	64-bit integer	8	1	1	8	Virtual value of MSR	0x9610000300000486
Virt. MSR Values	VIRTUAL_IA32_VMX_CR 0_FIXED1	None	RO	None	None	64-bit integer	8	1	1	8	Virtual value of MSR	0x9610000300000487
Virt. MSR Values	VIRTUAL_IA32_VMX_CR 4_FIXED0	None	RO	None	None	64-bit integer	8	1	1	8	Virtual value of MSR	0x9610000300000488
Virt. MSR Values	VIRTUAL_IA32_VMX_CR 4_FIXED1	None	RO	None	None	64-bit integer	8	1	1	8	Virtual value of MSR	0x9610000300000489
Virt. MSR Values	VIRTUAL_IA32_VMX_PROCBASED_CTLs2	None	RO	None	None	64-bit integer	8	1	1	8	Virtual value of MSR	0x961000030000048B
Virt. MSR Values	VIRTUAL_IA32_VMX_EPT_VPID_CAP	None	RO	None	None	64-bit integer	8	1	1	8	Virtual value of MSR	0x961000030000048C
Virt. MSR Values	VIRTUAL_IA32_VMX_TUE_PINBASED_CTLs	None	RO	None	None	64-bit integer	8	1	1	8	Virtual value of MSR	0x961000030000048D
Virt. MSR Values	VIRTUAL_IA32_VMX_TUE_PROCBASED_CTLs	None	RO	None	None	64-bit integer	8	1	1	8	Virtual value of MSR	0x961000030000048E
Virt. MSR Values	VIRTUAL_IA32_VMX_TUE_EXIT_CTLs	None	RO	None	None	64-bit integer	8	1	1	8	Virtual value of MSR	0x961000030000048F
Virt. MSR Values	VIRTUAL_IA32_VMX_TUE_ENTRY_CTLs	None	RO	None	None	64-bit integer	8	1	1	8	Virtual value of MSR	0x9610000300000490

Class	Field	VMM Prod. Access	VMM Debug Access	Guest Access	MigTD Access	Type	Field Size (Bytes)	Num Fields	Num Elem.	Elem. Size (Bytes)	Description	Base FIELD_ID (Hex)
Virt. MSR Values	VIRTUAL_IA32_VMX_VMFUNC	None	RO	None	None	64-bit integer	8	1	1	8	Virtual value of MSR	0x9610000300000491
Virt. MSR Values	VIRTUAL_IA32_VMX_PROCBASED_CTLS3	None	RO	None	None	64-bit integer	8	1	1	8	Virtual value of MSR	0x9610000300000492
Virt. MSR Values	VIRTUAL_IA32_VMX_EXIT_CTLS2	None	RO	None	None	64-bit integer	8	1	1	8	Virtual value of MSR	0x9610000300000493
CPUID	CPUID_VALUES	RO	RO	None	None	CPUID_RET	16	80	2	8	Values returned by CPUID leaves/sub-leaves: Element 0[31:0]: EAX Element 0[63:32]: EBX Element 1[31:0]: ECX Element 1[63:32]: EDX Field code is composed as follows: 31:17 Reserved, must be 0 16 Leaf number bit 31 15:9 Leaf number bit 6:0 8 Sub-leaf not applicable flag 7:1 Sub-leaf number bits 6:0 0 Element index within field	0x9410000300000000
Migration	MIG_DEC_KEY_SET	RO	RO	None	None	Boolean	1	1	1	1	Set when a new MIG_DEC_KEY is written, cleared when the MIG_DEC_KEY is copied to MIG_DEC_WORKING_KEY	0x9810000000000001
Migration	EXPORT_COUNT	RO	RO	None	None	32b Unsigned Integer	4	1	1	4	Counts the number of times this TD has been exported, included aborted export sessions. Incremented at the beginning of each export session (TDH.EXPORT.STATE.IMMUTABLE).	0x9810000200000002
Migration	IMPORT_COUNT	RO	RO	None	None	32b Unsigned Integer	4	1	1	4	Counts the number of times this TD has been imported. Incremented by TDH.IMPORT.COMMIT.	0x9810000200000003
Migration	MIG_EPOCH	RO	RO	None	None	32b Unsigned Integer	4	1	1	4	Migration epoch Starts from 0 on migration session start, incremented by 1 on each epoch token. A value of 0xFFFFFFFF indicates out-of-order phase.	0x9810000200000004
Migration	BW_EPOCH	RO	RO	None	None	64b Unsigned Integer	8	1	1	8	Blocking-for-write epoch Holds the value of TD_EPOCH at last time TDH.EXPORT.BLOCKW blocked a page for writing.	0x9810000300000005
Migration	TOTAL_MB_COUNT	RO	RO	None	None	Unsigned Integer	8	1	1	8	The total number of migration bundles exported or imported during the current migration sessions	0x9810000300000006
Migration	MIG_DEC_KEY	None	RO	None	RWS	KEY_256	32	1	4	8	Migration decryption key, as written by the Migration TD Special write behavior: - Acquire a shared lock on TDCS.OP_STATE to prevent concurrent migration session start. - Set MIG_DEC_KEY_SET	0x9810000300000010
Migration	MIG_DEC_WORKING_KEY	None	RO	None	RO	KEY_256	32	1	4	8	Pre-HSD1308803845: Migration working key, copied from MIG_KEY at the beginning of a migration session and used throughout the session. Post-HSD1308803845: Migration decryption working key Copied from MIG_DEC_KEY at the beginning of a migration session and used throughout the session.	0x9810000300000014
Migration	MIG_ENC_KEY	None	RO	None	RO	KEY_256	32	1	4	8	Migration encryption key This key is first generated by the TDX module on TDH.MNG.ADDCX, and is re-generated at the beginning of each migration session (TDH.EXPORT/IMPORT.STATE.IMMUTABLE) for use in a following session.	0x9810000300000018
Migration	MIG_ENC_WORKING_KEY	None	RO	None	RO	0	32	1	4	8	Migration encryption working key Copied from MIG_ENC_KEY at the beginning of a migration session (before a new MIG_ENC_KEY is generated) and used throughout the session.	0x981000030000001C
Migration	MIG_VERSION	RO	RO	None	RWS	16b Unsigned Integer	2	1	1	2	Migration protocol version, as written by the migration TD	0x9810000100000020
Migration	MIG_WORKING_VERSION	RO	RO	None	RO	16b Unsigned Integer	2	1	1	2	Migration working protocol version, copied from MIG_VERSION at the beginning of a migration session and used throughout the session	0x9810000100000021
Migration	DIRTY_COUNT	RO	RO	None	None	16b Unsigned Integer	8	1	1	8	Counts of the number of pages that must be re-exported, because their contents have been modified since they have been exported, before a start token may be generated	0x9810000300000030
Migration	MIG_COUNT	RO	RO	None	None	64b Unsigned Integer	8	1	1	8	Counts the number of SEPT entries that need to be cleaned up after an aborted migration	0x9810000300000031
Migration	NUM_MIGS	RO	RO	None	None	16b Unsigned Integer	2	1	1	2	Number of Migration Stream Context (MIGSC) pages that have been allocated (including the backward and forward MIGSC pages)	0x9810000100000032



Class	Field	VMM Prod. Access	VMM Debug Access	Guest Access	MigTD Access	Type	Field Size (Bytes)	Num Fields	Num Elem.	Elem. Size (Bytes)	Description	Base FIELD_ID (Hex)
Migration	NUM_MIGRATED_VCPUS	RO	RO	None	None	32b Unsigned Integer	4	1	1	4	Number of VCPUs that have been migrated	0x9810000200000034
Migration	PRE_IMPORT_UUID	RO	RO	RO	RO	256-bit blob	32	1	4	8	The original value of TD_UUID before is was overwritten as part of the immutable state import	0x9810000300000040
Service TD	SERVTD_HASH	RO	RO	RO	None	SHA384_HASH	48	1	6	8	SHA384 hash of the bound or pre-bound service TDs	0x9910000300000000
Service TD	SERVTD_NUM	RO	RO	RO	None	16-bit unsigned integer	2	1	1	2	Number of bound or pre-bound service TDs	0x9910000100000006
Service TD	SERVTD_BINDINGS_TABLE	RO	RO	RO	None	Array of SERVTD_BINDING entries	128	1	16	8	An array of service TD binding information entries	0x9910000300000080
MSR Bitmaps	MSR_BITMAPS	None	RO	None	None	MSR Exit Bitmaps	8	512	1	8	TD-scope RDMSR/WRMSR exit control bitmaps	0x2010000300000000
Secure EPT Root	SEPT_ROOT	None	RO	None	None	Secure EPT Entry	8	512	1	8	Secure EPT root page (PML5 or PML4)	0x2110000300000000
MIGSC Links	MIGSC_LINKS	RO	RO	None	None	MIGSC_LINK	8	512	1	8	An array of links to Migration Stream Contexts. Post-HSD1308803845: - Entry 0 is for the backward migration stream. - Entry [i + 1] is for forward migration stream i. Each entry contains the following information: Bit 51:12: MIGSC_HPA: Bits 52:12 of the MIGSC page HPA (without the HKID bits) Bit 0: LOCK: Mutex for controlling access to the MIGSC Bit 1: INITIALIZED: A boolean flag, indicating that the MIGSC has been initialized. Bit 2: ENABLED: A boolean flag, indicating that the MIGS is enabled The flags are held here, not in the MIGSC itself, to enable efficient state-related operations on all migration streams, e.g., disabling all streams.	0x9A10000300000000
L2 Secure EPT Root [1]	L2_SEPT_ROOT_1	None	RO	None	None	Secure EPT Entry	8	512	1	8	L2 VM's Secure EPT root page (PML5 or PML4)	0x2510000300000000
L2 Secure EPT Root [2]	L2_SEPT_ROOT_2	None	RO	None	None	Secure EPT Entry	8	512	1	8	L2 VM's Secure EPT root page (PML5 or PML4)	0x2910000300000000
L2 Secure EPT Root [3]	L2_SEPT_ROOT_3	None	RO	None	None	Secure EPT Entry	8	512	1	8	L2 VM's Secure EPT root page (PML5 or PML4)	0x2D10000300000000

## 4.2. **UPDATED:** TDVPS: VCPU-Scope Metadata

**Note:** This section describes TDVPS, as defined. Implementation may differ.

TDVPS is described in the [TDX Module Base Spec].

### 5 4.2.1. **UPDATED:** Overview

Logically, in the Intel TDX module's linear address space, TDVPS is a single structure that holds the state and control information for a single TD VCPU. The state is loaded to the LP on TD Entry and saved on TD exits.

Physically, TDVPS is composed of a root page (TDVPR) and multiple extension pages (TDCX). The pages need not be contiguous in physical memory.

- 10 **TDVPS is initialized by TDH.MNG.INITVP. For an TD being migrated, TDVPS is imported by TDH.IMPORT.STATE.VP, which initializes some state fields and migrates some fields from the source TD VPS state.**

TDVPS fields are divided into the following classes:

**Table 4.6: TDVPS Field Classes Definition**

Field Class	Description
<b>VCPU Management</b>	These fields are used to manage the TDVPS and the TD VCPU.
<b>TD VMCS</b>	The TD VCPU's architectural VMCS

Field Class	Description
VAPIC	The TD VCPU's Virtual APIC page
VE_INFO	Holds Virtualization Exception (#VE) information
Guest GPR State	TD VCPU's general-purpose register state
Guest MSR State	TD VCPU's MSR state
Guest Extended State	TD VCPU's extended state
VMCS[3:1]	VMCS pages of L2 VM 1, 2 and 3
MSR Bitmaps[3:1]	MSR bitmap pages of L2 VM 1, 2 and 3
MSR Bitmaps Shadow[3:1]	MSR bitmap shadow pages of L2 VM 1, 2 and 3

#### 4.2.2. How to Read the TDVPS (including TD VMCS) Tables

##### 4.2.2.1. **UPDATED:** Field Access

The access columns describe whether this field is accessible to the host VMM in production mode (ATTRIBUTES.DEBUG == 0) and debug mode (ATTRIBUTES.DEBUG == 1) and to the guest TD. Access is done using the TDX module metadata access functions, e.g., TDH.VP.RD. Possible values are shown in the table below.

Table 4.7: Field Access Definition

Access	Meaning
None	No access to the field
RO	This field can only be read.
ROS	This field can only be read. Reading is subject to some special handling as described per field.
RW	This field can be read and written. No limitations are imposed except for checking that the value fits in the field size.
RWS	This field can be read and written. Reading and/or writing is subject to the same limitations as if the field was modified by the guest TD (for guest state fields) and/or other limitation or special handling as described per field.

#### 4.2.3. TDVPS (excluding TD VMCS)

- Note:** The table below lists only TDVPS fields that may be accessed by the host VMM in either production or debug mode.

Table 4.8: TDVPS Definition

Class	Field	VMM Prod. Access	VMM Debug Access	Guest Access	Type	Field Size (Bytes)	Num Fields	Num Elem.	Elem. Size (Bytes)	Description	Base FIELD_ID (Hex)
Management	VCPU_STATE	None	RO	None	VCPU_STATE	1	1	1	1	The activity state of the VCPU. The values below are subject to change with new TDX module releases. 0x0: VCPU_UNINITIALIZED 0x1: VCPU_IMPORT 0x2: VCPU_READY 0x4: VCPU_ACTIVE 0x5: VCPU_DISABLED	0xA020000000000000
Management	LAST_TD_EXIT	None	RO	None	LAST_TD_EXIT	1	1	1	1	Type of the last TD exit. The values below are subject to change with new TDX module releases. 0x0: ASYNC_FAULT 0x1: ASYNC_TRAP 0x2: TDVMCALL	0xA02000000000000F
Management	VCPU_INDEX	RO	RO	RW	32b Unsigned Integer	4	1	1	4	Sequential index of the VCPU in the parent TD. VCPU_INDEX indicates the order of VCPU initialization (by TDINITVP), starting from 0, and is made available to the TD via TDINFO.	0xA020000200000002

Class	Field	VMM Prod. Access	VMM Debug Access	Guest Access	Type	Field Size (Bytes)	Num Fields	Num Elem.	Elem. Size (Bytes)	Description	Base FIELD_ID (Hex)
										VCPU_INDEX is in the range 0 to (TDCS.MAX_VCPUS - 1), up to 0xFFFE	
Management	NUM_TDVPS_PAGES	RO	RO	None	Unsigned Integer	1	1	1	1	Number of pages in this TDVPS	0xA020000000000003
Management	TDVPS_PAGE_PA	RO	RO	None	Array of PA	8	15	1	8	An array of TDVPS_PAGES physical address pointers to the TDVPS physical pages	0xA020000300000010
Management	ASSOC_LPID	RO	RO	None	Integer	4	1	1	4	The unique, hardware-derived identifier of the logical processor on which this VCPU is currently associated (either by TDENTER or by other VCPU-specific SEAMCALL flow): <ul style="list-style-type: none"> <li>A value of -1 indicates that VCPU is not associated with any LP.</li> <li>Initialized by TDH.VP.INIT to the LP_ID on which it ran.</li> </ul>	0xA020000200000004
Management	VCPU_EPOCH	RO	RO	None	Integer	8	1	1	8	The value of TDCS.TD_EPOCH at the time this VCPU entered TDX non-root mode	0xA020000300000006
Management	CPUID_SUPERVISOR_VE	RO	RO	RW	Boolean	1	1	1	1	When set, the Intel TDX module injects #VE on guest TD execution of CPUID in CPL = 0.	0xA020000000000007
Management	CPUID_USER_VE	RO	RO	RW	Boolean	1	1	1	1	When set, the Intel TDX module injects #VE on guest TD execution of CPUID in CPL > 0.	0xA020000000000008
Management	LAST_EXIT_TSC	None	RO	None	Unsigned 64b Integer	8	1	1	8	Initialized to the value returned rdtsc on TDH.VP.INIT	0xA02000030000000A
Management	PEND_NMI	RW	RW	None	Boolean	1	1	1	1	When set, the Intel TDX module injects an NMI to the guest TD at the next available opportunity (NMI window open after TDENTER). the Intel TDX module then clears PEND_NIM.	0x202000000000000B
Management	NMI_UNBLOCKING_DUE_TO_IRET	None	RO	None	Boolean	1	1	1	1	Flags that on the last VM exit NMI unblocking due to IRET was indicated	0xA020000000000040
Management	XFAM	RO	RWS	None	Bitmap	8	1	1	8	Copied from TDCS on TDH.VP.INIT. On TDH.VP.WR, checked for architectural and platform compatibility	0x202000030000000C
Management	LAST_EPF_GPA_LIST_IDX	None	RO	None	Unsigned Integer	1	1	1	1	Number of valid entries in LAST_EPF_GPA_LIST	0xA02000000000000D
Management	POSSIBLY_EPF_STEPPING	None	RO	None	Unsigned Integer	1	1	1	1	Number of possibly legal EPT Faults (EPFs) detected so far at this TD vCPU instruction	0xA02000000000000E
Management	HP_LOCK_BUSY_START	None	RO	None	Unsigned 64b Integer	8	1	1	8	TSC value at start of the host priority busy period	0xA020000300000030
Management	HP_LOCK_BUSY	None	RO	None	Boolean	1	1	1	1	Indicates that the guest has encountered a busy host priority lock	0xA020000000000031
Management	LAST_SEAMDB_INDEX	None	RO	None	64-bit unsigned integer	8	1	1	8	Value of PL.SEAMDB_INDEX, sampled on last VCPU-to-LP association	0xA020000300000032
Management	CURR_VM	None	RO	None	16-bit unsigned integer	2	1	1	2	VM index currently used for this VCPU	0xA020000100000041
Management	L2_EXIT_HOST_ROUTING	None	RO	None	0	1	1	1	1	Sticky status of L2-to-L1 routing by the host (TDH.VP.ENTER with RESUME_L1 set): 0: L2 TD exit not routed to L1 1: L2 async TD exit routed to L1 2: L2 sync (TDG.VP.VMCALL) TD exit routed to L1	0xA020000000000042
Management	VM_LAUNCHED	None	RO	None	Boolean	1	4	1	1	A Boolean flag per VM, indicating whether the VM has been VMLAUNCH'ed on this LP since it has last been associated with this VCPU. If TRUE, VM entry should use VMRESUME. Else, VM entry should use VMLAUNCH.	0xA020000000000044
Management	LP_DEPENDENT_HPA_UPDATE	None	RO	None	Boolean	1	4	1	1	A Boolean flag per VM, indicating that the LP-dependent HPA fields have been updated. Cleared after new VCPU-to-LP association.	0xA020000000000048
Management	MODULE_DEPENDENT_FIELDS_UPDATED	None	RO	None	Boolean	1	4	1	1	A Boolean flag per VM, indicating that the TDX module dependent HPA fields have been updated. Cleared after new VCPU-to-LP association that follows a TD preserving update.	0xA02000000000004C
Management	L2_CTLs	None	RWS	RWS	64-bit bitmap	8	4	1	8	L2 VM control flags, used by the L1 VMM: Bit 0: ENABLE_SHARED_EPTP Bit 1: ENABLE_TDVMCALL Bit 2: ENABLE_EXTENDED_VE Bits 63:3: RESERVED, must be 0	0xA020000300000050
Management	L2_DEBUG_CTLs	None	RW	None	64-bit bitmap	8	4	1	8	L2 VM debug control flags, used by the off-TD debugger: Bit 0: TD_EXIT_ON_L1_TO_L2 Bit 1: TD_EXIT_ON_L2_TO_L1 Bit 2: TD_EXIT_ON_L2_VM_EXIT Bits 63:3: RESERVED, must be 0	0xA020000300000054
Management	TSC_DEADLINE	None	RO	RWS	64-bit unsigned integer	8	4	1	8	TSC deadline, in virtual TSC ticks A value of -1 indicates no TSC deadline Applicable only to L2 VMs	0xA020000300000058
Management	SHADOW_TSC_DEADLINE	None	RO	None	64-bit unsigned integer	8	4	1	8	TSC deadline, in native TSC ticks Applicable only to L2 VMs	0xA02000030000005C

Class	Field	VMM Prod. Access	VMM Debug Access	Guest Access	Type	Field Size (Bytes)	Num Fields	Num Elem.	Elem. Size (Bytes)	Description	Base FIELD_ID (Hex)
Management	BASE_L2_CR0_GUEST_HOST_MASK	None	RW	None	64-bit bitmap	8	1	1	8	The base guest/host mask used for any L2 CR0 access by the L1 VMM. Bits 5, 29 and 30 can't be written even in debug mode.	0xA020000300000080
Management	BASE_L2_CR0_READ_SHADOW	None	RW	None	64-bit bitmap	8	1	1	8	The base read shadow used for any L2 CR0 access by the L1 VMM. Bits 0 and 5 can't be written even in debug mode.	0xA020000300000081
Management	BASE_L2_CR4_GUEST_HOST_MASK	None	RW	None	64-bit bitmap	8	1	1	8	The base guest/host mask used for any L2 CR4 access by the L1 VMM. Bits 6, 13 and 14 can't be written even in debug mode.	0xA020000300000082
Management	BASE_L2_CR4_READ_SHADOW	None	RW	None	64-bit bitmap	8	1	1	8	The base read shadow used for any L2 CR4 access by the L1 VMM. Bit 6 can't be written even in debug mode.	0xA020000300000083
Management	SHADOW_CR0_GUEST_HOST_MASK	None	RO	None	64-bit bitmap	8	4	1	8	The L2 VMCS CR0 guest/host mask original value, as set by the L1 VMM. Applicable only to L2	0xA020000300000084
Management	SHADOW_CR0_READ_SHADOW	None	RO	None	64-bit bitmap	8	4	1	8	The L2 VMCS CR0 read shadow original value, as set by the L1 VMM. Applicable only to L2	0xA020000300000088
Management	SHADOW_CR4_GUEST_HOST_MASK	None	RO	None	64-bit bitmap	8	4	1	8	The L2 VMCS CR4 guest/host mask original value, as set by the L1 VMM. Applicable only to L2	0xA02000030000008C
Management	SHADOW_CR4_READ_SHADOW	None	RO	None	64-bit bitmap	8	4	1	8	The L2 VMCS CR4 read shadow original value, as set by the L1 VMM. Applicable only to L2	0xA020000300000090
Management	SHADOW_NOTIFY_WINDOW	None	RO	None	32-bit unsigned integer	4	4	1	4	Shadow value of VMCS notify window, in crystal clock ticks. Applicable to all VMs	0xA020000200000094
Management	SHADOW_PID_HPA	None	RO	None	Shared HPA	8	1	1	8	Shadow value of VMCS posted interrupt descriptor address. Applicable only to L1	0xA020000300000098
Management	SHADOW_PINBASED_EXEC_CONTROLS	None	RO	None	32-bit bitmap	4	1	1	4	Shadow value of VMCS pin based execution controls. Applicable only to L1	0xA02000020000009C
Management	SHADOW_PLE_GAP	None	RO	None	32-bit unsigned integer	4	4	1	4	Shadow value of VMCS PLE_GAP, in virtual TSC ticks. Applicable only to L2 VMs	0xA0200002000000A4
Management	SHADOW_PLE_WINDOW	None	RO	None	32-bit unsigned integer	4	4	1	4	Shadow value of VMCS PLE_WINDOW, in virtual TSC ticks. Applicable only to L2 VMs	0xA0200002000000A8
Management	SHADOW_POSTED_INT_NOTIFICATION_VECTOR	None	RO	None	16-bit unsigned integer	2	1	1	2	Shadow value of VMCS posted interrupt notification window. Applicable only to L1	0xA0200001000000AC
Management	SHADOW_PROCBASED_EXEC_CONTROLS2	None	RO	None	32-bit bitmap	4	4	1	4	Shadow value of VMCS secondary processor based execution controls. Applicable to all VMs	0xA0200002000000B0
Management	SHADOW_SHARED_EPTP	None	RO	None	HPA	8	4	1	8	Shadow value of VMCS shared EPTP. Applicable to all VMs	0xA0200003000000B4
Management	L2_ENTER_GUEST_STATE_GPA	None	RO	None	GPA	8	4	1	8	GPA of TDG.VP.ENTER guest state output buffer. Applicable only to L2 VMs	0xA020000300000100
Management	L2_ENTER_GUEST_STATE_HPA	None	RO	None	HPA	8	4	1	8	HPA (incl. HKID) of TDG.VP.ENTER guest state output buffer. Applicable only to L2 VMs	0xA020000300000104
Management	L2_ENTER_GUEST_STATE_EPOCH	None	RO	None	Unsigned 64b Integer	8	4	1	8	TD epoch at the time GPA was translated to HPA. Applicable only to L2 VMs	0xA020000300000108
Management	VE_INFO_GPA	None	RO	None	GPA	8	4	1	8	Shadow GPA of the VE_INFO area. Applicable only to L2 VMs	0xA02000030000010C
Management	VE_INFO_HPA	None	RO	None	HPA	8	4	1	8	Shadow HPA (incl. HKID) of the VE_INFO area. Applicable only to L2 VMs	0xA020000300000110
Management	VE_INFO_EPOCH	None	RO	None	Unsigned 64b Integer	8	4	1	8	TD epoch at the time GPA was translated to HPA. Applicable only to L2 VMs	0xA020000300000114
Management	L2_VAPIC_GPA	None	RO	None	GPA	8	4	1	8	Shadow GPA of the L2 virtual APIC address (used by the L1 VMM). Applicable only to L2 VMs	0xA020000300000118
Management	L2_VAPIC_HPA	None	RO	None	HPA	8	4	1	8	Shadow HPA (incl. HKID) of the L2 virtual APIC address. Applicable only to L2 VMs	0xA02000030000011C
Management	L2_VAPIC_EPOCH	None	RO	None	Unsigned 64b Integer	8	4	1	8	TD epoch at the time GPA was translated to HPA. Applicable only to L2 VMs	0xA020000300000120
EPT Violation Log	LAST_EPF_GPA_LIST	None	RO	None	GPA	8	32	1	8	Array of GPAs that caused EPF so far at this TD vCPU instruction	0xA220000300000200
CPUID Control	CPUID_CONTROL	None	RO	RW	Array of 8-bit bitmaps	1	128	1	1	Bit 0: When set, the Intel TDX module injects #VE on guest TD execution of CPUID in CPL = 0. Bit 1: When set, the Intel TDX module injects #VE on guest TD execution of CPUID in CPL > 0. Other: Reserved, must be 0.	0xA120000000000000
TD VMCS	TD_VMCS	None	None	None	VMCS	1	2048	1	1		0x00200000FFFFFFFF
VAPIC	VAPIC	None	RO	None	Page	8	128	1	8	Virtual APIC Page	0x0120000300000000

Class	Field	VMM Prod. Access	VMM Debug Access	Guest Access	Type	Field Size (Bytes)	Num Fields	Num Elem.	Elem. Size (Bytes)	Description	Base FIELD_ID (Hex)
VE_INFO	EXIT_REASON	None	RO	None	0	4	1	1	4		0x0220000200000000
VE_INFO	VALID	None	RO	None	0	4	1	1	4	0xFFFFFFFF: valid 0x00000000: not valid	0x0220000200000001
VE_INFO	EXIT_QUALIFICATION	None	RO	None	0	8	1	1	8		0x0220000300000002
VE_INFO	GLA	None	RO	None	0	8	1	1	8		0x0220000300000003
VE_INFO	GPA	None	RO	None	0	8	1	1	8		0x0220000300000004
VE_INFO	EPTP_INDEX	None	RO	None	0	2	1	1	2		0x0220000100000005
VE_INFO	INSTRUCTION_LENGTH	None	RO	None	0	4	1	1	4		0x8220000200000010
VE_INFO	INSTRUCTION_INFORMATION	None	RO	None	0	4	1	1	4		0x8220000200000011
Guest GPR State	RAX	None	RW	None	0	8	1	1	8	0	0x1020000300000000
Guest GPR State	RCX	None	RW	None	0	8	1	1	8	Init value is provided as an input to TDH.VP.INIT (same value as R8)	0x1020000300000001
Guest GPR State	RDX	None	RW	None	0	8	1	1	8	Init Value: • Bits [31:00]: Same as RESET value, matches CPUID.1:EAX. CPU version information includes Family, Model and Stepping • Bits [63:32]: Set to 0	0x1020000300000002
Guest GPR State	RBX	None	RW	None	0	8	1	1	8	Init Value: • Bits [05:00]: GPAW is the effective GPA width (in bits) for this TD (do not confuse with MAXPA); SHARED bit is at GPA bit GPAW-1; only GPAW values 48 and 52 are possible • Bits [63:06]: Reserved for future additional details, set to 0, must be ignored by vBIOS	0x1020000300000003
Guest GPR State	RSP_PLACEHOLDER	None	None	None	0	8	1	1	8		0x1020000300000004
Guest GPR State	RBP	None	RW	None	0	8	1	1	8	Init Value: • Bits [31:00]: Virtual CPU index, starting from 0 and allocated sequentially on each successful TDH.VP.INIT • Bits [63:32]: Set to 0	0x1020000300000005
Guest GPR State	RSI	None	RW	None	0	8	1	1	8		0x1020000300000006
Guest GPR State	RD1	None	RW	None	0	8	1	1	8	Init value is provided as an input to TDH.VP.INIT (same value as RCX)	0x1020000300000007
Guest GPR State	R8	None	RW	None	0	8	1	1	8		0x1020000300000008
Guest GPR State	R9	None	RW	None	0	8	1	1	8		0x1020000300000009
Guest GPR State	R10	None	RW	None	0	8	1	1	8		0x102000030000000A
Guest GPR State	R11	None	RW	None	0	8	1	1	8		0x102000030000000B
Guest GPR State	R12	None	RW	None	0	8	1	1	8		0x102000030000000C
Guest GPR State	R13	None	RW	None	0	8	1	1	8		0x102000030000000D
Guest GPR State	R14	None	RW	None	0	8	1	1	8		0x102000030000000E
Guest GPR State	R15	None	RW	None	0	8	1	1	8		0x102000030000000F
Guest State	DR0	None	RW	None	0	8	1	1	8		0x1120000300000000
Guest State	DR1	None	RW	None	0	8	1	1	8		0x1120000300000001
Guest State	DR2	None	RW	None	0	8	1	1	8		0x1120000300000002
Guest State	DR3	None	RW	None	0	8	1	1	8		0x1120000300000003
Guest State	DR6	None	RW	None	0	8	1	1	8		0x1120000300000006
Guest State	XCR0	None	RO	None	0	8	1	1	8		0x1120000300000020
Guest State	CR2	None	RW	None	0	8	1	1	8	0	0x1120000300000028
Guest State	VCPU_STATE_DETAILS	ROS	ROS	None	0	8	1	1	8	Bit 0: INTR_PENDING: Indicates that a virtual interrupt is pending delivery, i.e. VMCS.RVI[7:4] > TDVPS.VAPIC.VPPR[7:4] Bits 63:2: Reserved, set to 0	0x9120000300000100
Guest MSR State	IA32_SPEC_CTRL	None	RW	None	0	8	1	1	8		0x1320000300000048
Guest MSR State	IA32_UMWAIT_CONTROL	None	RW	None	0	8	1	1	8		0x13200003000000E1
Guest MSR State	IA32_PERFVTSELx	None	RW	None	0	8	8	1	8		0x1320000300000186
Guest MSR State	MSR_OFFCORE_RSPx	None	RW	None	0	8	2	1	8		0x13200003000001A6

Class	Field	VMM Prod. Access	VMM Debug Access	Guest Access	Type	Field Size (Bytes)	Num Fields	Num Elem.	Elem. Size (Bytes)	Description	Base FIELD_ID (Hex)
Guest MSR State	IA32_XFD	None	RO	None	0	8	1	1	8		0x13200003000001C4
Guest MSR State	IA32_XFD_ERR	None	RO	None	0	8	1	1	8		0x13200003000001C5
Guest MSR State	IA32_FIXED_CTRx	None	RW	None	0	8	4	1	8		0x1320000300000309
Guest MSR State	IA32_PERF_METRICS	None	RW	None	0	8	1	1	8		0x1320000300000329
Guest MSR State	IA32_FIXED_CTR_CTRL	None	RW	None	0	8	1	1	8		0x132000030000038D
Guest MSR State	IA32_PERF_GLOBAL_STATUS	None	RO	None	0	8	1	1	8		0x132000030000038E
Guest MSR State	IA32_PEBs_ENABLE	None	RW	None	0	8	1	1	8		0x13200003000003F1
Guest MSR State	MSR_PEBs_DATA_CFG	None	RW	None	0	8	1	1	8		0x13200003000003F2
Guest MSR State	MSR_PEBs_LD_LAT	None	RW	None	0	8	1	1	8		0x13200003000003F6
Guest MSR State	MSR_PEBs_FRONTEND	None	RW	None	0	8	1	1	8		0x13200003000003F7
Guest MSR State	IA32_A_PMCx	None	RW	None	0	8	8	1	8		0x13200003000004C1
Guest MSR State	IA32_DS_AREA	None	RW	None	0	8	1	1	8		0x1320000300000600
Guest MSR State	IA32_XSS	None	RO	None	0	8	1	1	8		0x1320000300000DA0
Guest MSR State	IA32_LBR_DEPTH	None	RW	None	0	8	1	1	8		0x13200003000014CF
Guest MSR State	IA32_STAR	None	RO	None	0	8	1	1	8		0x1320000300002081
Guest MSR State	IA32_LSTAR	None	RO	None	0	8	1	1	8		0x1320000300002082
Guest MSR State	IA32_FMASK	None	RO	None	0	8	1	1	8		0x1320000300002084
Guest MSR State	IA32_KERNEL_GS_BASE	None	RO	None	0	8	1	1	8		0x1320000300002102
Guest MSR State	IA32_TSC_AUX	None	RW	None	0	8	1	1	8		0x1320000300002103
Guest Ext. State	XBUFF	None	RW	None	XSAVES buffer	8	1536	1	8		0x1220000300000000
VMCS[1]	L2_VMCS_1	None	None	None	VMCS	1	2048	1	1	VMCS for controlling the VCPU operation in L2 VM	0x2420000000000000
MSR Bitmaps[1]	L2_MSR_BITMAPS_1	None	RWS	RWS	MSR Exit Bitmaps	8	512	1	8	MSR exit bitmaps page, controlling L2 VM RDMSR/WRMSR VM exit	0x2520000300000000
MSR Bitmaps Shadow[1]	L2_SHADOW_MSR_BITMAPS_1	None	ROS	None	MSR Exit Bitmaps	8	512	1	8	Shadow MSR exit bitmaps page, defining the L2 VM policy for handling MSR access, set by the L1 VMM	0xA620000300000000
VMCS[2]	L2_VMCS_2	None	None	None	VMCS	1	2048	1	1	VMCS for controlling the VCPU operation in L2 VM	0x2C20000000000000
MSR Bitmaps[2]	L2_MSR_BITMAPS_2	None	RWS	RWS	MSR Exit Bitmaps	8	512	1	8	MSR exit bitmaps page, controlling L2 VM RDMSR/WRMSR VM exit	0x2D20000300000000
MSR Bitmaps Shadow[2]	L2_SHADOW_MSR_BITMAPS_2	None	ROS	None	MSR Exit Bitmaps	8	512	1	8	Shadow MSR exit bitmaps page, defining the L2 VM policy for handling MSR access, set by the L1 VMM	0xAE20000300000000
VMCS[3]	L2_VMCS_3	None	None	None	VMCS	1	2048	1	1	VMCS for controlling the VCPU operation in L2 VM	0x3420000000000000
MSR Bitmaps[3]	L2_MSR_BITMAPS_3	None	RWS	RWS	MSR Exit Bitmaps	8	512	1	8	MSR exit bitmaps page, controlling L2 VM RDMSR/WRMSR VM exit	0x3520000300000000
MSR Bitmaps Shadow[3]	L2_SHADOW_MSR_BITMAPS_3	None	ROS	None	MSR Exit Bitmaps	8	512	1	8	Shadow MSR exit bitmaps page, defining the L2 VM policy for handling MSR access, set by the L1 VMM	0xB620000300000000

#### 4.2.4. TD VMCS

##### Intel SDM, 24 Virtual Machine Control Structures

**Note:** This section describes TD VMCS usage, as defined. Implementation may differ.

- TD VMCS is a VMX format VMCS (with TDX ISA extensions) that is stored as part of TDVPS.

## 4.2.4.1. TD VMCS Guest State Area

## 4.2.4.1.1. TD VMCS Guest Register State Area

Intel SDM, Vol. 3, 9.1.1 Processor State after Reset  
 Intel SDM, Vol. 3, 24.4.1 Guest Register State

Table 4.9: TD VMCS Guest Register State Area Fields

Field	VMM Access		Init Value (after TDH.VP.INIT)
	Prod.	Debug	
Guest CR0	None	RWS	0x0021 • Bits PE (0) and NE (5) are set to 1. • All other bits are cleared to 0. The initial value is checked for compatibility with fixed-0 and fixed-1 bits according to IA32_VMX_CR0_FIXED* MSRs, except for PG (bit 31) which is allowed to be 0 since the guest TD runs as an unrestricted guest.
Guest CR3	None	RW	0
Guest CR4	None	RWS	0x2040 • Bits MCE (6) and VMXE (13) are set to 1 • All other bits are cleared to 0. The initial value is checked for compatibility with fixed-0 and fixed-1 bits according to IA32_VMX_CR4_FIXED* MSRs.
Guest DR7	None	RW	0x00000400
Guest RSP	None	RW	0
Guest RIP	None	RW	0xFFFFFFFF
Guest RFLAGS	None	RW	0x00000002
Guest ES selector	None	RW	0
Guest CS selector	None	RW	0
Guest SS selector	None	RW	0
Guest DS selector	None	RW	0
Guest FS selector	None	RW	0
Guest GS selector	None	RW	0
Guest LDTR selector	None	RW	0
Guest TR selector	None	RW	0
Guest ES base	None	RW	0
Guest CS base	None	RW	0
Guest SS base	None	RW	0
Guest DS base	None	RW	0
Guest FS base	None	RW	0
Guest GS base	None	RW	0
Guest LDTR base	None	RW	0
Guest TR base	None	RW	0
Guest GDTR base	None	RW	0
Guest IDTR base	None	RW	0
Guest ES limit	None	RW	0xFFFFFFFF
Guest CS limit	None	RW	0xFFFFFFFF
Guest SS limit	None	RW	0xFFFFFFFF
Guest DS limit	None	RW	0xFFFFFFFF
Guest FS limit	None	RW	0xFFFFFFFF
Guest GS limit	None	RW	0xFFFFFFFF
Guest LDTR limit	None	RW	0x0000FFFF
Guest TR limit	None	RW	0x0000FFFF
Guest GDTR limit	None	RW	0x0000FFFF
Guest IDTR limit	None	RW	0
Guest ES access rights	None	RW	0x0000C093 (Data, RW, Accessed, DPL=0, Present, 32b, 4KB granularity)
Guest CS access rights	None	RW	0x0000C09B (Code, RX, Accessed, DPL=0, Present, 32b)
Guest SS access rights	None	RW	0x0000C093 (Data, RW, Accessed, DPL=0, Present, 32b, 4KB granularity)
Guest DS access rights	None	RW	0x0000C093 (Data, RW, Accessed, DPL=0, Present, 32b, 4KB granularity)



Field	VMM Access		Init Value (after TDH.VP.INIT)
	Prod.	Debug	
Guest FS access rights	None	RW	0x0000C093 (Data, RW, Accessed, DPL=0, Present, 32b, 4KB granularity)
Guest GS access rights	None	RW	0x0000C093 (Data, RW, Accessed, DPL=0, Present, 32b, 4KB granularity)
Guest LDTR access rights	None	RW	0x00010082 (LDT, Present, 32b, 1B granularity, Unusable)
Guest TR access rights	None	RW	0x0000008B (32b TSS, Busy, Present, 32b, 1B granularity)
Guest SMBASE	None	None	0

#### 4.2.4.1.2. TD VMCS Guest MSRs

See also the MSR virtualization tables in 2.1.

Table 4.10: TD VMCS Guest MSRs

Field	VMM Access		Init Value (after TDH.VP.INIT)
	Prod.	Debug	
IA32_DEBUGCTL	None	RWS	0
IA32_SYSENTER_CS	None	RW	0
IA32_SYSENTER_ESP	None	RW	0
IA32_SYSENTER_EIP	None	RW	0
IA32_PERF_GLOBAL_CTRL	None	RW	0x000000FF • EN_PMCx (bits 0 to (NUM_PMC - 1)) are set to 1. • All other bits are cleared to 0.
IA32_PAT	None	RW	0x0007040600070406
IA32_EFER	None	RW	0x901 • SCE (bit 0) is set to 1. • LME (bit 8) is set to 1. • NXE (bit 11) is set to 1. • All other bits are cleared to 0.
GUEST_IA32_S_CET	None	RW	0
GUEST_SSP	None	RW	0
GUEST_IA32_INTERRUPT_SSP_TABLE_ADDR	None	RW	0
IA32_RTIT_CTL	None	RW	0
IA32_LBR_CTL	None	RW	0
IA32_BNDCFGS	None	None	The TDX module must not write this field
IA32_GUEST_PKRS	None	RW	0

5

#### 4.2.4.1.3. TD VMCS Guest Non-Register State Area

Intel SDM, 24.4.2

Guest Non-Register State

Table 4.11: TD VMCS Guest Non-Register State Area Fields

Field Name	VMM Access		Description	Initial State
	Prod.	Debug		
Activity State	None	RO	Saved/restored on VM exit/entry	Active (0)
Interruptibility State	None	RW	Saved/restored on VM exit/entry	0
Pending Debug Exceptions	None	RW	Saved/restored on VM exit/entry	0
VMCS Link Pointer	None	None		NULL_PA (-1)
VMX-Preemption Timer Value	None	RW	N/A: VMX-preemption timer is not used by guest TDs.	0
PDPTEn	None	RO	N/A: PAE paging is not used by TD guests.	NULL_PA (-1)



Field Name	VMM Access		Description	Initial State
	Prod.	Debug		
Guest Interrupt Status	None	RW	Includes RVI (lower byte) and SVI (upper byte): saved/restored on VM exit/entry	0
PML Index	None	RW		0
Guest UINV	None	RW		0

#### 4.2.4.2. TD VMCS Host State Area

Intel SDM, 24.5 Host-State Area

The host state area is not intended to be accessible outside the Intel TDX module.

#### 5 4.2.4.3. TD VMCS VM-Execution Control Fields

Intel SDM, 24.6 VM-Execution Control Fields

##### 4.2.4.3.1. TD VMCS Pin-Based VM-Execution Controls

Table 4.12: TD VMCS Pin-Based VM-Execution Controls

Bit	Name	VMM Access		Description	Init Value
		Prod.	Debug		
0	External-interrupt exiting	None	RO	The Intel TDX module performs TD Exit	1
1	Reserved	None	RO		MSR
2	Reserved	None	RO		MSR
3	NMI exiting	None	RO	The Intel TDX module performs TD Exit	1
4	Reserved	None	RO		MSR
5	Virtual NMIs	None	RO		1
6	Activate VMX-preemption timer	None	RO		0
7	Process posted interrupts	RWS	RWS	Set to 1 by TDH.VP.WR only if a valid posted interrupt descriptor and a valid posted interrupt notification vector are set.	0
8	Reserved	None	RO		MSR
9	Reserved	None	RO		MSR
10	Reserved	None	RO		MSR
11	Reserved	None	RO		MSR
12	Reserved	None	RO		MSR
13	Reserved	None	RO		MSR
14	Reserved	None	RO		MSR
15	Reserved	None	RO		MSR
16	Reserved	None	RO		MSR
17	Reserved	None	RO		MSR
18	Reserved	None	RO		MSR
19	Reserved	None	RO		MSR
20	Reserved	None	RO		MSR
21	Reserved	None	RO		MSR
22	Reserved	None	RO		MSR
23	Reserved	None	RO		MSR
24	Reserved	None	RO		MSR
25	Reserved	None	RO		MSR
26	Reserved	None	RO		MSR
27	Reserved	None	RO		MSR
28	Reserved	None	RO		MSR
29	Reserved	None	RO		MSR
30	Reserved	None	RO		MSR
31	Reserved	None	RO		MSR

10 Reserved bits are set based on IA32\_VMX\_TRUE\_PINBASED\_CTLMSR.

## 4.2.4.3.2. TD VMCS Processor-Based VM-Execution Controls

Table 4.13: TD VMCS Primary Processor-Based VM-Execution Controls

Bit	Name	VMM Access		Description	Init Value
		Prod.	Debug		
0	Reserved	None	RO		MSR
1	Reserved	None	RO		MSR
2	Interrupt-window exiting	None	RW		0
3	Use TSC offsetting	None	RO		1
4	Reserved	None	RO		MSR
5	Reserved	None	RO		MSR
6	Reserved	None	RO		MSR
7	HLT exiting	None	RO	The Intel TDX module injects a #VE into the guest TD	1
8	Reserved	None	RO		MSR
9	INVLPG exiting	None	RW		0
10	MWAIT exiting	None	RO	The Intel TDX module injects a #VE into the guest TD	1
11	RDPMS exiting	None	RW		~TDCS.ATTRIBUTES.PERFMON
12	RDTS exiting	None	RW		0
13	Reserved	None	RO		MSR
14	Reserved	None	RO		MSR
15	CR3-load exiting	None	RW		0
16	CR3-store exiting	None	RW		0
17	Activate tertiary controls	None	RO		1
18	Reserved	None	RO		MSR
19	CR8-load exiting	None	RW		0
20	CR8-store exiting	None	RW		0
21	Use TPR shadow	None	RO		1
22	NMI-window exiting	None	RO	Set by the Intel TDX module before entering the guest TD – based on TDVPS.PEND_NMI	0
23	MOV-DR exiting	None	RW		0
24	Unconditional I/O exiting	None	RW		1
25	Use I/O bitmaps	None	RO		0
26	Reserved	None	RO		MSR
27	Monitor trap flag	None	RW		0
28	Use MSR bitmaps	None	RO		1
29	MONITOR exiting	None	RW		1
30	PAUSE exiting	None	RW		0
31	Activate secondary controls	None	RO		1

Reserved bits are set based on IA32\_VMX\_TRUE\_PROCBASED\_CTLMSR.

5

Table 4.14: TD VMCS Secondary Processor-Based VM-Execution Controls

Bit	Name	VMM Access		Description	Init Value
		Prod.	Debug		
0	Virtualize APIC accesses	None	RO		0
1	Enable EPT	None	RO		1
2	Descriptor-table exiting	None	RW		0
3	Enable RDTSCP	None	RO		1
4	Virtualize x2APIC mode	None	RO		1
5	Enable VPID	None	RO		1
6	WBINVD exiting	None	RO		1
7	Unrestricted guest	None	RO		1
8	APIC-register virtualization	None	RO		1
9	Virtual-interrupt delivery	None	RO		1
10	PAUSE-loop exiting	None	RW		0
11	RDRAND exiting	None	RW		0
12	Enable INVPCID	None	RO		1
13	Enable VM functions	None	RO		1
14	VMCS shadowing	None	RO		0
15	Enable ENCLS exiting	None	RO		1

Bit	Name	VMM Access		Description	Init Value
		Prod.	Debug		
16	RDSEED exiting	None	RW		0
17	Enable PML	None	RWS	Can be set to 1 only if: - ATTRIBUTES.DEBUG is 1 - PML address is a valid shared physical address	0
18	EPT-violation #VE	None	RO		1
19	Conceal VMX from PT	None	RO		1
20	Enable XSAVES/XRSTORS	None	RW		1
21	PASID translation	None	RO		1
22	Mode-based execute control for EPT	None	RO		0
23	Enable SPP	None	RO		0
24	PT uses guest physical addresses (PT2GPA)	None	RO		1
25	Use TSC scaling	None	RO		1
26	Enable user-level wait and pause	None	RO		Set to the value of virtualized CPUID(0x7,0x0).ECX[5]
27	Enable PCONFIG	None	RO		Set to the value of virtualized CPUID(0x7,0x0).EDX[18]
28	Enable ENCLV exiting	None	RO		1
29	Enable EPC Virtualization Extensions	None	RO		0
30	Bus-lock detection	RW	RW	If enabled by the host VMM (using TDH.VP.WR), then the Intel TDX module performs TD Exit on VM exit.	0
31	Notification exiting	RW	RW	If enabled by the host VMM (using TDH.VP.WR), then the Intel TDX module performs TD Exit on VM exit.	0

Table 4.15: TD VMCS Tertiary Processor-Based VM-Execution Controls

Bit	Name	VMM Access		Description	Init Value (after TDH.VP.INIT)
		Prod.	Debug		
0	LOADIWKEY exiting	None	RW		0
1	Enable HLAT	None	RO		0
2	EPT paging-write control	None	RO		0
3	Guest-paging verification	None	RO		0
4	IPI virtualization	None	RO		0
5	GPAW	None	RO	0: GPA.SHARED bit is GPA[47] 1: GPA.SHARED bit is GPA[51]	Copied from TDCS.GPAW
6	Reserved	None	RO		MSR
7	Reserved	None	RO		MSR
8	Reserved	None	RO		MSR
9	Reserved	None	RO		MSR
10	Reserved	None	RO		MSR
11	Reserved	None	RO		MSR
12	Reserved	None	RO		MSR
13	Reserved	None	RO		MSR
14	Reserved	None	RO		MSR
15	Reserved	None	RO		MSR
16	Reserved	None	RO		MSR
17	Reserved	None	RO		MSR
18	Reserved	None	RO		MSR
19	Reserved	None	RO		MSR
20	Reserved	None	RO		MSR
21	Reserved	None	RO		MSR
22	Reserved	None	RO		MSR
23	Reserved	None	RO		MSR
24	Reserved	None	RO		MSR
25	Reserved	None	RO		MSR

Bit	Name	VMM Access		Description	Init Value (after TDH.VP.INIT)
		Prod.	Debug		
26	Reserved	None	RO		MSR
27	Reserved	None	RO		MSR
28	Reserved	None	RO		MSR
29	Reserved	None	RO		MSR
30	Reserved	None	RO		MSR
31	Reserved	None	RO		MSR
32	Reserved	None	RO		MSR
33	Reserved	None	RO		MSR
34	Reserved	None	RO		MSR
35	Reserved	None	RO		MSR
36	Reserved	None	RO		MSR
37	Reserved	None	RO		MSR
38	Reserved	None	RO		MSR
39	Reserved	None	RO		MSR
40	Reserved	None	RO		MSR
41	Reserved	None	RO		MSR
42	Reserved	None	RO		MSR
43	Reserved	None	RO		MSR
44	Reserved	None	RO		MSR
45	Reserved	None	RO		MSR
46	Reserved	None	RO		MSR
47	Reserved	None	RO		MSR
48	Reserved	None	RO		MSR
49	Reserved	None	RO		MSR
50	Reserved	None	RO		MSR
51	Reserved	None	RO		MSR
52	Reserved	None	RO		MSR
53	Reserved	None	RO		MSR
54	Reserved	None	RO		MSR
55	Reserved	None	RO		MSR
56	Reserved	None	RO		MSR
57	Reserved	None	RO		MSR
58	Reserved	None	RO		MSR
59	Reserved	None	RO		MSR
60	Reserved	None	RO		MSR
61	Reserved	None	RO		MSR
62	Reserved	None	RO		MSR
63	Reserved	None	RO		MSR

Reserved bits are set based on IA32\_VMX\_PROCBASED\_CTL3 MSR.

#### 4.2.4.3.3. TD VMCS Controls for APIC Virtualization

Table 4.16: TD VMCS Controls for APIC Virtualization

Field Name	VMM Access		Description	Initial Value
	Prod.	Debug		
APIC-access address	None	RO		NULL_PA (-1)
Virtual-APIC address	None	None	On VCPU-to-LP association, set by the Intel TDX module to the address of the VAPIC page in TDVPS, including the TD's ephemeral HKID	Address of the VAPIC page in TDVPS, including the TD's ephemeral HKID
TPR threshold	None	RO		0
EOI-exit bitmap n	None	RO		0
Posted-interrupt notification vector	RWS	RWS	TDH.VP.WR checks the value to be in the range 0 to 255. See process posted interrupt pin-based execution control.	0xFFFF

Field Name	VMM Access		Description	Initial Value
	Prod.	Debug		
Posted-interrupt descriptor address	RWS	RWS	Address must be: <ul style="list-style-type: none"> <li>Valid shared physical address (HKID bits encode a shared HKID).</li> <li>Aligned on 64B.</li> </ul> See process posted interrupt pin-based execution control.	NULL_PA (-1)

#### 4.2.4.3.4. EPTP and Shared EPTP

Table 4.17: EPTP (Copied from TDCS.EPTP on TDH.VP.INIT)

Bits	Field Name	VMM Access		Description	Initial Value
		Prod.	Debug		
2:0	EPT Memory Type	RO	RO	Set to WB	6
5:3	EPT Level	RO	RO	1 less than the EPT page-walk length	Copied from TDCS.EPTP
6	Enable A/D Bits	RO	RO		Copied from TDCS.EPTP
7	Enable supervisor shadow stack control	RO	RO		0
11:8	Reserved	RO	RO		0
51:12	EPML5/4 PA	RO	RO		
63:52	Reserved	RO	RO		0

Table 4.18: Shared EPTP

Bits	Field Name	VMM Access		Description	Initial Value
		Prod.	Debug		
11:0	Reserved	None	RO		0
51:12	EPML5/4 PA	RWS	RWS		
63:52	Reserved	None	RO		0

## 4.2.4.3.5. CR-Related TD VMCS VM-Execution Control Fields

Table 4.19: CR-Related VMCS VM-Execution Control Fields

Field Name	VMM Access		Description	Initial Value
	Prod.	Debug		
CR0 Guest/Host Mask	None	RW	Bits 0, 5, 29 and 30 can't be written even in debug mode	<p>The following bits are set to 1, indicating they are owned by the Intel TDX module:</p> <ul style="list-style-type: none"> <li>• PE (0)</li> <li>• NE (5)</li> <li>• NW (29)</li> <li>• CD (30)</li> <li>• Any bit set to 1 in IA32_VMX_CR0_FIXED0 (i.e., a bit whose value must be 1), except for PG(31) which is set to 0, since the guest TD runs as an unrestricted guest</li> <li>• Any bit set to 0 in IA32_VMX_CR0_FIXED1 (i.e., a bit whose value must be 0)</li> <li>• Bits known to the Intel TDX module as reserved (bits 63-32, 28-19, 17 and 15-6)</li> </ul> <p>All other bits are cleared to 0, indicating they are owned by the guest TD.</p>
CR0 Read Shadow	None	RW	Bits 0 and 5 can't be written even in debug mode	<p>The following bits are set to 1:</p> <ul style="list-style-type: none"> <li>• PE (0)</li> <li>• NE (5)</li> <li>• Any bit set to 1 in IA32_VMX_CR0_FIXED0 (i.e., a bit whose value must be 1)</li> </ul> <p>All other bits are cleared to 0.</p>
CR4 Guest/Host Mask	None	RW	Bits 6, 13 and 14 can't be written even in debug mode	<ul style="list-style-type: none"> <li>• Bits MCE (6), VMXE (13) and SMXE (14) are set to 1, indicating they are owned by the Intel TDX module.</li> <li>• Bit PKE (22) is set to <math>\sim</math>TDCS.XFAM[9] to intercept writes to CR4 if PK is not enabled.</li> <li>• If TDCS.XFAM[12:11] is 11, then bit CET (23) is cleared to 0. Otherwise (CET is not enabled), bit CET (23) is set to 1 to intercept writes to CR4.</li> <li>• Bit UINTR (25) is set <math>\sim</math>TDCS.XFAM[14] to intercept writes to CR4 if ULI is not enabled.</li> <li>• Bit KL (19) is set to <math>\sim</math>TDCS.ATTRIBUTES.KL to intercept writes to CR4 if KeyLocker is not enabled.</li> <li>• Bit PKS (24) is set to <math>\sim</math>TDCS.ATTRIBUTES.PKS to intercept writes to CR4 if PKS is not enabled.</li> <li>• Any bit set to 1 in IA32_VMX_CR4_FIXED0 (i.e., a bit whose value must be 1) is set to 1.</li> <li>• Any bit set to 0 in IA32_VMX_CR4_FIXED1 (i.e., a bit whose value must be 0) is set to 1.</li> <li>• Bits known to the Intel TDX module as reserved (bits 63-26 and bit 15) are set to 1.</li> <li>• All other bits are cleared to 0.</li> </ul>
CR4 Read Shadow	None	RW	Bit 6 can't be written even in debug mode	<ul style="list-style-type: none"> <li>• Bit MCE (6) is set to 1.</li> <li>• Bit VMXE (13) is cleared to 0.</li> <li>• Any other bit whose value is set to 1 in IA32_VMX_CR4_FIXED0 (i.e., a bit whose value must be 1) is set to 1.</li> <li>• All other bits are cleared to 0.</li> </ul>
CR3-Target Values	None	RW	N/A: The Intel TDX module does not control guest CR3	N/A
CR3-Target Count	None	RW	Set to 0: Intel TDX module does not control guest CR3	0

## 4.2.4.3.6. Other TD VMCS VM-Execution Control Fields

Table 4.20: Other TD VMCS VM-Execution Control Fields

Field Name	VMM Access		Description	Initial Value
	Prod.	Debug		
Exception Bitmap	None	RW	<ul style="list-style-type: none"> <li>Bit 18 (MCE) is set to 1, even in debug mode.</li> <li>Other bits are cleared to 0. They may be modified in debug mode.</li> </ul>	0x00040000
Page-fault error-code mask	None	RW		0
Page-fault error-code match	None	RW		0
I/O-Bitmap Address n	None	RO	Set to NULL_PA (-1): I/O bitmaps execution control is set to 0	NULL_PA (-1)
Time-Stamp Counter Offset	RO	RW		Copied from TDCS.TSC_OFFSET
Time-Stamp Counter Multiplier	RO	RW		Copied from TDCS.TSC_MULTIPLIER
MSR-Bitmap Address	RO	RO		
Executive-VMCS Pointer	None	None	N/A	NULL_PA (-1)
TD HKID	RO	RO		
VPID	None	RO	Unique identifier of the VM in the platform: <b>Bits 1:0:</b> VM index (0) <b>Bits 15:2:</b> TD's HKID	<b>Bits 1:0:</b> VM index (0) <b>Bits 15:2:</b> TD's HKID
PLE_GAP	RO	RW		0
PLE_Window	RO	RW		0
VM-Function Controls	RO	RO	The Intel TDX module injects a #UD into the TD.	0
EPTP-list address	RO	RO	VMFUNC is not supported.	NULL_PA (-1)
VMREAD-bitmap address	None	RO	VMCS shadowing is not supported.	NULL_PA (-1)
VMWRITE-bitmap address	None	RO	VMCS shadowing is not supported.	NULL_PA (-1)
ENCLS-Exiting Bitmap	None	RO	Set to all 1's – the Intel TDX module injects a #UD into the guest TD.	All 1s
ENCLV-Exiting Bitmap	None	RO	Set to all 1's – the Intel TDX module injects a #UD into the guest TD.	All 1s
PML address	RO	RW	Address must be: <ul style="list-style-type: none"> <li>Valid shared physical address (HKID bits encode a shared HKID).</li> <li>Aligned on 4KB.</li> </ul> See enable PML execution control.	NULL_PA (-1)
Virtualization-exception information address	None	RO		
EPTP index	None	RO		0
XSS-Exiting Bitmap	None	RW		0

Field Name	VMM Access		Description	Initial Value
	Prod.	Debug		
low PASID directory address	None	RO		
high PASID directory address	None	RO		
notify window	RWS	RWS		0
PCONFIG-Exiting Bitmap	None	RO		-1
HLAT pointer	None	RO		0
HLAT prefix size	None	RO		0

#### 4.2.4.4. TD VMCS VM-Exit Control Fields

Intel SDM, 24.7 VM-Exit Control Fields

Table 4.21: TD VMCS VM-Exit Controls

Bit	Name	VMM Access		Description	Init Value
		Prod.	Debug		
0	Reserved	None	RO		MSR
1	Reserved	None	RO		MSR
2	Save debug controls	None	RO		1
3	Reserved	None	RO		MSR
4	Reserved	None	RO		MSR
5	Reserved	None	RO		MSR
6	Reserved	None	RO		MSR
7	Reserved	None	RO		MSR
8	Reserved	None	RO		MSR
9	Host address-space size	None	RO		1
10	Reserved	None	RO		MSR
11	Reserved	None	RO		MSR
12	Load IA32_PERF_GLOBAL_CTRL	None	RO		= (TDCS.ATTRIBUTES.PERFMON   TDCS.ATTRIBUTES.DEBUG)
13	Reserved	None	RO		MSR
14	Reserved	None	RO		MSR
15	Acknowledge interrupt on exit	None	RO		1
16	Reserved	None	RO		MSR
17	Reserved	None	RO		MSR
18	Save IA32_PAT	None	RO		1
19	Load IA32_PAT	None	RO		1
20	Save IA32_EFER	None	RO		1
21	Load IA32_EFER	None	RO		1
22	Save VMX-preemption time value	None	RO		0
23	Clear IA32_BNDCFGS	None	RO	Deprecated	0
24	Conceal VMX from PT	None	RO		1
25	Clear IA32_RTIT_CTL	None	RO		1
26	Clear IA32_LBR_CTL	None	RO		1
27	Clear UINV	None	RO		1
28	Load host CET state	None	RO		1
29	Load host PKRS	None	RO		0
30	Save IA32_PERF_GLOBAL_CTRL	None	RO		= (TDCS.ATTRIBUTES.PERFMON   TDCS.ATTRIBUTES.DEBUG)
31	Activate secondary controls	None	RO		0

5

Reserved bits are set based on IA32\_VMX\_TRUE\_EXIT\_CTLs MSR.



Table 4.22: TD VMCS VM-Exit Controls for MSRs

Field Name	VMM Access		Description	Initial Value
	Prod.	Debug		
VM-exit MSR-store count	None	RO	Not used	0
VM-exit MSR-store address	None	RO	Not used	NULL_PA (-1)
VM-exit MSR-load count	None	RO	Not used	0
VM-exit MSR-load address	None	RO	Not used	NULL_PA (-1)

#### 4.2.4.5. TD VMCS VM-Entry Control Fields

Intel SDM, 24.8 VM-Entry Control Fields

5 Table 4.23: TD VMCS VM-Entry Controls

Bit	Name	VMM Access		Description	Init Value (after TDH.VP.INIT)
		Prod.	Debug		
0	Reserved	None	RO		MSR
1	Reserved	None	RO		MSR
2	Load debug controls	None	RO		1
3	Reserved	None	RO		MSR
4	Reserved	None	RO		MSR
5	Reserved	None	RO		MSR
6	Reserved	None	RO		MSR
7	Reserved	None	RO		MSR
8	Reserved	None	RO		MSR
9	IA-32e mode guest	None	RO	Written by the CPU on VM exit	0
10	Entry to SMM	None	RO		0
11	Deactivate dual-monitor treatment	None	RO		0
12	Reserved	None	RO		MSR
13	Load IA32_PERF_GLOBAL_CTRL	None	RO		= (TDCS.ATTRIBUTES.PERFMON   TDCS.ATTRIBUTES.DEBUG)
14	Load IA32_PAT	None	RO		1
15	Load IA32_EFER	None	RO		1
16	Load IA32_BNDCFGS	None	RO		0
17	Conceal VMX from PT	None	RO		1
18	Load IA32_RTIT_CTL	None	RO		1
19	Load UINV	None	RO		1
20	Load CET state	None	RO		1
21	Load IA32_LBR_CTL	None	RO		1
22	Load guest PKRS	None	RO		= (TDCS.ATTRIBUTES.PKRS   TDCS.ATTRIBUTES.DEBUG)
23	Reserved	None	RO		MSR
24	Reserved	None	RO		MSR
25	Reserved	None	RO		MSR
26	Reserved	None	RO		MSR
27	Reserved	None	RO		MSR
28	Reserved	None	RO		MSR
29	Reserved	None	RO		MSR
30	Reserved	None	RO		MSR
31	Reserved	None	RO		MSR

Reserved bits are set based on IA32\_VMX\_ENTRY\_CTLs MSR.

**Table 4.24: TD VMCS VM-Entry Controls for MSRs**

Field Name	VMM Access		Description	Initial Value
	Prod.	Debug		
VM-entry MSR-load count	None	RO	Not used	0
VM-entry MSR-load address	None	RO	Not used	NULL_PA (-1)

**Table 4.25: TD VMCS VM-Entry Controls for Event Injection**

Field Name	VMM Access		Description	Initial Value
	Prod.	Debug		
VM-entry interruption information	None	RO		
VM-entry exception error code	None	RO		
VM-entry instruction length	None	RO		

#### 5 4.2.4.6. TD VMCS VM-Exit Information Fields

Intel SDM, 24.9

VM-Exit Information Fields

**Table 4.26: TD VMCS Basic VM-Exit Information**

Field Name	VMM Access		Description	Initial Value
	Prod.	Debug		
Exit reason	None	RO	If the Intel TDX module decides to perform a TD exit, it returns this in RAX bits 31:0. Bit 27 (enclave mode) is not set. Bit 28 (Pending MTF VM exit) is not set. Bit 29 (VM exit from VMX root operation) is not set. Bit 31 (VM-entry failure) is not set.	N/A
Exit qualification	None	RO	If the Intel TDX module decides to perform a TD exit, it returns this in RCX. If the exit is due to EPT violation, bits 12-7 of the exit qualification are cleared to 0.	N/A
Guest-Linear Address	None	RO		N/A
Guest-physical Address	None	RO	If the Intel TDX module decides to perform a TD exit, it returns this in R8. If the EPT fault was caused by an access attempt to a private page, the Intel TDX module clears bits 11:0 to 0.	N/A

**Table 4.27: TD VMCS Information for VM Exits Due to Vectored Events**

Field Name	VMM Access		Description	Initial Value
	Prod.	Debug		
VM-exit interruption information	None	RO	On asynchronous TD exit, the Intel TDX module returns this in R9. Bits 63:32 are cleared to 0.	N/A
VM-exit interruption error code	None	RO		N/A

**Table 4.28: TD VMCS Information for VM Exits That Occur During Event Delivery**

Field Name	VMM Access		Description	Initial Value
	Prod.	Debug		
IDT-vectoring information	None	RO		
IDT-vectoring error code	None	RO		

**Table 4.29: TD VMCS Information for VM Exits Due to Instruction Execution**

Field Name	VMM Access		Description	Initial Value
	Prod.	Debug		
VM-exit instruction length	None	RO		
VM-exit instruction information	None	RO		
I/O RCX	None	RO		N/A
I/O RSI	None	RO		N/A
I/O RDI	None	RO		N/A
I/O RIP	None	RO		N/A

**Table 4.30: TD VMCS VM-Instruction Error Field**

Field Name	VMM Access		Description	Initial Value
	Prod.	Debug		
VM-instruction error	None	RO		N/A

**4.2.5. NEW: L2 VMCS**

Intel SDM, 24 Virtual Machine Control Structures

**Note:** This section describes L2 VMCS usage, as defined. Implementation may differ.

L2 VMCS is a VMX format VMCS (with TDX ISA extensions) that is stored as part of TDVPS.

**4.2.5.1. L2 VMCS Guest State Area****4.2.5.1.1. L2 VMCS Guest Register State Area**

Intel SDM, Vol. 3, 9.1.1 Processor State after Reset  
Intel SDM, Vol. 3, 24.4.1 Guest Register State

**Table 4.31: L2 VMCS Guest Register State Area Fields**

Field	Host VMM Access		L1 VMM Access	Init Value (after TDH.VP.INIT)
	Prod.	Debug		
Guest CR0	None	RWS	RWS	0x0000000000000021
Guest CR3	None	RW	RW	0
Guest CR4	None	RWS	RWS	0x0000000000002040
Guest DR7	None	RW	RW	0x00000400
Guest RSP	None	RW	RW	0
Guest RIP	None	RW	RW	0xFFFFFFFF0
Guest RFLAGS	None	RW	RW	0x00000002

Field	Host VMM Access		L1 VMM Access	Init Value (after TDH.VP.INIT)
	Prod.	Debug		
Guest ES selector	None	RW	RW	0
Guest CS selector	None	RW	RW	0
Guest SS selector	None	RW	RW	0
Guest DS selector	None	RW	RW	0
Guest FS selector	None	RW	RW	0
Guest GS selector	None	RW	RW	0
Guest LDTR selector	None	RW	RW	0
Guest TR selector	None	RW	RW	0
Guest ES base	None	RW	RW	0
Guest CS base	None	RW	RW	0
Guest SS base	None	RW	RW	0
Guest DS base	None	RW	RW	0
Guest FS base	None	RW	RW	0
Guest GS base	None	RW	RW	0
Guest LDTR base	None	RW	RW	0
Guest TR base	None	RW	RW	0
Guest GDTR base	None	RW	RW	0
Guest IDTR base	None	RW	RW	0
Guest ES limit	None	RW	RW	0xFFFFFFFF
Guest CS limit	None	RW	RW	0xFFFFFFFF
Guest SS limit	None	RW	RW	0xFFFFFFFF
Guest DS limit	None	RW	RW	0xFFFFFFFF
Guest FS limit	None	RW	RW	0xFFFFFFFF
Guest GS limit	None	RW	RW	0xFFFFFFFF
Guest LDTR limit	None	RW	RW	0x0000FFFF
Guest TR limit	None	RW	RW	0x0000FFFF
Guest GDTR limit	None	RW	RW	0x0000FFFF
Guest IDTR limit	None	RW	RW	0
Guest ES access rights	None	RW	RW	0x0000C093 (Data, RW, Accessed, DPL=0, Present, 32b, 4KB granularity)
Guest CS access rights	None	RW	RW	0x0000C09B (Code, RX, Accessed, DPL=0, Present, 32b)
Guest SS access rights	None	RW	RW	0x0000C093 (Data, RW, Accessed, DPL=0, Present, 32b, 4KB granularity)
Guest DS access rights	None	RW	RW	0x0000C093 (Data, RW, Accessed, DPL=0, Present, 32b, 4KB granularity)
Guest FS access rights	None	RW	RW	0x0000C093 (Data, RW, Accessed, DPL=0, Present, 32b, 4KB granularity)
Guest GS access rights	None	RW	RW	0x0000C093 (Data, RW, Accessed, DPL=0, Present, 32b, 4KB granularity)
Guest LDTR access rights	None	RW	RW	0x00010082 (LDT, Present, 32b, 1B granularity, Unusable)
Guest TR access rights	None	RW	RW	0x0000008B (32b TSS, Busy, Present, 32b, 1B granularity)
Guest SMBASE	None	None	None	0

#### 4.2.5.1.2. L2 VMCS Guest MSRs

See also the MSR virtualization tables in 2.1.

**Table 4.32: L2 VMCS Guest MSRs**

Field	Host VMM Access		L1 VMM Access	Init Value (after TDH.VP.INIT)
	Prod.	Debug		
IA32_DEBUGCTL	None	RWS	RWS	0
IA32_SYSENTER_CS	None	RW	RW	0
IA32_SYSENTER_ESP	None	RW	RW	0
IA32_SYSENTER_EIP	None	RW	RW	0
IA32_PERF_GLOBAL_CTRL	None	RW	RW	0x000000FF • EN_PMCx (bits 0 to (NUM_PMC - 1))

Field	Host VMM Access		L1 VMM Access	Init Value (after TDH.VP.INIT)
	Prod.	Debug		
				are set to 1. • All other bits are cleared to 0.
IA32_PAT	None	RW	RW	0x0007040600070406
IA32_EFER	None	RW	RW	0x901 • SCE (bit 0) is set to 1. • LME (bit 8) is set to 1. • NXE (bit 11) is set to 1. • All other bits are cleared to 0.
GUEST_IA32_S_CET	None	RW	RW	0
GUEST_SSP	None	RW	RW	0
GUEST_IA32_INTERRUPT_SSP_TABLE_ADDR	None	RW	RW	0
IA32_RTIT_CTL	None	RW	RW	0
IA32_LBR_CTL	None	RW	RW	0
IA32_BNDCFGS	None	None	None	The TDX module must not initialize this field
IA32_GUEST_PKRS	None	RW	RW	0

#### 4.2.5.1.3. L2 VMCS Guest Non-Register State Area

Intel SDM, 24.4.2 Guest Non-Register State

Table 4.33: L2 VMCS Guest Non-Register State Area Fields

Field Name	Host VMM Access		L1 VMM Access	Description	Initial State
	Prod.	Debug			
Activity State	None	RO	RO	Saved/restored on VM exit/entry	Active (0)
Interruptibility State	None	RW	RW	Saved/restored on VM exit/entry	0
Pending Debug Exceptions	None	RW	RW	Saved/restored on VM exit/entry	0
VMCS Link Pointer	None	None	None		NULL_PA (-1)
VMX-Preemption Timer Value	None	RW	None	VMX-preemption timer is used by the TDX module	0
PDPTEn	None	RO	RW		NULL_PA (-1)
Guest Interrupt Status	None	RW	RW	Includes RVI (lower byte) and SVI (upper byte): saved/restored on VM exit/entry	0
PML Index	None	RW	None		0
Guest UINV	None	RW	RW		0

5

#### 4.2.5.2. L2 VMCS Host State Area

Intel SDM, 24.5 Host-State Area

The host state area is not intended to be accessible outside the Intel TDX module.

#### 4.2.5.3. L2 VMCS VM-Execution Control Fields

10 Intel SDM, 24.6 VM-Execution Control Fields

##### 4.2.5.3.1. L2 VMCS Pin-Based VM-Execution Controls

Table 4.34: L2 VMCS Pin-Based VM-Execution Controls

Bit	Name	Host VMM Access		L1 VMM Access	Description	Init Value
		Prod.	Debug			
0	External-interrupt exiting	None	RO	RO	The Intel TDX module performs TD Exit	1

Bit	Name	Host VMM Access		L1 VMM Access	Description	Init Value
		Prod.	Debug			
1	Reserved	None	RO	RO		MSR
2	Reserved	None	RO	RO		MSR
3	NMI exiting	None	RO	RO	The Intel TDX module performs TD Exit	1
4	Reserved	None	RO	RO		MSR
5	Virtual NMIs	None	RO	RO		1
6	Activate VMX-preemption timer	None	RO	RO		0
7	Process posted interrupts	None	RO	RO	Posting interrupts to L2 is not allowed	0
8	Reserved	None	RO	RO		MSR
9	Reserved	None	RO	RO		MSR
10	Reserved	None	RO	RO		MSR
11	Reserved	None	RO	RO		MSR
12	Reserved	None	RO	RO		MSR
13	Reserved	None	RO	RO		MSR
14	Reserved	None	RO	RO		MSR
15	Reserved	None	RO	RO		MSR
16	Reserved	None	RO	RO		MSR
17	Reserved	None	RO	RO		MSR
18	Reserved	None	RO	RO		MSR
19	Reserved	None	RO	RO		MSR
20	Reserved	None	RO	RO		MSR
21	Reserved	None	RO	RO		MSR
22	Reserved	None	RO	RO		MSR
23	Reserved	None	RO	RO		MSR
24	Reserved	None	RO	RO		MSR
25	Reserved	None	RO	RO		MSR
26	Reserved	None	RO	RO		MSR
27	Reserved	None	RO	RO		MSR
28	Reserved	None	RO	RO		MSR
29	Reserved	None	RO	RO		MSR
30	Reserved	None	RO	RO		MSR
31	Reserved	None	RO	RO		MSR

Reserved bits are set based on IA32\_VMX\_TRUE\_PINBASED\_CTLMSR.

#### 4.2.5.3.2. L2 VMCS Processor-Based VM-Execution Controls

Table 4.35: L2 VMCS Primary Processor-Based VM-Execution Controls

Bit	Name	Host VMM Access		L1 VMM Access	Description	Init Value
		Prod.	Debug			
0	Reserved	None	RO	RO		MSR
1	Reserved	None	RO	RO		MSR
2	Interrupt-window exiting	None	RW	RW	VM exit causes L2-to-L1 exit	0
3	Use TSC offsetting	None	RO	RO	TBD - may need to open if L1 VMM needs to control TSC	1
4	Reserved	None	RO	RO		MSR
5	Reserved	None	RO	RO		MSR
6	Reserved	None	RO	RO		MSR
7	HLT exiting	None	RO	RO	#VE injection if TDVPS.ENABLE_EXTENDED_VE, else L2-to-L1 exit	1
8	Reserved	None	RO	RO		MSR
9	INVLPG exiting	None	RW	RW		0
10	MWAIT exiting	None	RO	RO	#VE injection if TDVPS.ENABLE_EXTENDED_VE, else L2-to-L1 exit	1
11	RDPMC exiting	None	RW	RWS	Writable only if ATTRIBUTES.PERFMON is 1; VM exit causes L2-to-L1 exit.	~TDCS.ATTRIBUTES.PERFMON
12	RDTSC exiting	None	RW	RW	VM exit causes L2-to-L1 exit	0
13	Reserved	None	RO	RO		MSR
14	Reserved	None	RO	RO		MSR
15	CR3-load exiting	None	RW	RW	VM exit causes L2-to-L1 exit	0

Bit	Name	Host VMM Access		L1 VMM Access	Description	Init Value
		Prod.	Debug			
16	CR3-store exiting	None	RW	RW	VM exit causes L2-to-L1 exit	0
17	Activate tertiary controls	None	RO	RO		1
18	Reserved	None	RO	RO		MSR
19	CR8-load exiting	None	RW	RW	VM exit causes L2-to-L1 exit	0
20	CR8-store exiting	None	RW	RW	VM exit causes L2-to-L1 exit	0
21	Use TPR shadow	None	RO	RW		1
22	NMI-window exiting	None	RO	RW	VM exit causes L2-to-L1 exit	0
23	MOV-DR exiting	None	RW	RW	VM exit causes L2-to-L1 exit	0
24	Unconditional I/O exiting	None	RW	None		1
25	Use I/O bitmaps	None	RO	None		0
26	Reserved	None	RO	RO		MSR
27	Monitor trap flag	None	RW	RW	TBD - interaction with 0/1-step	0
28	Use MSR bitmaps	None	RO	RO		1
29	MONITOR exiting	None	RW	RO	#VE injection if TDVPS.ENABLE_EXTENDED_VE, else L2-to-L1 exit	1
30	PAUSE exiting	None	RW	RW	VM exit causes L2-to-L1 exit	0
31	Activate secondary controls	None	RO	RO		1

Reserved bits are set based on IA32\_VMX\_TRUE\_PROCBASED\_CTLMSR.

**Table 4.36: L2 VMCS Secondary Processor-Based VM-Execution Controls**

Bit	Name	Host VMM Access		L1 VMM Access	Description	Init Value
		Prod.	Debug			
0	Virtualize APIC accesses	None	RO	RO		0
1	Enable EPT	None	RO	RO		1
2	Descriptor-table exiting	None	RW	RW		0
3	Enable RDTSCP	None	RO	RW		1
4	Virtualize x2APIC mode	None	RO	RO		1
5	Enable VPID	None	RO	RO		1
6	WBINVD exiting	None	RO	RO	#VE injection if TDVPS.ENABLE_EXTENDED_VE, else L2-to-L1 exit	1
7	Unrestricted guest	None	RO	RO		1
8	APIC-register virtualization	None	RO	RW		1
9	Virtual-interrupt delivery	None	RO	RW		1
10	PAUSE-loop exiting	None	RW	RW	VM exit causes L2-to-L1 exit	0
11	RDRAND exiting	None	RW	RW	VM exit causes L2-to-L1 exit	0
12	Enable INVPCID	None	RO	RW		1
13	Enable VM functions	None	RO	RW	VM-Function Controls is 0, so VMFUNC caused a VM exit and L2-to-L1 exit	1
14	VMCS shadowing	None	RO	RO		0
15	Enable ENCLS exiting	None	RO	RO	VM exit causes L2-to-L1 exit	1
16	RDSEED exiting	None	RW	RW	VM exit causes L2-to-L1 exit	0
17	Enable PML	None	RWS	RO	Can be set to 1 only if: - ATTRIBUTES.DEBUG is 1 - PML address is a valid shared physical address	0
18	EPT-violation #VE	None	RO	RO	Virtualization-exception information address shadow must be a valid private GPA to set this bit to 1	0
19	Conceal VMX from PT	None	RO	RO		1
20	Enable XSAVES/XRSTORS	None	RW	RW		1
21	PASID translation	None	RO	RO		1
22	Mode-based execute control for EPT	None	RO	RW		0
23	Enable SPP	None	RO	RO		0
24	PT uses guest physical addresses (PT2GPA)	None	RO	RO		1
25	Use TSC scaling	None	RO	RO		1

Bit	Name	Host VMM Access		L1 VMM Access	Description	Init Value
		Prod.	Debug			
26	Enable user-level wait and pause	None	RO	RWS	Writable only if the value of the TD's virtualized CPUID(0x7,0x0).ECX[5] (WAITPKG) is 1	Set to the value of virtualized CPUID(0x7,0x0).ECX[5] (WAITPKG)
27	Enable PCONFIG	None	RO	RWS	Writable only if the value of the TD's virtualized CPUID(0x7,0x0).EDX[18] (PCONFIG) is 1. PCONFIG-exiting bitmap is all 1, so VM exit causes an L2-to-L1 exit.	Set to the value of virtualized CPUID(0x7,0x0).EDX[18] (PCONFIG)
28	Enable ENCLV exiting	None	RO	RO		1
29	Enable EPC Virtualization Extensions	None	RO	RO		0
30	Bus-lock detection	RW	RW	RO		0
31	Notification exiting	RW	RW	RO		0

Table 4.37: L2 VMCS Tertiary Processor-Based VM-Execution Controls

Bit	Name	Host VMM Access		L1 VMM Access	Description	Init Value
		Prod.	Debug			
0	LOADIWKEY exiting	None	RW	RO		0
1	Enable HLAT	None	RO	RW		0
2	EPT paging-write control	None	RO	RW		0
3	Guest-paging verification	None	RO	RW		0
4	IPI virtualization	None	RO	RO		0
5	GPAW	None	RO	RO	0: GPA.SHARED bit is GPA[47] 1: GPA.SHARED bit is GPA[51]	Copied from TDCS.GPAW
6	Reserved	None	RO	RO		MSR
7	Reserved	None	RO	RO		MSR
8	Reserved	None	RO	RO		MSR
9	Reserved	None	RO	RO		MSR
10	Reserved	None	RO	RO		MSR
11	Reserved	None	RO	RO		MSR
12	Reserved	None	RO	RO		MSR
13	Reserved	None	RO	RO		MSR
14	Reserved	None	RO	RO		MSR
15	Reserved	None	RO	RO		MSR
16	Reserved	None	RO	RO		MSR
17	Reserved	None	RO	RO		MSR
18	Reserved	None	RO	RO		MSR
19	Reserved	None	RO	RO		MSR
20	Reserved	None	RO	RO		MSR
21	Reserved	None	RO	RO		MSR
22	Reserved	None	RO	RO		MSR
23	Reserved	None	RO	RO		MSR
24	Reserved	None	RO	RO		MSR
25	Reserved	None	RO	RO		MSR
26	Reserved	None	RO	RO		MSR
27	Reserved	None	RO	RO		MSR
28	Reserved	None	RO	RO		MSR
29	Reserved	None	RO	RO		MSR
30	Reserved	None	RO	RO		MSR
31	Reserved	None	RO	RO		MSR
32	Reserved	None	RO	RO		MSR
33	Reserved	None	RO	RO		MSR
34	Reserved	None	RO	RO		MSR
35	Reserved	None	RO	RO		MSR
36	Reserved	None	RO	RO		MSR
37	Reserved	None	RO	RO		MSR
38	Reserved	None	RO	RO		MSR
39	Reserved	None	RO	RO		MSR
40	Reserved	None	RO	RO		MSR
41	Reserved	None	RO	RO		MSR



Bit	Name	Host VMM Access		L1 VMM Access	Description	Init Value
		Prod.	Debug			
42	Reserved	None	RO	RO		MSR
43	Reserved	None	RO	RO		MSR
44	Reserved	None	RO	RO		MSR
45	Reserved	None	RO	RO		MSR
46	Reserved	None	RO	RO		MSR
47	Reserved	None	RO	RO		MSR
48	Reserved	None	RO	RO		MSR
49	Reserved	None	RO	RO		MSR
50	Reserved	None	RO	RO		MSR
51	Reserved	None	RO	RO		MSR
52	Reserved	None	RO	RO		MSR
53	Reserved	None	RO	RO		MSR
54	Reserved	None	RO	RO		MSR
55	Reserved	None	RO	RO		MSR
56	Reserved	None	RO	RO		MSR
57	Reserved	None	RO	RO		MSR
58	Reserved	None	RO	RO		MSR
59	Reserved	None	RO	RO		MSR
60	Reserved	None	RO	RO		MSR
61	Reserved	None	RO	RO		MSR
62	Reserved	None	RO	RO		MSR
63	Reserved	None	RO	RO		MSR

Reserved bits are set based on IA32\_VMX\_PROCBASED\_CTL3 MSR.

#### 4.2.5.3.3. L2 VMCS Controls for APIC Virtualization

Table 4.38: L2 VMCS Controls for APIC Virtualization

Field Name	VMM Access		L1 VMM Access	Description	Initial Value
	Prod.	Debug			
APIC-access address	None	RO	RO		NULL_PA (-1)
Virtual-APIC address	None	RO	RWS	L1 read and write access is to shadow Virtual-APIC address (as GPA). The TDX module translates this to HPA on demand.	NULL_PA (-1)
TPR threshold	None	RO	RW		0
EOI-exit bitmap n	None	RO	RW		0
Posted-interrupt notification vector	RO	RO	RO	Posting interrupts to L2 is not supported	0xFFFF
Posted-interrupt descriptor address	RO	RO	RO	Posting interrupts to L2 is not supported	NULL_PA (-1)

5

#### 4.2.5.3.4. EPTP and Shared EPTP

Table 4.39: EPTP (Copied from TDCS.EPTP on TDH.VP.INIT)

Bits	Field Name	VMM Access		L1 VMM Access	Description	Initial Value
		Prod.	Debug			
2:0	EPT Memory Type	RO	RO	RO	Set to WB	6
5:3	EPT Level	RO	RO	RO	1 less than the EPT page-walk length	Copied from TDCS.EPTP
6	Enable A/D Bits	RO	RO	RO		Copied from TDCS.EPTP

Bits	Field Name	VMM Access		L1 VMM Access	Description	Initial Value
		Prod.	Debug			
7	Enable supervisor shadow stack control	RO	RWS	RWS	Can be set to 1 only if CET is enabled to the TD (in XFAM)	0
11:8	Reserved	RO	RO	RO		0
51:12	EPML5/4 PA	RO	RO	None		
63:52	Reserved	RO	RO	RO		0

Table 4.40: Shared EPTP

Bits	Field Name	VMM Access		L1 VMM Access	Description	Initial Value
		Prod.	Debug			
11:0	Reserved	None	RO	RO		0
51:12	EPML5/4 PA	RWS	RWS	None	A guest TD attempt to access a Shared GPA before this field is updated by the host VMM results in an EPT Misconfiguration TD exit. TDH.VP.WR checks that this is a valid shared physical address (HKID bits encode a shared HKID) aligned on 4KB.	Initialized to bits [51:12] of the physical address (including private HKID) of TDCS.ZERO_PAGE, a page whose content is all 0.
63:52	Reserved	None	RO	RO		0

#### 4.2.5.3.5. CR-Related L2 VMCS VM-Execution Control Fields

5

Table 4.41: CR-Related VMCS VM-Execution Control Fields

Field Name	Host VMM Access		L1 VMM Access	Description	Initial State
	Prod.	Debug			
CR0 Guest/Host Mask	None	RO	RWS	Actual value is calculated by the TDX module from the TD VMCS' CR0 Guest/Host Mask and the value written by the L1 VMM	0xFFFFFFFFFFFFFFFF
CR0 Read Shadow	None	RO	RWS	Actual value is calculated by the TDX module from the TD VMCS' CR0 Read Shadow and the value written by the L1 VMM	0x0000000000000021
CR4 Guest/Host Mask	None	RO	RWS	Actual value is calculated by the TDX module from the TD VMCS' CR0 Guest/Host Mask and the value written by the L1 VMM	0xFFFFFFFFFFFFFFFF
CR4 Read Shadow	None	RO	RWS	Actual value is calculated by the TDX module from the TD VMCS' CR0 Read Shadow and the value written by the L1 VMM	0x00000000000002040
CR3-Target Values	None	RW	RW		0
CR3-Target Count	None	RW	RW		0

## 4.2.5.3.6. Other L2 VMCS VM-Execution Control Fields

Table 4.42: Other L2 VMCS VM-Execution Control Fields

Field Name	Host VMM Access		L1 VMM Access	Description	Initial State
	Prod.	Debug			
Exception Bitmap	None	RW	RW	<ul style="list-style-type: none"> <li>Bit 18 (MCE) is set to 1, even in debug mode.</li> <li>Other bits are cleared to 0. They may be modified in debug mode.</li> </ul>	0x00040000
Page-fault error-code mask	None	RW	RW		0
Page-fault error-code match	None	RW	RW		0
I/O-Bitmap Address n	None	RO	RO	Set to NULL_PA (-1): I/O bitmaps execution control is set to 0	NULL_PA (-1)
Time-Stamp Counter Offset	RO	RW	None		Copied from TDCS.TSC_OFFSET
Time-Stamp Counter Multiplier	RO	RW	None		Copied from TDCS.TSC_MULTIPLIER
MSR-Bitmap Address	RO	RO	None	The MSR bitmaps page is held as part of TDVPS. This field is set to the PA of that page.	PA (including HKID) of the L2 MSR Bitmaps page (in TDVPS)
Executive-VMCS Pointer	None	None	None	N/A	NULL_PA (-1)
TD HKID	RO	RO	None		Copied from TDCS
VPID	None	RO	None	Unique identifier of the VM in the platform: <b>Bits 1:0:</b> VM index <b>Bits 15:2:</b> TD's HKID	<b>Bits 1:0:</b> VM index <b>Bits 15:2:</b> TD's HKID
PLE_GAP	RO	RWS	RWS		0
PLE_Window	RO	RWS	RWS		0
VM-Function Controls	RO	RO	RO	The Intel TDX module injects a #UD into the TD.	0
EPTP-list address	RO	RO	None	VMFUNC is not supported.	NULL_PA (-1)
VMREAD-bitmap address	None	RO	None	VMCS shadowing is not supported.	NULL_PA (-1)
VMWRITE-bitmap address	None	RO	None	VMCS shadowing is not supported.	NULL_PA (-1)
ENCLS-Exiting Bitmap	None	RO	RO	Set to all 1's – the Intel TDX module injects a #UD into the guest TD.	All 1s
ENCLV-Exiting Bitmap	None	RO	RO	Set to all 1's – the Intel TDX module injects a #UD into the guest TD.	All 1s
PML address	RO	RW	None	Address must be: <ul style="list-style-type: none"> <li>Valid shared physical address (HKID bits encode a shared HKID).</li> <li>Aligned on 4KB.</li> </ul> See enable PML execution control.	NULL_PA (-1)
Virtualization-exception information address	None	RO	RO	L1 read and write access is to shadow VE info address (as GPA). The TDX module translates this to HPA on demand. "EPT Violation #VE" control must be 0 if the shadow value of this field is NULL_PA (-1).	NULL_PA (-1)
EPTP index	None	RO	None		0
XSS-Exiting Bitmap	None	RW	RW		0

Field Name	Host VMM Access		L1 VMM Access	Description	Initial State
	Prod.	Debug			
low PASID directory address	None	RO	None		Implementation-dependent
high PASID directory address	None	RO	None		Implementation-dependent
notify window	RW	RW	RO	TO BE COMPLETED	0
PCONFIG-Exiting Bitmap	None	RO	None		-1
HLAT pointer	None	RWS	RWS		0
HLAT prefix size	None	RW	RW		0

#### 4.2.5.4. L2 VMCS VM-Exit Control Fields

Intel SDM, 24.7 VM-Exit Control Fields

Table 4.43: L2 VMCS VM-Exit Controls

Bit	Name	Host VMM Access		L1 VMM Access	Description	Init Value
		Prod.	Debug			
0	Reserved	None	RO	RO		MSR
1	Reserved	None	RO	RO		MSR
2	Save debug controls	None	RO	RO		1
3	Reserved	None	RO	RO		MSR
4	Reserved	None	RO	RO		MSR
5	Reserved	None	RO	RO		MSR
6	Reserved	None	RO	RO		MSR
7	Reserved	None	RO	RO		MSR
8	Reserved	None	RO	RO		MSR
9	Host address-space size	None	RO	RO		1
10	Reserved	None	RO	RO		MSR
11	Reserved	None	RO	RO		MSR
12	Load IA32_PERF_GLOBAL_CTRL	None	RO	RO		= (TDCS.ATTRIBUTES.PERFMON   TDCS.ATTRIBUTES.DEBUG)
13	Reserved	None	RO	RO		MSR
14	Reserved	None	RO	RO		MSR
15	Acknowledge interrupt on exit	None	RO	RO		1
16	Reserved	None	RO	RO		MSR
17	Reserved	None	RO	RO		MSR
18	Save IA32_PAT	None	RO	RO		1
19	Load IA32_PAT	None	RO	RO		1
20	Save IA32_EFER	None	RO	RO		1
21	Load IA32_EFER	None	RO	RO		1
22	Save VMX-preemption time value	None	RO	RO		0
23	Clear IA32_BNDCFGS	None	RO	RO	Deprecated	0
24	Conceal VMX from PT	None	RO	RO		1
25	Clear IA32_RTIT_CTL	None	RO	RO		1
26	Clear IA32_LBR_CTL	None	RO	RO		1
27	Clear UINV	None	RO	RO		1
28	Load host CET state	None	RO	RO		1
29	Load host PKRS	None	RO	RO		0
30	Save IA32_PERF_GLOBAL_CTRL	None	RO	RO		= (TDCS.ATTRIBUTES.PERFMON   TDCS.ATTRIBUTES.DEBUG)
31	Activate secondary controls	None	RO	RO		0

Reserved bits are set based on IA32\_VMX\_TRUE\_EXIT\_CTLMSR.

**Table 4.44: L2 VMCS VM-Exit Controls for MSRs**

Field Name	Host VMM Access		L1 VMM Access	Description	Initial State
	Prod.	Debug			
VM-exit MSR-store count	None	RO	None	Not used	0
VM-exit MSR-store address	None	RO	None	Not used	NULL_PA (-1)
VM-exit MSR-load count	None	RO	None	Not used	0
VM-exit MSR-load address	None	RO	None	Not used	NULL_PA (-1)

#### 5 4.2.5.5. L2 VMCS VM-Entry Control Fields

Intel SDM, 24.8 VM-Entry Control Fields

**Table 4.45: L2 VMCS VM-Entry Controls**

Bit	Name	Host VMM Access		L1 VMM Access	Description	Init Value
		Prod.	Debug			
0	Reserved	None	RO	RO		MSR
1	Reserved	None	RO	RO		MSR
2	Load debug controls	None	RO	RO		1
3	Reserved	None	RO	RO		MSR
4	Reserved	None	RO	RO		MSR
5	Reserved	None	RO	RO		MSR
6	Reserved	None	RO	RO		MSR
7	Reserved	None	RO	RO		MSR
8	Reserved	None	RO	RO		MSR
9	IA-32e mode guest	None	RO	RW	Written by the CPU on VM exit	0
10	Entry to SMM	None	RO	RO		0
11	Deactivate dual-monitor treatment	None	RO	RO		0
12	Reserved	None	RO	RO		MSR
13	Load IA32_PERF_GLOBAL_CTRL	None	RO	RO		= (TDCS.ATTRIBUTES.PERFMON   TDCS.ATTRIBUTES.DEBUG)
14	Load IA32_PAT	None	RO	RO		1
15	Load IA32_EFER	None	RO	RO		1
16	Load IA32_BNDCFGS	None	RO	RO		0
17	Conceal VMX from PT	None	RO	RO		1
18	Load IA32_RTIT_CTL	None	RO	RO		1
19	Load UINV	None	RO	RO		1
20	Load CET state	None	RO	RO		1
21	Load IA32_LBR_CTL	None	RO	RO		1
22	Load guest PKRS	None	RO	RO		= (TDCS.ATTRIBUTES.PKRS   TDCS.ATTRIBUTES.DEBUG)
23	Reserved	None	RO	RO		MSR
24	Reserved	None	RO	RO		MSR
25	Reserved	None	RO	RO		MSR
26	Reserved	None	RO	RO		MSR
27	Reserved	None	RO	RO		MSR
28	Reserved	None	RO	RO		MSR
29	Reserved	None	RO	RO		MSR
30	Reserved	None	RO	RO		MSR
31	Reserved	None	RO	RO		MSR

Reserved bits are set based on IA32\_VMX\_ENTRY\_CTLMSR.

Table 4.46: L2 VMCS VM-Entry Controls for MSRs

Field Name	Host VMM Access		L1 VMM Access	Description	Initial State
	Prod.	Debug			
VM-entry MSR-load count	None	RO	None	Not used	0
VM-entry MSR-load address	None	RO	None	Not used	NULL_PA (-1)

Table 4.47: L2 VMCS VM-Entry Controls for Event Injection

Field Name	Host VMM Access		L1 VMM Access	Description	Initial State
	Prod.	Debug			
VM-entry interruption information	None	RO	RW		N/A
VM-entry exception error code	None	RO	RW		N/A
VM-entry instruction length	None	RO	RW		N/a

## 5 4.2.5.6. L2 VMCS VM-Exit Information Fields

Intel SDM, 24.9

VM-Exit Information Fields

Table 4.48: L2 VMCS Basic VM-Exit Information

Field Name	Host VMM Access		L1 VMM Access	Description	Initial State
	Prod.	Debug			
Exit reason	None	RO	RW	If the Intel TDX module decides to perform a TD exit, it returns this in RAX bits 31:0.	N/A
Exit qualification	None	RO	RW	If the Intel TDX module decides to perform a TD exit, it returns this in RCX. If the exit is due to EPT violation, bits 12-7 of the exit qualification are cleared to 0.	N/A
Guest-Linear Address	None	RO	RW		N/A
Guest-physical Address	None	RO	RW	If the Intel TDX module decides to perform a TD exit, it returns this in R8. If the EPT fault was caused by an access attempt to a private page, the Intel TDX module clears bits 11:0 to 0.	N/A

Table 4.49: L2 VMCS Information for VM Exits Due to Vectored Events

Field Name	Host VMM Access		L1 VMM Access	Description	Initial State
	Prod.	Debug			
VM-exit interruption information	None	RO	RW	On asynchronous TD exit, the Intel TDX module returns this in R9. Bits 63:32 are cleared to 0.	N/A
VM-exit interruption error code	None	RO	RW		N/A

10

Table 4.50: L2 VMCS Information for VM Exits That Occur During Event Delivery

Field Name	Host VMM Access		L1 VMM Access	Description	Initial State
	Prod.	Debug			
IDT-vectoring information	None	RO	RW		N/A
IDT-vectoring error code	None	RO	RW		N/A

**Table 4.51: L2 VMCS Information for VM Exits Due to Instruction Execution**

Field Name	Host VMM Access		L1 VMM Access	Description	Initial State
	Prod.	Debug			
VM-exit instruction length	None	RO	RW		N/A
VM-exit instruction information	None	RO	RW		N/A
I/O RCX	None	RO	RW		N/A
I/O RSI	None	RO	RW		N/A
I/O RDI	None	RO	RW		N/A
I/O RIP	None	RO	RW		N/A

**Table 4.52: L2 VMCS VM-Instruction Error Field**

Field Name	Host VMM Access		L1 VMM Access	Description	Initial State
	Prod.	Debug			
VM-instruction error	None	RO	RW		N/A

DRAFT

## 5. UPDATED: Interface Functions

### 5.1. How to Read the Interface Function Definitions

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 5 A table of operands is provided for any function that has explicit and/or implicit memory operands or implicit resources. Table 5.1 below describes how to read it. Most of the background is detailed in the [TDX Module Base Spec].

**Table 5.1: How to Read the Operands Information Tables**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Alignment Check	Concurrency Restrictions		
								Resource	Contain. 2MB	Contain. 1GB
The operand may be specified explicitly or may be implicit	Register used as a pointer to the operand	How the operand is referenced: HPA, GPA, GPA and level or index	Resource (memory or CPU internal) for this operand	Data type of the resource, as defined in Chapter 3 or Chapter 4	Type of memory or resource access: R, RW, or Ref	Shared, Private, Opaque or Hidden	Required alignment of the operand	<p>Concurrency restrictions are described in the [TDX Module Base Spec].</p> <p>For explicit memory accesses using HPA, there are additional concurrency restrictions on the 1GB and 2MB blocks that contain the accessed HPA. For other types of accesses, only the operand concurrency is applicable.</p> <p>Shared(h) and Exclusive(h) indicate shared access with host-side priority.</p> <p>Sh./Ex.(h) indicates either shared or exclusive access with host-side priority, depending on the platform type</p> <ul style="list-style-type: none"> <li>On server platforms, access is shared.</li> <li>On client platforms, access is exclusive.</li> </ul> <p>Shared(i) and Exclusive(i) indicate that the resource is implicitly restricted.</p>		

### 5.2. NEW: Common Algorithms Used by Multiple Interface Functions

- 10 This section describes common algorithms that are used by multiple interface functions.

#### 5.2.1. VCPU Association with an LP

The following algorithm is used for associating the current VCPU with the current LP. It is used with VCPU-specific host-side interface functions such as TDH.VP.ENTER, TDH.VP.RD etc.

1. Check that the VCPU has been initialized and is not being torn down.
- 15 2. Atomically check that the VCPU is not associated with another LP, and associate it with the current LP.
3. If this is a new association, and the TD's ephemeral HKID has changed since last association, update all TD VMCS physical pointers and the TD HKID execution control.
4. Update the TD VMCS host state fields with any Intel TDX module LP-specific values.



## 5.2.2. Metadata Access

### 5.2.2.1. Single Metadata Field Read

The following algorithm is used when reading a single metadata field based on a provided field identifier. This algorithm is used by TDH.MNG.RD, TDH.VP.RD and TDG.VM.RD, TDG.VP.RD.

- 5 **Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.
1. Check that the field identifier is valid and derive a read mask depending on whether this algorithm is used by a host-side or a guest-side interface function, and whether the TD runs in debug mode (ATTRIBUTES.DEBUG is 1).
  2. If the read mask is 0, then fail; the field is not readable.
- 10 If the above checks passed:
3. Read the field value from the control structure using the proper method per field class.
  4. Mask the field value with the read mask derived above, and return the resulting value.
    - 4.1. In some cases, special handling is required. E.g., the field value may need to be translated to another format, or some other action may be needed.

### 5.2.2.2. Single Metadata Field Write

The following algorithm is used when writing a single metadata field based on provided field identifier, input value and write mask. This algorithm is used by TDH.MNG.WR, TDH.VP.WR, TDG.VM.WR and TDG.VP.WR.

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 20
  1. Check that the field identifier is valid and derive the field attributes (read mask, write mask) depending on whether this algorithm is used by a host-side or a guest-side interface function, and whether the TD runs in debug mode (ATTRIBUTES.DEBUG is 1).
  2. If the write mask is 0, then fail, the field is not writable.
- If passed:
- 25
  3. Calculate an effective write mask:
    - 3.1. If a write mask is provided as an input, derive the effective write mask by bitwise-anding the write mask derived above with the write mask provided as an input.
    - 3.2. Else, the effective write mask is the write mask derived above.
  4. If the effective write mask is 0, then fail, the field is not writable.
- 30 If passed:
5. Read the old field value from the control structure using the proper method per field class.
  6. Calculate a new field value based on the input value and the effective write mask, and write to the control structure using the proper method per field class.
    - 6.1. In some cases, special handling is required. E.g., the new field value may need to be checked for validity, or some other action may be needed.
- 35

If passed:

7. Mask the old field value with the read mask derived above, and return the resulting value.

### 5.2.2.3. Multiple Metadata Fields Write based on a Metadata List

The following algorithm is used when writing multiple metadata fields based on a provided metadata list. This algorithm is used by TDH.IMPORT.STATE.\*.

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

1. Check the list header to be valid (NUM\_SEQUENCES > 0).

If passed:

- 45
  2. For each sequence in the list:
    - 2.1. Check that the list did not cross 4KB page boundary.

2.2. Read the sequence header and check it is valid.

If the above checks passed:

2.3. For each field in the sequence:

2.3.1. Check that the list did not cross 4KB page boundary.

2.3.2. Check that the field identifier is valid and derive a write mask depending on whether this algorithm is used by a host-side or a guest-side interface function, and whether the TD runs in debug mode (ATTRIBUTES.DEBUG is 1).

2.3.3. If the write mask is 0, then fail, the field is not writable.

If the above checks passed:

2.3.4. Calculate an effective write mask:

2.3.4.1. If a write mask is provided for each field in the current sequence, derive the effective write mask by bitwise-anding the write mask derived above with the write mask provided with the field.

2.3.4.2. Else, the effective write mask is the write mask derived above.

2.3.5. If the effective write mask is 0, then fail, the field is not writable.

If passed:

2.3.6. Read the existing field value from the control structure using the proper method per field class.

2.3.7. Calculate a new field value based on the input value and the effective write mask, and write to the control structure using the proper method per field class.

2.3.7.1. In some cases, special handling is required. E.g., the new field value may need to be checked for validity, or some other action may be needed.

DRAFT

### 5.3. **UPDATED:** Host-Side (SEAMCALL) Interface Functions

The SEAMCALL instruction enters the Intel TDX module. It is designed to call host-side Intel TDX functions, either local or a TD entry to a guest TD, as selected by RAX.

#### 5.3.1. **UPDATED:** SEAMCALL Instruction (Common)

- 5 This section describes the common functionality of SEAMCALL. Leaf functions are described in the following sections.

**Table 5.2: SEAMCALL Input Operands Definition**

Parameter	Description		
RAX	Leaf and version numbers, as defined in the [TDX Module Base Spec]. See Table 5.4 below for SEAMCALL leaf numbers.		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version
	63:24	Reserved	Must be 0
Other	See individual SEAMCALL leaf functions.		

**Table 5.3: SEAMCALL Output Operands Definition**

Parameter	Description
RAX	Instruction return code, indicating the outcome of execution of the instruction. See the [TDX Module Base Spec] for details.
Other	See individual SEAMCALL leaf functions.

**Table 5.4: SEAMCALL Instruction Leaf Numbers Definition**

Leaf #	Interface Function Name	Description
0	TDH.VP.ENTER	Enter TDX non-root operation
1	TDH.MNG.ADDCX	Add a control structure page to a TD
2	TDH.MEM.PAGE.ADD	Add a 4KB private page to a TD during TD build time
3	TDH.MEM.SEPT.ADD	Add and map a 4KB Secure EPT page to a TD
4	TDH.VP.ADDCX	Add a control structure page to a TD VCPU
5	TDH.MEM.PAGE.RELOCATE	Relocate a 4KB mapped page from its HPA to another
6	TDH.MEM.PAGE.AUG	Dynamically add a 4KB private page to an initialized TD
7	TDH.MEM.RANGE.BLOCK	Block a TD private GPA range
8	TDH.MNG.KEY.CONFIG	Configure the TD private key on a single package
9	TDH.MNG.CREATE	Create a guest TD and its TDR root page
10	TDH.VP.CREATE	Create a guest TD VCPU and its TDVPR root page
11	TDH.MNG.RD	Read TD metadata
12	TDH.MEM.RD	Read from private memory of a debuggable guest TD
13	TDH.MNG.WR	Write TD metadata
14	TDH.MEM.WR	Write to private memory of a debuggable guest TD
15	TDH.MEM.PAGE.DEMOTE	Split a 2MB or a 1GB private TD page mapping into 512 4KB or 2MB page mappings respectively
16	TDH.MR.EXTEND	Extend the guest TD measurement register during TD build
17	TDH.MR.FINALIZE	Finalize the guest TD measurement register

Leaf #	Interface Function Name	Description
18	TDH.VP.FLUSH	Flush the address translation caches and cached TD VMCS associated with a TD VCPU
19	TDH.MNG.VPFLUSHDONE	Check all of a guest TD's VCPUs have been flushed by TDH.VP.FLUSH
20	TDH.MNG.KEY.FREEID	Mark the guest TD's HKID as free
21	TDH.MNG.INIT	Initialize per-TD control structures
22	TDH.VP.INIT	Initialize the per-VCPU control structures
23	TDH.MEM.PAGE.PROMOTE	Merge 512 consecutive 4KB or 2MB private TD page mappings into one 2MB or 1GB page mapping respectively
24	TDH.PHYMEM.PAGE.RDMD	Read the metadata of a page in a TDMR
25	TDH.MEM.SEPT.RD	Read a Secure EPT entry
26	TDH.VP.RD	Read VCPU metadata
27	TDH.MNG.KEY.RECLAIMID	Does nothing; provided for backward compatibility
28	TDH.PHYMEM.PAGE.RECLAIM	Reclaim a physical memory page owned by a TD (i.e., TD private page, Secure EPT page or a control structure page)
29	TDH.MEM.PAGE.REMOVE	Remove a private page from a guest TD
30	TDH.MEM.SEPT.REMOVE	Remove a Secure EPT page from a TD
31	TDH.SYS.KEY.CONFIG	Configure the Intel TDX global private key on the current package
32	TDH.SYS.INFO	Get Intel TDX module information
33	TDH.SYS.INIT	Globally initialize the Intel TDX module
34	TDH.SYS.RD	Read a TDX Module global-scope metadata field
35	TDH.SYS.LP.INIT	Initialize the Intel TDX module per logical processor
36	TDH.SYS.TDMR.INIT	Partially initialize a Trust Domain Memory Region (TDMR)
37	TDH.SYS.RDALL	Read all host-readable TDX Module global-scope metadata fields
38	TDH.MEM.TRACK	Increment the TD's TLB tracking counter
39	TDH.MEM.RANGE.UNBLOCK	Remove the blocking of a TD private GPA range
40	TDH.PHYMEM.CACHE.WB	Write back the contents of the cache on a package
41	TDH.PHYMEM.PAGE.WBINVD	Write back and invalidate all cache lines associated with the specified memory page and HKID
43	TDH.VP.WR	Write VCPU metadata
44	TDH.SYS.LP.SHUTDOWN	Does nothing; provided for backward compatibility
45	TDH.SYS.CONFIG	Globally configure the Intel TDX module
48	TDH.SERVTD.BIND	Bind a service TD to a target TD
49	TDH.SERVTD.PREBIND	Pre-bind a service TD to a target TD
52	TDH.SYS.SHUTDOWN	Shutdown the Intel TDX module and prepare handoff data
53	TDH.SYS.UPDATE	Populate Intel TDX module state from handoff data
64	TDH.EXPORT.ABORT	Abort an export session
65	TDH.EXPORT.BLOCKW	Block a TD private page for writing
66	TDH.EXPORT.RESTORE	Restore a list of TD private 4KB pages' Secure EPT entry states after an export abort
68	TDH.EXPORT.MEM	Export a list of TD private pages contents and/or cancellation requests
70	TDH.EXPORT.PAUSE	Pause the exported TD
71	TDH.EXPORT.TRACK	End the current in-order export phase epoch and either start a new epoch or start the out-of-order export phase
72	TDH.EXPORT.STATE.IMMUTABLE	Start an export session and export the TD's immutable state
73	TDH.EXPORT.STATE.TD	Export the TD's mutable state
74	TDH.EXPORT.STATE.VP	Export a VCPU mutable state
75	TDH.EXPORT.UNBLOCKW	Unblock a page that has been blocked for writing
80	TDH.IMPORT.ABORT	Abort an import session
81	TDH.IMPORT.END	End an import session
82	TDH.IMPORT.COMMIT	Commit the import session and allow the imported TD to run
83	TDH.IMPORT.MEM	Import a list of TD private pages contents and/or cancellation requests based on a migration bundle in shared memory
84	TDH.IMPORT.TRACK	End the current in-order import phase epoch and either start a new epoch or start the out-of-order import phase

Leaf #	Interface Function Name	Description
85	TDH.IMPORT.STATE.IMMUTABLE	Start an import session and import the TD's immutable state
86	TDH.IMPORT.STATE.TD	Import the TD's mutable state
87	TDH.IMPORT.STATE.VP	Import a VCPU mutable state
96	TDH.MIG.STREAM.CREATE	Create a migration stream

### Instruction Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 5 On entry, the Intel TDX module performs the checks listed below at a high level. Errors cause a SEAMRET with RAX set to the proper completion status code.
1. The leaf number in RAX is supported by the Intel TDX module.
  2. If the Intel TDX module's state is not SYS\_READY, only **TDH.SYS.RD\***, TDH.SYS.INFO, TDH.SYS.INIT, TDH.SYS.LP.INIT, TDH.SYS.CONFIG, TDH.SYS.KEY.CONFIG and TDH.SYS.SHUTDOWN leaf functions are allowed. Those leaf functions
- 10 then perform other initialization state checks.

If all checks pass, the Intel TDX module calls the leaf function according to the leaf number in RAX. See the following sections for individual leaf function details.

### Completion Status Codes

**Table 5.5: SEAMCALL Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_SUCCESS	SEAMCALL is successful.
TDX_SYS_SHUTDOWN	
Other	See individual leaf functions.

### 5.3.2. NEW: TDH.EXPORT.ABORT Leaf

TDH.EXPORT.ABORT aborts an export session and allows the source TD to resume normal operation, depending on export state and an abort token received from the destination platform.

**Table 5.6: TDH.EXPORT.ABORT Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	HPA of the source TD's TDR page (HKID bits must be 0)		
R8	If an abort token is available, R8 provides the HPA and size of memory of an MBMD structure in memory, as described below. Otherwise, R8's value must be 0.		
	Bits	Name	Description
	51:0	HPA	Bits 51:0 of the host physical address (including HKID bits)
	63:52	Size	Size of the memory buffer containing MBMD, in bytes
R10	Migration stream index:		
	Bits	Name	Description
	15:0	MIGS_INDEX	Migration stream index – must be 0
	63:16	RESERVED	Reserved: must be 0

**Table 5.7: TDH.EXPORT.ABORT Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code, see 5.3.1
Other	Unmodified

#### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

#### Leaf Function Description

TDH.EXPORT.ABORT aborts an export session. If successful, i.e., the target TD does not run, the source TD becomes runnable. If called during the out-of-order phase, an abort token received from the destination platform is required.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.8: TDH.EXPORT.ABORT Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDR page	TDR	R	Opaque	4KB	Shared	Shared	Shared
Explicit	R8	HPA	MBMD buffer	MBMD	R	Shared	128B	None	None	None
Explicit	R10	Index	Mig. Stream context	Mig. Stream context	RW	Opaque	N/A	Exclusive	N/A	N/A
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Exclusive	N/A	N/A
Implicit	N/A	N/A	Migration context	N/A	RW	Opaque	N/A	Exclusive(i)	N/A	N/A

TDH.EXPORT.ABORT checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

5 The function checks the following conditions:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
4. TDCS has been allocated (TDR.NUM\_TDCX is the required number).
- 10 5. An export session is in progress but has not been committed yet: TDCS.OP\_STATE is LIVE\_EXPORT, PAUSED\_EXPORT or POST\_EXPORT.
6. The migration stream index is lower than TDCS.NUM\_MIGS.

If successful, the function does the following:

7. If the export session is in the post-copy phase (TDCS.OP\_STATE is POST\_EXPORT):
  - 15 7.1. Check that the buffer provided for MBMD is large enough.
  - 7.2. Copy the MBMD into a temporary buffer.
  - 7.3. Check the MBMD fields.
- If passed:
  - 20 7.4. If the migration stream has not been initialized, initialize it.
  - 7.5. Build the 96b IV for this migration bundle by concatenating the stream index and the MBMD's IV\_COUNTER.
  - 7.6. Calculate MAC based on the MAC'ed fields of MBMD and check that its value is the same as the MBMD's MAC field's value.
8. Else (the export session is in the pre-copy phase – TDCS.OP\_STATE is LIVE\_EXPORT or PAUSED\_EXPORT):
  - 25 8.1. Check that the MBMD HPA and size provided in R8 is 0.
  - 8.2. Check that the migration stream index provided in R10 is 0.

If passed:

9. Terminate the export session:
  - 9.1. Set all migration streams' INITIALIZED and ENABLED flags to FALSE.
  - 9.2. Set TDCS.OP\_STATE to RUNNABLE.

### 30 Completion Status Codes

**Table 5.9: TDH.EXPORT.ABORT Completion Status Codes (Returned in RAX) Definition [TO BE COMPLETED]**

Completion Status Code	Description
TDX_OPERAND_INVALID	

Completion Status Code	Description
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	Operation is successful
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

DRAFT



### 5.3.3. NEW: TDH.EXPORT.BLOCKW Leaf

Block a list of TD private 4KB pages for writing and for attributes modification.

**Table 5.10: TDH.EXPORT.BLOCKW Input Operands Definition**

Operand	Name	Description		
RAX	Leaf and Version	SEAMCALL instruction leaf number and version, see 5.3.1		
		Bits	Field	Description
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
		63:24	Reserved	Must be 0
RCX	GPA_LIST_INFO	GPA_LIST_INFO: HPA of a GPA list page in shared memory, and first and last entries to process, as defined in 3.12.2 FORMAT must be GPA_ONLY.		
RDX	TDR	HPA of the source TD’s TDR page (HKID bits must be 0)		

**Table 5.11: TDH.EXPORT.BLOCKW Output Operands Definition**

Operand	Name	Description
RAX	STATUS	SEAMCALL instruction return code, see 5.3.1
RCX	GPA_LIST_INFO	Same as the input value, except that FIRST_ENTRY is updated to the index of the next entry to be processed. If all entries have been processed, FIRST_ENTRY is updated to (LAST_ENTRY + 1) Modulo 512.
Other		Unmodified

#### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- For each 4KB page in the GPA list, if a blocking operation has been requested, TDH.EXPORT.BLOCKW finds the Secure EPT entry for the provided page. If the entry state is correct (MAPPED, PENDING, EXPORTED\_DIRTY or PENDING\_EXPORTED\_DIRTY), TDH.EXPORT.BLOCKW blocks it for writing, by saving and clearing the W bit and setting the Secure EPT entry state (to BLOCKEDW, PENDING\_BLOCKEDW, EXPORTED\_DIRTY\_BLOCKEDW or PENDING\_EXPORTED\_DIRTY\_BLOCKEDW respectively). It records the current TD's TLB epoch in the TD's global BW\_EPOCH, and marks the GPA list entry as ready for export. If the TD is partitioned, TDH.EXPORT.BLOCKW also blocks any L2 SEPT entries mapping the 4KB page.

**List Entry Error:** If a page can't be blocked for writing, TDH.EXPORT.BLOCKW marks its GPA list entry as unsuccessful. List processing is not aborted, it continues to the next entry, if applicable. The return status in RAX indicates the number of such cases encountered during operation.

- Interruptibility:** TDH.EXPORT.BLOCKW is interruptible. If a pending interrupt is detected during operation, TDH.EXPORT.BLOCKW returns with a TDX\_INTERRUPTED\_RESUMABLE status in RAX. RCX is updated with the next list entry index to process, so the host BMM may re-invoke TDH.EXPORT.BLOCKW immediately after handling the interrupt.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

Table 5.12: TDH.EXPORT.BLOCKW Memory Operands Information

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	GPA List page	GPA_LIST	RW	Shared	4KB	None	None	None
Explicit	RDX	HPA	TDR page	TDR	R	Opaque	4KB	Shared	Shared	Shared
Explicit	N/A	GPA	TD private pages (via GPA list)	Block	None	Private	4KB	None	None	None
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	R	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	N/A	Migration context	N/A	RW	Opaque	N/A	None	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT tree	N/A	RW	Private	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT entry	SEPT Entry	RW	Private	N/A	Exclusive(h)	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT trees	N/A	RW	Private	N/A	Shared(i)	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT entries	SEPT Entry	RW	Private	N/A	Exclusive(i)	N/A	N/A

TDH.MEM.RANGE.BLOCKW checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

- 5 The function checks the following conditions:
  1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
  2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
  3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
  4. TDCS is allocated (TDR.NUM\_TDCX is the required number).
- 10 5. Export session is in the in-order phase and the TD has not been paused yet (TDCS.OP\_STATE is LIVE\_EXPORT).

If passed, process the GPA list:

**Note:** Error conditions that impact a single GPA list entry do not cause an abort of TDH.EXPORT.BLOCKW. Instead, the GPA list entry is updates with a proper status code, and the corresponding migration buffer list entry is marked as invalid.

- 15 6. For each entry in the GPA list, starting with RCX.FIRST\_ENTRY and ending with RCX.LAST\_ENTRY, if OPERATION indicates a BLOCKW request:
  - 6.1. Check the GPA list entry fields value.
- If passed:
  - 6.2. Walk the L1 Secure EPT based on the GPA operand and find the Secure EPT entry to be blocked.
  - 20 6.3. Check the Secure EPT entry state: it should be either of MAPPED, PENDING, EXPORTED\_DIRTY or PENDING\_EXPORTED\_DIRTY.
  - 6.4. If passed, update the SEPT entry and record the TD epoch:
    - 6.4.1. Save the original value of SEPT.W into SEPT.TDW.
    - 6.4.2. Clear SEPT.W.
    - 25 6.4.3. Atomically set the SEPT entry state to BLOCKEDW, PENDING\_BLOCKEDW, EXPORTED\_DIRTY\_BLOCKEDW or PENDING\_EXPORTED\_DIRTY\_BLOCKEDW as appropriate.

- 6.4.4. If the page state is MAPPED or EXPORTED\_DIRTY, then for each L2 mapping of the page:
- 6.4.4.1. Walk the L2 SEPT tree based on the GPA operand and find the L2 Secure EPT entry to be blocked for writing.
  - 6.4.4.2. Save the original value of SEPT.W into SEPT.TDW.
  - 6.4.4.3. Clear SEPT.W.
  - 6.4.4.4. Set the L2 SEPT entry state to L2\_BLOCKED

**Note:** If the page state is one of the PENDING\* states, then the L2 SEPT entry state is already L2\_BLOCKED, no change is required.

6.4.5. Copy the TD's epoch (TDCS.TD\_EPOCH) to TDCS.BW\_EPOCH.

6.5. Else:

6.5.1. Set the GPA list entry's OPERATION field to NOP and STATUS field to the applicable status.

6.6. If this is not the last entry in the list, and there is a pending interrupt, terminate TDH.EXPORT.BLOCKW with a TDX\_INTERRUPTED\_RESUMABLE status.

### Completion Status Codes

**Table 5.13: TDH.EXPORT.BLOCKW Completion Status Codes (Returned in RAX) Definition [TO BE COMPLETED]**

Completion Status Code	Description
TDX_EPT_ENTRY_FREE	
TDX_EPT_WALK_FAILED	
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	Operation is successful. <b>Note:</b> Processing of some GPA list entries may have encountered errors, but this did not cause an abort of the overall operation. The number such errors is reported in the lower 32 bits of the completion status.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

**5.3.4. NEW: TDH.EXPORT.MEM Leaf**

TDH.EXPORT.MEM exports a list of TD private pages contents and/or cancellation requests and prepares a migration bundle in shared memory.

**Table 5.14: TDH.EXPORT.MEM Input Operands Definition**

Operand	Name	Description		
RAX	Leaf and Version	SEAMCALL instruction leaf number and version, see 5.3.1		
		Bits	Field	Description
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
		63:24	Reserved	Must be 0
RCX	GPA_LIST_INFO	HPA of a GPA list page in shared memory, and first and last entries to process, as defined in 3.12.2  On a new invocation, FIRST_ENTRY must be 0. On a resumed invocation, FIRST_ENTRY must be the index of the next GPA list entry to export.		
RDX	TDR	HPA of the source TD's TDR page (HKID bits must be 0)		
R8	MBMD	HPA and size of memory of a memory buffer to use for MBMD:		
		Bits	Name	Description
		51:0	HPA	Bits 51:0 of the host physical address (including HKID bits)
		63:52	Size	Size of the memory buffer containing MBMD, in bytes
R9	MIG_BUFF_LIST	HPA (including HKID bits) of a migration buffer list in shared memory, corresponding to the GPA list pointed by RCX – see 3.12.3.		
R10	MIG_STREAM	Migration stream and resume flag:		
		Bits	Name	Description
		15:0	MIGS_INDEX	Migration stream index
		62:16	RESERVED	Reserved: must be 0
		63	RESUME	0: This is a new invocation 1: This is resumption of a previously interrupted operation
R11	MAC_LIST_0	HPA (including HKID bits) of a MAC list in shared memory, corresponding to the first 256 entries of the GPA list pointed by RCX – see 3.12.3.  If GPA_LIST_INFO.FIRST_ENTRY >= 256, then MAC_LIST_0 is ignored.		
R12	MAC_LIST_1	HPA (including HKID bits) of a MAC list in shared memory, corresponding to the last 256 entries of the GPA list pointed by RCX – see 3.12.3.  If GPA_LIST_INFO.LAST_ENTRY < 256, then MAC_LIST_1 is ignored.		
R14	ATTRIB_LIST	If GPA_LIST_INFO.FORMAT is GPA_AND_L2_ATTR, then R14 contains the HPA (including HKID bits) of a page attributes list in shared memory – see 3.12.4.  Else, R14 is ignored.		

Table 5.15: TDH.EXPORT.MEM Output Operands Definition

Operand	Name	Description
RAX	STATUS	SEAMCALL instruction return code, see 5.3.1
RCX	GPA_LIST_INFO	Same as the input value, except that FIRST_ENTRY is updated to the index of the next entry to be processed.  If all entries have been processed, FIRST_ENTRY is updated to (LAST_ENTRY + 1) Modulo 512.
RDX	NUM_EXPORTED	If TDH.EXPORT.MEM is successful, RDX returns the number of exported 4KB migration buffers, including: <ul style="list-style-type: none"> <li>The GPA list page</li> <li>One or two MAC pages (depending on GPA_LIST_INFO.FIRST_ENTRY and GPA_LIST_INFO.LAST_ENTRY)</li> <li>Up to 512 encrypted memory pages</li> </ul> If TDH.EXPORT.MEM is not successful, RDX is unmodified.
AVX, AVX2 and AVX512 state		May be reset to the architectural INIT state
Other		Unmodified

### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

[TO BE UPDATED]

TDH.EXPORT.MEM exports a list of up to 512 TD private 4KB pages as a migration bundle, which includes an MBMD, set of 4KB pages encrypted with the migration session key, a 4KB page containing the GPA and attributes list, an optional 4KB containing page attributes list, and two 4KB pages containing page MACs.

A GPA list is provided as an input. For each page in the list, the requested operation may be either to export the page, to re-export a previously exported page or to cancel a previous page export. It is also possible to skip entries in the least by requesting no operation for specific entries. The GPA list format is described in 3.12.2. It is designed to be compatible with TDH.EXPORT.BLOCKW.

A list of 4KB page buffers is provided as an input. In case no data is exported (PENDING page, page cancellation or some state error) TDH.EXPORT.PAGE marks the applicable list entry as invalid.

**Blocking and TLB Tracking:** If the TD may be running, the exported pages must be blocked and TLB tracked. Else (e.g., the TD has been paused for export), no blocking and tracking is required.

**Export Error:** If a page can't be exported, TDH.EXPORT.MEM marks its GPA list entry as unsuccessful, but does not abort. It continues to the next entry, if applicable. The return status in RAX indicates the number of such cases encountered during operation.

**Interruptibility:** TDH.EXPORT.MEM is interruptible. If a pending interrupt is detected during operation, TDH.EXPORT.MEM returns with a TDX\_INTERRUPTED\_RESUMABLE status in RAX. RCX is updated with the next list entry index to process, so the host VMM may re-invoke TDH.EXPORT.MEM immediately after handling the interrupt, keeping the same inputs except setting R10.RESUME to 1.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

Table 5.16: TDH.EXPORT.MEM Operands Information Definition

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	GPA List page	GPA_LIST	RW	Shared	4KB	None	None	None
Explicit	RDX	HPA	TDR page	TDR	R	Opaque	4KB	Shared	Shared	Shared
Explicit	R8	HPA	Memory to use for MBMD	MBMD	RW	Shared	128B	None	None	None
Explicit	R9	HPA	Migration buffer list	PAGE_LIST	R	Shared	4KB	None	None	None
Explicit	R10	Index	Mig. Stream context	Mig. Stream context	RW	Opaque	N/A	Exclusive	N/A	N/A
Explicit	R11	HPA	MAC list page 1	MAC list	RW	Shared	4KB	None	None	None
Explicit	R12	HPA	MAC list page 2	MAC list	RW	Shared	4KB	None	None	None
Explicit	R14	HPA	attributes list page	page attributes	R	Shared	4KB	None	None	None
Explicit	N/A	GPA	TD private pages (via GPA list)	Blob	None	Private	4KB	None	None	None
Explicit	N/A	HPA	Migration buffer pages (via page list)	Blob	RW	Shared	4KB	None	None	None
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	R	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	N/A	Migration context	N/A	RW	Opaque	N/A	None	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT tree	N/A	RW	Private	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT entry	SEPT Entry	RW	Private	N/A	Exclusive(h)	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT trees	N/A	RW	Private	N/A	Shared(i)	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT entries	SEPT Entry	RW	Private	N/A	Exclusive(i)	N/A	N/A

TDH.EXPORT.MEM checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

- 5 The function checks the following conditions:
  1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
  2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
  3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
  4. TDCS is allocated (TDR.NUM\_TDCX is the required number).

5. An export session is in progress.
6. The migration stream index is lower than TDCS.NUM\_MIGS.
7. The buffer provided for MBMD is large enough.

If successful, the function does the following:

- 5 8. If the RESUME input flag is 0, indicating that this is a new (not resumed) invocation of TDH.EXPORT.MEM:
  - 8.1. If the migration stream has not been initialized, initialize it.
  - 8.2. Increment the migration stream context's IV\_COUNTER
  - 8.3. Build the 96b IV for this migration bundle by concatenating the stream index and the stream context's IV\_COUNTER.
  - 10 8.4. Build a local copy of the MBMD.
  - 8.5. Calculate the MBMD MAC.
  - 8.6. Write the MBMD to memory.
9. Else (this is a resumption of a previously interrupted TDH.EXPORT.MEM):
  - 9.1. Check that the migration stream has been initialized.
  - 15 9.2. Check that the stream context's INTERRUPTED\_FUNC contains TDH.EXPORT.MEM's leaf number.
  - 9.3. Check that the current inputs are the same as saved in the stream context when the function was interrupted.

If passed, process the GPA list:

**Note:** Error conditions that impact a single GPA list entry do not cause an abort of TDH.EXPORT.MEM. Instead, the GPA list entry is updated with a proper status code, and the corresponding migration buffer list entry is marked as invalid.

10. For each entry in the GPA list, starting with RCX.FIRST\_ENTRY and ending with RCX.LAST\_ENTRY:
  - 10.1. If no operation is requested, mark the corresponding migration buffer list entry as invalid and continue to the next GPA list entry.
  - 10.2. Check the GPA list entry fields value.

If passed:

- 10.3. Walk the L1 SEPT tree based on the GPA and level operands and find the leaf entry for the page.
- 10.4. Check that the SEPT entry state is allowed for page export.
- 10.5. If the TD is running (TDCS.OP\_STATE is LIVE\_EXPORT) and TLB tracking is required, check TLB tracking vs. TDCS.BW\_EPOCH set previously by TDH.EXPORT.BLOCKW.
- 10.6. Check that the requested operation is allowed in the current export phase.
- 10.7. Check that the requested operation is allowed for the current SEPT entry state.
- 10.8. If the page has any L2 mappings, check that a page attributes list has been provided (GPA\_LIST\_INFO.FORMAT is GPA\_AND\_ATTR).

**Note:** TDH.EXPORT.MEM does not check that the page has not been exported in the current migration epoch during the in-order phase. This is checked when the page is imported by TDH.IMPORT.MEM.

If passed:

- 10.9.1. For each L2 mapping of the page:
  - 10.9.1.1. Walk the L2 SEPT based on the GPA and level operands and find the leaf entry for the page (if any).
- 10.9.2. Update the page attributes list entry.
- 10.9.3. Update the L1 SEPT entry state, GPA list entry and migration buffer list entry.

10.10. Else:

- 10.10.1. Update the GPA list entry and migration buffer list entry with error status.
- 10.10.2. For each L2 mapping of the page:
  - 10.10.2.1. Walk the L2 SEPT based on the GPA and level operands and find the leaf entry for the page (if any).
- 10.10.3. Update the page attributes list entry.

10.11. Increment the migration stream context's IV\_COUNTER

10.12. Build the 96b IV for this page by concatenating 0 as the direction bit, the stream index and the stream context's IV\_COUNTER.

10.13. Accumulate page MAC based on the GPA list entry.

10.14. If a page attributes list has been provided, accumulate MAC based on the page attributes list entry.

10.15. If the page content is to be exported, encrypt the TD private page into the migration buffer and accumulate MAC.

10.16. Write the page MAC to the MAC list.

10.17. If this is not the last round and there is a pending interrupt:

10.17.1. Save intermediate state in the migration stream context.

10.17.2. Terminate TDH.EXPORT.MEM with a TDX\_INTERRUPTED\_RESUMABLE status.

10.18. Else, advance to the next entry in the GPA list, if applicable.

5 11. Once the GPA list has been fully processed, update the migration stream next MB counter field.

### Completion Status Codes

**Table 5.17: TDH.EXPORT.MEM Completion Status Codes (Returned in RAX) Definition** [TO BE COMPLETED]

Completion Status Code	Description
TDX_INTERRUPTED_RESUMABLE	
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_OPERAND_PAGE_INVALID	
TDX_SUCCESS	Operation is successful. <b>Note:</b> Processing of some GPA list entries may have encountered errors, but this did not cause an abort of the overall operation. The number such errors is reported in the lower 32 bits of the completion status.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	



### 5.3.5. NEW: TDH.EXPORT.PAUSE Leaf

TDH.EXPORT.PAUSE starts the TDX-enforced blackout period on the source platform, where the source TD is paused.

**Table 5.18: TDH.EXPORT.PAUSE Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	HPA of Source TD TDR page (HKID bits must be 0)		

**Table 5.19: TDH.EXPORT.PAUSE Output Operands Definitions**

Operand	Description
RAX	SEAMCALL instruction return code, see 5.3.1
Other	Unmodified

#### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.EXPORT.PAUSE starts the Live Migration Blackout period on the source platform.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.20: TDH.EXPORT.PAUSE Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDR page	TDR	R	Opaque	4KB	Exclusive	Shared	Shared
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	N/A	Migration context	N/A	RW	Opaque	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	N/A	TDCS Epoch Tracking Fields	N/A	RW	Opaque	N/A	Exclusive(i)	N/A	N/A

- 15 TDH.EXPORT.PAUSE checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

The function checks the following conditions:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
- 5 4. TDCS is allocated (TDR.NUM\_TDCX is the required number).
5. TDCS.OP\_STATE is LIVE\_EXPORT.

**Note:** All TD VCPUs have stopped executing and no other TD-specific SEAMCALL is running. This is implicit, since TDH.EXPORT.PAUSE has an exclusive access to TDR and TDCS.

If successful, the function does the following:

- 10 6. Increment the TD's epoch counter (TDCS.TD\_EPOCH).

**Note:** This allows memory management operations to skip the need for blocking and TLB tracking while the TD is paused. If the export session is aborted, the first TDH.VP.ENTER on each VCPU will flush TLB.

7. Set the TDCS.OP\_STATE to PAUSED\_EXPORT.

### Completion Status Codes

15 **Table 5.21: TDH.EXPORT.PAUSE Completion Status Codes (Returned in RAX) Definition** [TO BE COMPLETED]

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	Operation is successful
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

### 5.3.6. NEW: TDH.EXPORT.RESTORE Leaf

TDH.EXPORT.RESTORE restores a list of TD private 4KB pages' Secure EPT entry states after an export abort.

**Table 5.22: TDH.EXPORT.RESTORE Input Operands Definition**

Operand	Name	Description		
RAX	Leaf and Version	SEAMCALL instruction leaf number and version, see 5.3.1		
		Bits	Field	Description
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
		63:24	Reserved	Must be 0
RCX	GPA_LIST_INFO	GPA_LIST_INFO: HPA of a GPA list page in shared memory, and first and last entries to process, as defined in 3.12.2 FORMAT must be GPA_ONLY.		
RDX	TDR	HPA of the source TD's TDR page (HKID bits must be 0)		

**Table 5.23: TDH.EXPORT.RESTORE Output Operands Definition**

Operand	Name	Description
RAX	STATUS	SEAMCALL instruction return code, see 5.3.1
RCX	GPA_LIST_INFO	Same as the input value, except that FIRST_ENTRY is updated to the index of the next entry to be processed. If all entries have been processed, FIRST_ENTRY is updated to (LAST_ENTRY + 1) Modulo 512.
Other		Unmodified

#### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.EXPORT.RESTORE restores a list of TD private 4KB pages' Secure EPT entry states after an aborted export session. It reverts each L1 Secure EPT entry and any applicable L2 Secure EPT entries to their original non-exported state.

**List Entry Error:** If a page's Secure EPT entry can't be restored, TDH.EXPORT.RESTORE marks its GPA list entry as unsuccessful. List process is not aborted; it continues to the next entry, if applicable. The return status in RAX indicates the number of such cases encountered during operation.

- 15 **Interruptibility:** TDH.EXPORT.RESTORE is interruptible. If a pending interrupt is detected during operation, TDH.EXPORT.RESTORE returns with a TDX\_INTERRUPTED\_RSUMABLE status in RAX. RCX is updated with the next list entry index to process, so the host VMM may re-invoke TDH.EXPORT.RESTORE immediately after handling the interrupt.

- 20 To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

Table 5.24: TDH.EXPORT.RESTORE Operands Information Definition

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	GPA List page	GPA_LIST	RW	Shared	4KB	None	None	None
Explicit	RDX	HPA	TDR page	TDR	R	Opaque	4KB	Shared	Shared	Shared
Explicit	N/A	GPA	TD private pages (via GPA list)	Block	None	Private	4KB	None	None	None
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	R	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	N/A	Migration context	N/A	RW	Opaque	N/A	None	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT tree	N/A	RW	Private	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT entry	SEPT Entry	RW	Private	N/A	Exclusive(h)	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT trees	N/A	RW	Private	N/A	Shared(i)	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT entries	SEPT Entry	RW	Private	N/A	Exclusive(i)	N/A	N/A

TDH.EXPORT.RESTORE checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

- 5 The function checks the following conditions:
  1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
  2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
  3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
  4. TDCS is allocated (TDR.NUM\_TDCX is the required number).
- 10 5. TDCS.OP\_STATE is RUNNABLE.

If passed, process the GPA list:

**Note:** Error conditions that impact a single GPA list entry do not cause an abort of TDH.EXPORT.RESTORE. Instead, the GPA list entry is updates with a proper status code, and the corresponding migration buffer list entry is marked as invalid.

- 15 6. For each entry in the GPA list, starting with RCX.FIRST\_ENTRY and ending with RCX.LAST\_ENTRY, if OPERATION indicates a RESTORE request:
  - 6.1. Check the GPA list entry fields value.

If passed:

  - 6.2. Walk the L1 SEPT based on the GPA and level operands and find the leaf entry for the page.
  - 20 6.3. Check that the SEPT entry state is one of the EXPORTED\_\* or PENDING\_EXPORTED\_\* states.
  - 6.4. If passed, update the SEPT entry:
    - 6.4.1. Atomically decrement TDCS.MIG\_COUNT.
    - 6.4.2. If the SEPT state is one of the \*\_DIRTY\* states, atomically decrement TDCS.DIRTY\_COUNT.
    - 6.4.3. If the SEPT state is one of the PENDING\_\* states, update it to PENDING. Else, update it to MAPPED.
    - 25 6.4.4. If the page has any L2 mappings, and the SEPT state was one of the non-PENDING but BLOCKEDW states, then for each L2 SEPT:

6.6.1.1. Walk the L2 SEPT tree based on the GPA operand and find the Secure EPT entry to be blocked.

6.6.1.2. If found:

6.6.1.2.1. Save the original value of SEPT.W into SEPT.TDW.

6.6.1.2.2. Clear SEPT.W.

6.5. Else:

6.5.1. Set the GPA list entry's OPERATION field to NOP and STATUS field to the applicable status.

6.6. If this is not the last entry in the list, and there is a pending interrupt, terminate TDH.EXPORT.RESTORE with a TDX\_INTERRUPTED\_RESUMABLE status.

## Completion Status Codes

**Table 5.25: TDH.EXPORT.RESTORE Completion Status Codes (Returned in RAX) Definition [TO BE COMPLETED]**

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_OPERAND_PAGE_INVALID	
TDX_SUCCESS	Operation is successful. <b>Note:</b> Processing of some GPA list entries may have encountered errors, but this did not cause an abort of the overall operation. The number such errors is reported in the lower 32 bits of the completion status.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

**5.3.7. NEW: TDH.EXPORT.STATE.IMMUTABLE Leaf**

TDH.EXPORT.STATE.IMMUTABLE starts a new export session and exports the TD's immutable state as a multi-page migration bundle.

**Table 5.26: TDH.EXPORT.STATE.IMMUTABLE Input Operands Definition**

Operand	Name	Description		
RAX	Leaf and Version	SEAMCALL instruction leaf number and version, see 5.3.1		
		Bits	Field	Description
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
		63:24	Reserved	Must be 0
RCX	TDR	HPA of the source TD's TDR page (HKID bits must be 0)		
R8	MBMD	HPA and size of memory of a memory buffer to use for MBMD:		
		Bits	Name	Description
		51:0	HPA	Bits 51:0 of the host physical address (including HKID bits)
		63:52	Size	Size of the memory buffer containing MBMD, in bytes
R9	PAGE_LIST_INFO	Migration buffers list information – see 3.12.6.1 PAGE_LIST_INFO.FIRST_ENTRY must be 0.		
R10	MIG_STREAM	Migration stream and resume flag:		
		Bits	Name	Description
		15:0	MIGS_INDEX	Migration stream index – must be 0
		62:16	RESERVED	Reserved: must be 0
		63	RESUME	0: This is a new invocation 1: This is resumption of a previously interrupted operation

5

**Table 5.27: TDH.EXPORT.STATE.IMMUTABLE Output Operands Definition**

Operand	Name	Description
RAX	STATUS	SEAMCALL instruction return code, see 5.3.1
RDX	NUM_EXPORTED	Number of exported 4KB migration buffers
AVX, AVX2 and AVX512 state		May be reset to the architectural INIT state
Other		Unmodified

## Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.EXPORT.STATE.IMMUTABLE starts a new export session. It exports the TD's immutable state as a migration bundle, which includes an MBMD and a set of 4KB pages, encrypted with the migration session key. The migration bundle is protected by a MAC that is stored in the MBMD.

TDH.EXPORT.STATE.IMMUTABLE is interruptible. The host VMM is expected to invoke it in a loop until it returns with either a success indication or with a non-recoverable error indication.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.28: TDH.EXPORT.STATE.IMMUTABLE Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDR page	TDR	R	Opaque	4KB	Shared	Shared	Shared
Explicit	R8	HPA	Memory to use for MBMD	MBMD	RW	Shared	128B	None	None	None
Explicit	R9	HPA	Page list	PAGE_LIST	RW	Shared	4KB	None	None	None
Explicit	R10	N/A	Mig. Stream context	Mig. Stream context	RW	Opaque	N/A	Exclusive	N/A	N/A
Explicit	N/A	HPA	Destination pages (via page list)	Blob	RW	Shared	4KB	None	None	None
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Exclusive	N/A	N/A
Implicit	N/A	N/A	Migration context	N/A	RW	Opaque	N/A	Exclusive(i)	N/A	N/A

TDH.EXPORT.STATE.IMMUTABLE checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

The function checks the following conditions:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
4. TDCS is allocated (TDR.NUM\_TDCX is the required number).
5. The TD build and measurement have been finalized, or the TD has been imported, and no export session is in progress (TDCS.OP\_STATE is either RUNNABLE or LIVE\_IMPORT).
6. The TD is migratable: TDCS.ATTRIBUTES.MIGRATABLE is set to 1.
7. Any previous aborted export session has been cleaned up: TDCS.MIG\_COUNT is 0.
8. MIGS\_INDEX is 0.
9. The buffer provided for MBMD is large enough.
10. PAGE\_LIST\_INFO.FIRST\_ENTRY is 0 and the number of pages in the page list is large enough to hold the exported state.

**Note:** The required number of pages is enumerated by TDH.SYS.RD\*.

If successful, the function does the following:

11. If the RESUME input flag is 0, indicating that this is a new invocation of TDH.EXPORT.STATE.IMMUTABLE (not a resumption of a previously interrupted one):

- 11.1. Check that a valid migration decryption key has been set by the Migration TD. If this is not the first migration session, then the migration key must have been set after the previous migration session has started.

**Note:** There is no explicit check that a migration TD is bound; this is implied by the above check.

If passed:

- 11.2. Initialize the migration context in TDCS:

- 11.2.1. Copy the migration keys to working migration keys that will be used throughout the export session.

If passed:

- 11.2.2. Set all migration streams' INITIALIZED flags to 0 and ENABLED flags to 1.

- 11.3. Initialize the current migration stream.

- 11.4. Increment the migration stream context's IV\_COUNTER.

- 11.5. Build the 96b IV for this migration bundle by concatenating 0 as the direction bit, the stream index and the stream context's IV\_COUNTER.

- 11.6. Build the MBMD in the migration stream context.

- 11.7. Accumulate MAC in the stream context based on the MAC'ed fields of MBMD.

12. Else (this is a resumption of a previously interrupted TDH.EXPORT.STATE.IMMUTABLE):

- 12.1. Check that the resumption is valid:

- 12.1.1. The stream context indicates there's a valid interruption state.

- 12.1.2. The current SEAMCALL leaf number and the PAGE\_OR\_LIST operand have the same value as in the interruption state.

- 12.2. Check that the migration stream is enabled.

- 12.3. Restore the previously saved page list index from the migration context.

- 12.4. Build the 96b IV for this migration bundle by concatenating the stream index and the stream context's IV\_COUNTER.

If passed:

13. Repeat exporting 4KB pages until all immutable state is exported or until a pending interrupt is detected:

- 13.1. Get the 4KB next page HPA from it from the page list.

- 13.2. Dump the next set of metadata fields as a metadata list of field sequences, into an internal temporary 4KB buffer.

- 13.3. Use the migration key and the migration stream context to encrypt the 4KB internal buffer into the destination data page and update the MAC calculation.

- 13.4. If all immutable state has been exported:

- 13.4.1. Write the accumulated MAC to the MBMD in the stream context.

- 13.4.2. Write the MBMD to the memory buffer provided by the host VMM.

- 13.4.3. Mark the migration stream context's interrupted state as invalid.

- 13.4.4. Increment the migration stream context's NEXT\_MB\_COUNTER.

- 13.4.5. Set TDCS.TOTAL\_MB to 1.

- 13.4.6. Set TDCS.OP\_STATE to LIVE\_EXPORT.

- 13.4.7. Clear TDCS.DIRTY\_COUNT to 0.

- 13.4.8. Terminate TDH.EXPORT.STATE.IMMUTABLE with a TDX\_SUCCESS status.

- 13.5. Else, if there is a pending interrupt:

- 13.5.1. Save the interruption state to the stream context

- 13.5.2. Terminate TDH.EXPORT.STATE.IMMUTABLE with a TDX\_INTERRUPTED\_RESUMABLE status.

## Completion Status Codes

**Table 5.29: TDH.EXPORT.STATE.IMMUTABLE Completion Status Codes (Returned in RAX) Definition [TO BE COMPLETED]**

Completion Status Code	Description
TDX_INTERRUPTED_RESUMABLE	
TDX_OPERAND_INVALID	



Completion Status Code	Description
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_RND_NO_ENTROPY	Failed to generate a random migration encryption key. This is typically caused by an entropy error of the CPU's random number generator, and may be impacted by RDSEED, RDRAND or PCONFIG executing on other LPs. The operation should be retried.
TDX_SUCCESS	Operation is successful
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

DRAFT

**5.3.8. NEW: TDH.EXPORT.STATE.TD Leaf**

TDH.EXPORT.STATE.TD exports a paused TD's mutable state as a multi-page migration bundle.

**Table 5.30: TDH.EXPORT.STATE.TD Input Operands Definition**

Operand	Name	Description		
RAX	Leaf and Version	SEAMCALL instruction leaf number and version, see 5.3.1		
		Bits	Field	Description
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
		63:24	Reserved	Must be 0
RCX	TDR	HPA of the source TD's TDR page (HKID bits must be 0)		
R8	MBMD	HPA and size of memory of a memory buffer to use for MBMD:		
		Bits	Name	Description
		51:0	HPA	Bits 51:0 of the host physical address (including HKID bits)
		63:52	Size	Size of the memory buffer containing MBMD, in bytes
R9	PAGE_LIST_INFO	Migration buffers list information – see 3.12.6.1 PAGE_LIST_INFO.FIRST_ENTRY must be 0.		
R10	MIG_STREAM	Migration stream and resume flag:		
		Bits	Name	Description
		15:0	MIGS_INDEX	Migration stream index – must be 0
		62:16	RESERVED	Reserved: must be 0
		63	RESUME	0: This is a new invocation 1: This is resumption of a previously interrupted operation

5

**Table 5.31: TDH.EXPORT.STATE.TD Output Operands Definition**

Operand	Name	Description
RAX	STATUS	SEAMCALL instruction return code, see 5.3.1
RDX	NUM_EXPORTED	Number of exported 4KB migration buffers
AVX, AVX2 and AVX512 state		May be reset to the architectural INIT state
Other		Unmodified

## Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.EXPORT.STATE.TD exports the TD's mutable state as a migration bundle, which includes an MBMD and a set of 4KB pages, encrypted with the migration session key. The migration bundle is protected by a MAC that is stored in the MBMD. The TD must have been paused by a TDH.EXPORT.PAUSE.

**Interruptibility:** TDH.EXPORT.STATE.TD is interruptible. The host VMM is expected to invoke it in a loop until it returns with either a success indication or with a non-recoverable error indication.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.32: TDH.EXPORT.STATE.TD Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDR page	TDR	R	Opaque	4KB	Shared	Shared	Shared
Explicit	R8	HPA	Memory to use for MBMD	MBMD	RW	Shared	128B	None	None	None
Explicit	R9	HPA	Page list	PAGE_LIST	R	Shared	4KB	None	None	None
Explicit	R10	Index	Mig. Stream context	Mig. Stream context	RW	Opaque	N/A	Exclusive	N/A	N/A
Explicit	N/A	HPA	Destination pages (via page list)	Blob	RW	Shared	4KB	None	None	None
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	R	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	N/A	Migration context	N/A	RW	Opaque	N/A	None	N/A	N/A

TDH.EXPORT.STATE.TD checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

The function checks the following conditions:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
4. TDCS is allocated (TDR.NUM\_TDCX is the required number).
5. An export session is in progress, and the TD has been paused: TDCS.OP\_STATE is PAUSED\_EXPORT.
6. Migration stream index is 0.
7. The migration stream is enabled and initialized.
8. The buffer provided for MBMD is large enough.
9. PAGE\_LIST\_INFO.FIRST\_ENTRY is 0 and the number of pages in the page list is large enough to hold the exported state.

**Note:** The required number of pages is enumerated by TDH.SYS.RD\*.

If successful, the function does the following:

10. If the RESUME input flag is 0, indicating that this is a new invocation of TDH.EXPORT.STATE.TD (not a resumption of a previously interrupted one):
  - 10.1. Increment the migration stream context's IV\_COUNTER.
  - 10.2. Build the 96b IV for this migration bundle by concatenating 0 as the direction bit, the stream index (0) and the stream context's IV\_COUNTER.
  - 10.3. Build the MBMD in the migration stream context.
  - 10.4. Accumulate MAC in the stream context based on the MAC'ed fields of MBMD.
11. Else (this is a resumption of a previously interrupted TDH.EXPORT.STATE.TD):
  - 11.1. Check that the resumption is valid:
    - 11.1.1. The stream context indicates there's a valid interruption state.
    - 11.1.2. The current SEAMCALL leaf number and the PAGE\_OR\_LIST operand have the same value as in the interruption state.
  - 11.2. Build the 96b IV for this migration bundle by concatenating the stream index and the stream context's IV\_COUNTER.
  - 11.3. Restore the previously saved page list index from the migration context.

If passed:

12. Repeat exporting 4KB pages until all mutable TD state is exported or until a pending interrupt is detected:
  - 12.1. Get the 4KB next page HPA from it from the page list.
  - 12.2. Dump the next set of metadata fields as a metadata list of field sequences, into an internal temporary 4KB buffer.
  - 12.3. Use the migration key and the migration stream context to encrypt the 4KB internal buffer into the destination data page and update the MAC calculation.
  - 12.4. If all TD state has been exported:
    - 12.4.1. Write the accumulated MAC to the MBMD in the stream context.
    - 12.4.2. Write the MBMD to the memory buffer provided by the host VMM.
    - 12.4.3. Mark the migration stream context's interrupted state as invalid.
    - 12.4.4. Increment the migration stream context's NEXT\_MB\_COUNTER.
    - 12.4.5. Increment TDCS.TOTAL\_MB.
    - 12.4.6. Terminate TDH.EXPORT.STATE.TD with a TDX\_SUCCESS status.
  - 12.5. Else, if there is a pending interrupt:
    - 12.5.1. Save the interruption state to the stream context
    - 12.5.2. Terminate TDH.EXPORT.STATE.TD with a TDX\_INTERRUPTED\_RESUMABLE status.

### Completion Status Codes

**Table 5.33: TDH.EXPORT.STATE.TD Completion Status Codes (Returned in RAX) Definition** **[TO BE COMPLETED]**

Completion Status Code	Description
TDX_INTERRUPTED_RESUMABLE	
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	Operation is successful
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

**5.3.9. NEW: TDH.EXPORT.STATE.VP Leaf**

TDH.EXPORT.STATE.VP exports a paused TD's VCPU mutable state as a multi-page migration bundle.

**Table 5.34: TDH.EXPORT.STATE.VP Input Operands Definition**

Operand	Name	Description		
RAX	Leaf and Version	SEAMCALL instruction leaf number and version, see 5.3.1		
		Bits	Field	Description
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
		63:24	Reserved	Must be 0
RCX	TDVPR	HPA of the source TD VCPU's TDVPR page (HKID bits must be 0)		
R8	MBMD	HPA and size of memory of a memory buffer to use for MBMD:		
		Bits	Name	Description
		51:0	HPA	Bits 51:0 of the host physical address (including HKID bits)
		63:52	Size	Size of the memory buffer containing MBMD, in bytes
R9	PAGE_LIST_INFO	Migration buffers list information – see 3.12.6.1 PAGE_LIST_INFO.FIRST_ENTRY must be 0.		
R10	MIG_STREAM	Migration stream and resume flag:		
		Bits	Name	Description
		15:0	MIGS_INDEX	Migration stream index
		62:16	RESERVED	Reserved: must be 0
		63	RESUME	0: This is a new invocation 1: This is resumption of a previously interrupted operation

5

**Table 5.35: TDH.EXPORT.STATE.VP Output Operands Definition**

Operand	Name	Description
RAX	STATUS	SEAMCALL instruction return code, see 5.3.1
RDX	NUM_EXPORTED	Number of exported 4KB migration buffers
AVX, AVX2 and AVX512 state		May be reset to the architectural INIT state
Other		Unmodified

## Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.EXPORT.STATE.VP exports a TD's VCPU mutable state as a migration bundle, which includes an MBMD and a set of 4KB pages, encrypted with the migration session key. The migration bundle is protected by a MAC that is stored in the MBMD. The TD must have been paused by a TDH.EXPORT.PAUSE.

**Interruptibility:** TDH.EXPORT.STATE.VP is interruptible. The host VMM is expected to invoke it in a loop until it returns with either a success indication or with a non-recoverable error indication.

**VCPU Association:** TDH.EXPORT.VP associates the TD VCPU with the current LP. This requires that the VCPU will not be associated with another LP – for details, see the [TDX Module Base Spec].

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.36: TDH.EXPORT.STATE.VP Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDVPR page	TDVPS	RW	Opaque	4KB	Exclusive	Shared	Shared
Explicit	R8	HPA	Memory to use for MBMD	MBMD	RW	Shared	128B	None	None	None
Explicit	R9	HPA	Page list	PAGE_LIST	R	Shared	4KB	None	None	None
Explicit	R10	Index	Mig. Stream context	Mig. Stream context	RW	Opaque	N/A	Exclusive	N/A	N/A
Explicit	N/A	HPA	Destination pages (via page list)	Blob	RW	Shared	4KB	None	None	None
Implicit	N/A	HPA	TDR page	TDR	R	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	R	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	N/A	Migration context	N/A	RW	Opaque	N/A	None	N/A	N/A

TDH.EXPORT.STATE.VP checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

The function checks the following conditions:

1. The TDVPR page metadata in PAMT must be correct (PT must be PT\_TDVPR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
4. TDCS is allocated (TDR.NUM\_TDCX is the required number).
5. An export session is in progress, and the TD has been paused: TDCS.OP\_STATE is PAUSED\_EXPORT.
6. Migration stream index is lower than TDCS.NUM\_MIGS.
7. The migration stream is enabled.
8. The buffer provided for MBMD is large enough.
9. PAGE\_LIST\_INFO.FIRST\_ENTRY is 0 and the number of pages in the page list is large enough to hold the exported state.

**Note:** The required number of pages is enumerated by TDH.SYS.RD\*.

If successful, the function does the following:

10. Associate the VCPU with the current LP, and update TD VMCS using the algorithm described in 5.2.1.

If passed:

11. If the RESUME input flag is 0, indicating that this is a new invocation of TDH.EXPORT.STATE.VP (not a resumption of a previously interrupted one):
  - 11.1. If the migration stream has not been initialized, initialize it.
  - 11.2. Increment the migration stream context's IV\_COUNTER.
  - 11.3. Build the MBMD in the migration stream context.
  - 11.4. Build the 96b IV for this migration bundle by concatenating the stream index and the stream context's IV\_COUNTER.
  - 11.5. Accumulate MAC in the stream context based on the MAC'ed fields of MBMD.
12. Else (this is a resumption of a previously interrupted TDH.EXPORT.STATE.VP):
  - 12.1. Check that the resumption is valid:
    - 12.1.1. The stream context indicates there's a valid interruption state.
    - 12.1.2. The current SEAMCALL leaf number, and the TDVPR HPA and PAGE\_OR\_LIST operands are the same as in the interruption state.
  - 12.2. Increment the migration stream context's IV\_COUNTER.
  - 12.3. Restore the previously saved page list index from the migration context.
13. Repeat exporting 4KB pages until all immutable state is exported or until a pending interrupt is detected:
  - 13.1. Get the 4KB next page HPA from it from the page list.
  - 13.2. Dump the next set of metadata fields as a metadata list of field sequences, into an internal temporary 4KB buffer.
  - 13.3. Use the migration key and the migration stream context to encrypt the 4KB internal buffer into the destination data page and update the MAC calculation.
  - 13.4. If all VCPU state has been exported:
    - 13.4.1. Write the accumulated MAC to the MBMD in the stream context.
    - 13.4.2. Write the MBMD to the memory buffer provided by the host VMM.
    - 13.4.3. Mark the migration stream context's interrupted state as invalid.
    - 13.4.4. Increment the migration stream context's NEXT\_MB\_COUNTER.
    - 13.4.5. Increment TDCS.TOTAL\_MB.
    - 13.4.6. Terminate TDH.EXPORT.STATE.VP with a TDX\_SUCCESS status.
  - 13.5. Else, if there is a pending interrupt:
    - 13.5.1. Save the interruption state to the stream context
    - 13.5.2. Terminate TDH.EXPORT.STATE.VP with a TDX\_INTERRUPTED\_RESUMABLE status.

### Completion Status Codes

**Table 5.37: TDH.EXPORT.STATE.VP Completion Status Codes (Returned in RAX) Definition** [TO BE COMPLETED]

Completion Status Code	Description
TDX_INTERRUPTED_RESUMABLE	
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	Operation is successful
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

**5.3.10. NEW: TDH.EXPORT.TRACK Leaf**

TDH.EXPORT.TRACK ends the current in-order export phase epoch and either starts a new epoch or starts the out-of-order export phase. Generate an epoch token to be exported to the destination platform.

**Table 5.38: TDH.EXPORT.TRACK Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	<b>Bits</b>	<b>Field</b>	<b>Description</b>
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	HPA of the source TD's TDR page (HKID bits must be 0)		
R8	HPA and size of memory of a memory buffer to use for MBMD:		
	<b>Bits</b>	<b>Name</b>	<b>Description</b>
	51:0	HPA	Bits 51:0 of the host physical address (including HKID bits)
	63:52	Size	Size of the memory buffer containing MBMD, in bytes
R10	Migration stream and flags:		
	<b>Bits</b>	<b>Name</b>	<b>Description</b>
	15:0	MIGS_INDEX	Migration stream index – must be 0
	62:16	RESERVED	Reserved: must be 0
	63	IN_ORDER_DONE	Indicates that the in-order export phase is done

5

**Table 5.39: TDH.EXPORT.TRACK Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code, see 5.3.1
AVX, AVX2 and AVX512 state	May be reset to the architectural INIT state
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

If R10.IN\_ORDER\_DONE is 0, TDH.EXPORT.TRACK starts a new export epoch.

Else (R10.IN\_ORDER\_DONE is 1), TDH.EXPORT.TRACK checks that no memory exported so far needs to be re-exported. If so, it ends the in-order export phase and starts the out-of-order phase.

In both cases, TDH.EXPORT.TRACK generates an epoch token, to be exported on the specified migration stream.

10



To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.40: TDH.EXPORT.TRACK Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDR page	TDR	R	Opaque	4KB	Shared	Shared	Shared
Explicit	R8	HPA	Memory to use for MBMD	MBMD	RW	Shared	128B	None	None	None
Explicit	R10	Index	Mig. Stream context	Mig. Stream context	RW	Opaque	N/A	Exclusive	N/A	N/A
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Exclusive	N/A	N/A
Implicit	N/A	N/A	Migration context	N/A	RW	Opaque	N/A	None	N/A	N/A

- 5 TDH.EXPORT.TRACK checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

The function checks the following conditions:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
- 10 3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
4. TDCS is allocated (TDR.NUM\_TDCX is the required number).
5. An export session is in the in-order phase: TDCS.OP\_STATE is either LIVE\_EXPORT or PAUSED\_EXPORT.
6. The migration stream index is 0.
7. The migration stream is initialized.
- 15 8. The buffer provided for MBMD is large enough.

If successful, the function does the following:

9. If R10.IN\_ORDER\_DONE is 0:
  - 9.1. Increment TDCS.MIG\_EPOCH
10. Else (R10.IN\_ORDER\_DONE is 1):
  - 20 10.1. Check that an export session is in the in-order phase and the TD has been paused: TDCS.OP\_STATE is PAUSED\_EXPORT.
  - 10.2. Check that TDCS.DIRTY\_COUNT is 0, indicating that no unexported newer versions of any memory page exported so far remain. Memory pages that have not yet been exported may remain, and may later be exported (out-of-order).
  - 25 10.3. **[TBD – REMOVE THIS CHECK]** The TD mutable state has been exported (by TDH.EXPORT.STATE.TD).
- If passed:
  - 10.4. Start the out-of-order phase:
    - 10.4.1. Set TDCS.OP\_STATE to POST\_EXPORT.
    - 10.4.2. Set TDCS.MIG\_EPOCH to 0xFFFFFFFF.
  - 30 11. Increment the migration stream context's IV\_COUNTER.
  12. Build the 96b IV for this migration bundle by concatenating the stream index and the stream context's IV\_COUNTER.
  13. Create an epoch token MBMD with the following fields:
    - 13.1. The number of the new epoch that have just begun. Bit 63 indicates the beginning of the out-of-order phase.
    - 13.2. The total number of migration bundles (including the current one) that have been exported in the current migration session.
  - 35 14. Accumulate MAC based on the MAC'ed fields of MBMD and write to the MBMD's MAC field's value.

15. Write the MBMD to the provided memory buffer.

#### Completion Status Codes

**Table 5.41: TDH.EXPORT.TRACK Completion Status Codes (Returned in RAX) Definition** [TO BE COMPLETED]

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	Operation is successful
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

DRAFT

**5.3.11. NEW: TDH.EXPORT.UNBLOCKW Leaf**

Remove the write-blocking of a 4KB TD private page previously blocked by TDH.EXPORT.BLOCKW.

**Table 5.42: TDH.EXPORT.UNBLOCKW Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	EPT mapping information:		
	Bits	Name	Description
	2:0	Level	Level of the Secure EPT entry that maps the page to be blocked for writing – see 3.6.1: must be 0 (4KB)
	11:3	Reserved	Reserved: must be 0
	51:12	GPA	Bits 51:12 of the GPA to be unblocked for writing
	63:52	Reserved	Reserved: must be 0
RDX	Host physical address of the parent TDR page (HKID bits must be 0)		

**Table 5.43: TDH.EXPORT.UNBLOCKW Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code, see 5.3.1
RCX	Extended error information part 1 In case of EPT walk error, Secure EPT entry where the error was detected In other cases, RCX returns 0
RDX	Extended error information part 2 In case of EPT walk error, EPT level where the error was detected In other cases, RDX returns 0
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.EXPORT.UNBLOCKW finds the write blocked Secure EPT entry for the given GPA and level. It verifies that the entry has been blocked for writing and TLB tracking has been done, then marks the entry as non-blocked for writing (MAPPED, PENDING, EXPORTED\_DIRTY or PENDING\_EXPORTED\_DIRTY as appropriate). If the page has any L2 mappings, TDH.EXPORT.UNBLOCKW unblocks them.

- 15 To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

Table 5.44: TDH.EXPORT.UNBLOCKW Operands Information Definition

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	GPA and Level	Secure EPT page or TD private page	Blob	None	Private	$2^{12+9*Level}$ Bytes	None	None	None
Explicit	RDX	HPA	TDR page	TDR	R	Opaque	4KB	Shared	Shared	Shared
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	R	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	N/A	Migration context	N/A	RW	Opaque	N/A	None	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT tree	N/A	RW	Private	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT entry	SEPT Entry	RW	Private	N/A	Exclusive(h)	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT trees	N/A	RW	Private	N/A	Shared(i)	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT entries	SEPT Entry	RW	Private	N/A	Exclusive(i)	N/A	N/A

TDH.EXPORT.UNBLOCKW checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

- 5 The function checks the following conditions:
  1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
  2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
  3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
  4. TDCS is allocated (TDR.NUM\_TDCX is the required number).
- 10 5. Either of the following is true:
  - 5.1. An export session is in progress.
  - 5.2. The TD is allowed to run (TDCS.OP\_STATE is either RUNNABLE, LIVE\_EXPORT, PAUSED\_EXPORT or POST\_EXPORT). In these states, TDH.EXPORT.UNBLOCKW is used to clean up after an aborted export session.
6. The specified level is 0 (4KB).
- 15 If successful, the function does the following:
  7. Walk the Secure EPT based on the GPA operand and find the Secure EPT page or TD private page to be unblocked for writing.
  8. Check the Secure EPT entry state is blocked for writing: BLOCKEDW, PENDING\_BLOCKEDW, EXPORTED\_DIRTY\_BLOCKEDW or PENDING\_EXPORTED\_DIRTY\_BLOCKEDW.
- 20 9. If the TD is allowed to run, check that TLB tracking was done.
- If passed:
  10. If the page state is not one of the PENDING\* states:
    - 10.1. Restore the original value of SEPT.W from SEPT.TDW.
  11. If the page has not been exported (Secure EPT entry state is BLOCKEDW or PENDING\_BLOCKEDW), unblock the Secure EPT entry for writing by atomically setting its state to MAPPED or PENDING, respectively.
  - 25 12. Else (Secure EPT entry state is EXPORTED\_DIRTY\_BLOCKEDW or PENDING\_EXPORTED\_DIRTY\_BLOCKEDW):
    - 12.1. Unblock the Secure EPT entry for writing by atomically setting its state to EXPORTED\_DIRTY or PENDING\_EXPORTED\_DIRTY, respectively.

12.2. Atomically increment TDCS.DIRTY\_COUNT.

13. If the updated page state is MAPPED or EXPORTED\_DIRTY, then for each L2 mapping of the page:

13.1. Walk the L2 SEPT tree based on the GPA operand and find the L2 Secure EPT entry to be unblocked for writing.

13.2. Restore the original value of SEPT.W from SEPT.TDW.

13.3. Set the L2 SEPT entry state to L2\_MAPPED

**Note:** If the updated page state is one of the PENDING\* states, then the L2 SEPT entry state is already L2\_BLOCKED, no change is required.

### Completion Status Codes

**Table 5.45: TDH.EXPORT.UNBLOCKW Completion Status Codes (Returned in RAX) Definition [TO BE COMPLETED]**

Completion Status Code	Description
TDX_EPT_WALK_FAILED	
TDX_NOT_WRITE_BLOCKED	
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.EXPORT.UNBLOCKW is successful
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	
TDX_TLB_TRACKING_NOT_DONE	

**5.3.12. NEW: TDH.IMPORT.ABORT Leaf**

Abort an import session; after this the target TD can only be destroyed. Generate an abort token that is to be consumed by the source platform.

**Table 5.46: TDH.IMPORT.ABORT Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	HPA of the source TD's TDR page (HKID bits must be 0)		
R8	HPA and size of memory of a memory buffer to use for MBMD:		
	Bits	Name	Description
	51:0	HPA	Bits 51:0 of the host physical address (including HKID bits)
	63:52	Size	Size of the memory buffer containing MBMD, in bytes
R10	Migration stream index – must be 0		

**Table 5.47: TDH.IMPORT.ABORT Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code, see 5.3.1
AVX, AVX2 and AVX512 state	May be reset to the architectural INIT state
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.IMPORT.ABORT generates an abort token MBMD and sets the destination TD's OP\_STATE to IMPORT\_FAILED. In this state, the destination TD will not run; it can only be destroyed. This is indicated by the FATAL bit (61) of the completion status returned in RAX.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

Table 5.48: TDH.IMPORT.ABORT Operands Information Definition

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDR page	TDR	R	Opaque	4KB	Exclusive	Shared	Shared
Explicit	R8	HPA	Memory to use for MBMD	MBMD	RW	Shared	128B	None	None	None
Explicit	R10	Index	Mig. Stream context	Mig. Stream context	RW	Opaque	N/A	Exclusive	N/A	N/A
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	N/A	Migration context	N/A	RW	Opaque	N/A	Exclusive(i)	N/A	N/A

TDH.IMPORT.ABORT checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

- 5 The function checks the following conditions:
1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
  2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
  3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
  4. TDCS is allocated (TDR.NUM\_TDCX is the required number).
  5. An import session is in progress but has not been committed yet (TDCS.OP\_STATE is one of MEMORY\_IMPORT, STATE\_IMPORT, POST\_IMPORT or FAILED\_IMPORT).
  6. The migration stream index is 0.
  7. The buffer provided for MBMD is large enough.

If successful, the function does the following:

8. Set TDCS.OP\_STATE to FAILED\_IMPORT.
9. If the migration stream has not been initialized, initialize it.
10. Increment the stream context's IV\_COUNTER.
11. Build the 96b IV for this migration bundle by concatenating the stream index and the stream context's IV\_COUNTER.
12. Create an abort token MBMD.
13. Accumulate MAC based on the MAC'ed fields of MBMD and write to the MBMD's MAC field's value.
14. Write the MBMD to the provided memory buffer.
15. Increment the stream context's NEXT\_MB\_COUNTER.

#### Completion Status Codes

Table 5.49: TDH.IMPORT.ABORT Completion Status Codes (Returned in RAX) Definition [TO BE COMPLETED]

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	

Completion Status Code	Description
TDX_SUCCESS	Operation is successful
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

DRAFT



**5.3.13. NEW: TDH.IMPORT.COMMIT Leaf**

Commit an import session and allow the imported TD to run.

**Table 5.50: TDH.IMPORT.COMMIT Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	HPA of the source TD's TDR page (HKID bits must be 0)		

**Table 5.51: TDH.IMPORT.COMMIT Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code, see 5.3.1
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.IMPORT.COMMIT commits an import session and allows the important TD to run. Post-copy memory import may continue.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.52: TDH.IMPORT.COMMIT Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDR page	TDR	R	Opaque	4KB	Exclusive	Shared	Shared
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	N/A	Migration context	N/A	RW	Opaque	N/A	Exclusive(i)	N/A	N/A

TDH.IMPORT.COMMIT checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

The function checks the following conditions:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).

3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
4. TDCS is allocated (TDR.NUM\_TDCX is the required number).
5. An import session is in the out-of-order phase: TDCS.OP\_STATE is POST\_IMPORT.

If successful, the function does the following:

5. 6. Set TDCS.OP\_STATE to LIVE\_IMPORT.

#### Completion Status Codes

**Table 5.53: TDH.IMPORT.COMMIT Completion Status Codes (Returned in RAX) Definition** **[TO BE COMPLETED]**

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	Operation is successful
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

DRAFT

**5.3.14. NEW: TDH.IMPORT.END Leaf**

End an import session.

**Table 5.54: TDH.IMPORT.END Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	HPA of the source TD's TDR page (HKID bits must be 0)		

**Table 5.55: TDH.IMPORT.END Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code, see 5.3.1
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.IMPORT.END ends an import session and allows the important TD to run (if not already allowed by TDH.IMPORT.COMMIT).

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.56: TDH.IMPORT.END Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDR page	TDR	R	Opaque	4KB	Exclusive	Shared	Shared
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	N/A	Migration context	N/A	RW	Opaque	N/A	Exclusive(i)	N/A	N/A

TDH.IMPORT.END checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

The function checks the following conditions:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).

3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
4. TDCS is allocated (TDR.NUM\_TDCX is the required number).
5. An import session is in the out-of-order phase.

If successful, the function does the following:

5. 6. Set TDCS.OP\_STATE to RUNNABLE.

#### Completion Status Codes

**Table 5.57: TDH.IMPORT.END Completion Status Codes (Returned in RAX) Definition [TO BE COMPLETED]**

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	Operation is successful
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

DRAFT

**5.3.15. NEW: TDH.IMPORT.MEM Leaf**

TDH.IMPORT.MEM imports a list of TD private pages contents and/or cancellation requests based on a migration bundle in shared memory.

**Table 5.58: TDH.IMPORT.MEM Input Operands Definition**

Operand	Name	Description		
RAX	Leaf and Version	SEAMCALL instruction leaf number and version, see 5.3.1		
		Bits	Field	Description
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
		63:24	Reserved	Must be 0
RCX	GPA_LIST_INFO	HPA of a GPA list page in shared memory, and first and last entries to process, as defined in 3.12.2  On a new invocation, FIRST_ENTRY must be 0. On a resumed invocation, FIRST_ENTRY must be the index of the next GPA list entry to export.		
RDX	TDR	HPA of the destination TD's TDR page (HKID bits must be 0)		
R8	MBMD	HPA and size of memory of an MBMD structure in memory:		
		Bits	Name	Description
		51:0	HPA	Bits 51:0 of the host physical address (including HKID bits)
		63:52	Size	Size of the memory buffer containing MBMD, in bytes
R9	MIG_BUFF_LIST	HPA (including HKID bits) of a migration buffer list in shared memory, corresponding to the GPA list pointed by RCX – see 3.12.3.  No migration buffers are required for PENDING pages and for migration cancellation requests. The list entries for such pages are skipped.		
R10	MIG_STREAM	Migration stream and resume flag:		
		Bits	Name	Description
		15:0	MIGS_INDEX	Migration stream index
		62:16	RESERVED	Reserved: must be 0
		63	RESUME	0: This is a new invocation 1: This is resumption of a previously interrupted operation
R11	MAC_LIST_0	HPA (including HKID bits) of a MAC list in shared memory, corresponding to the first 256 entries of the GPA list pointed by RCX – see 3.12.3.  If GPA_LIST_INFO.FIRST_ENTRY >= 256, then MAC_LIST_0 is ignored.		
R12	MAC_LIST_1	HPA (including HKID bits) of a MAC list in shared memory, corresponding to the last 256 entries of the GPA list pointed by RCX – see 3.12.3.  If GPA_LIST_INFO.LAST_ENTRY < 256, then MAC_LIST_1 is ignored.		

R13	PAGE_LIST	<p>If in-place import is requested for all pages imported for the first-time in the current import session, or for the first-time after a previous import cancellation, R13 should be set to NULL_PA (all 1's).</p> <p>Otherwise, if some pages are to be imported in a non-in-place mode, R13 should be set to the HPA (including HKID bits) of a destination page list in shared memory, corresponding to the GPA list pointed by RCX – see 3.10.6. The page list allows selecting in-place or non-in-place import for each page imported for the first-time in the current import session, or for the first-time after a previous import cancellation:</p> <ul style="list-style-type: none"> <li>To select in-place import, the page list entry's INVALID bit should be set to 1 (it is possible to set the whole entry to NULL_PA).</li> <li>To select non-in-place import, the page list entry should be set to the HPA (including HKID) of the page to become a new TD private page.</li> </ul>
R14	ATTRIB_LIST	<p>If GPA_LIST_INFO.FORMAT is GPA_AND_L2_ATTR, then R14 contains the HPA (including HKID bits) of a page L2 attributes list in shared memory – see 3.12.4.</p> <p>Else, R14 is ignored.</p>

Table 5.59: TDH.IMPORT.MEM Output Operands Definition

Operand	Name	Description
RAX	STATUS	SEAMCALL instruction return code, see 5.3.1
RCX	GPA_LIST_INFO	<p>Same as the input value, except that FIRST_ENTRY is updated to the index of the next entry to be processed.</p> <p>If all entries have been processed, FIRST_ENTRY is updated to (LAST_ENTRY + 1) Modulo 512.</p>
AVX, AVX2 and AVX512 state		May be reset to the architectural INIT state
Other		Unmodified

### Leaf Function Description

- 5 **Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.IMPORT.MEM imports a list of up to 512 TD private 4KB pages based on a migration bundle, which includes an MBMD, set of 4KB pages encrypted with the migration session key, a 4KB page containing the GPA and attributes list, an optional 4KB containing page attributes list, and two 4KB pages containing page MACs.

- 10 For each page in the migration bundle's GPA list, the requested operation may either be to import the page, to re-import a newer version of the page (after a previous import) or to cancel a previous page import. It is also possible to skip entries in the list by requesting no operation for specific entries. The GPA list format is described in 3.12.2.

**Re-Import:** Re-import is only allowed during the in-order import phase. The imported pages replace an older version of the same pages, as long as the SEPT entry state is compatible:

- 15
- If the old SEPT state is PENDING, it may be overwritten by a new version that is either PENDING or MAPPED.
  - If the old SEPT state is MAPPED, it may be overwritten by a newer version that is MAPPED.

Page attributes (e.g., RWX etc.) of a new page version may be different than those of a previously imported version.

- 20 If the out-of-order import phase, the imported pages may not overwrite an older version of the same pages.

**In-Place Import:** First-time import of a page during the current import session, or following a previous import cancellation, may be done in-place; the same physical pages that are provided as input are converted to TD private pages. Alternatively, a list of 4KB pages to be used as the destination TD new private pages may be provided. In any case, either a migration buffer or a new page must be provided, even if the imported page is PENDING and no content is imported.

Re-import of a page is always done over the TD private page that holds the previously imported version.

**Import Abort:** In many cases, an error during import aborts the import session because the memory state of the imported TD can't be guaranteed to be correct.

If the import session has not been committed yet (by THD.IMPORT.COMMIT) and not yet entered the LIVE\_IMPORT state where the TD is allowed to run, a failed TDH.IMPORT.MEM is considered fatal to the import session (except in cases where the imported TD state has not been modified). The target TD is marked as IMPORT\_FAILED and, by design, will not run. This is indicated by the FATAL bit (61) of the completion status returned in RAX.

If the import session has been committed and the entered the LIVE\_IMPORT state where the TD is allowed to run, then a failed TDH.IMPORT.MEM terminates the import session (except in cases where the imported TD state has not been modified) but does not impact the TD's ability to run. This is indicated by the FATAL bit (61) of the completion status returned in RAX.

**Interruptibility:** TDH.IMPORT.MEM is interruptible. If a pending interrupt is detected during operation, TDH.IMPORT.MEM returns with a TDX\_INTERRUPTED\_RSUMABLE status in RAX. RCX is updated with the next list entry index to process, so the host VMM may re-invoke TDH.IMPORT.MEM immediately after handling the interrupt, keeping the same inputs except setting R10.RESUME to 1.

**Cache Flush and Init:** After any private pages have been removed by a CANCEL operation, the host VMM should flush the physical pages' cache lines and initialize their content before they are reused, as described in the [Base Spec].

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.60: TDH.IMPORT.MEM Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	GPA List page	GPA_LIST	RW	Shared	4KB	None	None	None
Explicit	RDX	HPA	TDR page	TDR	RW	Opaque	4KB	Shared	Shared	Shared
Explicit	R8	HPA	MBMD	MBMD	R	Shared	128B	None	None	None
Explicit	R9	HPA	Migration buffer list	PAGE_LIST	R	Shared	4KB	None	None	None
Explicit	R10	Index	Mig. Stream context	Mig. Stream context	RW	Opaque	N/A	Exclusive	N/A	N/A
Explicit	R11	HPA	MAC list page 1	MAC list	R	Shared	4KB	None	None	None
Explicit	R12	HPA	MAC list page 2	MAC list	R	Shared	4KB	None	None	None
Explicit	R13	HPA	Destination page list	Blob	RW	Private	4KB	Exclusive	Shared	Shared
Explicit	R14	HPA	L2 attributes list page	L2 page attributes	R	Shared	4KB	None	None	None

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	N/A	GPA	TD private pages (via GPA list)	Blob	None	Private	4KB	None	None	None
Explicit	N/A	HPA	Migration buffer pages (via page list)	Blob	RW	Shared	4KB	None	None	None
Explicit	N/A	HPA	Destination pages (via page list)	Blob	RW	Private	4KB	Exclusive	Shared	Shared
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	N/A	Migration context	N/A	RW	Opaque	N/A	None	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT tree	N/A	RW	Private	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT entries	SEPT Entry	RW	Private	N/A	Exclusive	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT trees	N/A	RW	Private	N/A	Shared(i)	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT entries	SEPT Entry	RW	Private	N/A	Exclusive(i)	N/A	N/A

TDH.IMPORT.MEM checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

The function checks the following conditions:

- 5    1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
4. TDCS is allocated (TDR.NUM\_TDCX is the required number).
5. An import session is in progress.
- 10   6. The migration stream index is lower than TDCS.NUM\_MIGS.

If successful, the function does the following:

7. If the RESUME input flag is 0, indicating that this is a new (not resumed) invocation of TDH.IMPORT.MEM:
  - 7.1. Initialize the migration stream if not done so far.
  - 7.2. Copy the MBMD into a temporary buffer.
  - 15   7.3. Check the MBMD fields.
- If passed:
  - 7.4. Build the 96b IV for this migration bundle by concatenating 0 as the direction bit, the stream index and MBMD's MB\_COUNTER.
  - 7.5. Check the MAC based on the MAC'ed fields of MBMD.
  - 20   8. Else (this is a resumption of a previously interrupted TDH.IMPORT.MEM):
    - 8.1. Check that the stream context's INTERRUPTED\_FUNC contains TDH.IMPORT.MEM's leaf number.
    - 8.2. Check that the current inputs are the same as saved in the stream context when the function was interrupted.



If passed, process the GPA list:

**Note:** Error conditions that impact a single GPA list entry, but do not cause an import session about, do not cause an abort of TDH.IMPORT.MEM. Instead, the GPA list entry is updates with a proper status code.

9. For each entry in the GPA list, starting with RCX.FIRST\_ENTRY and ending with RCX.LAST\_ENTRY:

9.1. Increment the migration stream context's IV\_COUNTER

9.2. Build the 96b IV for this migration bundle by concatenating the stream index and the stream context's IV\_COUNTER.

9.3. Accumulate page MAC based on the GPA list entry.

9.4. If a page L2 attributes list was provided, accumulate page MAC based on the page L2 attributes list entry.

9.5. If no operation is requested:

9.5.1. Check that the calculated MAC value is equal to the provided page MAC value.

If passed:

9.5.2. Mark the corresponding new page list entry (if available) as invalid and continue to the next GPA list entry.

9.6. Walk the SEPT based on the GPA and level operands and find the leaf entry for the page.

9.7. Check that the SEPT entry state is allowed for page import.

9.8. If import is in the out-of-order phase, check that the requested operation in first-time import.

9.9. If the requested operation if import or re-import, and the page state is not PENDING, check that a migration buffer is provided, and its address is a valid shared address.

9.10. If the requested operation is first-time migrate:

9.10.1. Check that the SEPT entry state is either FREE or REMOVED.

9.10.2. If the SEPT entry state is REMOVED, check that the has not been removed in the current migration epoch.

If passed:

9.10.3. If no new page list entry is provided, and a migration buffer is provided, this indicates in-place import. If the page is not PENDING, copy the migration buffer content to a temporary buffer. The migration buffer page will become the new TD private page.

9.10.4. Else, check that the new page list entry is a valid shared HPA.

9.10.5. If the page is not PENDING, decrypt the migration buffer or temporary buffer into the new TD page. Use direct writes (MOVDIR64B), and accumulate MAC.

9.10.6. Check that the calculated MAC value is equal to the provided page MAC value.

If passed:

9.10.7. Update the new TD page PAMT entry; record the current migration epoch value in PAMT.BEPOCH.

9.10.8. Update the SEPT entry.

9.10.9. If a page L2 attributes list was provided, then for each valid L2 attributes entry in the in the page attributes list entry:

9.10.9.1. Check that the alias VM index is not higher than TDCS.NUM\_L2VMS

9.10.9.2. Walk the L2 SEPT based on the GPA and level operands and find the FREE entry for the page alias.

9.10.9.3. Update the L2 SEPT entry based on the page L2 attributes list entry and the new TD page HPA.

9.11. Else, if the requested operation is re-migrate:

9.11.1. Check that the SEPT entry state is either MAPPED or PENDING.

9.11.2. Using the page's PAMT.BEPOCH, check that the page has not been imported in the current migration epoch.

If passed:

9.11.3. Record the current migration epoch value in PAMT.BEPOCH.

9.11.4. If the page is not PENDING, decrypt the migration buffer or temporary buffer into the new TD page. Use direct writes (MOVDIR64B), and accumulate MAC.

9.11.5. Check that the calculated MAC value is equal to the provided page MAC value.

If passed:

9.11.6. For each existing L2 page mapping or, if a page attributes list was provided, each valid L2 page attributes entry in the page attributes list entry:

- 9.11.6.1. In the page attributes list entry (if provided), check that the alias VM index is not higher than TDCS.NUM\_L2VMS
- 9.11.6.2. Walk the L2 SEPT based on the GPA and level operands and find the leaf entry for the L2 page.
- 9.11.6.3. Update the L2 SEPT entry based on the page attributes list entry (if provided). If there is an existing L2 page mapping and no new L2 page attributes were provided, set the L2 SEPT entry state to L2\_FREE.
- 9.11.7. Update the SEPT entry.
- 9.12. Else, if the requested operation is migration cancel:
- 9.12.1. Check that the SEPT entry state indicates that the page has been exported.
- 9.12.2. Calculate MAC over the GPA list entry and check that the value is equal to the provided page MAC value.
- 9.12.3. Using the page's PAMT.BEPOCH, check that the page has not been imported in the current migration epoch.
- If passed:
- 9.12.4. For each existing L2 mapping of the page:
- 9.12.4.1. Walk the L2 SEPT based on the GPA and level operands and find the leaf entry for the page.
- 9.12.4.2. Set the L2 SEPT entry state to L2\_FREE.
- 9.12.5. Update the SEPT entry; set the state to REMOVED and record the current migration epoch in the HPA.
- 9.13. If this is not the last round and there is a pending interrupt:
- 9.13.1. Save intermediate state in the migration stream context.
- 9.13.2. Terminate TDH.EXPORT.MEM with a TDX\_INTERRUPTED\_RESUMABLE status.
- 9.14. Else, advance to the next entry in the GPA list, if applicable.
10. Once the GPA list has been fully processed, update the migration stream expected MB counter field.

#### Completion Status Codes

**Table 5.61: TDH.IMPORT.MEM Completion Status Codes (Returned in RAX) Definition [TO BE COMPLETED]**

Completion Status Code	Description
TDX_INTERRUPTED_RESUMABLE	
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	Operation is successful. <b>Note:</b> Processing of some GPA list entries may have encountered errors, but this did not cause an abort of the overall operation. The number such errors is reported in the lower 32 bits of the completion status.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

**5.3.16. NEW: TDH.IMPORT.STATE.IMMUTABLE Leaf**

TDH.IMPORT.STATE.IMMUTABLE starts a new import session and exports the TD's immutable state as a multi-page migration bundle.

**Table 5.62: TDH.IMPORT.STATE.IMMUTABLE Input Operands Definition**

Operand	Name	Description		
RAX	Leaf and Version	SEAMCALL instruction leaf number and version, see 5.3.1		
		Bits	Field	Description
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
		63:24	Reserved	Must be 0
RCX	TDR	HPA of Destination TD TDR page (HKID bits must be 0)		
R8	MBMD	HPA and size of memory of an MBMD structure in memory:		
		Bits	Name	Description
		51:0	HPA	Bits 51:0 of the host physical address (including HKID bits)
		63:52	Size	Size of the memory buffer containing MBMD, in bytes
R9	PAGE_LIST_INFO	Migration buffers list information – see 3.12.6.1 PAGE_LIST_INFO.FIRST_ENTRY must be 0.		
R10	MIG_STREAM	Migration stream and resume flag:		
		Bits	Name	Description
		15:0	MIGS_INDEX	Migration stream index – must be 0
		62:16	RESERVED	Reserved: must be 0
		63	RESUME	0: This is a new invocation 1: This is resumption of a previously interrupted operation

5

**Table 5.63: TDH.IMPORT.STATE.IMMUTABLE Output Operands Definition**

Operand	Name	Description
RAX	STATUS	SEAMCALL instruction return code, see 5.3.1
RCX	EXTENDED_ERROR_INFO1	In case of an interruption, as indicated by RAX returning TDX_INTERRUPTED_RESUMABLE, RCX is unmodified.  In case of an error related to non-memory state field import, as indicated by RAX, RCX contains the offending field identifier.  In other cases, RCX returns 0.
RDX	EXTENDED_ERROR_INFO2	In case of an interruption, as indicated by RAX returning TDX_INTERRUPTED_RESUMABLE, RDX is unmodified.  In other cases, RDX returns 0.

Operand	Name	Description
AVX, AVX2 and AVX512 state		May be reset to the architectural INIT state
Other		Unmodified

### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 5 TDH.IMPORT.STATE.IMMUTABLE starts a new import session. It imports the TD's immutable state migration bundle previously exported by TDH.EXPORT.STATE.IMMUTABLE. The migration bundle includes an MBMD and a set of 4KB pages.

TD immutable state is verified by TDH.IMPORT.STATE.IMMUTABLE against target platform capabilities and Intel TDX module version, capabilities and configuration. The checks are similar, but not identical, to the TD\_PARAMS checks done on the source platform by TDH.MNG.INIT.

**Interruptibility:** TDH.IMPORT.STATE.IMMUTABLE is interruptible. The host VMM is expected to invoke it in a loop until it returns with either a success indication or with a non-recoverable error indication.

**Import Abort:** A failed TDH.IMPORT.STATE.IMMUTABLE marks (except in cases where the imported TD state has not been modified) the target TD as IMPORT\_FAILED; by design, it will not run. This is indicated by the FATAL bit (61) of the completion status returned in RAX.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.64: TDH.IMPORT.STATE.IMMUTABLE Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDR page	TDR	RW	Opaque	4KB	Exclusive	Shared	Shared
Explicit	R8	HPA	MBMD	MBMD	R	Shared	128B	None	None	None
Explicit	R9	HPA	Page list	PAGE_LIST	R	Shared	4KB	None	None	None
Explicit	R10	Index	Mig. Stream context	Mig. Stream context	RW	Opaque	N/A	Exclusive	N/A	N/A
Explicit	N/A	HPA	Source pages (via page list)	Blob	R	Shared	4KB	None	None	None
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	N/A	Migration context	N/A	RW	Opaque	N/A	Exclusive(i)	N/A	N/A

- 20 TDH.IMPORT.STATE.IMMUTABLE checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

The function checks the following conditions:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
- 5 4. TDCS is allocated (TDR.NUM\_TDCX is the required number).
5. The TD has not been initialized (TDCS.OP\_STATE is UNINITIALIZED).
6. A Migration TD has been bound to the source TD, and no migration session is in progress: Migration Session State is MIG\_TD\_BOUND.
7. The migration stream index is 0.
- 10 8. The buffer provided for MBMD is large enough and fits within a 4KB page.
9. PAGE\_LIST\_INFO.FIRST\_ENTRY is 0.

If successful, the function does the following:

10. If the RESUME input flag is 0, indicating this is a new invocation of TDH.IMPORT.STATE.IMMUTABLE (not a resumption of a previously interrupted one):
  - 15 10.1. Check that a valid migration decryption key has been set by the Migration TD. If this is not the first migration session, then the migration key must have been set after the previous migration session has started.

**Note:** There is no explicit check that a migration TD is bound; this is implied by the above check.

If passed:

- 10.2. Initialize the migration context in TDCS:

- 20 10.2.1. Copy the migration keys to working migration keys that will be used throughout the import session.
- 10.2.2. Generate a new migration encryption key, to be used in the next migration session.

If passed:

- 10.2.3. Set all migration streams' INITIALIZED flag to 0 and ENABLED flags to 1.

- 10.3. Initialize the current migration stream.

- 25 10.4. Copy the MBMD into the migration context.

- 10.5. Check the MBMD fields.

If passed:

- 10.6. Build the 96b IV for this migration bundle by concatenating the stream index and the stream context's IV\_COUNTER.

- 30 10.7. Accumulate MAC based on the MAC'ed fields of MBMD.

11. Else (this is a resumption of a previously interrupted TDH.IMPORT.STATE.IMMUTABLE):

- 11.1. Check that the resumption is valid:

- 11.1.1. The stream context indicates there's a valid interruption state.

- 35 11.1.2. The current SEAMCALL leaf number and the PAGE\_OR\_LIST operand are the same as in the interruption state.

- 11.2. Check that the migration stream is enabled.

- 11.3. Restore the previously saved page list index from the migration context.

If passed:

12. Repeat importing 4KB pages until all immutable state is imported or until a pending interrupt is detected:

- 40 12.1. Get the 4KB next page HPA from it from the page list.

- 12.2. Use the migration key and the migration stream context to decrypt the 4KB internal buffer into an internal temporary 4KB buffer and update the MAC calculation.

- 12.3. Parse the metadata list and write the control structure fields using the algorithm described in 5.2.2.3. Check each TDR or TDCS field for compatibility.

45 If passed:

- 12.4. If all metadata lists have been imported:

- 12.4.1. Check that the accumulated MAC value is equal to the saved MBMD's MAC value.

- 12.4.2. Check that all global, TDR and TDCS metadata fields required to be imported by TDH.IMPORT.STATE.IMMUTABLE have indeed been imported.

- 50 12.4.3. Initialize TDR and TDCS fields that need to be initialized at the beginning of the import session.

- 12.4.4. Mark the migration stream context's interrupted state as invalid.

- 12.4.5. Increment the migration stream context's EXPECTED\_MB\_COUNTER.

- 12.4.6. Set TDCS.TOTAL\_MB to 1.

12.4.7. Set TDCS.OP\_STATE to MEMORY\_IMPORT.

12.4.8. Terminate TDH.IMPORT.STATE.IMMUTABLE with a TDX\_SUCCESS status.

12.5. Else, if there is a pending interrupt:

12.5.1. Save the interruption state to the stream context

12.5.2. Terminate TDH.IMPORT.STATE.IMMUTABLE with a TDX\_INTERRUPTED\_RESUMABLE status.

### Completion Status Codes

**Table 5.65: TDH.IMPORT.STATE.IMMUTABLE Completion Status Codes (Returned in RAX) Definition [TO BE COMPLETED]**

Completion Status Code	Description
TDX_INTERRUPTED_RESUMABLE	
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_RND_NO_ENTROPY	Failed to generate a random migration encryption key. This is typically caused by an entropy error of the CPU's random number generator, and may be impacted by RDSEED, RDRAND or PCONFIG executing on other LPs. The operation should be retried.
TDX_SUCCESS	Operation is successful
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

**5.3.17. NEW: TDH.IMPORT.STATE.TD Leaf**

TDH.IMPORT.STATE.TD imports the TD-scope mutable state as a multi-page migration bundle.

**Table 5.66: TDH.IMPORT.STATE.TD Input Operands Definition**

Operand	Name	Description		
RAX	Leaf and Version	SEAMCALL instruction leaf number and version, see 5.3.1		
		Bits	Field	Description
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
		63:24	Reserved	Must be 0
RCX	TDR	HPA of Destination TD TDR page (HKID bits must be 0)		
R8	MBMD	HPA and size of memory of an MBMD structure in memory:		
		Bits	Name	Description
		51:0	HPA	Bits 51:0 of the host physical address (including HKID bits)
		63:52	Size	Size of the memory buffer containing MBMD, in bytes
R9	PAGE_LIST_INFO	Migration buffers list information – see 3.12.6.1 PAGE_LIST_INFO.FIRST_ENTRY must be 0.		
R10	MIG_STREAM	Migration stream and resume flag:		
		Bits	Name	Description
		15:0	MIGS_INDEX	Migration stream index – must be 0
		62:16	RESERVED	Reserved: must be 0
		63	RESUME	0: This is a new invocation 1: This is resumption of a previously interrupted operation

5

**Table 5.67: TDH.IMPORT.STATE.TD Output Operands Definition**

Operand	Name	Description
RAX	STATUS	SEAMCALL instruction return code, see 5.3.1
RCX	EXTENDED_ERROR_INFO1	In case of an interruption, as indicated by RAX returning TDX_INTERRUPTED_RESUMABLE, RCX is unmodified.  In case of an error related to non-memory state field import, as indicated by RAX, RCX contains the offending field identifier.  In other cases, RCX returns 0.
RDX	EXTENDED_ERROR_INFO2	In case of an interruption, as indicated by RAX returning TDX_INTERRUPTED_RESUMABLE, RDX is unmodified.  In other cases, RDX returns 0.

Operand	Name	Description
AVX, AVX2 and AVX512 state		May be reset to the architectural INIT state
Other		Unmodified

### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 5 TDH.IMPORT.STATE.TD imports the TD-scope mutable state migration bundle previously exported by TDH.EXPORT.STATE.TD. The migration bundle includes an MBMD and a set of 4KB pages.

TD-scope mutable state is verified by TDH.IMPORT.STATE.TD against target platform capabilities and Intel TDX module version, capabilities and configuration.

- 10 **Interruptibility:** TDH.IMPORT.STATE.TD is interruptible. The host VMM is expected to invoke it in a loop until it returns with either a success indication or with a non-recoverable error indication.

**Import Abort:** A failed TDH.IMPORT.STATE.TD marks (except in cases where the imported TD state has not been modified) the target TD as IMPORT\_FAILED; by design, it will not run. This is indicated by the FATAL bit (61) of the completion status returned in RAX.

- 15 To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.68: TDH.IMPORT.STATE.VP Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDR page	TDR	R	Opaque	4KB	Exclusive	Shared	Shared
Explicit	R8	HPA	MBMD	MBMD	R	Shared	128B	None	None	None
Explicit	R9	HPA	Page list	PAGE_LIST	R	Shared	4KB	None	None	None
Explicit	R10	Index	Mig. Stream context	Mig. Stream context	RW	Opaque	N/A	Exclusive	N/A	N/A
Explicit	N/A	HPA	Source pages (via page list)	Blob	R	Shared	4KB	None	None	None
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	N/A	Migration context	N/A	RW	Opaque	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	N/A	Service TD bindings table	N/A	R	Hidden	N/A	Exclusive(i)	N/A	N/A

TDH.IMPORT.STATE.TD checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.



The function checks the following conditions:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
- 5 4. TDCS is allocated (TDR.NUM\_TDCX is the required number).
5. An import session is in progress, but TD-scope mutable state has not been imported yet (TDCS.OP\_STATE is MEMORY\_IMPORT).
6. The migration stream index is 0.
7. The migration stream is enabled.
- 10 8. The buffer provided for MBMD is large enough and fits within a 4KB page.
9. PAGE\_LIST\_INFO.FIRST\_ENTRY is 0.

If successful, the function does the following:

10. If the RESUME input flag is 0, indicating this is a new invocation of TDH.IMPORT.STATE.TD (not a resumption of a previously interrupted one):
  - 10.1. Copy the MBMD into the migration context.
  - 10.2. Check the MBMD fields.

If passed:

- 10.3. Build the 96b IV for this migration bundle by concatenating the stream index and the stream context's IV\_COUNTER.
- 10.4. Accumulate MAC based on the MAC'ed fields of MBMD.
11. Else (this is a resumption of a previously interrupted TDH.IMPORT.STATE.IMMUTABLE):
  - 11.1. Check that the resumption is valid:
    - 11.1.1. The stream context indicates there's a valid interruption state.
    - 11.1.2. The current SEAMCALL leaf number and the PAGE\_OR\_LIST operand are the same as in the interruption state.
  - 11.2. Restore the previously saved page list index from the migration context.

If passed:

12. Repeat importing 4KB pages until all immutable state is imported or until a pending interrupt is detected:
  - 12.1. Get the 4KB next page HPA from it from the page list.
  - 12.2. Use the migration key and the migration stream context to decrypt the 4KB internal buffer into an internal temporary 4KB buffer and update the MAC calculation.
  - 12.3. Parse the metadata list and write the control structure fields using the algorithm described in 5.2.2.3. Check each TDR or TDCS field for compatibility.

If passed:

- 12.4. If all metadata lists have been imported:
  - 12.4.1. Check that the accumulated MAC value is equal to the saved MBMD's MAC value.
  - 12.4.2. Check that all TDR and TDCS fields required to be imported by TDH.IMPORT.STATE.TD have indeed been imported.
  - 12.4.3. Initialize TDR and TDCS fields that need to be initialized at the end of the import session.
  - 12.4.4. Mark the migration stream context's interrupted state as invalid.
  - 12.4.5. Increment the migration stream context's EXPECTED\_MB\_COUNTER.
  - 12.4.6. Increment TDCS.TOTAL\_MB.
  - 12.4.7. Set TDCS.OP\_STATE to STATE\_IMPORT.
  - 12.4.8. Terminate TDH.IMPORT.STATE.TD with a TDX\_SUCCESS status.
- 12.5. Else, if there is a pending interrupt:
  - 12.5.1. Save the interruption state to the stream context
  - 12.5.2. Terminate TDH.IMPORT.STATE.TD with a TDX\_INTERRUPTED\_RESUMABLE status.

## Completion Status Codes

**Table 5.69: TDH.IMPORT.STATE.TD Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_INTERRUPTED_RESUMABLE	

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	Operation is successful
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

DRAFT

**5.3.18. NEW: TDH.IMPORT.STATE.VP Leaf**

TDH.IMPORT.STATE.VP imports the VCPU-scope mutable state as a multi-page migration bundle.

**Table 5.70: TDH.IMPORT.STATE.VP Input Operands Definition**

Operand		Description		
RAX	Leaf and Version	SEAMCALL instruction leaf number and version, see 5.3.1		
		Bits	Field	Description
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
		63:24	Reserved	Must be 0
RCX	TDVPR	HPA of the destination TD VCPU's TDVPR page (HKID bits must be 0)		
R8	MBMD	HPA and size of memory of an MBMD structure in memory:		
		Bits	Name	Description
		51:0	HPA	Bits 51:0 of the host physical address (including HKID bits)
		63:52	Size	Size of the memory buffer containing MBMD, in bytes
R9	PAGE_LIST_INFO	Migration buffers list information – see 3.12.6.1 PAGE_LIST_INFO.FIRST_ENTRY must be 0.		
R10	MIG_STREAM	Migration stream and resume flag:		
		Bits	Name	Description
		15:0	MIGS_INDEX	Migration stream index – must be 0
		62:16	RESERVED	Reserved: must be 0
		63	RESUME	0: This is a new invocation 1: This is resumption of a previously interrupted operation

5

**Table 5.71: TDH.IMPORT.STATE.VP Output Operands Definition**

Operand	Name	Description
RAX	STATUS	SEAMCALL instruction return code, see 5.3.1
RCX	EXTENDED_ERROR_INFO1	In case of an interruption, as indicated by RAX returning TDX_INTERRUPTED_RESUMABLE, RCX is unmodified.  In case of an error related to non-memory state field import, as indicated by RAX, RCX contains the offending field identifier.  In other cases, RCX returns 0.
RDX	EXTENDED_ERROR_INFO2	In case of an interruption, as indicated by RAX returning TDX_INTERRUPTED_RESUMABLE, RDX is unmodified.  In other cases, RDX returns 0.

Operand	Name	Description
AVX, AVX2 and AVX512 state		May be reset to the architectural INIT state
Other		Unmodified

### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 5 TDH.IMPORT.STATE.VP imports the VCPU-scope mutable state migration bundle previously exported by TDH.EXPORT.STATE.VP. The migration bundle includes an MBMD and a set of 4KB pages.

VCPU-scope mutable state is verified by TDH.IMPORT.STATE.VP against target platform capabilities and Intel TDX module version, capabilities and configuration.

- 10 **Interruptibility:** TDH.IMPORT.STATE.VP is interruptible. The host VMM is expected to invoke it in a loop until it returns with either a success indication or with a non-recoverable error indication.

**Import Abort:** A failed TDH.IMPORT.STATE.VP marks (except in cases where the imported TD state has not been modified) the target TD as IMPORT\_FAILED; by design, it will not run. This is indicated by the FATAL bit (61) of the completion status returned in RAX.

- 15 **VCPU Association:** TDH.IMPORT.VP associates the TD VCPU with the current LP. This requires that the VCPU will not be associated with another LP – for details, see the [TDX Module Base Spec].

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.72: TDH.IMPORT.STATE.VP Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDVPR page	TDVPS	RW	Opaque	4KB	Exclusive	Shared	Shared
Explicit	R8	HPA	MBMD	MBMD	R	Shared	128B	None	None	None
Explicit	R9	HPA	Page list	PAGE_LIST	R	Shared	4KB	None	None	None
Explicit	R10	Index	Mig. Stream context	Mig. Stream context	RW	Opaque	N/A	Exclusive	N/A	N/A
Explicit	N/A	HPA	Source pages (via page list)	Blob	R	Shared	4KB	None	None	None
Implicit	N/A	HPA	TDR page	TDR	R	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	N/A	Migration context	N/A	RW	Opaque	N/A	None	N/A	N/A

- 20 TDH.IMPORT.STATE.VP checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

The function checks the following conditions:

1. The TDVPR page metadata in PAMT must be correct (PT must be PT\_TDVPR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
- 5 4. TDCS is allocated (TDR.NUM\_TDCX is the required number).
5. An import session is in progress and TD-scope mutable state has been imported (TDCS.OP\_STATE is STATE\_IMPORT).
6. The migration stream index is lower than TDCS.NUM\_MIGS.
7. The number of pages allocated to this TDVPS is correct.
8. The VCPU has not been initialized yet (TDVPS.VCPU\_STATE is VCPU\_UNINITIALIZED).
- 10 9. The buffer provided for MBMD is large enough.
10. PAGE\_LIST\_INFO.FIRST\_ENTRY is 0.

If successful, the function does the following:

11. Associate the VCPU with the current LP, and update TD VMCS using the algorithm described in 5.2.1.

If passed:

- 15 12. If the RESUME input flag is 0, indicating this is a new invocation of a previously interrupted TDH.IMPORT.STATE.VP (not a resumption of a previously interrupted one):
  - 12.1. Copy the MBMD into the migration context.
  - 12.2. Check the MBMD fields.

If passed:

- 20 12.3. Build the 96b IV for this migration bundle by concatenating the stream index and the stream context's IV\_COUNTER.
- 12.4. Accumulate MAC based on the MAC'ed fields of MBMD.
- 12.5. Atomically increment the TD's migrated VCPU counter (TDCS.NUM\_MIGRATED\_VCPUS), and check that number of VCPUs (TDCS.NUM\_VCPUS) has not been exceeded.
- 25 13. Else (this is a resumption of a previously interrupted TDH.IMPORT.STATE.VP):
  - 13.1. Check that the resumption is valid:
    - 13.1.1. The stream context indicates there's a valid interruption state.
    - 13.1.2. The current SEAMCALL leaf number and the PAGE\_OR\_LIST operand are the same as in the interruption state.

If passed:

14. Repeat importing 4KB pages until all TD-scope state is imported or until a pending interrupt is detected:
  - 14.1. Get the 4KB next page HPA from it from the page list.
  - 14.2. Use the migration key and the migration stream context to decrypt the 4KB internal buffer into an internal temporary 4KB buffer and update the MAC calculation.
  - 35 14.3. Parse the metadata list and write the control structure fields using the algorithm described in 5.2.2.3. Check each TDVPS field for compatibility.

If passed:

- 14.4. If all metadata lists have been imported:
  - 14.4.1. Check that the accumulated MAC value is equal to the saved MBMD's MAC value.
  - 40 14.4.2. Check that all TDVPS fields required to be imported by TDH.IMPORT.STATE.VP have indeed been imported.
  - 14.4.3. Initialize TDVPS fields that need to be initialized at the end of the import session.
  - 14.4.4. Mark the migration stream context's interrupted state as invalid.
  - 14.4.5. Increment the migration stream context's EXPECTED\_MB\_COUNTER.
  - 45 14.4.6. Increment TDCS.TOTAL\_MB.
  - 14.4.7. Terminate TDH.IMPORT.STATE.VP with a TDX\_SUCCESS status.
- 14.5. Else, if there is a pending interrupt:
  - 14.5.1. Save the interruption state to the stream context
  - 14.5.2. Terminate TDH.IMPORT.STATE.VP with a TDX\_INTERRUPTED\_RESUMABLE status.

**Completion Status Codes****Table 5.73: TDH.IMPORT.STATE.VP Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_INTERRUPTED_RESUMABLE	
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	Operation is successful
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

DRAFT

**5.3.19. NEW: TDH.IMPORT.TRACK Leaf**

TDH.IMPORT.TRACK consumes an epoch token received from the source platform. It ends the current in-order import phase epoch and either starts a new epoch or starts the out-of-order import phase.

**Table 5.74: TDH.IMPORT.TRACK Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function Must be 0
	23:16	Version Number	Selects the SEAMCALL interface function version
	63:24	Reserved	Must be 0
RCX	HPA of the source TD's TDR page (HKID bits must be 0)		
R8	HPA and size of memory of an MBMD structure in memory:		
	Bits	Name	Description
	51:0	HPA	Bits 51:0 of the host physical address (including HKID bits)
	63:52	Size	Size of the memory buffer containing MBMD, in bytes
R10	Migration stream index – must be 0		

**Table 5.75: TDH.IMPORT.TRACK Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code, see 5.3.1
AVX, AVX2 and AVX512 state	May be reset to the architectural INIT state
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.IMPORT.TRACK parses an epoch token received from the source platform. It checks that the epoch number indicated by the token is correct, and that all migration bundles indicated by the token have been received.

If successful, it ends the current import epoch, and as indicated by the epoch token either starts a new epoch or starts the out-of-order import phase.

**Import Abort:** A failure may mark the target TD as IMPORT\_FAILED; by design, it will not run. This is indicated by the FATAL bit (61) of the completion status returned in RAX.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

Table 5.76: TDH.IMPORT.TRACK Operands Information Definition

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDR page	TDR	R	Opaque	4KB	Exclusive	Shared	Shared
Explicit	R8	HPA	MBMD buffer	MBMD	R	Shared	128B	None	None	None
Explicit	R10	Index	Mig. Stream context	Mig. Stream context	RW	Opaque	N/A	Exclusive	N/A	N/A
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	N/A	Migration context	N/A	RW	Opaque	N/A	Exclusive(i)	N/A	N/A

TDH.IMPORT.TRACK checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

- 5 The function checks the following conditions:
  1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
  2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
  3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
  4. TDCS is allocated (TDR.NUM\_TDCX is the required number).
  - 10 5. An import session is in the in-order phase: TDCS.OP\_STATE is either MEMORY\_IMPORT or STATE\_IMPORT.
  6. An import session is in progress and the TD-scope state has been imported: TDCS.OP\_STATE is STATE\_IMPORT.
  7. The migration stream index is 0.
  8. The migration stream is initialized.
  9. The buffer provided for MBMD is large enough.
- 15 If successful, the function does the following:
  10. Copy the MBMD into a temporary buffer.
  11. Check the MBMD fields:
    - 11.1. Check that SIZE is large enough.
    - 11.2. Check that MB\_TYPE indicates an epoch token.
    - 20 11.3. Check that MIGS\_INDEX is 0.
    - 11.4. Check that the MB\_COUNTER value is equal to the migration stream's EXPECTED\_RX\_COUNTER.
    - 11.5. Check that MIG\_EPOCH is higher than TDCS.MIG\_EPOCH.
    - 11.6. Check that TOTAL\_MB is equal to TDCS.TOTAL\_MB + 1.
    - 11.7. Check that reserved fields are 0.
  - 25 If passed:
    12. Build the 96b IV for this migration bundle by concatenating the stream index and the stream context's IV\_COUNTER.
    13. Accumulate MAC based on the MAC'ed fields of MBMD and check that the value is the same as the MBMD's MAC field's value.
  - If passed:
    - 30 14. If the MIG\_EPOCH value provided in the MBMD is 0xFFFFFFFF, indicating the start of out-of-order phase:
      - 14.1. Check that all VCPUs have been imported
      - If passed:
        - 14.2. Start the out-of-order import phase: set TDCS.OP\_STATE to POST\_IMPORT.
      15. Set the stream context's EXPECTED\_MB\_COUNTER to 1.
      - 35 16. Increment TDCS.TOTAL\_MB.



17. Set TDCS.MIG\_EPOCH to the MIG\_EPOCH value provided in the MBMD.

#### Completion Status Codes

**Table 5.77: TDH.IMPORT.TRACK Completion Status Codes (Returned in RAX) Definition** **[TO BE COMPLETED]**

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	Operation is successful
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

DRAFT

**5.3.20. UPDATED: TDH.MEM.PAGE.ADD Leaf**

Add a 4KB private page to a TD, mapped to the specified GPA, filled with the given page image and encrypted using the TD ephemeral key, and update the TD measurement with the page properties.

**Table 5.78: TDH.MEM.PAGE.ADD Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	<b>Bits</b>	<b>Field</b>	<b>Description</b>
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	EPT mapping information:		
	<b>Bits</b>	<b>Name</b>	<b>Description</b>
	2:0	Level	Level of the EPT entry that will map the new page – see 3.6.1: must be 0
	11:3	Reserved	Reserved: must be 0
	51:12	GPA	Bits 51:12 of the guest physical address to be mapped for the new Secure EPT page
	63:52	Reserved	Reserved: must be 0
RDX	Host physical address of the parent TDR page (HKID bits must be 0)		
R8	Host physical address of the target page to be added to the TD (HKID bits must be 0)		
R9	Host physical address (including HKID bits) of the source page image		

**Table 5.79: TDH.MEM.PAGE.ADD Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
RCX	Extended error information part 1 In case of EPT walk error, Secure EPT entry architectural content where the error was detected – see 3.6.2 The architectural content represents how the Secure EPT maps a private memory page or a Secure EPT page, and may be different than the actual contents of the Secure EPT entry. Software should consult the Secure EPT information returned in RDX. In other cases, RCX returns 0.
RDX	Extended error information part 2 In case of EPT walk error, Secure EPT entry level and state where the error was detected – see 3.6.2 In other cases, RDX returns 0.
Other	Unmodified

## Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.MEM.PAGE.ADD adds a 4KB private page to a TD and maps it to the provided GPA. It copies the provided source page image to specified physical page using the TD's ephemeral private key and updates the TD measurement with the page properties. TDH.MEM.PAGE.ADD is used during TD build before the TD is initialized.

**In-Place Add:** It is allowed to set the TD page HPA in R8 to the same address as the source page HPA in R9. In this case the source page is converted to be a TD private page.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.80: TDH.MEM.PAGE.ADD Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	GPA	TD private page (GPA) <sup>2</sup>	Blob	RW	Private	4KB	N/A	N/A	N/A
Explicit	RDX	HPA	TDR page	Blob	RW	Opaque	4KB	Exclusive	Shared	Shared
Explicit	R8	HPA	TD private page (HPA) <sup>2</sup>	Blob	RW	Private	4KB	Exclusive	Shared	Shared
Explicit	R9	HPA	Source page	Blob	R	Shared	4KB	None	None	None
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	GPA	Secure EPT tree	N/A	RW	Private	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	GPA	Secure EPT entry	SEPT Entry	RW	Private	N/A	Exclusive(i)	N/A	N/A

TDH.MEM.PAGE.ADD checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

The function checks the following conditions:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
4. The TD must have been initialized but not finalized (TDR.NUM\_TDCX is the required number and TDCS.OP\_STATE is INITIALIZED).
5. The target page metadata in PAMT must be correct (PT must be PT\_NDA).

If successful, the function does the following:

6. Walk the Secure EPT based on the GPA operand, and find the leaf EPT entry for the 4KB page.

If the Secure EPT entry is marked as FREE, the function does the following:

7. Copy the source image to the target TD page using the TD's ephemeral private HKID, and direct write (MOVDIR64B).
8. Update the parent Secure EPT entry with the target page HPA and MAPPED state.
9. Extend TDCS.MRTD with the target page GPA. Extension is done using SHA384 with a 128B extension buffer composed as follows:
  - Bytes 0 through 11 contain the ASCII string "MEM.PAGE.ADD".

<sup>2</sup> RCX and R8 denote the same TD private page operand, using HPA and GPA respectively

- Bytes 16 through 23 contain the GPA (in little-endian format).
- All the other bytes contain 0.

10. Increment TDR.CHLDCNT.

11. Update the PAMT entry with the PT\_REG page type and the TDR physical address as the OWNER.

## 5 Completion Status Codes

**Table 5.81: TDH.MEM.PAGE.ADD Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_EPT_ENTRY_NOT_FREE	
TDX_EPT_WALK_FAILED	
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.MEM.PAGE.ADD is successful
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_FINALIZED	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

**5.3.21. UPDATED: TDH.MEM.PAGE.AUG Leaf**

Dynamically add a 4KB or a 2MB private page to an initialized TD, mapped to the specified GPAs.

**Table 5.82: TDH.MEM.PAGE.AUG Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	EPT mapping information:		
	Bits	Name	Description
	2:0	Level	Level of the EPT entry that will map the new page – see 3.6.1: must be 0 (4KB) or 1 (2MB)
	11:3	Reserved	Reserved: must be 0
	51:12	GPA	Bits 51:12 of the guest physical address to be mapped for the new Secure EPT page
	63:52	Reserved	Reserved: must be 0
RDX	Host physical address of the parent TDR page (HKID bits must be 0)		
R8	Host physical address of the target page to be added to the TD (HKID bits must be 0)		

5

**Table 5.83: TDH.MEM.PAGE.AUG Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
RCX	Extended error information part 1 In case of EPT walk error, Secure EPT entry architectural content where the error was detected – see 3.6.2 The architectural content represents how the Secure EPT maps a private memory page or a Secure EPT page, and may be different than the actual contents of the Secure EPT entry. Software should consult the Secure EPT information returned in RDX. In other cases, RCX returns 0.
RDX	Extended error information part 2 In case of EPT walk error, Secure EPT entry level and state where the error was detected – see 3.6.2 In other cases, RDX returns 0.
Other	Unmodified

## Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.MEM.PAGE.AUG adds a 4KB or a 2MB private page to a TD and maps it to the provided GPA. The new page is mapped in a pending state and can be accessed only by the guest TD after it accepts it using TDCALL(TDG.MEM.PAGE.ACCEPT). TDH.MEM.PAGE.AUG does not initialize the new page and does not update the TD measurement.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.84: TDH.MEM.PAGE.AUG Operands Information Definition**

Explicit/ Implicit	Register	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	GPA	TD private page (GPA) <sup>3</sup>	Blob	None	Private	2 <sup>12+9*Level</sup> Bytes	N/A	N/A	N/A
Explicit	RDX	HPA	TDR page	TDR	RW	Opaque	4KB	Shared	Shared	Shared
Explicit	R8	HPA	TD private page (HPA) <sup>3</sup>	Blob	None	Private	2 <sup>12+9*Level</sup> Bytes	Exclusive	Shared <sup>4</sup>	Shared
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	Secure EPT tree	N/A	RW	Private	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	Secure EPT entry	SEPT Entry	RW	Private	N/A	Exclusive	N/A	N/A

TDH.MEM.PAGE.AUG checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

The function checks the following conditions:

- The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
- The TD is not in a FATAL state (TDR.FATAL is FALSE).
- The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
- The TD must be in one of the following states:
  - The TD has been initialized locally by TDH.MNG.INIT and no migration session is in progress
  - An export session is in progress its live export phase; TDH.EXPORT.PAUSE has not been invoked yet.
  - An import session is in its live import phase, initiated by TDH.IMPORT.COMMIT.
- The target page metadata in PAMT must be correct (PT must be PT\_NDA for the entire 4KB or 2MB range).

If successful, the function does the following:

- Walk the Secure EPT based on the GPA operand, and find the leaf EPT entry for the 4KB or 2MB page.

If the Secure EPT entry is marked as FREE, the function does the following:

- Update the parent Secure EPT entry with the target page HPA and PENDING state.
- Atomically increment TDR.CHLCNT by 1 (for a 4KB page) or by 512 (for a 2MB page).
- Update the PAMT entry with the PT\_REG page type and the TDR physical address as the OWNER.

<sup>3</sup> RCX and R8 denote the same TD private page operand, using HPA and GPA respectively

<sup>4</sup> Applicable for 4KB pages only

## Completion Status Codes

Table 5.85: TDH.MEM.PAGE.AUG Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_EPT_ENTRY_NOT_FREE	
TDX_EPT_WALK_FAILED	
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.MEM.PAGE.AUG is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_FINALIZED	
TDX_TD_NOT_INITIALIZED	

DRAFT

**5.3.22. UPDATED: TDH.MEM.PAGE.DEMOTE Leaf**

Split a large private TD page (2MB or 1GB) into 512 small pages (4KB or 2MB, respectively).

**Table 5.86: TDH.MEM.PAGE.DEMOTE Input Operands Definition**

Operand	Description				
RAX	SEAMCALL instruction leaf number and version, see 5.3.1				
	Bits	Field	Description		
	15:0	Leaf Number	Selects the SEAMCALL interface function		
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0		
	63:24	Reserved	Must be 0		
RCX	EPT mapping information:				
	Bits	Name	Description		
	2:0	Level	Level of the Secure EPT entry that maps the large page to be split: either 1 (2MB) or 2 (1GB) – see 3.6.1		
	11:3	Reserved	Reserved: must be 0		
	51:12	GPA	Bits 51:12 of the guest physical address of the large page to be split  Depending on the level, the following least significant bits must be 0:  Level 1 (2MB):      Bits 20:12 Level 2 (1GB):      Bits 29:12		
	63:52	Reserved	Reserved: must be 0		
RDX	TD handle and flags:				
	Bits	Name	Description		
	0	L2_SEPT_ADD_MODE	New L2 SEPT pages addition mode:		
			Value	Name	Description
			0	DENSE	New L2 SEPT pages are added, if provided by R9, R10 or R11.
			1	SPARSE	New L2 SEPT pages are added, if provided by R9, R10 or R11, but only for L2 VMs where the L1 VMM has created a page alias (using TDG.MEM.PAGE.ATTR.WR).
			In both cases, a new L2 SEPT must be provided for L2 VMs where a page alias exists.		
	11:1	Reserved	Reserved: must be 0		
	51:12	TDR_HPA	Bits 51:12 of the host physical address of the parent TDR page (HKID bits must be 0)		
	63:52	Reserved	Reserved: must be 0		



Operand	Description
R8	Host physical address of the new L1 Secure EPT page to be added to the TD (HKID bits must be 0)
R9	<p>If the number of L2 VMs is <math>\geq 1</math>, R9 contains the host physical address of the new Secure EPT page to be added to L2 VM #1's SEPT tree (HKID bits must be 0).</p> <p>Else (the number of L2 VMs is 0), R9 is ignored.</p> <p>If the value of R9 is NULL_PA (-1), no new SEPT page is added to L2 VM #1's SEPT. If the demoted TD private page has an L2 page alias for L2 VM #1, this is an error.</p> <p>Else, bit 63 of R9 is ignored. If L2_SEPT_ADD_MODE is 1, the new SEPT page is only used if the demoted TD private page has an L2 page alias for L2 VM #1. Else, the new SEPT page is always used.</p>
R10	<p>If the number of L2 VMs is <math>\geq 2</math>, R10 contains the host physical address of the new Secure EPT page to be added to L2 VM #2's SEPT tree (HKID bits must be 0).</p> <p>Else (the number of L2 VMs is 0 or 1), R10 is ignored.</p> <p>If the value of R10 is NULL_PA (-1), no new SEPT page is added to L2 VM #2's SEPT. If the demoted TD private page has an L2 page alias for L2 VM #2, this is an error.</p> <p>Else, bit 63 of R10 is ignored. If L2_SEPT_ADD_MODE is 1, the new SEPT page is only used if the demoted TD private page has an L2 page alias for L2 VM #2. Else, the new SEPT page is always used.</p>
R11	<p>If the number of L2 VMs is <math>\geq 3</math>, R11 contains the host physical address of the new Secure EPT page to be added to L2 VM #3's SEPT tree (HKID bits must be 0).</p> <p>Else (the number of L2 VMs is 0, 1 or 2), R11 is ignored.</p> <p>If the value of R11 is NULL_PA (-1), no new SEPT page is added to L2 VM #3's SEPT. If the demoted TD private page has an L2 page alias for L2 VM #3, this is an error.</p> <p>Else, bit 63 of R11 is ignored. If L2_SEPT_ADD_MODE is 1, the new SEPT page is only used if the demoted TD private page has an L2 page alias for L2 VM #3. Else, the new SEPT page is always used.</p>

Table 5.87: TDH.MEM.PAGE.DEMOTE Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
RCX	<p>In case of an interruption, as indicated by RAX returning TDX_INTERRUPTED_RESTARTABLE, RCX is unmodified.</p> <p>Else, RCX returns extended error information part 1.</p> <p>In case of EPT walk error, Secure EPT entry architectural content where the error was detected – see 3.6.2</p> <p>The architectural content represents how the Secure EPT maps a private memory page or a Secure EPT page; it may be different than the actual contents of the Secure EPT entry. Software should consult the Secure EPT information returned in RDX.</p> <p>In other cases, RCX returns 0.</p>
RDX	<p>In case of an interruption, as indicated by RAX returning TDX_INTERRUPTED_RESTARTABLE, RDX is unmodified.</p> <p>Else, RDX returns extended error information part 2.</p> <p>In case of EPT walk error, Secure EPT entry level and state where the error was detected – see 3.6.2</p> <p>In other cases, RDX returns 0.</p>

R9	If TDH.MEM.PAGE.DEMOTE terminated successfully, the number of L2 VMs is $\geq 1$ and the page whose HPA was provided in R9 was not used as an SEPT page for any reason, R9 is updated with bit 63 set to 1. Else, R9 is unmodified.
R10	If TDH.MEM.PAGE.DEMOTE terminated successfully, the number of L2 VMs is $\geq 2$ and the page whose HPA was provided in R10 was not used as an SEPT page for any reason, R10 is updated with bit 63 set to 1. Else, R10 is unmodified.
R11	If TDH.MEM.PAGE.DEMOTE terminated successfully, the number of L2 VMs is $\geq 3$ and the page whose HPA was provided in R11 was not used as an SEPT page for any reason, R11 is updated with bit 63 set to 1. Else, R11 is unmodified.
Other	Unmodified

### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 5 TDH.MEM.PAGE.DEMOTE splits a large TD private page (2MB or 1GB) into 512 small pages (4KB or 2MB, respectively) and adds a new Secure EPT page to map those small pages. If the large page is mapped in any L2 SEPTs, TDH.MEM.PAGE.DEMOTE splits those mapping and adds new L2 Secure EPT pages to map the demoted page.

10 **Blocking and TLB Tracking:** If the TD may be running, the demoted page must be blocked and TLB tracked. Else (e.g., TDH.MR.FINALIZE has not yet been executed, or the TD has been paused for export), no blocking and tracking is required.

**L2 SEPT Population:** TDH.MEM.PAGE.DEMOTE supports multiple host VMM policies for populating the L2 SEPT trees.

- Dense mode is when the host VMM maintains an L2 SEPT page for each L1 SEPT page. This mode is selected by setting L2\_SEPT\_ADD\_MODE to 0.
- Sparse mode is when the host VMM only maintains an L2 SEPT page for a certain L1 SEPT page on demand, i.e., when there's a need to map a TD private page in an L2 VM's GPA space. This mode is selected by setting L2\_SEPT\_ADD\_MODE to 0. The host VMM provides new SEPT pages, but they are only used if a page alias exists for the relevant L2 VM.

20 The host VMM may also choose to maintain L2 SEPT trees only for a subset of the L2 VMs (e.g., if a TD is created with a certain number of L2 VMs but not all of them are currently in use). The host VMM can do so by providing NULL\_PA as the new SEPT page(s) HPA.

25 **Interruptibility:** TDH.MEM.PAGE.DEMOTE is interruptible but not resumable. If a pending interrupt is detected during operation, TDH.MEM.PAGE.DEMOTE returns with a TDX\_INTERRUPTED\_RESTARTABLE status in RAX. No demote operation is done and no output operands except RAX are modified.

In such case, TDH.MEM.PAGE.DEMOTE should be invoked in a loop until it terminates successfully. The host VMM should be designed to avoid cases where interrupt storms prevent successful completion of TDH.MEM.PAGE.DEMOTE.

30 The table below shows the values of the SEPT HPA arguments in R9 – R11, when no error occurs (RAX returns TDX\_SUCCESS).

**Table 5.88: Meaning of TDH.MEM.PAGE.DEMOTE's SEPT HPA Arguments on Input and Output (No Error)**

Value	R9 – R11 on Input	R9 – R11 on Output
NULL_PA (-1)	No new SEPT page to add	Unmodified
Bits 62:0: Valid HPA, HKID bits are 0 Bit 63: 0	Bits 62:0: HPA of new SEPT page to add Bit 63: Ignored	TDH.MEM.PAGE.DEMOTE terminated successfully and the new SEPT page has been added
Bits 62:0: Valid HPA, HKID bits are 0 Bit 63: 1		TDH.MEM.PAGE.DEMOTE terminated successfully and the new SEPT page has not been added

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

5

**Table 5.89: TDH.MEM.PAGE.DEMOTE Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource Name	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	GPA and Level	TD private page to split	Blob	None	Private	2 <sup>12+9*level</sup> bytes	Exclusive	None	None
Explicit	RDX	HPA	TDR page	TDR	RW	Opaque	4KB	Shared	Shared	Shared
Explicit	R8	HPA	New Secure EPT page	SEPT_PAGE	RW	Private	4KB	Exclusive	Shared	Shared
Explicit	R9, R10, R11	HPA	New L2 Secure EPT pages	SEPT_PAGE	RW	Private	4KB	Exclusive	Shared	Shared
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT Tree	N/A	RW	Private	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT entry	SEPT Entry	RW	Private	N/A	Exclusive	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT Trees	N/A	RW	Private	N/A	Shared(i)	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT entries	SEPT Entry	RW	Private	N/A	Exclusive(i)	N/A	N/A

TDH.MEM.PAGE.DEMOTE checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

The function checks the following conditions:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
4. The TD must have been initialized (TDR.NUM\_TDCX is the required number and TDCS.OP\_STATE is either INITIALIZED or RUNNING).
5. The specified page level is either 1 (2MB) or 2 (1GB). See 3.6.1 for a definition of EPT level.

If successful, the function does the following:

6. Walk the L1 Secure EPT based on the GPA operand and locate the large TD private page to be demoted.
7. If the TD may run (its OP\_STATE in either RUNNING, LIVE\_EXPORT or LIVE\_IMPORT):
  - 7.1. Check the Secure EPT entry: It must be a leaf BLOCKED or PENDING\_BLOCKED entry.
  - 7.2. Check that TLB tracking has been done, based on the large TD page's PAMT.BEPOCH.
  - Else:
  - 7.3. Check the Secure EPT entry: It must be a leaf MAPPED, BLOCKED, PENDING or PENDING\_BLOCKED entry.

If passed:

8. Check that the new SEPT page metadata in PAMT is correct (PT must be PT\_NDA).
9. Check that the new L2 SEPT pages metadata in PAMT is correct (PT must be PT\_NDA).

If passed:

10. Split the large TD private page PAMT entry into 512 PAMT entries at the lower level:
  - 10.1. Set the parent PAMT\_2M or PAMT\_1G entry state to PT\_NDA.
  - 10.2. Set the 512 child PAMT4K or PAMT\_2M entries respectively to PT\_REG.
11. Initialize the new Secure EPT page's 512 entries to MAPPED (if the original page was MAPPED or BLOCKED) or PENDING (if the original page was PENDING or PENDING\_BLOCKED) pointing to the 512 consecutive small pages above. Use the TD's ephemeral private HKID and direct write (MOVDIR64B).
12. Atomically set the original Secure EPT entry to NL\_MAPPED non-leaf entry pointing to the new Secure EPT page.
13. For each L2 VM, if L2\_SEPT\_ADD\_MODE is 0 or there is an L2 mapping of the page to be demoted:
  - 13.1. Walk the L2 Secure EPT based on the GPA operand, and locate the L2 Secure EPT parent entry of the page to be demoted.

**Note:** If there is an L2 mapping of the page, this walk should not fail. The L2 SEPT entry state is implicit: It must be a leaf, mapped (L2\_MAPPED) or blocked (L2\_BLOCKED) entry. Else, the walk may fail – for this reason the update is done below only if all SEPT walks succeeded.

If passed:

14. For each L2 VM:
  - 14.1. If there is an L2 mapping of the page to be demoted:
    - 14.1.1. Initialize the new L2 Secure EPT page's 512 entries to L2\_MAPPED (if the original page was MAPPED or BLOCKED) or PENDING (if the original page was PENDING or PENDING\_BLOCKED) pointing to the 512 consecutive small pages above. Use the TD's ephemeral private HKID and direct write (MOVDIR64B).
  - 14.2. Else, if L2\_SEPT\_ADD\_MODE is 0:
    - 14.2.1. Initialize the new L2 Secure EPT page's 512 entries to L2\_FREE. Use the TD's ephemeral private HKID and direct write (MOVDIR64B).
  - 14.3. Atomically set the original L2 Secure EPT entry to NL\_MAPPED non-leaf entry pointing to the new L2 Secure EPT page.
15. Atomically increment TDR.CHLDCNT by 1.
  - 15.1. Note that CHLDCNT counts the number of 4KB pages. The change is due only to the addition of the new Secure EPT page.
16. Update the PAMT entry of the new Secure-EPT page with the PT\_EPT page type and the TDR physical address as the OWNER.
17. For each L2 VM:
  - 17.1. If there was an L2 mapping of the page and the new L2 SEPT page was used:
    - 17.1.1. Update the PAMT entry of each new L2 SEPT page with the PT\_EPT page type and the TDR physical address as the OWNER.
    - 17.1.2. Atomically increment TDR.CHLDCNT by 1.
  - Else:
  - 17.1.3. Set bit 63 of the applicable output GPR (R9, R10 or R11) to 1, indicating that this page was not used as an SEPT page.

## Completion Status Codes

Table 5.90: TDH.MEM.PAGE.DEMOTE Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_EPT_ENTRY_NOT_LEAF	
TDX_EPT_WALK_FAILED	
TDX_GPA_RANGE_NOT_BLOCKED	
TDX_INTERRUPTED_RESTARTABLE	TDH.MEM.PAGE.DEMOTE's operation has been interrupted by an external event; it may be restarted (from its beginning) by calling it again.
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.MEM.PAGE.DEMOTE is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	
TDX_TLB_TRACKING_NOT_DONE	

**5.3.23. UPDATED: TDH.MEM.PAGE.PROMOTE Leaf**

Merge 512 consecutive small private TD pages (4KB or 2MB) into one large page (2MB or 1GB, respectively).

**Table 5.91: TDH.MEM.PAGE.PROMOTE Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version <b>Versions 0 and 1 are supported.</b>
	63:24	Reserved	Must be 0
RCX	EPT mapping information:		
	Bits	Name	Description
	2:0	Level	Level of the Secure EPT entry that will map the merged large page: either 1 (2MB) or 2 (1GB) (see 3.6.1)
	11:3	Reserved	Reserved: must be 0
	51:12	GPA	Bits 51:12 of the guest physical address of the merged large page Depending on the level, the following least significant bits must be 0: Level 1 (2MB): Bits 20:12 Level 2 (1GB): Bits 29:12
	63:52	Reserved	Reserved: must be 0
RDX	Host physical address of the parent TDR page (HKID bits must be 0)		

5

**Table 5.92: TDH.MEM.PAGE.PROMOTE Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
RCX	<p>If TDH.MEM.PAGE.PROMOTE succeeded, RCX returns the HPA of the removed SEPT page.</p> <p><b>In case of an interruption, as indicated by RAX returning TDX_INTERRUPTED_RESTARTABLE, RCX is unmodified.</b></p> <p>Else, RCX returns extended error information part 1.</p> <p>In case of EPT walk error, Secure EPT entry architectural content where the error was detected – see 3.6.2</p> <p>The architectural content represents how the Secure EPT maps a private memory page or a Secure EPT page; it may be different than the actual contents of the Secure EPT entry. Software should consult the Secure EPT information returned in RDX.</p> <p>In other cases, RCX returns 0.</p>

Operand	Description
RDX	<p>In case of an interruption, as indicated by RAX returning <code>TDX_INTERRUPTED_RESTARTABLE</code>, RDX is unmodified.</p> <p>Else, RDX returns extended error information part 2.</p> <p>In case of EPT walk error, Secure EPT entry level and state where the error was detected – see 3.6.2</p> <p>In other cases, RDX returns 0.</p>
R9	<p>If <code>TDH.MEM.PAGE.PROMOTE</code> version is 1 or higher:</p> <ul style="list-style-type: none"> <li>If L2 VM #1's L2 SEPT page has been removed, R9 returns the HPA of that SEPT page.</li> <li>Else, R9 returns <code>NULL_PA</code> (-1).</li> </ul> <p>Else, R9 is unmodified.</p>
R10	<p>If <code>TDH.MEM.PAGE.PROMOTE</code> version is 1 or higher:</p> <ul style="list-style-type: none"> <li>If L2 VM #2's L2 SEPT page has been removed, R10 returns the HPA of that SEPT page.</li> <li>Else, R10 returns <code>NULL_PA</code> (-1).</li> </ul> <p>Else, R10 is unmodified.</p>
R11	<p>If <code>TDH.MEM.PAGE.PROMOTE</code> version is 1 or higher:</p> <ul style="list-style-type: none"> <li>If L2 VM #3's L2 SEPT page has been removed, R11 returns the HPA of that SEPT page.</li> <li>Else, R11 returns <code>NULL_PA</code> (-1).</li> </ul> <p>Else, R11 is unmodified.</p>
Other	Unmodified

### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 5 TDH.MEM.PAGE.PROMOTE merges 512 private pages, which are consecutive both in the HPA space and in the GPA space. The L1 SEPT page and all existing L2 SEPT pages at the requested GPA and level are removed.

All merged private pages must have the same Secure EPT leaf entry attributes and state, which must be either `MAPPED` or `PENDING`. All merged private pages must have the same set of L2 mappings and L2 attributes.

- 10 **Blocking and TLB Tracking:** If the TD may be running, the promoted GPA range must be blocked and TLB tracked. Else (e.g., `TDH.MR.FINALIZE` has not yet been executed, or the TD has been paused for export), no blocking and tracking is required.

- 15 **Interruptibility:** TDH.MEM.PAGE.PROMOTE is interruptible but not resumable. If a pending interrupt is detected during operation, TDH.MEM.PAGE.PROMOTE returns with a `TDX_INTERRUPTED_RESTARTABLE` status in RAX. No promote operation is done and no output operands except RAX are modified.

In such case, TDH.MEM.PAGE.PROMOTE should be invoked in a loop until it terminates successfully. The host VMM should be designed to avoid cases where interrupt storms prevent successful completion of TDH.MEM.PAGE.PROMOTE.

- 20 **Cache Flush and Init:** After the SEPT pages have been removed, the host VMM should flush the physical pages' cache lines and initialize their content before they are reused, as described in the [Base Spec].

The table below shows the values of the SEPT HPA output arguments in RCX and R9 – R11, when version 1 or higher is selected and no error occurs (RAX returns `TDX_SUCCESS`).

**Table 5.93: Meaning of TDH.MEM.PAGE.PROMOTE's SEPT HPA Arguments on Output (No Error)**

Value	RCX on Output	R9 – R11 on Output
NULL_PA (-1)	N/A	No removed SEPT page for this L2 VM
Valid HPA	Removed L1 SEPT page	Removed L2 SEPT page

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

5

**Table 5.94: TDH.MEM.PAGE.PROMOTE Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	GPA and Level	Removed Secure EPT page	SEPT_PAGE	R	Private	2 <sup>12+9*Level</sup> Bytes	Exclusive	None	None
Explicit	RDX	HPA	TDR page	TDR	RW	Opaque	4KB	Shared	Shared	Shared
Implicit	N/A	HPA	Merged HPA range	Blob	None	Private	N/A	Exclusive	None	None
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT Tree	N/A	RW	Private	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	Large page L1 Secure EPT entry	SEPT Entry	RW	Private	N/A	Exclusive	N/A	N/A
Implicit	N/A	GPA	Small pages L1 Secure EPT entries	SEPT Entry	RW	Private	N/A	Exclusive	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT Trees	N/A	RW	Private	N/A	Shared(i)	N/A	N/A
Implicit	N/A	GPA	L2 Large page Secure EPT entry	SEPT Entry	RW	Private	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	GPA	L2 Small pages Secure EPT entries	SEPT Entry	RW	Private	N/A	Exclusive(i)	N/A	N/A

TDH.MEM.PAGE.PROMOTE checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

The function checks the following conditions:

10

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).



4. The TD must have been initialized or its metadata has been imported (TDR.NUM\_TDCX is the required number and TDCS.OP\_STATE is either INITIALIZED, RUNNING, \*\_EXPORT, POST\_IMPORT or LIVE\_IMPORT).
5. The specified merged page level is either 1 (2MB) or 2 (1GB) – see 3.6.1 for a definition of EPT level.

If successful, the function does the following:

6. Walk the Secure EPT based on the GPA operand, and locate the Secure EPT parent entry of the GPA range to be promoted to a merged large page.
7. Get the HPA of the Secure EPT page, which currently maps the GPA range to be promoted, from the Secure EPT above. Get its PAMT entry.
8. If the TD may run (its OP\_STATE in either RUNNING, LIVE\_EXPORT or LIVE\_IMPORT):
  - 8.1. Check the Secure EPT entry: It must be a non-leaf, blocked (NL\_BLOCKED) entry.
  - 8.2. Check that TLB tracking has been done, based on the above Secure EPT page's PAMT.BEPOCH.
  - Else:
    - 8.3. Check the Secure EPT entry: It must be a non-leaf, mapped (NL\_MAPPED) or blocked (NL\_BLOCKED) entry.

If passed:

9. Scan the content of the above Secure EPT page and check all 512 entries:
  - 9.1. They are leaf entries (this also implies that the corresponding pages are PT\_REG).
  - 9.2. Their state is either MAPPED or PENDING.
  - 9.3. They have contiguous HPA mapping aligned to the promoted range size.
  - 9.4. They have L2 mappings for the same set of L2 VMs.
10. For each L2 VM:
  - 10.1. Walk the L2 Secure EPT based on the GPA operand, and locate the L2 Secure EPT parent entry of the GPA range to be promoted to a merged large page.
  - 10.2. If L2 mapping was found above to exist for this L2 VM:
    - 10.2.1. The above SEPT walk should not fail. The L2 SEPT entry state is implicit: It must be a non-leaf, mapped (NL\_MAPPED) or blocked (NL\_BLOCKED) entry.
    - 10.2.2. Scan the content of the above L2 Secure EPT page and check all 512 entries:
      - 10.2.2.1. They should all have the same L2 attributes.
      - 10.2.2.2. The following are implicit: All L2 SEPT entries are leaf entries, they all have the same state, and they have contiguous HPA mapping aligned to the promoted range size.
  - 10.3. Else (L2 mapping was not found above to exist for this L2 VM):
    - 10.3.1. The walk may fail; this is not considered an error.
    - 10.3.2. If the walk succeeded, it implicitly arrives at an empty SEPT page.

If successful, the above checks imply that:

- The 2MB or 1GB GPA range to be promoted has a corresponding single HPA range and a single PAMT entry (PAMT\_2M or PAMT\_1G, respectively) owned by the current guest TD, and its current PAMT.PT is PAMT\_NDA.
- The 512 child PAMT entries (PAMT\_2M or PAMT\_4K, respectively) of the above are owned by the current guest TD, and their PAMT.PT is PAMT\_REG.

The function then does the following:

11. Merge the corresponding 512 physical pages into a single larger physical page:
  - 11.1. Set the small page (PAMT\_4K or PAMT\_2M) entries state to PT\_NDA.
  - 11.2. Set the parent (PAMT\_2M or PAMT\_1G respectively) entry to PT\_REG.
12. Atomically set the promoted Secure EPT entry to MAPPED or PENDING (depending on the small pages' Secure EPT entry state) leaf entry pointing to the merged HPA range.
13. Remove the Secure EPT page that previously mapped the 512 physical pages:
  - 13.1. Atomically decrement TDR.CHLDCNT by 1.
    - 13.1.1. Note that CHLDCNT counts the number of 4KB pages. The change is due only to the removal of the Secure EPT page.
  - 13.2. Update the PAMT entry of the removed Secure EPT page to PT\_NDA.
14. For each L2 VM where L2 mapping was found above to exist:
  - 14.1. Remove the Secure EPT page that previously mapped the 512 physical pages:
    - 14.1.1. Atomically decrement TDR.CHLDCNT by 1.
    - 14.1.2. Update the PAMT entry of the removed Secure EPT page to PT\_NDA.

## Completion Status Codes

Table 5.95: TDH.MEM.PAGE.PROMOTE Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_EPT_ENTRY_FREE	
TDX_EPT_ENTRY_LEAF	
TDX_EPT_INVALID_PROMOTE_CONDITIONS	
TDX_EPT_WALK_FAILED	
TDX_INTERRUPTED_RESTARTABLE	TDH.MEM.PAGE.PROMOTE's operation has been interrupted by an external event; it may be restarted (from its beginning) by calling it again.
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.MEM.PAGE.PROMOTE is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	
TDX_TLB_TRACKING_NOT_DONE	

**5.3.24. UPDATED: TDH.MEM.PAGE.RELOCATE Leaf**

Relocate a 4KB mapped page from its current host physical address to another.

**Table 5.96: TDH.MEM.PAGE.RELOCATE Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	EPT mapping information:		
	Bits	Name	Description
	2:0	Level	Level of the Secure EPT entry that maps the private page to be relocated, must be 0 (i.e., 4KB) (see 3.6.1).
	11:3	Reserved	Reserved: must be 0
	51:12	GPA	Bits 51:12 of the guest physical address of the private page to be relocated
	63:52	Reserved	Reserved: must be 0
RDX	Host physical address of the parent TDR page (HKID bits must be 0)		
R8	Host physical address of the relocated page target (HKID bits must be 0)		

5

**Table 5.97: TDH.MEM.PAGE.RELOCATE Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
RCX	<p>If TDH.MEM.PAGE.RELOCATE succeeded, RCX returns the HPA of the old physical page that has been removed.</p> <p>In case of EPT walk error, Secure EPT entry architectural content where the error was detected – see 3.6.2</p> <p>The architectural content represents how the Secure EPT maps a private memory page or a Secure EPT page, and may be different than the actual contents of the Secure EPT entry. Software should consult the Secure EPT information returned in RDX.</p> <p>In other cases, RCX returns 0.</p>
RDX	<p>Extended error information part 2</p> <p>In case of EPT walk error, Secure EPT entry level and state where the error was detected – see 3.6.2</p> <p>In other cases, RDX returns 0.</p>
Other	Unmodified

## Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.MEM.PAGE.RELOCATE replaces a mapped 4KB page mapping target HPA by moving the current page content to a new target HPA and updating the Secure-EPT mapping to the new target HPA. On successful operation, the previous mapped HPA target is marked is free in the PAMT.

**Blocking and TLB Tracking:** If the TD may be running, the relocated page must be blocked and TLB tracked. Else (e.g., TDH.MR.FINALIZE has not yet been executed, or the TD has been paused for export), no blocking and tracking is required.

**Cache Flush and Init:** After the page has been relocated, the host VMM should flush the old physical page's cache lines and initialize its content before it is reused, as described in the [Base Spec].

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.98: TDH.MEM.PAGE.RELOCATE Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	GPA and Level	TD private page	Blob	R	Private	4KB	Exclusive	None	None
Explicit	RDX	HPA	TDR page	TDR	R	Opaque	4KB	Shared	Shared	Shared
Explicit	R8	HPA	Target physical page	Blob	RW	Private	4KB	Exclusive	Shared	Shared
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT tree	N/A	RW	Private	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT entry	SEPT Entry	RW	Private	N/A	Exclusive	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT trees	N/A	RW	Private	N/A	Shared(i)	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT entries	SEPT Entry	RW	Private	N/A	Exclusive (i)	N/A	N/A

TDH.MEM.PAGE.RELOCATE checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

The function checks the following conditions:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
4. The TD must have been initialized (TDR.NUM\_TDCX is the required number and TDCS.OP\_STATE is either INITIALIZED or RUNNING).
5. The target page metadata in PAMT must be correct (PT must be PT\_NDA).

If successful, the function does the following:

6. Walk the Secure EPT based on the GPA operand and level and find the currently mapped HPA.
7. If TLB tracking is required (based on the Secure EPT entry state and the TD's OP\_STATE):
  - 7.1. Check that the SEPT entry is BLOCKED or PENDING\_BLOCKED.
  - 7.2. Check that TLB tracking was done.

Else:

7.3. Check that the SEPT entry is MAPPED, BLOCKED, BLOCKEDW, EXPORTED\*, PENDING, PENDING\_BLOCKED, PENDING\_BLOCKEDW or PENDING\_EXPORTED\*.

8. Check that the currently mapped HPA is different than the target HPA.

5 If successful, the function does the following:

9. If the page state is **not one of the PENDING\* states**, copy the currently mapped page content to the target page, using the TD's ephemeral private HKID and direct writes (MOVDIR64B).

10. Free the currently mapped HPA by setting its PAMT.PT to PT\_NDA.

11. Update the target page's PAMT entry with the PT\_REG page type and the TDR physical address as the OWNER.

10 12. Update the Secure EPT entry:

12.1. Set the HPA to point to the target page.

12.2. Unblock the SEPT entry: if its state was BLOCKED or PENDING\_BLOCKED, update it to MAPPED or PENDING, respectively.

13. For each L2 VM where the page has L2 mapping:

15 13.1. Walk the L2 Secure EPT based on the GPA operand and find the leaf L2 SEPT entry mapping the page to be relocated.

13.2. Update the L2 Secure EPT entry:

13.2.1. Set the HPA to point to the target page.

20 13.2.2. If the updated L1 SEPT entry state is one of the PENDING\* states, set the L2 SEPT entry state to L2\_BLOCKED. Else, set the L2 SEPT entry state to L2\_MAPPED.

## Completion Status Codes

Table 5.99: TDH.MEM.PAGE.RELOCATE Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_EPT_WALK_FAILED	
TDX_GPA_RANGE_NOT_BLOCKED	
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.MEM.PAGE.RELOCATE is successful
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_FINALIZED	
TDX_TD_NOT_INITIALIZED	
TDX_TLB_TRACKING_NOT_DONE	

**5.3.25. UPDATED: TDH.MEM.PAGE.REMOVE Leaf**

Remove a GPA-mapped 4KB, 2MB or 1GB private page from a TD.

**Table 5.100: TDH.MEM.PAGE.REMOVE Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	EPT mapping information:		
	Bits	Name	Description
	2:0	Level	Level of the Secure EPT entry that maps the private page to be removed: either 0 (4KB), 1 (2MB) or 2 (1GB) – see 3.6.1.
	11:3	Reserved	Reserved: must be 0
	51:12	GPA	Bits 51:12 of the guest physical address of the private page to be removed
	63:52	Reserved	Reserved: must be 0
RDX	Host physical address of the parent TDR page (HKID bits must be 0)		

**Table 5.101: TDH.MEM.PAGE.REMOVE Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
RCX	<p>If TDH.MEM.PAGE.REMOVE succeeded, RCX returns the HPA of the removed page.</p> <p>In case of EPT walk error, Secure EPT entry architectural content where the error was detected – see 3.6.2</p> <p>The architectural content represents how the Secure EPT maps a private memory page or a Secure EPT page, and may be different than the actual contents of the Secure EPT entry. Software should consult the Secure EPT information returned in RDX.</p> <p>In other cases, RCX returns 0.</p>
RDX	<p>Extended error information part 2</p> <p>In case of EPT walk error, Secure EPT entry level and state where the error was detected – see 3.6.2</p> <p>In other cases, RDX returns 0.</p>
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.MEM.PAGE.REMOVE removes a 4KB, 2MB or 1GB private page from the TD's Secure EPT tree (marks the SEPT entry as FREE). If the page is mapped in any L2 Secure EPT, the applicable L2 SEPT entries are marked as L2\_FREE. On successful operation, TDH.MEM.PAGE.REMOVE marks the physical page as free in PAMT.

**Blocking and TLB Tracking:** If the TD may be running, the removed page must be blocked and TLB tracked. Else (e.g., TDH.MR.FINALIZE has not yet been executed, or the TD has been paused for export), no blocking and tracking is required.

**Cache Flush and Init:** After the page has been removed, the host VMM should flush the physical page's cache lines and initialize its content before it is reused, as described in the [Base Spec].

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.102: TDH.MEM.PAGE.REMOVE Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	GPA and Level	TD private page	Blob	R	Private	2 <sup>12+9*Level</sup> Bytes	Exclusive	None	None
Explicit	RDX	HPA	TDR page	TDR	RW	Opaque	4KB	Shared	Shared	Shared
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT tree	N/A	RW	Private	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT entry	SEPT Entry	RW	Private	N/A	Exclusive	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT trees	N/A	RW	Private	N/A	Shared(i)	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT entries	SEPT Entry	RW	Private	N/A	Exclusive(i)	N/A	N/A

TDH.MEM.PAGE.REMOVE checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

The function checks the following conditions:

- The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
- The TD is not in a FATAL state (TDR.FATAL is FALSE).
- The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
- The TD must be in one of the following states:
  - The TD has been initialized locally by TDH.MNG.INIT.
  - An import session is in progress, and either the out-of-order phase has started (TDH.IMPORT.TRACK has been successfully executed with an Epoch Token MBMD indicating a MIG\_EPOCH value of 0xFFFFFFFF), or import has failed.
- The specified level is either 0 (4KB), 1 (2MB) or 2 (1GB) – see 3.6.1 for a definition of EPT level.

If successful:

- Walk the Secure EPT based on the GPA operand, and find the leaf entry of the page to be removed.
- If TLB tracking is required (based on the Secure EPT entry state and the TD's OP\_STATE):
  - Check that the SEPT entry is BLOCKED or PENDING\_BLOCKED.
  - Check that TLB tracking was done.

Else:

7.3. Check that the SEPT entry is MAPPED, BLOCKED, BLOCKEDW, PENDING, PENDING\_BLOCKED or PENDING\_BLOCKEDW.

If successful:

- 5    8. For each L2 VM where the page is mapped:
  - 8.1. Walk the L2 Secure EPT based on the GPA operand and find the page mapping to be removed.
  - 8.2. Set the L2 SEPT entry state to L2\_FREE.
9. If an import session is in progress:
  - 9.1. Set the SEPT entry state to REMOVED.
  - 10    9.2. Record the migration epoch in the SEPT entry.
- Else:
  - 9.3. Set the SEPT entry state to FREE.
10. Atomically decrement TDR.CHLDCNT by 1, 512 or 512<sup>2</sup> depending on the removed TD private page size (4KB, 2MB or 1GB, respectively).
- 15    11. Free the physical page: Set the PAMT entry of the removed TD private page to PT\_NDA.

### Completion Status Codes

**Table 5.103: TDH.MEM.PAGE.REMOVE Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_EPT_WALK_FAILED	
TDX_GPA_RANGE_NOT_BLOCKED	
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.MEM.PAGE.REMOVE is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_FINALIZED	
TDX_TD_NOT_INITIALIZED	
TDX_TLB_TRACKING_NOT_DONE	



**5.3.26. UPDATED: TDH.MEM.RANGE.BLOCK Leaf**

Block a TD private GPA range (i.e., a Secure EPT page or a TD private page) at any level (4KB, 2MB, 1GB, 512GB, 256TB, etc.) from creating new GPA-to-HPA address translations.

**Table 5.104: TDH.MEM.RANGE.BLOCK Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	EPT mapping information:		
	Bits	Name	Description
	2:0	Level	Level of the Secure EPT entry that maps the GPA range to be blocked – see 3.6.1  Level must be between 0 and 3 for a 4-level EPT or between 0 and 4 for a 5-level EPT.
	11:3	Reserved	Reserved: must be 0
	51:12	GPA	Bits 51:12 of the GPA range to be blocked Depending on the level, the following least significant bits must be 0: Level 0 (EPTE):     None Level 1 (EPDE):     Bits 20:12 Level 2 (EPDPTE):   Bits 29:12 Level 3 (EPML4E):   Bits 38:12 Level 4 (EPML5E):   Bits 47:12
	63:52	Reserved	Reserved: must be 0
RDX	Host physical address of the parent TDR page (HKID bits must be 0)		

5

**Table 5.105: TDH.MEM.RANGE.BLOCK Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
RCX	Extended error information part 1  In case of EPT walk error, Secure EPT entry architectural content where the error was detected – see 3.6.2  The architectural content represents how the Secure EPT maps a private memory page or a Secure EPT page, and may be different than the actual contents of the Secure EPT entry. Software should consult the Secure EPT information returned in RDX.  In other cases, RCX returns 0.

Operand	Description
RDX	Extended error information part 2 In case of EPT walk error, Secure EPT entry level and state where the error was detected – see 3.6.2 In other cases, RDX returns 0.
Other	Unmodified

### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- TDH.MEM.RANGE.BLOCK finds the Secure EPT entry for the given GPA and level, and it marks it as blocked (BLOCKED or PENDING\_BLOCKED as appropriate). It records the current TD's TLB epoch in the PAMT entry of the physical Secure EPT page or TD private page mapped by the blocked Secure EPT entry.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

10

**Table 5.106: TDH.MEM.RANGE.BLOCK Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	GPA and Level	Secure EPT page or TD private page	Blob	None	Private	$2^{12+9*\text{Level}}$ Bytes	None	None	None
Explicit	RDX	HPA	TDR page	TDR	R	Opaque	4KB	Shared	Shared	Shared
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT tree	N/A	RW	Private	N/A	Exclusive	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT entry	SEPT Entry	RW	Private	N/A	Exclusive(h)	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT trees	N/A	RW	Private	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT entries	SEPT Entry	RW	Private	N/A	Exclusive(i)	N/A	N/A

TDH.MEM.RANGE.BLOCK checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

The function checks the following conditions:

- The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
- The TD is not in a FATAL state (TDR.FATAL is FALSE).
- The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
- The TD must have been initialized (TDR.NUM\_TDCX is the required number and TDCS.OP\_STATE is either INITIALIZED or RUNNING).
- The specified level is of an EPT entry – i.e., 0 to 3 for 4-level EPT or 0 to 4 for 5-level EPT. See 3.6.1 for a definition of EPT level.

If successful, the function does the following:

6. Walk the Secure EPT based on the GPA operand and find the Secure EPT entry to be blocked.
7. Check the Secure EPT entry is not free and not blocked (its state should be **NL\_MAPPED**, **MAPPED** or **PENDING**).

If passed:

- 5 8. Block the Secure EPT entry. Set its state to **NL\_BLOCKED** (if it was **NL\_MAPPED**), **BLOCKED** (if it was **MAPPED**) or **PENDING\_BLOCKED** (if it was **PENDING**).
9. For each L2 VM where there is an SEPT entry for the given GPA and level:
  - 9.1. Walk the L2 Secure EPT based on the GPA and level operand and find the L2 SEPT entry to be blocked.
  - 9.2. If the L1 SEPT entry is a leaf entry, then set the L2 SEPT entry state to **L2\_BLOCKED** and save its attributes. Note that if the page was blocked for writing, then the W bit has already been saved.
  - 10 9.3. Else (L1 SEPT entry is a non-leaf, mapping an SEPT page), set the non-leaf L2 SEPT entry state to **NL\_BLOCKED**.  
**Note:** There is no need to write **TD\_EPOCH** to the L2 SEPT page's PAMT. Blocked epoch is implicit from the L1 SEPT page.

If passed:

- 15 10. Read the TD's epoch (**TDCS.TD\_EPOCH**), and write it to the PAMT entry of the blocked Secure EPT page or TD private page (**PAMT.BEPOCH**).

### Completion Status Codes

**Table 5.107: TDH.MEM.RANGE.BLOCK Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
<b>TDX_EPT_ENTRY_FREE</b>	
<b>TDX_EPT_WALK_FAILED</b>	
<b>TDX_OPERAND_ADDR_RANGE_ERROR</b>	
<b>TDX_OPERAND_BUSY</b>	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
<b>TDX_OPERAND_INVALID</b>	
<b>TDX_OPERAND_PAGE_METADATA_INCORRECT</b>	
<b>TDX_SUCCESS</b>	<b>TDH.MEM.RANGE.BLOCK</b> is successful.
<b>TDX_SYS_NOT_READY</b>	
<b>TDX_SYS_SHUTDOWN</b>	
<b>TDX_TD_FATAL</b>	
<b>TDX_TD_KEYS_NOT_CONFIGURED</b>	
<b>TDX_TD_NOT_INITIALIZED</b>	

**5.3.27. UPDATED: TDH.MEM.RANGE.UNBLOCK Leaf**

Remove the blocking of a TD private GPA range (i.e., a Secure EPT page or a TD private page), at any level (4KB, 2MB, 1GB, 512GB, 256TB etc.) previously blocked by TDH.MEM.RANGE.BLOCK.

**Table 5.108: TDH.MEM.RANGE.UNBLOCK Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	EPT mapping information:		
	Bits	Name	Description
	2:0	Level	Level of the Secure EPT entry that maps the GPA range to be unblocked – see 3.6.1  Level must be between 0 and 3 for a 4-level EPT or between 0 and 4 for a 5-level EPT.
	11:3	Reserved	Reserved: must be 0
	51:12	GPA	Bits 51:12 of the guest physical address range to be unblocked Depending on the level, the following least significant bits must be 0: Level 0 (EPTE):     None Level 1 (EPDE):     Bits 20:12 Level 2 (EPDPTE):   Bits 29:12 Level 3 (EPML4E):   Bits 38:12 Level 4 (EPML5E):   Bits 47:12
	63:52	Reserved	Reserved: must be 0
RDX	Host physical address of the parent TDR page (HKID bits must be 0)		

5

**Table 5.109: TDH.MEM.RANGE.UNBLOCK Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
RCX	Extended error information part 1  In case of EPT walk error, Secure EPT entry architectural content where the error was detected – see 3.6.2  The architectural content represents how the Secure EPT maps a private memory page or a Secure EPT page, and may be different than the actual contents of the Secure EPT entry. Software should consult the Secure EPT information returned in RDX.  In other cases, RCX returns 0.

Operand	Description
RDX	Extended error information part 2 In case of EPT walk error, Secure EPT entry level and state where the error was detected – see 3.6.2 In other cases, RDX returns 0.
Other	Unmodified

### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 5 TDH.MEM.RANGE.UNBLOCK finds the blocked Secure EPT entry for the given GPA and level. It checks that the entry has been blocked and TLB tracking has been done, and then it marks the entry as non-blocked (MAPPED or PENDING as appropriate).

**Blocking and TLB Tracking:** If the TD may be running, the unblocked GPA range must be blocked and TLB tracked. Else (e.g., TDH.MR.FINALIZE has not yet been executed, or the TD has been paused for export), no blocking and tracking is required.

10

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.110: TDH.MEM.RANGE.UNBLOCK Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	GPA and Level	Secure EPT page or TD private page	Blob	None	Private	$2^{12+9*\text{Level}}$ Bytes	None	None	None
Explicit	RDX	HPA	TDR page	TDR	R	Opaque	4KB	Shared	Shared	Shared
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT tree	N/A	RW	Private	N/A	Exclusive	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT entry	SEPT Entry	RW	Private	N/A	Exclusive(h)	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT tree	N/A	RW	Private	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT entries	SEPT Entry	RW	Private	N/A	Exclusive(i)	N/A	N/A

- 15 TDH.MEM.RANGE.UNBLOCK checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

The function checks the following conditions:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
- 20 3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).

4. The TD must have been initialized (TDR.NUM\_TDCX is the required number and TDCS.OP\_STATE is either INITIALIZED or RUNNING).
5. The specified level is of an EPT entry (i.e., 0 to 3 for 4-level EPT or 0 to 4 for 5-level EPT) – see 3.6.1 for a definition of EPT level.
- 5 If successful, the function does the following:
  6. Walk the Secure EPT based on the GPA operand and find the Secure EPT page or TD private page to be unblocked.
  7. Check the page's parent Secure EPT entry is blocked (NL\_BLOCKED, BLOCKED or PENDING\_BLOCKED).
  8. If TLB tracking is required (based on the Secure EPT entry state and the TD's OP\_STATE):
    - 8.1. Check that TLB tracking was done.
- 10 If successful, the function does the following:
  9. For each L2 VM where there is an SEPT entry for the given GPA and level:
    - 9.1. Walk the L2 Secure EPT based on the GPA operand and find the L2 SEPT entry to be unblocked.
    - 9.2. If the updated L1 SEPT entry is a leaf entry, and its state is not one of the PENDING\* states, set the L2 SEPT entry state to L2\_MAPPED and restore the L2 SEPT attributes.
    - 15 9.3. Else (L1 SEPT entry is a non-leaf, mapping an SEPT page), set the non-leaf L2 SEPT entry state to NL\_MAPPED.
  10. Unblock the Secure EPT entry. Atomically set its state to NL\_MAPPED (if it was NL\_BLOCKED), MAPPED (if it was BLOCKED) or PENDING (if it was PENDING\_BLOCKED).

### Completion Status Codes

**Table 5.111: TDH.MEM.RANGE.UNBLOCK Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_EPT_WALK_FAILED	
TDX_GPA_RANGE_NOT_BLOCKED	
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.MEM.RANGE.UNBLOCK is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	
TDX_TLB_TRACKING_NOT_DONE	

### 5.3.28. TDH.MEM.RD Leaf

Read a 64b chunk from a debuggable guest TD private memory.

**Table 5.112: TDH.MEM.RD Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	The guest physical address of a naturally aligned 8-byte chunk of a guest TD private page		
RDX	Host physical address of the parent TDR page (HKID bits must be 0)		

**Table 5.113: TDH.MEM.RD Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
RCX	Extended error information part 1 In case of EPT walk error, Secure EPT entry architectural content where the error was detected – see 3.6.2 The architectural content represents how the Secure EPT maps a private memory page or a Secure EPT page, and may be different than the actual contents of the Secure EPT entry. Software should consult the Secure EPT information returned in RDX. In other cases, RCX returns 0.
RDX	Extended error information part 2 In case of EPT walk error, Secure EPT entry level and state where the error was detected – see 3.6.2 In other cases, RDX returns 0.
R8	Content of the memory chunk In case of an error, as indicated by RAX, R8 returns 0
Other	Unmodified

#### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.MEM.RD reads a 64b chunk from a debuggable guest TD private memory.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

Table 5.114: TDH.MEM.RD Operands Information Definition

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	GPA	TD private memory	Blob	R	Private	8B	None	None	None
Explicit	RDX	HPA	TDR page	TDR	R	Opaque	4KB	Shared	Shared	Shared
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	R	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	Secure EPT tree	N/A	R	Private	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	Secure EPT entry	SEPT Entry	R	Private	N/A	Exclusive	N/A	N/A

TDH.MEM.RD checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

- 5 The function checks the following conditions:
  1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
  2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
  3. The TD keys are configured on the hardware (TDR.KEY\_STATE is TD\_KEYS\_CONFIGURED).
  4. TDCS must have been initialized (TDR.INIT is TRUE).
  5. The TD is debuggable (TDCS.ATTRIBUTES.DEBUG is 1).

If successful, the function does the following:

6. Walk the Secure EPT based on the GPA operand and find the leaf entry.
7. Check that the Secure EPT entry state is PRESENT.

If passed:

- 15 8. Read the content of the memory chunk.

#### Completion Status Codes

Table 5.115: TDH.MEM.RD Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TO BE COMPLETED	
TDX_SUCCESS	



**5.3.29. UPDATED: TDH.MEM.SEPT.ADD Leaf**

Add and map 4KB **L1 and L2** Secure EPT pages to a TD.

**Table 5.116: TDH.MEM.SEPT.ADD Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version <b>Versions 0 and 1 are supported.</b>
	63:24	Reserved	Must be 0
RCX	EPT mapping information:		
	Bits	Name	Description
	2:0	Level	Level of the non-leaf Secure EPT entry that will map the new Secure EPT page – see 3.6.1  Level must be between 1 and 3 for a 4-level EPT or between 1 and 4 for a 5-level EPT.
	11:3	Reserved	Reserved: must be 0
	51:12	GPA	Bits 51:12 of the guest physical address of to be mapped for the new Secure EPT page  Depending on the level, the following least significant bits must be 0: Level 1 (EPT):       Bits 20:12 Level 2 (EPD):       Bits 29:12 Level 3 (EPDPT):     Bits 38:12 Level 4 (EPML4):     Bits 47:12
	63:52	Reserved	Reserved: must be 0
RDX	TD handle and flags:		
	Bits	Name	Description
	0	ALLOW_EXISTING	Flags that TDH.MEM.SEPT.ADD should not fail if an SEPT page to be added already exists in the L1 or L2 SEPT tree. Instead, it should just return an indication in the output operand, as described below.
	11:1	Reserved	Reserved: must be 0
	51:12	TDR_HPA	Bits 51:12 of the host physical address of the parent TDR page (HKID bits must be 0)
	63:52	Reserved	Reserved: must be 0
R8	Host physical address of the new <b>L1</b> Secure EPT page to be added to the TD (HKID bits must be 0) <b>For TDH.MEM.SEPT.ADD version 1 or higher:</b> <ul style="list-style-type: none"> <li>If the value of R8 is NULL_PA (-1), no L1 SEPT page is added.</li> <li>Else, bit 63 of R8 is ignored.</li> </ul>		

Operand	Description
R9	<p>For TDH.MEM.SEPT.ADD version 1 or higher, R9 specifies the HPA of a new L2 VM #1 Secure EPT page to be added to the TD (HKID bits must be 0).</p> <ul style="list-style-type: none"> <li>• If the value of R9 is NULL_PA (-1), no L2 VM #1 SEPT page is added.</li> <li>• Else, bit 63 of R9 is ignored.</li> </ul>
R10	<p>For TDH.MEM.SEPT.ADD version 1 or higher, R10 specifies the HPA of a new L2 VM #2 Secure EPT page to be added to the TD (HKID bits must be 0).</p> <ul style="list-style-type: none"> <li>• If the value of R10 is NULL_PA (-1), no L2 VM #2 SEPT page is added.</li> <li>• Else, bit 63 of R10 is ignored.</li> </ul>
R11	<p>For TDH.MEM.SEPT.ADD version 1 or higher, R11 specifies the HPA of a new L2 VM #3 Secure EPT page to be added to the TD (HKID bits must be 0).</p> <ul style="list-style-type: none"> <li>• If the value of R11 is NULL_PA (-1), no L2 VM #3 SEPT page is added.</li> <li>• Else, bit 63 of R11 is ignored.</li> </ul>

Table 5.117: TDH.MEM.SEPT.ADD Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
RCX	<p>In case of an interruption, as indicated by RAX returning TDX_INTERRUPTED_RESUMABLE, RCX is unmodified.</p> <p>Else, RCX returns extended error information part 1.</p> <p>In case of EPT walk error, Secure EPT entry architectural content where the error was detected – see 3.6.2</p> <p>The architectural content represents how the Secure EPT maps a private memory page or a Secure EPT page; it may be different than the actual contents of the Secure EPT entry. Software should consult the Secure EPT information returned in RDX.</p> <p>In other cases, RCX returns 0.</p>
RDX	<p>In case of an interruption, as indicated by RAX returning TDX_INTERRUPTED_RESUMABLE, RDX is unmodified.</p> <p>Else, RDX returns extended error information part 2.</p> <p>In case of EPT walk error, Secure EPT entry level and state where the error was detected – see 3.6.2</p> <p>In other cases, RDX returns 0.</p>
R8	<p>For TDH.MEM.SEPT.ADD version 1 or higher:</p> <ul style="list-style-type: none"> <li>• If a provided L1 SEPT page has been added, R8 returns NULL_PA (-1).</li> <li>• Else, if an L1 SEPT page already exists, bit 63 of R8 is set to 1, other bits are unmodified.</li> <li>• Else, bit 63 of R8 is cleared to 0, other bits are unmodified.</li> </ul> <p>For TDH.MEM.SEPT.ADD version 0, R8 is unmodified.</p>
R9	<p>For TDH.MEM.SEPT.ADD version 1 or higher:</p> <ul style="list-style-type: none"> <li>• If a provided L2 VM #1 SEPT page has been added, R9 returns NULL_PA (-1).</li> <li>• Else, if an L2 VM #1 SEPT page already exists, bit 63 of R9 is set to 1, other bits are unmodified.</li> <li>• Else, bit 63 of R9 is cleared to 0, other bits are unmodified.</li> </ul> <p>For TDH.MEM.SEPT.ADD version 0, R9 is unmodified.</p>

Operand	Description
R10	<p>For TDH.MEM.SEPT.ADD version 1 or higher:</p> <ul style="list-style-type: none"> <li>If a provided L2 VM #2 SEPT page has been added, R10 returns NULL_PA (-1).</li> <li>Else, if an L2 VM #2 SEPT page already exists, bit 63 of R10 is set to 1, other bits are unmodified.</li> <li>Else, bit 63 of R10 is cleared to 0, other bits are unmodified.</li> </ul> <p>For TDH.MEM.SEPT.ADD version 0, R10 is unmodified.</p>
R11	<p>For TDH.MEM.SEPT.ADD version 1 or higher:</p> <ul style="list-style-type: none"> <li>If a provided L2 VM #3 SEPT page has been added, R11 returns NULL_PA (-1).</li> <li>Else, if an L2 VM #3 SEPT page already exists, bit 63 of R11 is set to 1, other bits are unmodified.</li> <li>Else, bit 63 of R11 is cleared to 0, other bits are unmodified.</li> </ul> <p>For TDH.MEM.SEPT.ADD version 0, R11 is unmodified.</p>
Other	Unmodified

### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 5 TDH.MEM.SEPT.ADD adds a set of 4KB Secure EPT pages to a TD and maps them to the provided GPA and level. SEPT pages can be added to the main (L1) SEPT tree and/or to one or more of the L2 VMs' SEPT trees. TDH.MEM.SEPT.ADD initializes the SEPT pages to hold 512 free entries using the TD's ephemeral private key.

10 L2 SEPT trees may not be deeper than the L1 SEPT tree. To add an L2 SEPT page at some level, there must either already be an L1 SEPT page at that level, or an L1 SEPT page at that level is being added by the current TDH.MEM.SEPT.ADD invocation.

**Interruptibility:** TDH.MEM.SEPT.ADD is interruptible. If a pending interrupt is detected during operation, TDH.MEM.SEPT.ADD returns with a TDX\_INTERRUPTED\_RESUMABLE status in RAX. The SEPT page HPA values in R8, R9, R10 and R11 are updated.

TDH.MEM.SEPT.ADD is designed to be invoked in a loop until all required SEPT pages have been added:

- 15
1. Call TDH.MEM.SEPT.ADD.
  2. While RAX indicates TDX\_INTERRUPTED\_RESUMABLE:
    - 2.1. Call TDH.MEM.SEPT.ADD with the GPR values as returned by the previous call.
    - 2.2. If an error indication other than TDX\_INTERRUPTED\_RESUMABLE is returned, abort.

20 The table below shows the values of the SEPT HPA arguments in R8 – R11, when version 1 or higher is selected and no error occurs (RAX returns TDX\_SUCCESS or TDX\_INTERRUPTED\_RESUMABLE).

**Table 5.118: Meaning of TDH.MEM.SEPT.ADD's SEPT HPA Arguments on Input and Output (Version > 0, No Error)**

Value	R8 – R11 on Input	R8 – R11 on Output
NULL_PA (-1)	No new SEPT page to add	New SEPT page has been added
Bits 62:0: Valid HPA, HKID bits are 0 Bit 63: 0	Bits 62:0: HPA of new SEPT page to add	N/A
Bits 62:0: Valid HPA, HKID bits are 0 Bit 63: 1	Bit 63: Ignored	SEPT page already exists, new SEPT page has not been used

**Atomicity:** Unless terminated with a TDX\_INTERRUPTED\_RESUMABLE indication, TDH.MEM.SEPT.ADD either fully succeeds in adding the requested SEPT pages, or doesn't add any page.

In case of an interrupt (TDX\_INTERRUPTED\_RESUMABLE), if the host VMM invokes TDH.MEM.SEPT.ADD in a loop as described above, and doesn't initiate other operations that impact TDH.MEM.SEPT.ADD (e.g., TDH.MEM.RANGE.BLOCK), then this atomicity still holds at the end of the loop.

- 5 To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.119: TDH.MEM.SEPT.ADD Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	GPA and Level	Secure EPT page (GPA) <sup>5</sup>	SEPT_PAGE	RW	Private	2 <sup>12+9*Level</sup> Bytes	N/A	N/A	N/A
Explicit	RDX	HPA	TDR page	TDR	RW	Opaque	4KB	Shared	Shared	Shared
Explicit	R8	HPA	Secure EPT page (HPA) <sup>5</sup>	SEPT_PAGE	RW	Private	4KB	Exclusive	Shared	Shared
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT tree	N/A	RW	Private	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT entry	SEPT Entry	RW	Private	N/A	Exclusive	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT trees	N/A	RW	Private	N/A	Shared(i)	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT entry	SEPT Entry	RW	Private	N/A	Exclusive(i)	N/A	N/A

- 10 TDH.MEM.SEPT.ADD checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

The function checks the following conditions:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
- 15 4. The TD must have been initialized (TDR.NUM\_TDCX is the required number and TDCS.OP\_STATE is either INITIALIZED or RUNNING).
5. The specified level is of an EPT non-leaf entry – i.e., 1 to 3 for 4-level EPT or 1 to 4 for 5-level EPT. See 3.6.1 for a definition of EPT level.

If successful, the function does the following:

- 20 6. SEPT physical pages checks:
  - 6.1. If the version number indicated in RAX is 1 or higher, check and lock the new SEPT physical pages:
    - 6.1.1. For each of the SEPT page HPAs provided in R8, R9, R10 and R11, if the value is not NULL\_PA, lock the PAMT entry and check that the page metadata is correct (PT must be PT\_NDA).

<sup>5</sup> RCX and R8 denote the same Secure EPT page operand, using HPA and GPA respectively

6.2. Else (version is 0):

- 6.2.1. For the L1 SEPT page HPA provided in R8, lock the PAMT entry and check that the page metadata is correct (PT must be PT\_NDA).

If passed:

5 7. SEPT trees walk and state checks:

- 7.1. Walk the L1 Secure EPT based on the requested GPA and level and find the SEPT entry.

If L1 SEPT walk succeeded:

7.2. If requested to add an L1 SEPT page (i.e., R8 is not a NULL\_PA):

- 7.2.1. The L1 SEPT entry state should either be FREE,

10 7.2.2. Or, if ALLOW\_EXISTING is 1, the L1 SEPT entry state can be NL\_MAPPED (i.e., there's already an L1 SEPT page at the requested level). In this case, mark the return value in R8 to indicate that the new L1 SEPT page was not used.

- 7.2.3. On other L1 SEPT entry states, abort with an error indication in RAX.

7.3. Else (no L1 SEPT page is to be added):

15 7.3.1. Check that an L1 SEPT page exists (the L1 SEPT entry state is NL\_MAPPED).

If passed:

7.4. If the version number indicated in RAX is 1 or higher, then for each L2 VM do the following:

- 7.4.1. Walk the L2 Secure EPT based on the requested GPA and level and find the L2 SEPT entry.

If L2 SEPT walk succeeded:

- 20 7.4.2. The L2 SEPT entry state should either be L2\_FREE,

7.4.3. Or, if ALLOW\_EXISTING is 1, the L2 SEPT entry state can be L2\_NL\_MAPPED (i.e., there's already an L2 SEPT page at the requested level). In this case, mark the return value in R9, R10 or R11 to indicate that the new SEPT page was not used.

- 7.4.4. On other L2 SEPT entry states, abort with an error indication in RAX.

25 If passed:

8. SEPT page additions:

8.1. If the L1 SEPT entry state is FREE:

- 8.1.1. Initialize the new L1 Secure EPT page, indicating 512 entries in the FREE state, using the TD's ephemeral private HKID and direct writes (MOVDIR64B).

30 8.1.2. Update the parent L1 Secure EPT entry with the new Secure EPT page HPA and NL\_MAPPED state.

- 8.1.3. Increment TDR.CHLDCNT.

8.1.4. Update the new Secure EPT page's PAMT entry with the PT\_EPT page type and the TDR physical address as the OWNER.

35 8.1.5. If the version number indicated in RAX is 1 or higher, set the returned value of R8 to NULL\_PA, indicating that a new SEPT page has been added.

If passed:

8.2. If the version number indicated in RAX is 1 or higher, then for each L2 VM do the following:

8.2.1. If the L2 SEPT entry state is L2\_FREE:

- 8.2.1.1. If an interrupt is pending, abort with a TDX\_INTERRUPTED\_RESUMABLE status.

40 8.2.1.2. Initialize the new L2 Secure EPT page, indicating 512 entries in the L2\_FREE state, using the TD's ephemeral private HKID and direct writes (MOVDIR64B).

- 8.2.1.3. Update the parent Secure EPT entry with the new Secure EPT page HPA and NL\_MAPPED state.

- 8.2.1.4. Increment TDR.CHLDCNT.

45 8.2.1.5. Update the new L2 Secure EPT page's PAMT entry with the PT\_EPT page type and the TDR physical address as the OWNER.

- 8.2.1.6. Set the returned value of R9, R10 or R11 to NULL\_PA, indicating that a new L2 SEPT page has been added.

## Completion Status Codes

Table 5.120: TDH.MEM.SEPT.ADD Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_EPT_ENTRY_NOT_FREE	
TDX_EPT_WALK_FAILED	
TDX_INTERRUPTED_RESUMABLE	TDH.MEM.SEPT.ADD's operation has been interrupted by an external event; it may be resumed from the point it was interrupted by calling it again.
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.MEM.SEPT.ADD is successful
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

**5.3.30. UPDATED: TDH.MEM.SEPT.RD Leaf**

Read a Secure EPT entry.

**Table 5.121: TDH.MEM.SEPT.RD Input Operands Definition**

Operand		Description		
RAX	Leaf and Version	SEAMCALL instruction leaf number and version, see 5.3.1		
		Bits	Field	Description
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version
		63:24	Reserved	Must be 0
RCX	GPA Mapping	EPT mapping information:		
		Bits	Name	Description
		2:0	Level	Level of the Secure EPT entry to read – see 3.6.1 Level must be between 0 and 3 for a 4-level EPT or between 0 and 4 for a 5-level EPT.
		11:3	Reserved	Reserved: must be 0
		51:12	GPA	Bits 51:12 of the guest physical address for the Secure EPT entry to read Depending on the level, the following least significant bits must be 0: Level 0 (EPTE): None Level 1 (EPDE): Bits 20:12 Level 2 (EPDPTE): Bits 29:12 Level 3 (EPML4E): Bits 38:12 Level 4 (EPML5E): Bits 47:12
		63:52	Reserved	Reserved: must be 0
RDX	TD Handle and Flags	TD handle and flags:		
		Bits	Name	Description
		0	READ_L2_ATTR	Flags that L2 attributes should be returned in R8
		11:1	Reserved	Reserved: must be 0
		51:12	HPA	Bits 51:12 of the host physical address of the parent TDR page (HKID bits must be 0)
		63:52	Reserved	Reserved: must be 0

**Table 5.122: TDH.MEM.SEPT.RD Output Operands Definition**

Operand		Description
RAX	Status	SEAMCALL instruction return code – see 5.3.1

Operand		Description
RCX	SEPT Entry	Secure EPT entry architectural content – see 3.6.2 The architectural content represents how the Secure EPT maps a private memory page or a Secure EPT page; it may be different than the actual contents of the Secure EPT entry. Software should consult the Secure EPT information returned in RDX. <ul style="list-style-type: none"> <li>In case of successful operation, the requested entry's architectural content is returned.</li> <li>In case of EPT walk error, the architectural content of the Secure EPT entry where the error was detected is returned.</li> </ul> In other cases, RCX returns 0.
RDX	SEPT Level and State	Secure EPT entry level and state – see 3.6.2 <ul style="list-style-type: none"> <li>In case of successful operation, the requested entry's information is returned.</li> <li>In case of EPT walk error, the information of the Secure EPT entry where the error was detected is returned.</li> </ul> In other cases, RDX returns 0.
R8	L2 Attributes	If the TD's ATTRIBUTES.DEBUG is 1 and READ_L2_ATTR is 1, R8 returns the L2 attributes of the applicable L2 SEPT entries, in the format defined in 3.6.3. Else, R8 is unmodified.

#### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 5 TDH.MEM.SEPT.RD reads a Secure EPT entry. If the TD's ATTRIBUTES.DEBUG is 1, then TDH.MEM.SEPT.RD can return the page's L2 attributes.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.123: TDH.MEM.SEPT.RD Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	GPA and Level	Secure EPT entry	SEPT_ENTRY	R	Private	2 <sup>12+9*Level</sup> Bytes	None	None	None
Explicit	RDX	HPA	TDR page	TDR	R	Opaque	4KB	Shared	Shared	Shared
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT Tree	N/A	R	Private	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT entry	SEPT Entry	RW	Private	N/A	None	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT Tree	N/A	R	Private	N/A	Shared(i)	N/A	N/A



Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Implicit	N/A	GPA	L2 Secure EPT entries	SEPT Entry	RW	Private	N/A	Exclusive(i)	N/A	N/A

TDH.MEM.SEPT.RD checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

The function checks the following conditions:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
4. The TD must have been initialized (TDR.NUM\_TDCX is the required number and TDCS.OP\_STATE is either INITIALIZED or RUNNING).
5. The specified level is of an EPT entry (i.e., 0 to 3 for 4-level EPT or 0 to 4 for 5-level EPT) – see 3.6.1 for a definition of EPT level.

If successful, the function does the following:

6. If READ\_L2\_ATTR is set, check that TDCS.ATTRIBUTES.DEBUG is 1.

If passed:

7. Walk the L1 Secure EPT based on the GPA and level operand and find the Secure EPT entry.
8. Read the L1 Secure EPT entry contents.
9. If READ\_L2\_ATTR is set:
  - 9.1. If the L1 SEPT entry is a leaf entry, then for each L2 VM where the page is mapped
    - 9.1.1. Walk the L2 SEPT based on the GPA and level operand and find the L2 SEPT entry.
    - 9.1.2. Read the leaf L2 SEPT entry and build the L2 attributes to be returned.
  - 9.2. If the L1 SEPT entry is a non-leaf entry, then for each L2 VM:
    - 9.2.1. Walk the L2 SEPT based on the GPA and level operand, and find the non-leaf L2 SEPT entry, if any.
    - 9.2.2. Read the non-leaf L2 SEPT entry and build the L2 attributes to be returned.
  - 9.3. Else (the L1 SEPT entry is FREE):
    - 9.3.1. Return 0 as the L2 attributes.

## Completion Status Codes

Table 5.124: TDH.MEM.SEPT.RD Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_EPT_WALK_FAILED	
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.MEM.SEPT.RD is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	

Completion Status Code	Description
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

DRAFT

**5.3.31. UPDATED: TDH.MEM.SEPT.REMOVE Leaf**

Remove an empty L1 Secure EPT page and any associated L2 SEPT pages from a TD.

**Table 5.125: TDH.MEM.SEPT.REMOVE Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Versions 0 and 1 are supported.
	63:24	Reserved	Must be 0
RCX	EPT mapping information:		
	Bits	Name	Description
	2:0	Level	Level of the non-leaf Secure EPT entry that maps the Secure EPT page to be removed – see 3.6.1  Level must be between 1 and 3 for a 4-level EPT or between 1 and 4 for a 5-level EPT.
	11:3	Reserved	Reserved: must be 0
	51:12	GPA	Bits 51:12 of the guest physical address for the Secure EPT page to be removed  Depending on the level, the following least significant bits must be 0: Level 1 (EPT): Bits 20:12 Level 2 (EPD): Bits 29:12 Level 3 (EPDPT): Bits 38:12 Level 4 (EPML4): Bits 47:12
	63:52	Reserved	Reserved: must be 0
RDX	Host physical address of the parent TDR page (HKID bits must be 0)		

5

**Table 5.126: TDH.MEM.SEPT.REMOVE Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
RCX	<p>If TDH.MEM.SEPT.REMOVE succeeded, RCX returns the HPA of the removed SEPT page.</p> <p>In case of EPT walk error, Secure EPT entry architectural content where the error was detected – see 3.6.2</p> <p>The architectural content represents how the Secure EPT maps a private memory page or a Secure EPT page, and may be different than the actual contents of the Secure EPT entry. Software should consult the Secure EPT information returned in RDX.</p> <p>In other cases, RCX returns 0.</p>

Operand	Description
RDX	Extended error information part 2 In case of EPT walk error, Secure EPT entry level and state where the error was detected – see 3.6.2 In other cases, RDX returns 0.
R9	If TDH.MEM.SEPT.REMOVE version is 1 or higher: <ul style="list-style-type: none"> <li>If L2 VM #1's L2 SEPT page has been removed, R9 returns the HPA of that SEPT page.</li> <li>Else, R9 returns NULL_PA (-1).</li> </ul> Else, R9 is unmodified.
R10	If TDH.MEM.SEPT.REMOVE version is 1 or higher: <ul style="list-style-type: none"> <li>If L2 VM #2's L2 SEPT page has been removed, R10 returns the HPA of that SEPT page.</li> <li>Else, R10 returns NULL_PA (-1).</li> </ul> Else, R10 is unmodified.
R11	If TDH.MEM.SEPT.REMOVE version is 1 or higher: <ul style="list-style-type: none"> <li>If L2 VM #3's L2 SEPT page has been removed, R11 returns the HPA of that SEPT page.</li> <li>Else, R11 returns NULL_PA (-1).</li> </ul> Else, R11 is unmodified.
Other	Unmodified

### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 5 TDH.MEM.SEPT.REMOVE removes an empty Secure EPT page **or pages**, with all 512 entries marked as FREE, from the TD's Secure EPT trees.

The L1 SEPT page and all existing L2 SEPT pages at the requested GPA and level are removed.

On successful operation, it TDH.MEM.SEPT.REMOVE marks the **removed** 4KB physical pages as free in PAMT.

- 10 **Blocking and TLB Tracking:** If the TD may be running, the removed GPA range must be blocked and TLB tracked. Else (e.g., TDH.MR.FINALIZE has not yet been executed, or the TD has been paused for export), no blocking and tracking is required.

**Atomicity:** TDH.MEM.SEPT.REMOVE either fully succeeds in removing the requested SEPT pages or doesn't remove any page.

- 15 **Cache Flush and Init:** After the SEPT pages have been removed, the host VMM should flush the physical pages' cache lines and initialize their content before they are reused, as described in the [Base Spec].

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.127: TDH.MEM.SEPT.REMOVE Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	GPA and Level	Secure EPT page	SEPT_PAGE	R	Private	2 <sup>12+9*Level</sup> Bytes	Exclusive	None	None

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RDX	HPA	TDR page	TDR	RW	Opaque	4KB	Shared	Shared	Shared
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT Tree	N/A	RW	Private	N/A	Exclusive	N/A	N/A
Implicit	N/A	GPA	L1 Secure EPT entry	SEPT Entry	RW	Private	N/A	Exclusive(h)	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT Trees	N/A	RW	Private	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	GPA	L2 Secure EPT entries	SEPT Entry	RW	Private	N/A	Exclusive(i)	N/A	N/A

TDH.MEM.SEPT.REMOVE checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

The function checks the following conditions:

- 5 1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
4. The TD must either have been initialized by TDH.MNG.INIT, or an import session has begun by TDH.IMPORT.STATE.IMMUTABLE.
- 10 5. The specified level is of a non-leaf EPT entry (i.e., 1 to 3 for 4-level EPT or 1 to 4 for 5-level EPT) – see 3.6.1 for a definition of EPT level.

If successful, the function does the following:

6. Walk the L1 Secure EPT based on the GPA operand and find the non-leaf SEPT entry of the SEPT page to be removed.
7. If TLB tracking is required (based on the Secure EPT entry state and the TD's OP\_STATE):
  - 15 7.1. Check the L1 Secure EPT entry is a non-leaf blocked (NL\_BLOCKED) entry.
  - 7.2. Check that TLB tracking was done.
8. Scan the L1 Secure EPT page content and check all 512 entries are FREE.

If passed:

9. For each L2 VM:
  - 20 9.1. Walk the L2 Secure EPT based on the GPA and level operand and find the applicable non-leaf SEPT entry.
  - 9.2. If an L2 SEPT entry was found, and its state is not FREE:
    - 9.2.1. Atomically decrement TDR.CHLDCNT.
    - 9.2.2. Set the PAMT entry of the removed Secure EPT page to PT\_NDA.
    - 9.2.3. Set the parent L2 Secure EPT entry to FREE.

25 If passed:

10. Set the parent L1 Secure EPT entry to FREE.
11. Atomically decrement TDR.CHLDCNT.
12. Set the PAMT entry of the removed L1 Secure EPT page to PT\_NDA.

## Completion Status Codes

Table 5.128: TDH.MEM.SEPT.REMOVE Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_EPT_ENTRY_NOT_FREE	
TDX_EPT_WALK_FAILED	
TDX_GPA_RANGE_NOT_BLOCKED	
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.MEM.SEPT.REMOVE is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	
TDX_TLB_TRACKING_NOT_DONE	

**5.3.32. UPDATED: TDH.MEM.TRACK Leaf**

Increment the TD's TLB epoch counter.

**Table 5.129: TDH.MEM.TRACK Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	The physical address of the parent TDR page (HKID bits must be 0)		

5

**Table 5.130: TDH.MEM.TRACK Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.MEM.TRACK increments the TD's TLB epoch counter.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.131: TDH.MEM.TRACK Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDR page	TDR	R	Opaque	4KB	Shared	Shared	Shared
Implicit	N/A	N/A	TDCS structure	TDR	RW	Opaque	4KB	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	N/A	TDCS Epoch Tracking Fields	N/A	RW	Opaque	N/A	Exclusive	N/A	N/A

- 15 In addition to the memory operand checks per the table above, the function checks the following:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
4. The TD must have been initialized (by TDH.MNG.INIT or TDH.IMPORT.STATE.IMMUTABLE).

If successful, the function does the following as a critical section, protected by exclusively locking the TDCS epoch tracking fields TD\_EPOCH and REFCOUNT. A concurrent TDH.VP.ENTER may cause this locking to fail with a TDX\_OPERAND\_BUSY status code; in this case the caller is expected to retry TDH.MEM.TRACK.

5. Lock the TDCS epoch tracking fields in exclusive mode.
6. Check that the TD's previous epoch's REFCOUNT is 0. This helps ensure that no REFCOUNT information will be lost when TD\_EPOCH is incremented in the next step.
7. If successful, increment the TD's epoch counter (TDCS.TD\_EPOCH).
8. Release the exclusive mode locking of the epoch tracking fields.

### Completion Status Codes

**Table 5.132: TDH.MEM.TRACK Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.  Note the special case where the indicated operand is TLB_EPOCH. This may happen due to a conflict with TDH.VP.ENTER. The host VMM should retry TDH.MEM.TRACK.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_PREVIOUS_TLB_EPOCH_BUSY	
TDX_SUCCESS	TDH.MEM.TRACK is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_FINALIZED	
TDX_TD_NOT_INITIALIZED	



### 5.3.33. TDH.MEM.WR Leaf

Write a 64b chunk from a debuggable guest TD private memory.

**Table 5.133: TDH.MEM.WR Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	The guest physical address of a naturally aligned 8-byte chunk of a guest TD private page		
RDX	Host physical address of the parent TDR page (HKID bits must be 0)		
R8	Data to be written to memory		

**Table 5.134: TDH.MEM.WR Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
RCX	Secure EPT entry architectural content – see 3.6.2 The architectural content represents how the Secure EPT maps a private memory page or a Secure EPT page, and may be different than the actual contents of the Secure EPT entry. Software should consult the Secure EPT information returned in RDX. <ul style="list-style-type: none"> <li>In case of successful operation, the requested entry's architectural content is returned.</li> <li>In case of EPT walk error, the architectural content of the Secure EPT entry where the error was detected is returned.</li> </ul> In other cases, RCX returns 0.
RDX	Secure EPT entry level and state – see 3.6.2 <ul style="list-style-type: none"> <li>In case of successful operation, the requested entry's information is returned.</li> <li>In case of EPT walk error, the information of the Secure EPT entry where the error was detected is returned.</li> </ul> In other cases, RDX returns 0.
R8	Previous content of the memory chunk In case of an error, as indicated by RAX, R8 returns 0
Other	Unmodified

#### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.MEM.WR writes a 64b chunk to a debuggable guest TD private memory.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.135: TDH.MEM.WR Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	GPA	TD private memory	Blob	RW	Private	8B	None	None	None
Explicit	RDX	HPA	TDR page	TDR	R	Opaque	4KB	Shared	Shared	Shared
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	R	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	Secure EPT tree	N/A	R	Private	N/A	Shared	N/A	N/A
Implicit	N/A	GPA	Secure EPT entry	SEPT Entry	R	Private	N/A	Exclusive	N/A	N/A

- 5 TDH.MEM.WR checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

The function checks the following conditions:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
- 10 3. The TD keys are configured on the hardware (TDR.KEY\_STATE is TD\_KEYS\_CONFIGURED).
4. TDCS must have been initialized (TDR.INIT is TRUE).
5. The TD is debuggable (TDCS.ATTRIBUTES.DEBUG is 1).

If successful, the function does the following:

6. Walk the Secure EPT based on the GPA operand and find the leaf entry.
- 15 7. Check that the Secure EPT entry state is PRESENT.

If passed:

8. Read the content of the memory chunk.
9. Write the new content of the memory chunk.

#### Completion Status Codes

20 **Table 5.136: TDH.MEM.WR Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TO BE COMPLETED	
TDX_SUCCESS	

DRAFT

**5.3.34. NEW: TDH.MIG.STREAM.CREATE Leaf**

Create a Migration Stream and its MIGSC control structure.

**Table 5.137: TDH.MIG.STREAM.CREATE Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	The physical address of a page where MIGSC will be created		
RDX	The physical address of the owner TDR page (HKID bits must be 0)		

**Table 5.138: TDH.MIG.STREAM.CREATE Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.MIG.STREAM.CREATE creates a new Migration Stream and its MIGSC control structure. This function can be invoked at any time after the TDCS pages have been allocated.

TDH.MIG.STREAM.CREATE can only be successfully invoked if no migration session is in progress.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.139: TDH.MIG.STREAM.CREATE Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	MIGSC page	MIGSC	RW	Opaque	4KB	Exclusive	Shared	Shared
Explicit	RDX	HPA	TDR page	TDR	RW	Opaque	4KB	Shared	Shared	Shared
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	4KB	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	N/A	Migration context	N/A	RW	Opaque	N/A	Exclusive	N/A	N/A

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Implicit	N/A	N/A	Mig. Stream context	Mig. Stream context	RW	Opaque	N/A	Exclusive(i)	N/A	N/A

In addition to the explicit memory operand checks per the table above, the function checks the following conditions:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
- 5 3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
4. TDCS pages have been allocated (TDR.NUM\_TDCX is the required number).
5. No migration session is in progress (TDCS.OP\_STATE is none of \*\_EXPORT or \*\_IMPORT).
6. The MIGSC page metadata in PAMT is correct (PT is PT\_NDA).
7. The number of already created migration streams is lower than the maximum allowed.
- 10 If successful, the function does the following:
  8. Increment the number of migration streams (TDCS.NUM\_MIG\_STREAMS).
  9. Initialize the MIGSC page contents using direct write (MOVDIR64B).
  10. Initialize the applicable forward link entry in TDCS (TDCS.MIGSC\_LINK):
    - o Set MIGSC\_PA to the MIGSC page HPA.
    - 15 o Clear the INITIALIZED and ENABLED flags.
  11. Atomically increment TDR.CHLD CNT.
  12. Initialize the MIGSC page metadata in PAMT (Set PT to PT\_TDCX, OWNER to the TDR HPA).

#### Completion Status Codes

**Table 5.140: TDH.MIG.STREAM.CREATE Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.MIG.STREAM.CREATE is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_FINALIZED	
TDX_TD_KEYS_NOT_CONFIGURED	

#### 5.3.35. **UPDATED:** TDH.MNG.ADDCX Leaf

Add a TDCX page to a guest TD.

Table 5.141: TDH.MNG.ADDCX Input Operands Definition

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	The physical address of a page where TDCX will be added (HKID bits must be 0)		
RDX	The physical address of the owner TDR page (HKID bits must be 0)		

Table 5.142: TDH.MNG.ADDCX Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
Other	Unmodified

### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.MNG.ADDCX adds a TDCX page, which is a child of the specified TDR.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

Table 5.143: TDH.MNG.ADDCX Operands Information Definition

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDCX page	Blob	RW	Opaque	4KB	Exclusive	Shared	Shared
Explicit	RDX	HPA	TDR page	TDR	RW	Opaque	4KB	Exclusive	Shared	Shared

In addition to the explicit memory operand checks per the table above, the function checks the following conditions:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
- ~~3. The TD must not have been initialized (TDR.INIT is FALSE).~~
4. The number of TDCX pages (TDR.NUM\_TDCX) is smaller than the required number.
5. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
6. The new TDCX page metadata in PAMT must be correct (PT must be PT\_NDA).

If successful, the function does the following:

7. Initialize the TDCX page contents using direct writes (MOVDIR64B).
8. Set the TDCX pointer entry in the TDR.TDCX\_PA array.
9. Increment TDR.NUM\_TDCX.
10. If TDR.NUM\_TDCX is equal to the required number of TDCX pages:

- 10.1. Mark the TD as uninitialized (set TDCS.OP\_STATE to UNINITIALIZED).
- 10.2. Generate a migration encryption key, to be used in the next migration session.

If failed:

- 10.3. Decrement TDR.NUM\_TDCX.

## 5 Completion Status Codes

**Table 5.144: TDH.MNG.ADDCX Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_RND_NO_ENTROPY	Failed to generate a random migration encryption key. This is typically caused by an entropy error of the CPU's random number generator, and may be impacted by RDSEED, RDRAND or PCONFIG executing on other LPs. The operation should be retried.
TDX_SUCCESS	TDH.MNG.ADDCX is successful
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_INITIALIZED	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TDCX_NUM_INCORRECT	

### 5.3.36. TDH.MNG.CREATE Leaf

Create a new guest TD and its TDR root page.

**Table 5.145: TDH.MNG.CREATE Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	The physical address of a page where TDR will be created (HKID bits must be 0)		
RDX	Bits	Name	Description
	15:0	HKID	The TD's ephemeral private HKID
	63:16	Reserved	Reserved: must be 0

**Table 5.146: TDH.MNG.CREATE Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
Other	Unmodified

#### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.MNG.CREATE creates a TDR page which is the root page of a new guest TD.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.147: TDH.MNG.CREATE Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDR page	TDR	RW	Opaque	4KB	Exclusive	Shared	Shared
Implicit	N/A	N/A	KOT	KOT	N/A	Hidden	N/A	Exclusive	N/A	N/A

- 15 In addition to the explicit memory operand checks per the table above, the function checks the following conditions:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_NDA).
2. The value of the specified HKID must be in the range configured for TDX.
3. The KOT entry for the specified HKID must be marked as HKID\_FREE.



If successful, the function does the following:

4. Zero out the TDR page contents using direct write (MOVDIR64B).
5. Initialize the key management fields.
6. Initialize the state variables.
7. Initialize the TD management fields.
8. Initialize the TD preserving fields (handoff version and current SEAMDB entry's index/nonce pair).
9. Mark the KOT entry for the specified HKID as HKID\_ASSIGNED.
10. Initialize the TDR page metadata in PAMT.

### Completion Status Codes

**Table 5.148: TDH.MNG.CREATE Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_HKID_NOT_FREE	
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_RND_NO_ENTROPY	Random TD_UUID generation (e.g., RDRAND or RDSEED) failed because the hardware random number generator did not have enough entropy. The host VMM should retry the operation.
TDX_SUCCESS	TDH.MNG.CREATE is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	

**5.3.37. UPDATED: TDH.MNG.INIT Leaf**

Initialize TD-scope control structures TDR and TDCS.

**Table 5.149: TDH.MNG.INIT Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	The physical address of a TDR page (HKID bits must be 0)		
RDX	The physical address (including HKID bits) of an input TD_PARAMS_STRUCT		

**Table 5.150: TDH.MNG.INIT Output Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction return code – see 5.3.1		
RCX	Extended error information In case of a TD_PARAMS_STRUCT.CPUID_CONFIG error, RCX returns the applicable CPUID information as shown below. In all other cases, RCX returns 0.		
	Bits	Name	Description
	31:0	LEAF	CPUID leaf number
	63:32	SUBLEAF	CPUID sub-leaf number: if sub-leaf is not applicable, value is -1 (0xFFFFFFFF).
Other	Unmodified		

**Leaf Function Latency**

TDH.MNG.INIT execution time may be longer than most TDX module interface functions execution time. No interrupts (including NMI and SMI) are processed by the logical processor during that time.

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.MNG.INIT initializes the TD-scope control structures TDR and TDCS based on a set of TD parameters provided as input.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

Table 5.151: TDH.MNG.INIT Operands Information Definition

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDR page	TDR	RW	Opaque	4KB	Exclusive	Shared	Shared
Explicit	RDX	HPA	TD Parameters	TD_PARAMS	R	Shared	1024B	None	N/A	N/A
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Exclusive(i)	N/A	N/A

In addition to the explicit memory operand checks per the table above, the function checks the following conditions:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
  2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
  - ~~3. The TD must not have been initialized (TDR.INIT is FALSE).~~
  4. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
  5. All the **required** TDCS pages have been added (by TDH.MNG.ADDCX) but the **TD has not have been initialized (TDCS.OP\_STATE is UNINITIALIZED)**.
- If successful, the function does the following:
6. Set the TDCS TD management fields to their initial values.
  7. Read the input parameters structure fields.
  8. Check the input parameters and initialize the TDCS logical structure.
    - 8.1. Check that ATTRIBUTES and XFAM bits that must be fixed-0 or fixed-1 are set correctly.
    - 8.2. Check XFAM bit groups that must have certain values (e.g., AVX bits 7:5).
- If passed:
9. Initialize EPTP to point to TDCS.SEPT\_ROOT.
  10. Initialize the MSR bitmaps based on ATTRIBUTES and XFAM.
  11. Initialize the TDCS measurement fields.
  12. Mark the TD as initialized (set **TDCS.OP\_STATE to INITIALIZED**).

### Completion Status Codes

Table 5.152: TDH.MNG.INIT Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.MNG.INIT is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_INITIALIZED	

Completion Status Code	Description
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TDCX_NUM_INCORRECT	

DRAFT

### 5.3.38. TDH.MNG.KEY.CONFIG Leaf

Configure the TD ephemeral private key on a single package.

**Table 5.153: TDH.MNG.KEY.CONFIG Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	The physical address of a TDR page (HKID bits must be 0)		

**Table 5.154: TDH.MNG.KEY.CONFIG Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
Other	Unmodified

#### Leaf Function Latency

TDH.MNG.KEY.CONFIG execution time may be longer than most TDX module interface functions execution time. No interrupts (including NMI and SMI) are processed by the logical processor during that time.

#### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.MNG.KEY.CONFIG configures the TD's ephemeral private key on a single package.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.155: TDH.MNG.KEY.CONFIG Operands Information**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDR page	TDR	RW	Opaque	4KB	Exclusive	Shared	Shared
Implicit	N/A	N/A	KETs on current package	N/A	N/A	Hidden	N/A	Exclusive	N/A	N/A

In addition to the memory operand checks per the table above, the function checks the following conditions:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. HKID has been assigned to the TD; TDR.LIFECYCLE\_STATE is TD\_HKID\_ASSIGNED.

If successful, the function does the following:

4. Configure the TD ephemeral private key on the package.
  - 4.1. This operation may fail due to a conflict with a concurrent TDH.MNG.KEY.CONFIG or PCONFIG running on the same package.
  - 4.2. A CPU-generated random key is used. The operation may fail due to lack of entropy.
5. If the key has been configured on all the packages, set TDR.LIFECYCLE\_STATE to TD\_KEYS\_CONFIGURED.

#### Completion Status Codes

**Table 5.156: TDH.MNG.KEY.CONFIG Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_KEY_CONFIGURED	
TDX_KEY_GENERATION_FAILED	Failed to generate a random key. This is typically caused by an entropy error of the CPU's random number generator, and may be impacted by RDSEED, RDRAND or PCONFIG executing on other LPs. The operation should be retried.
TDX_LIFECYCLE_STATE_INCORRECT	
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.  Specifically, key configuration may fail due to a concurrently running PCONFIG instruction.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.MNG.KEY.CONFIG is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	

### 5.3.39. TDH.MNG.KEY.FREEID Leaf

End the platform cache flush sequence, and mark applicable HKIDs in KOT as free.

**Table 5.157: TDH.MNG.KEY.FREEID Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	The physical address of a TDR page (HKID bits must be 0)		

**Table 5.158: TDH.MNG.KEY.FREEID Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
Other	Unmodified

#### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.MNG.KEY.FREEID ends the platform cache flush sequence for the HKIDs associated with the specified TD after TDH.PHYMEM.CACHE.WB has been executed on all the required packages. It marks the TD's HKIDs in KOT as free, and the TD itself as being torn down.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.159: TDH.MNG.KEY.FREEID Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDR page	TDR	RW	Opaque	4KB	Exclusive	Shared	Shared
Implicit	N/A	N/A	KOT	KOT	N/A	Hidden	N/A	Exclusive	N/A	N/A

In addition to the memory operand checks per the table above, the function checks the following conditions:

- The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
- TLB and VMCS caches associated with the HKID have been flushed, and no memory associated with this HKID may be accessed:
  - TDR.LIFECYCLE\_STATE is TD\_BLOCKED.
  - The KOT entry for the TD's private HKID is marked as HKID\_FLUSHED.
  - The KOT entry for the TD's private HKID indicates that TDH.PHYMEM.CACHE.WB has been executed on all applicable packages or cores.

If successful, the function does the following:

3. Mark the KOT entry as HKID\_FREE.
4. Set TDR.LIFECYCLE\_STATE to TD\_TEARDOWN.

#### Completion Status Codes

**Table 5.160: TDH.MNG.KEY.FREEID Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_LIFECYCLE_STATE_INCORRECT	
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.MNG.KEY.FREEID is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_WBCACHE_NOT_COMPLETE	

DRAFT



**5.3.40. TDH.MNG.KEY.RECLAIMID Leaf (Deprecated)**

This function is deprecated; it is provided for backward compatibility.

**Table 5.161: TDH.MNG.KEY.RECLAIMID Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0

5

**Table 5.162: TDH.MNG.KEY.RECLAIMID Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.MNG.KEY.RECLAIMID is provided for backward compatibility. It does not do anything except returning a constant TDX\_SUCCESS status.

**Completion Status Codes****Table 5.163: TDH.MNG.KEY.RECLAIMID Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_SUCCESS	TDH.MNG.KEY.RECLAIMID is successful.

**5.3.41. UPDATED: TDH.MNG.RD Leaf**

Read a TD-scope metadata field (control structure field) of a TD.

**Table 5.164: TDH.MNG.RD Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version <b>May be 0 or 1</b>
	63:24	Reserved	Must be 0
RCX	The physical address of a TDR page (HKID bits must be 0)		
RDX	Field <b>identifier</b> – see 3.10 The <b>LAST_ELEMENT_IN_FIELD</b> and <b>LAST_FIELD_IN_SEQUENCE</b> components of the field identifier must be 0. <b>WRITE_MASK_VALID</b> , <b>INC_SIZE</b> , <b>CONTEXT_CODE</b> and <b>ELEMENT_SIZE_CODE</b> components of the field identifier are ignored. For TDH.MNG.RD version 1 or higher, a value of -1 is a special case: it is not a valid field identifier; in this case the first readable field identifier is returned in RDX.		

**Table 5.165: TDH.MNG.RD Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
RDX	For TDH.MNG.RD version 0, RDX is unmodified. For TDH.MNG.RD version 1 or higher: <ul style="list-style-type: none"> <li>If the input field identifier was -1, RDX returns the first readable field identifier.</li> </ul> Else, in there case of an error, RDX returns -1. On success, RDX returns the next readable field identifier. A value of -1 indicates no next field identifier is available.
R8	Contents of the field In case of no success, as indicated by RAX, R8 returns 0.
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

10 TDH.MNG.RD reads a TD-scope metadata field (control structure field) of a TD.

If version 1 or higher is specified in RAX, RDX returns the next host-side readable field identifier. This may be used by the host VMM to dump the host readable TD metadata. To read all the available fields, the host VMM can invoke TDH.MNG.RD in a loop, starting with field identifier -1 as an input, until RDX returns -1.

15 To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.166: TDH.MNG.RD Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDR page	TDR	R	Opaque	4KB	Shared	Shared	Shared
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Shared	N/A	N/A

In addition to the memory operand checks per the table above, the function checks the following conditions:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
4. All the required TDCS pages have been added (TDR.NUM\_TDCX is the required number).

If the above checks passed:

5. Read the control structure field using the algorithm described in 5.2.2.1.

#### Completion Status Codes

**Table 5.167: TDH.MNG.RD Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_METADATA_FIRST_FIELD_ID_IN_CONTEXT	Indicates that the first field ID in context is returned
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.MNG.RD is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

### 5.3.42. TDH.MNG.VPFLUSHDONE Leaf

Check that none of the TD's VCPUs are associated with an LP.

**Table 5.168: TDH.MNG.VPFLUSHDONE Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	The physical address of a TDR page (HKID bits must be 0)		

**Table 5.169: TDH.MNG.VPFLUSHDONE Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
Other	Unmodified

#### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.MNG.VPFLUSHDONE checks that none of the TD's VCPUs are associated with an LP, and it then prepares for cache flushing by TDH.PHYMEM.CACHE.WB.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.170: TDH.MNG.VPFLUSHDONE Operands Information**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDR page	TDR	RW	Opaque	4KB	Exclusive	Shared	Shared
Implicit	N/A	N/A	TDCS structure	TDCS	R	Opaque	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	N/A	KOT	KOT	N/A	Hidden	N/A	Exclusive	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	None	Opaque	N/A	Exclusive(i)	N/A	N/A

In addition to the memory operand checks per the table above, the function checks the following conditions:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. TDR.LIFECYCLE\_STATE is either TD\_HKID\_ASSIGNED or TD\_KEYS\_CONFIGURED.
3. The KOT entry for the TD's assigned HKID in the list must be marked as HKID\_ASSIGNED.
- 20 4. None of the TD's VCPUs are associated with an LP (either the TD has not been initialized by TDH.MNG.INIT, or TDCS.NUM\_ASSOC\_VCPUS is 0).

If successful, the function does the following:

5. Set a bitmap in the KOT entry to track the required subsequent TDH.PHYMEM.CACHE.WB operations.
6. Set TDR.LIFECYCLE\_STATE to TD\_BLOCKED.
7. Mark the KOT entry as HKID\_FLUSHED.

## 5 Completion Status Codes

**Table 5.171: TDH.MNG.VPFLUSHDONE Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_FLUSHVP_NOT_DONE	
TDX_LIFECYCLE_STATE_INCORRECT	
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.MNG.VPFLUSHDONE is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	

DRAFT

**5.3.43. UPDATED: TDH.MNG.WR Leaf**

Write a TD-scope metadata field (control structure field) of a TD.

**Table 5.172: TDH.MNG.WR Input Operands Definitions**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	The physical address of a TDR page (HKID bits must be 0)		
RDX	Field <b>identifier</b> – see 3.10  The <b>LAST_ELEMENT_IN_FIELD</b> and <b>LAST_FIELD_IN_SEQUENCE</b> components of the field identifier must be 0.  <b>WRITE_MASK_VALID</b> , <b>INC_SIZE</b> , <b>CONTEXT_CODE</b> and <b>ELEMENT_SIZE_CODE</b> components of the field identifier are ignored.		
R8	Data to write to the field		
R9	A 64b write mask to indicate which bits of the value in R8 are to be written to the field		

**Table 5.173: TDH.MNG.WR Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
R8	Previous content of the field In case of an error, as indicated by RAX, R8 returns 0.
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.MNG.WR writes a TD-scope metadata field (control structure field) of a TD. The specific bits of the value (R8) are written as specified by the write mask (R9). Writing is subject to the field's writability.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.174: TDH.MNG.WR Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDR page	TDR	RW	Opaque	4KB	Shared	Shared	Shared

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Shared	N/A	N/A

In addition to the memory operand checks per the table above, the function checks the following conditions:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR)
2. The TD is not in a FATAL state (TDR.FATAL is FALSE)
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED)
4. All the required TDCS pages have been added (TDR.NUM\_TDCX is the required number).
5. The TD is debuggable (TDCS.ATTRIBUTES.DEBUG is 1)

If the above checks passed:

6. Write the control structure field and return its old value, using the algorithm described in 5.2.2.2.

## 10 Completion Status Codes

**Table 5.175: TDH.MNG.WR Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.MNG.WR is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NON_DEBUG	
TDX_TD_NOT_INITIALIZED	

**5.3.44. UPDATED: TDH.MR.EXTEND Leaf**

Extend the MRTD measurement register in the TDCS with the measurement of the indicated chunk of a TD page.

**Table 5.176: TDH.MR.EXTEND Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	The GPA of the TD page chunk to be measured		
RDX	The physical address of the TDR page of the target TD (HKID bits must be 0)		

**Table 5.177: TDH.MR.EXTEND Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
RCX	Extended error information part 1 In case of EPT walk error, Secure EPT entry where the error was detected In other cases, RCX returns 0.
RDX	Extended error information part 2 In case of EPT walk error, EPT level where the error was detected In other cases, RDX returns 0.
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.MR.EXTEND updates the MRTD measurement register in the TDCS with the measurement of the indicated chunk of a TD private page. For pages whose contents need to be measured, once the page is copied into the TD memory area, the host VMM will call TDH.MR.EXTEND multiple times to measure the pages contents into MRTD. TDEXEND can be executed only before TDH.MR.FINALIZE.

**Note:** TDH.MR.EXTEND works on a 256B chunk of a page, not on a full page, due to instruction latency considerations.

- 15 To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.



Table 5.178: TDH.MR.EXTEND Operands Information Definition

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	GPA	TD private page chunk	Blob	R	Private	256B	None	None	None
Explicit	RDX	HPA	TDR page	TDR	R	Opaque	4KB	Exclusive	Shared	Shared
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	4KB	Exclusive(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	GPA	Secure EPT tree	N/A	R	Private	N/A	Exclusive(i)	N/A	N/A
Implicit	N/A	GPA	Secure EPT entry	SEPT Entry	RW	Private	N/A	Exclusive(i)	N/A	N/A

In addition to the memory operand checks per the table above, the function checks the following:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
4. The TD must have been initialized but not finalized (TDR.NUM\_TDCX is the required number and TDCS.OP\_STATE is INITIALIZED).
5. The page must be mapped and accessible in the Secure EPT.

If successful, the function does the following:

6. Update the TD measurement in TDCS based on the chunk's GPA and contents.
7. Extend TDCS.MRTD with the chunk's GPA and contents. Extension is done using SHA384, with three 128B extension buffers. The first extension buffer is composed as follows:
  - Bytes 0 through 8 contain the ASCII string "MR.EXTEND".
  - Bytes 16 through 23 contain the GPA (in little-endian format).
  - All the other bytes contain 0.

The other two extension buffers contain the chunk's contents.

#### Completion Status Codes

Table 5.179: TDH.MR.EXTEND Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_EPT_ENTRY_NOT_PRESENT	
TDX_EPT_WALK_FAILED	
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.MR.EXTEND is successful.
TDX_SYS_NOT_READY	

Completion Status Code	Description
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_FINALIZED	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

DRAFT

**5.3.45. UPDATED: TDH.MR.FINALIZE Leaf**

TDH.MR.FINALIZE completes measurement of the initial TD contents and marks the TD as ready to run.

**Table 5.180: TDH.MR.FINALIZE Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	The physical address of the parent TDR page (HKID bits must be 0)		

**Table 5.181: TDH.MR.FINALIZE Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.MR.FINALIZE completes the measurement of the initial TD contents and marks the TD as finalized.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.182: TDH.MR.FINALIZE Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDR page	TDR	R	Opaque	4KB	Exclusive	Shared	Shared
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	4KB	Exclusive(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Exclusive(i)	N/A	N/A

- 15 In addition to the memory operand checks per the table above, the function checks the following:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
4. The TD must have been initialized but not finalized (TDR.NUM\_TDCX is the required number and TDCS.OP\_STATE is INITIALIZED).

If successful, the function does the following:

5. Finalize the TD measurement, i.e., SHA384 calculation of TDCS.MRTD that has been accumulated so far by TDH.MEM.PAGE.ADD and TDH.MR.EXTEND.
6. Calculate TDCS.SERVTD\_HASH:
  - 5 6.1. Get all service TD binding slots whose SERVTD\_BINDING\_STATE is not NOT\_BOUND.
    - 6.1.1. If no service TD binding slots apply, set TDCS.SERVTD\_HASH to 0.
  - 6.2. Sort in ascending order by SERVTD\_TYPE as the primary key, SERVTD\_INFO\_HASH as a secondary key (if multiple service TDs of the same type are bound).
  - 6.3. Concatenate SERVTD\_INFO\_HASH, SERVTD\_TYPE and SERVTD\_ATTR of each slot in a temporary buffer:
    - 10 6.3.1. SERVTD\_INFO\_HASH in bytes 5:0
    - 6.3.2. SERVTD\_TYPE in bytes 7:6
    - 6.3.3. SERVTD\_ATTR in bytes 15:8
    - 6.3.4. Concatenate all buffers.
    - 6.3.5. Calculate SHA384 and store in TDCS.SERVTD\_HASH.
- 15 7. Mark the TD as finalized.

### Completion Status Codes

**Table 5.183: TDH.MR.FINALIZE Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.MR.FINALIZE is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_FINALIZED	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

### 5.3.46. TDH.PHYMEM.CACHE.WB Leaf

TDH.PHYMEM.CACHE.WB is an interruptible and restartable function to write back the cache hierarchy on a package or a core.

**Table 5.184: TDH.PHYMEM.CACHE.WB Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	Command, as described below:		
	Value	Name	Description
	0	WB_START_CMD	Start a new TDH.PHYMEM.CACHE.WB cycle with no cache invalidation.
	1	WB_RESUME_CMD	Resume a previously interrupted TDH.PHYMEM.CACHE.WB cycle with no cache invalidation.
	Other		Reserved

**Table 5.185: TDH.PHYMEM.CACHE.WB Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
Other	Unmodified

#### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.PHYMEM.CACHE.WB writes back the cache hierarchy to memory and updates the KOT state to allow reuse of HKIDs.

**Interruptibility:** TDH.PHYMEM.CACHE.WB is interruptible. If a pending interrupt is detected during operation, TDH.PHYMEM.CACHE.WB returns with a TDX\_INTERRUPTED\_RESUMABLE status in RAX.

The host's VMM initially calls TDH.PHYMEM.CACHE.WB with RCX indicating WB\_START\_CMD. If TDH.PHYMEM.CACHE.WB returns TDX\_INTERRUPTED\_RESUMABLE (or any other recoverable error) status in RAX, the host VMM should call TDH.PHYMEM.CACHE.WB with RCX indicating WB\_RESUME\_CMD in a loop until it completes its operation, as indicated by TDX\_SUCCESS status.

**Warning:** When TDH.PHYMEM.CACHE.WB is interrupted, the CPU still considers this as a cache write-back operation in progress. The host VMM should complete the cache write-back operation by resuming TDH.PHYMEM.CACHE.WB (i.e., with RCX indicating WB\_RESUME\_CMD) until completed successfully. **Failure to do so will result in memory performance impact that would only be resolved by a restart.** The host VMM should also refrain from executing WBINVD or WBNOINVD while the TDH.PHYMEM.CACHE.WB cycle is in progress.

Other TDH.PHYMEM.CACHE.WB characteristics:

- TDH.PHYMEM.CACHE.WB does not invalidate cache lines.
- The function operates on cache lines associated with any HKID.
- The function is designed to ensure write back of at least those cache lines where the state of that HKID (in the KOT) was HKID\_FLUSHED at the time of the first invocation (RCX == WB\_START\_CMD).
- Depending on the implementation, the instruction may write back additional cache lines.
- The scope at which TDH.PHYMEM.CACHE.WB operates (e.g., package or core) is determined at Intel TDX module initialization time.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.186: TDH.PHYMEM.CACHE.WB (Implicit) Operands Information**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Implicit	N/A	N/A	KOT	KOT	N/A	Hidden	N/A	Shared	N/A	N/A
Implicit	N/A	N/A	WBT entry for current scope	WBT_ENTRY	N/A	Hidden	N/A	Exclusive	N/A	N/A

In addition to the memory operand checks per the table above, the function checks the following conditions:

1. The command value is one of the supported ones.
2. If the command is to start a new TDH.PHYMEM.CACHE.WB cycle (RCX == 0), then:
  - 2.1. Clear the internally saved interruption state.
  - 2.2. Scan the KOT: mark those HKIDs whose state is HKID\_FLUSHED in an internal table; only those HKIDs will be later marked as written back and invalidated upon successful completion of TDH.PHYMEM.CACHE.WB.
  - 2.3. If none of the KOT entries for the requested set of HKIDs (either single or all) is in HKID\_FLUSHED state, then abort with an informational code (it achieved its goal: write back and invalidate at least the HKIDs that are in the HKID\_FLUSHED state).
3. Run cache write back operation on the cache hierarchy of the current package or core. This operation is long and may be interrupted by external events.
  - 3.1. If a previous TDH.PHYMEM.CACHE.WB has been interrupted, the operation resumes from the interruption point which has been recorded.
  - 3.2. In case of interruption, the current point in the write back and invalidation flow and the current HKID are recorded.
4. If the operation has not been interrupted, update the KOT as follows:
  - 4.1. For each KOT entry, if the entry was marked as HKID\_FLUSHED at the start of the TDH.PHYMEM.CACHE.WB cycle as discussed above, use the KOT entry's bitmap to indicate that TDH.PHYMEM.CACHE.WB has been executed on this package or core.

## Error and Informational Codes

**Table 5.187: TDH.PHYMEM.CACHE.WB Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_INTERRUPTED_RESUMABLE	TDH.PHYMEM.CACHE.WB was interrupted; it is recommended to resume it with RCX indicating WB_RESUME_CMD
TDX_NO_HKID_READY_TO_WBCACHE	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	

Completion Status Code	Description
TDX_SUCCESS	TDH.PHYMEM.CACHE.WB is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	

DRAFT

**5.3.47. TDH.PHYMEM.PAGE.RDMD Leaf**

Read the metadata of a page (or the metadata of the containing large page) in TDMR.

**Table 5.188: TDH.PHYMEM.PAGE.RDMD Operands**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	A physical address of a 4KB page in TDMR (HKID bits must be 0)		

5

**Table 5.189: TDH.PHYMEM.PAGE.RDMD Output Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction return code – see 5.3.1		
RCX	Page Type (PT):		
	Value	Name	Description
	0	PT_NDA	The physical page is <b>Not Directly Assigned</b> to the Intel TDX module.
	1	PT_RSVD	The physical page is reserved for non-TDX usage.
	3	PT_REG	The physical page holds TD private memory.
	4	PT_TDR	The physical page holds the TD Root (TDR) control structure.
	8:5		The physical page holds a TD control structure.
	Other		Reserved
	In case of an error, RCX returns 0.		
RDX	OWNER: the HPA of the TD's TDR control structure page (if applicable) In case of an error, as indicated by RAX, RDX returns 0.		
R8	Bits	Name	Description
	2:0	Size	Size of the containing 4KB, 2MB or 1GB page – see 3.5.1
	63:3	Reserved	Set to 0
	In case of an error, as indicated by RAX, R8 returns 0.		
R9	BEPOCH In case of an error, as indicated by RAX, R9 returns 0.		
R10	Reserved: set to 0		
R11	Reserved: set to 0		



Operand	Description
Other	Unmodified

### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- TDH.PHYMEM.PAGE.RDMD finds the containing page (4KB, 2MB or 2GB) of the given page in TDMR and reads its metadata from its PAMT entry.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.190: TDH.PHYMEM.PAGE.RDMD Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	Target page	Blob	None	Opaque/ Private	4KB	Shared	Shared	Shared

If the memory operand checks per the table above pass, the function does the following:

- Do a PAMT walk, and find the containing page and its size.

If passed:

- Read the PAMT entry.

### Completion Status Codes

**Table 5.191: TDH.PHYMEM.PAGE.RDMD Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_SUCCESS	TDH.PHYMEM.PAGE.RDMD is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	

**5.3.48. TDH.PHYMEM.PAGE.RECLAIM Leaf**

Reclaim a physical 4KB, 2MB or 1GB TD-owned page (i.e., TD private page, Secure EPT page or a control structure page) from a TD, given its HPA.

**Table 5.192: TDH.PHYMEM.PAGE.RECLAIM Input Operands Definition**

Operand	Name	Description		
RAX	Leaf and Version	SEAMCALL instruction leaf number and version, see 5.3.1		
		<b>Bits</b>	<b>Field</b>	<b>Description</b>
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
		63:24	Reserved	Must be 0
RCX	PAGE	The physical address of a 4KB, 2MB or 1GB page to be reclaimed (HKID bits must be 0)		

5

**Table 5.193: TDH.PHYMEM.PAGE.RECLAIM Output Operands Definition**

Operand	Name	Description		
RAX	STATUS	SEAMCALL instruction return code – see 5.3.1		
RCX	PT	Page Type (PT):		
		<b>Value</b>	<b>Name</b>	<b>Description</b>
		0	PT_NDA	The physical page is <b>Not Directly Assigned</b> to the Intel TDX module.
		1	PT_RSVD	The physical page is reserved for non-TDX usage.
		3	PT_REG	The physical page holds TD private memory.
		4	PT_TDR	The physical page holds the TD Root (TDR) control structure.
		8:5		The physical page holds a TD control structure.
		Other		Reserved
		In multiple error cases, as indicated by RAX, RDX returns 0. In other error cases, RDX still returns the PT information. See the completion status codes table below for details.		
RDX	OWNER	The HPA of the TD's TDR control structure page (if applicable)  In multiple error cases, as indicated by RAX, RDX returns 0. In other error cases, RDX still returns the OWNER information. See the completion status codes table below for details.		
R8	SIZE	<b>Bits</b>	<b>Name</b>	<b>Description</b>
		2:0	Size	Size of the containing 4KB, 2MB or 1GB page – see 3.5.1
		63:3	Reserved	Set to 0
		In multiple error cases, as indicated by RAX, RDX returns 0. In other error cases, RDX still returns the size information. See the completion status codes table below for details.		

Operand	Name	Description
R9		Reserved: set to 0
R10		Reserved: set to 0
R11		Reserved: set to 0
Other		Unmodified

### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 5 TDH.PHYMEM.PAGE.RECLAIM reclaims a TD-owned physical page from the TD.

TDH.PHYMEM.PAGE.RECLAIM can reclaim pages only if the owner TD is in the TD\_TEARDOWN state.

**Cache Flush and Init** After the physical page has been reclaimed, the host VMM should flush its cache lines (required only for TDR pages) and initialize its content before it is reused, as described in the [Base Spec].

- 10 To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.194: TDH.PHYMEM.PAGE.RECLAIM Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	Target page	Blob	RW	Opaque/ Private	4KB, 2MB or 1GB	Exclusive	Shared	Shared
Implicit	N/A	N/A	TDR page <sup>6</sup>	TDR	RW	Opaque	4KB	Shared	N/A	N/A

TDH.PHYMEM.PAGE.RECLAIM checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

- 15 The function works as follows:

1. Check that the target page metadata in PAMT are correct (PT must not be PT\_NDA nor PT\_RSVD).
2. If the target page is not a TDR (PT is not PT\_TDR):
  - 2.1. Get the TDR page (pointed by the target page's PAMT.OWNER).
  - 2.2. Check that the TD is in teardown state (TDR.LIFECYCLE\_STATE is TD\_TEARDOWN).
  - 2.3. Atomically decrement TDR.CHLDCNT.
3. Else (target page is a TDR):
  - 3.1. Check that the TD is in teardown state (TDR.LIFECYCLE\_STATE is TD\_TEARDOWN).
  - 3.2. Check that TDR.CHLDCNT is 0.
4. Update the PAMT entry of the reclaimed page to PT\_NDA.
5. Return the page metadata (as they were before PAMT update above).

<sup>6</sup> Except when TDR is the target page

## Completion Status Codes

Table 5.195: TDH.PHYMEM.PAGE.RECLAIM Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_LIFECYCLE_STATE_INCORRECT	RCX, RDX and R9 return the actual PT, OWNER and SIZE information.
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.  If the page is not a TDR page but the owner TDR is busy, then RCX, RDX and R9 return the actual PT, OWNER and SIZE information.
TDX_OPERAND_INVALID	If the page physical address is not aligned on its size, then RCX, RDX and R9 return the actual PT, OWNER and SIZE information.
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.PHYMEM.PAGE.RECLAIM is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_ASSOCIATED_PAGES_EXIST	RCX, RDX and R9 return the actual PT, OWNER and SIZE information.

### 5.3.49. TDH.PHYMEM.PAGE.WBINVD Leaf

Write back and invalidate all cache lines associated with the specified memory page and HKID.

**Table 5.196: TDH.PHYMEM.PAGE.WBINVD Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	Physical address (including HKID bits) of a 4KB page in TDMR		

**Table 5.197: TDH.PHYMEM.PAGE.WBINVD Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
Other	Unmodified

#### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.PHYMEM.PAGE.WBINVD performs cache write back and invalidation on all the cache lines associated with the specified page and HKID. The page must not be in use by the Intel TDX module (i.e., not assigned to a TD as a private page or a Secure EPT page), nor used as a control structure page.
- It is the responsibility of the host VMM to track which HKID is associated with the target page; the function does not check it.
- 15 To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.198: TDH.PHYMEM.PAGE.WBINVD Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	Target page	Blob	R	Private/ Opaque	4KB	Shared	Shared	Shared

In addition to the memory operand checks per the table above, the function checks the following conditions:

- 20 1. The target page must be marked in PAMT as not controlled by the Intel TDX module (PT must be PT\_NDA).
- If successful, the function performs the following:
2. Write back and invalidate all the cache lines for the given target HPA and HKID.

## Completion Status Codes

Table 5.199: TDH.PHYMEM.PAGE.WBINVD Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.PHYMEM.PAGE.WBINVD is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	

DRAFT

**5.3.50. NEW: TDH.SERVTD.BIND Leaf**

Bind a service TD to a target TD.

**Table 5.200: TDH.SERVTD.BIND Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	The physical address of the target TD's TDR page (HKID bits must be 0)		
RDX	The physical address of the service TD's TDR page (HKID bits must be 0)		
R8	Index (slot number) in the target TD's service TD binding table		
R9	SERVTD_TYPE: Service TD type		
R10	SERVTD_ATTR: Service TD attributes		

**Table 5.201: TDH.SERVTD.BIND Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
RCX	Binding handle In case of an error, as indicated by RAX, RCX returns 0.
R10	TD_UUID bits 63:0 In case of an error, as indicated by RAX, R10 returns 0.
R11	TD_UUID bits 127:64 In case of an error, as indicated by RAX, R11 returns 0.
R12	TD_UUID bits 191:128 In case of an error, as indicated by RAX, R12 returns 0.
R13	TD_UUID bits 255:192 In case of an error, as indicated by RAX, R13 returns 0.
AVX, AVX2 and AVX512 state	May be reset to the architectural INIT state
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.SERVTD.BIND binds a service TD to a target TD.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.202: TDH.SERVTD.BIND Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	Target TD's TDR page	TDR	R	Opaque	4KB	Shared	Shared	Shared
Explicit	RDX	HPA	Service TD's TDR page	TDR	R	Opaque	4KB	Shared	Shared	Shared
Implicit	N/A	N/A	Target TD's TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	Target TD's TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	N/A	Binding table		RW	Opaque	N/A	Exclusive	None	None
Implicit	N/A	N/A	Service TD's TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	Service TD's TDCS.OP_STATE	OP_STATE	R	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	N/A	Service TD's TDCS.RTMR	SHA384 HASH	N/A	Opaque	N/A	Shared	N/A	N/A

In addition to the memory operand checks per the table above, the function checks the following conditions:

1. Target TD checks:
    - 1.1. The target TD's TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
    - 1.2. The target TD is not in a FATAL state (TDR.FATAL is FALSE).
    - 1.3. The target TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
    - 1.4. The target TD's TDCS pages must have been allocated (TDR.NUM\_TDCX is the required number).
    - 1.5. The target TD has not been paused for export and is not in the in-order import phase.
  2. Service TD checks:
    - 2.1. The service TD's TDR HPA must be different than the target TD's TDR HPA
    - 2.2. The service TD's TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
    - 2.3. The service TD is not in a FATAL state (TDR.FATAL is FALSE).
    - 2.4. The service TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
    - 2.5. The service TD's TDCS pages must have been allocated (TDR.NUM\_TDCX is the required number).
    - 2.6. Either the service TD's measurements have been finalized (by TDH.MR.FINALIZE) or it is being imported and import is in the out-of-order phase.
  3. Binding slot number does not exceed the number of available slots.
  4. SERVTD\_TYPE is supported.
  5. If only one service TD binding instance is supported by SERVTD\_TYPE, no other binding slot whose BINDIND\_STATE is not NOT\_BOUND may have the same SERVTD\_TYPE.
  6. SERVTD\_ATTR is supported.
- If the above checks passed:
7. If the binding slot's SERVTD\_BINDING\_STATE is NOT\_BOUND (i.e., this is an **initial binding**):
    - 7.1. Check that the target TD's measurements have not been finalized (by TDH.MR.FINALIZE).
    - 7.2. Copy the provided SERVTD\_TYPE and SERVTD\_ATTR to the binding slot.
    - 7.3. Calculate the service TD's SERVTD\_INFO\_HASH and write to the binding slot's SERVTD\_INFO\_HASH.



- 7.4. Copy the SERVTD's TD\_UUID to the binding slot's SERVTD\_UUID.
8. Else, if the binding slot's SERVTD\_BINDING\_STATE is PRE\_BOUND (i.e., this is a **late initial binding**):
- 8.1. Check that the requested SERVTD\_TYPE is equal to the binding slot's SERVTD\_TYPE.
- 8.2. Check that the requested SERVTD\_ATTR is equal to the binding slot's SERVTD\_ATTR.
- 5 8.3. Calculate the service TD's SERVTD\_INFO\_HASH and check that it is equal to the binding slot's SERVTD\_INFO\_HASH.
- 8.4. Copy the SERVTD's TD\_UUID to the binding slot's SERVTD\_UUID.
9. Else, if the binding slot's SERVTD\_BINDING\_STATE is BOUND (i.e., this is a **rebinding**):
- 9.1. Check that the requested SERVTD\_TYPE is equal to the binding slot's SERVTD\_TYPE.
- 10 9.2. Check that the requested SERVTD\_ATTR is equal to the binding slot's SERVTD\_ATTR.
- 9.3. Calculate the service TD's SERVTD\_INFO\_HASH.
- 9.4. If SERVTD\_ATTR.INSTANCE\_BINDING is set:
- 9.4.1. Check that the SERVTD's TD\_UUID is equal to the binding slot's SERVTD\_UUID.
- 9.4.2. Copy the calculated service TD's SERVTD\_INFO\_HASH to the binding slot's SERVTD\_INFO\_HASH.
- 15 9.5. Else:
- 9.5.1. Check that the service TD's SERVTD\_INFO\_HASH is equal to the binding slot's SERVTD\_INFO\_HASH.
- 9.5.2. Copy the SERVTD's TD\_UUID to the binding slot's SERVTD\_UUID.

If passed:

10. Set the binding slot's SERVTD\_BINDING\_STATE to BOUND.
- 20 11. Calculate and return the binding handle.

### Completion Status Codes

**Table 5.203: TDH.SERVTD.BIND Completion Status Codes (Returned in RAX) Definition** **[TO BE EDITED]**

Completion Status Code	Description
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.SERVTD.BIND is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

**5.3.51. NEW: TDH.SERVTD.PREBIND Leaf**

Pre-bind a service TD to a target TD.

**Table 5.204: TDH.SERVTD.PREBIND Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	The physical address of the target TD's TDR page (HKID bits must be 0)		
RDX	The physical address (including HKID bits) of SERVTD_INFO_HASH, the expected SHA384 hash of the service TD's TDINFO_STRUCT		
R8	Index (slot number) in the target TD's service TD binding table		
R9	SERVTD_TYPE: Expected service TD type		
R10	SERVTD_ATTR: Expected service TD attributes		

**Table 5.205: TDH.SERVTD.PREBIND Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.SERVTD.PREBIND pre-binds a service TD to a target TD, by setting its expected binding parameters.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.206: TDH.SERVTD.PREBIND Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	Target TD's TDR page	TDR	R	Opaque	4KB	Shared	Shared	Shared
Explicit	RDX	HPA	SERVTD_INFO_HASH	SHA384_HASH	R	Shared	64B	N/A	N/A	N/A
Implicit	N/A	N/A	Target TD's TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Implicit	N/A	N/A	Target TD's TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	N/A	Binding table		RW	Opaque	N/A	Exclusive	None	None

In addition to the memory operand checks per the table above, the function checks the following conditions:

1. The target TD's TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The target TD is not in a FATAL state (TDR.FATAL is FALSE).
- 5 3. The target TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
4. The target TD's TDCS pages must have been allocated (TDR.NUM\_TDCX is the required number).
5. The target TD's measurements have not been finalized (by TDH.MR.FINALIZE).
6. Binding slot number does not exceed the number of available slots.
7. SERVTD\_TYPE is supported.
- 10 8. If only one service TD binding instance is supported by SERVTD\_TYPE, no other binding slot whose BINDIND\_STATE is not NOT\_BOUND may have the same SERVTD\_TYPE.
9. SERVTD\_ATTR is supported.
10. The binding slot's SERVTD\_BINDING\_STATE is either NOT\_BOUND or PRE\_BOUND.

If the above checks passed:

- 15 11. Copy the provided SERVTD\_TYPE, SERVTD\_ATTR and SERVTD\_INFO\_HASH to the binding slot.
12. Set the binding slot's SERVTD\_BINDING\_STATE to PRE\_BOUND.

#### Completion Status Codes

**Table 5.207: TDH.SERVTD.PREBIND Completion Status Codes (Returned in RAX) Definition** [TO BE EDITED]

Completion Status Code	Description
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.SERVTD.PREBIND is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

### 5.3.52. TDH.SYS.CONFIG Leaf

Globally configure the Intel TDX module.

**Table 5.208: TDH.SYS.CONFIG Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	The physical address of an array of pointers, each containing the physical address of a single TDMR_INFO entry (see 3.3.2).  The pointer array must be sorted such that TDMR base addresses (TDMR_INFO.TDMR_BASE) are sorted from the lowest to the highest base address, and TDMRs do not overlap with each other.		
RDX	The number of pointers in the above buffer, between 1 and 64		
R8	Bits	Name	Description
	15:0	HKID	Intel TDX global private HKID value
	63:16	Reserved	Reserved: must be 0

**Table 5.209: TDH.SYS.CONFIG Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
Other	Unmodified

#### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.SYS.CONFIG performs global (platform-scope) configuration of the Intel TDX module. This function is intended to be executed during OS/VMM boot, and thus it has relaxed latency requirements.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.210: TDH.SYS.CONFIG Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDMR Info Pointers	Array of HPA	R	Shared	512B	None	N/A	N/A
Explicit	N/A	HPA	TDMR Info	TDMR_INFO	R	Shared	512B	None	N/A	N/A

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Implicit	N/A	N/A	All Intel TDX module internal variables	N/A	RW	Hidden	N/A	Exclusive	N/A	N/A

In addition to the memory operand checks per the table above, the function checks the following conditions:

1. Global and LP-scope initialization has been done:
  - 1.1. PL.SYS\_STATE is SYSINIT\_DONE.
  - 1.2. TDH.SYS.LP.INIT has been executed on all LPs.
2. The number of TDMR\_INFO entries is at least 1 and does not exceed the supported number of TDMRs.
3. Check each physical address of to TDMR\_INFO; read the applicable TDMR\_INFO entry; check and update the internal TDMR\_TABLE with TDMR, reserved areas and PAMT setup. The order of checks is not required to be exactly the same as described below.
  - TDMRs must be sorted in an ascending base address order.
  - For each TDMR:
    - TDMR base address must be aligned on 1GB.
    - TDMR size must be greater than 0 and a whole multiple of 1GB.
    - Any address within the TDMR must comply with the platform's maximum PA, and its HKID bits must be 0.
    - For each PAMT region (1G, 2M and 4K) of each TDMR:
      - PAMT base address must comply with the alignment requirements.
      - Any address within the PAMT range must comply with the platform's maximum PA, and its HKID bits must be 0.
      - The size of each PAMT region must be large enough to contain the PAMT for its associated TDMR.
    - Reserved areas within TDMR must be sorted in an ascending offset order.
    - A null reserved area (indicated by a size of 0) may be followed only by other null reserved areas.
    - For each reserved area within TDMR:
      - Offset and size must comply with the alignment and granularity requirements.
      - Reserved areas must not overlap.
      - Reserved areas must be fully contained within their TDMR.
  - TDMRs must not overlap with other TDMRs.
  - PAMTs must not overlap with other PAMTs.
  - TDMRs' non-reserved parts and PAMTs must not overlap (PAMTs may reside within TDMR reserved areas).
  - TDMRs' non-reserved parts must be contained in convertible memory – i.e., in CMRs.
  - PAMTs must be contained in convertible memory – i.e., in CMRs.
4. Check and set the Intel TDX global private HKID. The provided HKID must be in the TDX HKID range.

If successful, the function does the following:

5. Complete the initialization of the Intel TDX module at platform scope.
6. Set PL.SYS\_STATE to SYSCONFIG\_DONE.

### Completion Status Codes

**Table 5.211: TDH.SYS.CONFIG Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_INVALID_PAMT	
TDX_INVALID_RESERVED_IN_TDMR	
TDX_INVALID_TDMR	
TDX_NON_ORDERED_RESERVED_IN_TDMR	
TDX_NON_ORDERED_TDMR	

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_PAMT_OUTSIDE_CMRS	
TDX_PAMT_OVERLAP	
TDX_SUCCESS	TDH.SYS.CONFIG is successful.
TDX_SYS_BUSY	The operation was invoked when another TDX module operation was in progress. The operation may be retried.
TDX_SYS_SHUTDOWN	
TDX_SYSINIT_NOT_DONE	
TDX_SYSINITLP_NOT_DONE	
TDX_TDMR_ALREADY_INITIALIZED	
TDX_TDMR_OUTSIDE_CMRS	

DRAFT

**5.3.53. UPDATED: TDH.SYS.INFO Leaf**

Provide information about the Intel TDX module and the convertible memory.

**Note:** TDH.SYS.INFO is provided for backward compatibility. TDH.SYS.RDALL is the recommended method to read Intel TDX module information.

**Table 5.212: TDH.SYS.INFO Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	The physical address (including HKID bits) of a buffer where the output TDSYSINFO_STRUCT will be written		
RDX	The number of bytes in the above buffer		
R8	The physical address (including HKID bits) of a buffer where an array of CMR_INFO will be written		
R9	The number of CMR_INFO entries in the above buffer		

**Table 5.213: TDH.SYS.INFO Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
RDX	The actual number of bytes written to the above buffer In case of an error, as indicated by RAX, RDX returns 0.
R9	The number of CMR_INFO entries actually written to the above buffer In case of an error, as indicated by RAX, R9 returns 0.
Other	Unmodified

**Leaf Function Description**

- 10 **Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.SYS.INFO provides information about the Intel TDX module and about the memory configuration.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.214: TDH.SYS.INFO Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDX system information structure	TDSYSINFO_STRUCT	RW	Shared	1024B	None	N/A	N/A
Explicit	R8	HPA	CMR table	CMR_INFO_ARRAY	RW	Shared	512B	None	N/A	N/A

In addition to the memory operand checks per the table above, the function checks the following conditions:

1. Global and LP-scope initialization has been done:
    - 1.1. TDH.SYS.INIT has been executed.
    - 1.2. TDH.SYS.LP.INIT has been executed on the current LP.
  2. The number of bytes provided for returning TDSYSINFO\_STRUCT (in RDX) must be at least the size of that structure.
  3. The number of entries provided for returning CMR\_INFO\_ARRAY (in R9) must be at least the number of CMRs supported by TDX.
- 10 If successful, the function does the following:
4. Write the TDSYSINFO\_STRUCT, and set RDX to the actual number of bytes written.
  5. Write the CMR\_INFO\_ARRAY based on the CMR information in SEAMCFG, and set R9 to the number of CMRs.

#### Completion Status Codes

**Table 5.215: TDH.SYS.INFO Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_SUCCESS	TDH.SYS.INFO is successful.
TDX_SYS_SHUTDOWN	
TDX_SYSINITLP_NOT_DONE	



### 5.3.54. TDH.SYS.INIT Leaf

Globally initialize the Intel TDX module.

**Table 5.216: TDH.SYS.INIT Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	Intel TDX module attributes		
	Bits	Name	Description
	63:0	RESERVED	Reserved: must be 0

**Table 5.217: TDH.SYS.INIT Output Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction return code – see 5.3.1		
RCX	Extended error information part 1 If RAX returns TDX_INCORRECT_CPUID_VALUE, RCX returns the applicable CPUID information as shown below. In all other cases, RCX returns 0.		
	Bits	Name	Description
	31:0	LEAF	CPUID leaf number
	63:32	SUBLEAF	CPUID sub-leaf number: if sub-leaf is not applicable, value is -1 (0xFFFFFFFF).
RDX	Extended error information part 2 If RAX returns TDX_INCORRECT_CPUID_VALUE, RDX returns the value masks as shown below. A bit value of 1 indicates a bit position that was checked against the required value. In all other cases, RDX returns 0.		
	Bits	Name	Description
	31:0	MASK_EAX	Mask of the value returned by CPUID in EAX
	63:32	MASK_EBX	Mask of the value returned by CPUID in EBX
R8	Extended error information part 3 If RAX returns TDX_INCORRECT_CPUID_VALUE, R8 returns the value masks as shown below. A bit value of 1 indicates a bit position that was checked against the required value. In all other cases, R8 returns 0.		
	Bits	Name	Description
	31:0	MASK_ECX	Mask of the value returned by CPUID in ECX

Operand	Description		
	63:32	MASK_EDX	Mask of the value returned by CPUID in EDX
R9	Extended error information part 4 If RAX returns TDX_INCORRECT_CPUID_VALUE, R9 returns the expected values as shown below. In all other cases, R9 returns 0.		
	Bits	Name	Description
	31:0	VALUE_EAX	Value expected to be returned by CPUID in EAX
	63:32	VALUE_EBX	Value expected to be returned by CPUID in EBX
R10	Extended error information part 5 If RAX returns TDX_INCORRECT_CPUID_VALUE, R10 returns the expected values as shown below. In all other cases, R10 returns 0.		
	Bits	Name	Description
	31:0	VALUE_ECX	Value expected to be returned by CPUID in ECX
	63:32	VALUE_EDX	Value expected to be returned by CPUID in EDX
Other	Unmodified		

### Special Environment Requirements

If the IA32\_TSX\_CTRL MSR is supported by the CPU, as enumerated by IA32\_ARCH\_CAPABILITIES.TSX\_CTRL (bit 7), then the values of its following bits must be 0:

- 5 • RTM\_DISABLE (bit 0)
- TSX\_CPUID\_CLEAR (bit 1)

The IA32\_MISC\_PACKAGE\_CTRL MSR must be supported by the CPU, as enumerated by IA32\_ARCH\_CAPABILITIES.MISC\_PACKAGE\_CTRL (bit 11). IA32\_MISC\_PACKAGE\_CTL.ENERGY\_FILTERING\_ENABLE (bit 0) must be set to 1.

### Leaf Function Latency

- 10 TDH.SYS.INIT execution time may be longer than most TDX module interface functions execution time. No interrupts (including NMI and SMI) are processed by the logical processor during that time.

### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 15 TDH.SYS.INIT performs global (platform-scope) initialization of the Intel TDX module. This function is intended to be executed during OS/VMM boot and thus it has relaxed latency requirements.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.218: TDH.SYS.INIT Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Implicit	N/A	N/A	All Intel TDX module internal variables	N/A	RW	Hidden	N/A	Exclusive	N/A	N/A

In addition to the memory operand checks per the table above, the function checks the following conditions:

1. Check that PL.SYS\_STATE is SYSINIT\_PENDING.
2. Do any global Intel TDX module initializations required for running this flow.
3. Check the memory operands per the table above.
- 5 4. Check the following conditions (no specific order is implied):
  - Enumerate CPU and platform information, and check Intel TDX module compatibility. If the Intel TDX module is compatible with multiple variants of CPU and platform features, sample the current LP's features enumeration – to be later checked to be the same on all LPs by TDH.SYS.LP.INIT. Examples of compatibility checks are:
    - 10 ○ The CPU must support any ISA that the Intel TDX module relies upon, such as SHA-NI.
    - The CPU must support the WBINVD scope for which the Intel TDX module was built.
  - Sample and check the platform configuration on the current LP – to be later checked to be the same on all LPs by TDH.SYS.LP.INIT. For example:
    - Sample SMRR and SMRR2, check they are locked and do not overlap any CMR, and store their values to be checked later on each LP.
- 15 If successful, the function does the following:
  5. Complete the initialization of the Intel TDX module at platform scope.
  6. Set PL.SYS\_STATE to SYSINIT\_DONE.

### Completion Status Codes

**Table 5.219: TDH.SYS.INIT Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_BOOT_NT4_SET	
TDX_CPUID_LEAF_1F_FORMAT_UNRECOGNIZED	
TDX_CPUID_LEAF_1F_NOT_SUPPORTED	
TDX_CPUID_LEAF_0D_INCONSISTENT	
TDX_INCORRECT_CPUID_VALUE	Additional information is provided in RCX – R10
TDX_INCORRECT_MSR_VALUE	
TDX_INVALID_WBINVD_SCOPE	
TDX_SMRR_LOCK_NOT_SUPPORTED	
TDX_SMRR_NOT_LOCKED	
TDX_SMRR_NOT_SUPPORTED	
TDX_SMRR_OVERLAPS_CMR	
TDX_SUCCESS	TDH.SYS.INIT is successful.
TDX_SYS_BUSY	The operation was invoked when another TDX module operation was in progress. The operation may be retried.
TDX_SYS_SHUTDOWN	
TDX_SYSINIT_NOT_PENDING	

### 5.3.55. TDH.SYS.KEY.CONFIG Leaf

Configure the Intel TDX global private key on the current package.

**Table 5.220: TDH.SYS.KEY.CONFIG Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0

**Table 5.221: TDH.SYS.KEY.CONFIG Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
Other	Unmodified

#### Leaf Function Latency

TDH.SYS.KEY.CONFIG execution time may be longer than most TDX module interface functions execution time. No interrupts (including NMI and SMI) are processed by the logical processor during that time.

#### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.SYS.KEY.CONFIG performs package-scope Intel TDX global private key configuration.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.222: TDH.SYS.KEY.CONFIG Operands Information**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Implicit	N/A	N/A	All Intel TDX module internal variables	N/A	RW	Hidden	N/A	Exclusive	N/A	N/A

In addition to the memory operand checks per the table above, the function checks the following conditions:

1. Check that TDH.SYS.CONFIG has completed successfully (PL.SYS\_STATE is SYSCONFIG\_DONE).

If successful, the function does the following:

2. Do the following as an atomic operation (e.g., LOCK BTS) on PL.PKG\_CONFIG\_BITMAP:
  - 2.1. Check the package has not yet been configured.
  - 2.2. Mark it as configured.
3. Execute PCONFIG to configure the Intel TDX global private HKID on the package with a CPU-generated random key.

PCONFIG may fail due to an entropy error or a device busy error. In these cases, the VMM should retry TDH.SYS.KEY.CONFIG.

If successful:

4. If this was the last package on which TDH.SYS.KEY.CONFIG has executed, set PL.STATE to SYS\_READY.

## 5 Completion Status Codes

**Table 5.223: TDH.SYS.KEY.CONFIG Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_KEY_CONFIGURED	
TDX_KEY_GENERATION_FAILED	Failed to generate a random key. This is typically caused by an entropy error of the CPU's random number generator, and may be impacted by RDSEED, RDRAND or PCONFIG executing on other LPs. The operation should be retried.
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.  Specifically, key configuration may fail due to a concurrently running PCONFIG instruction.
TDX_SUCCESS	TDH.SYS.KEY.CONFIG is successful.
TDX_SYS_BUSY	The operation was invoked when another TDX module operation was in progress. The operation may be retried.
TDX_SYS_SHUTDOWN	
TDX_SYSINIT_NOT_DONE	

### 5.3.56. TDH.SYS.LP.INIT Leaf

Initialize the Intel TDX module at the current logical processor scope.

**Table 5.224: TDH.SYS.LP.INIT Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0

5

**Table 5.225: TDH.SYS.LP.INIT Output Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction return code – see 5.3.1		
RCX	Extended error information part 1 If RAX returns TDX_INCONSISTENT_CPUID_FIELD, RCX returns the applicable CPUID information as shown below. In all other cases, RCX returns 0.		
	Bits	Name	Description
	31:0	LEAF	CPUID leaf number
	63:32	SUBLEAF	CPUID sub-leaf number: if sub-leaf is not applicable, value is -1 (0xFFFFFFFF).
RDX	Extended error information part 2 If RAX returns TDX_INCONSISTENT_CPUID_FIELD, RDX returns the value masks as shown below. A bit value of 1 indicates a bit position that was checked against the same CPUID leaf value checked during TDH.SYS.INIT. In all other cases, RDX returns 0.		
	Bits	Name	Description
	31:0	MASK_EAX	Mask of the value returned by CPUID in EAX
	63:32	MASK_EBX	Mask of the value returned by CPUID in EBX
R8	Extended error information part 3 If RAX returns TDX_INCONSISTENT_CPUID_FIELD, R8 returns the value masks as shown below. A bit value of 1 indicates a bit position that was checked against the same CPUID leaf value checked during TDH.SYS.INIT. In all other cases, R8 returns 0.		
	Bits	Name	Description
	31:0	MASK_ECX	Mask of the value returned by CPUID in ECX
	63:32	MASK_EDX	Mask of the value returned by CPUID in EDX

Operand	Description
Other	Unmodified

### Special Environment Requirements

If the IA32\_TSX\_CTRL MSR is supported by the CPU, as enumerated by IA32\_ARCH\_CAPABILITIES.TSX\_CTRL (bit 7), then the values of its following bits must be 0:

- 5 • RTM\_DISABLE (bit 0)
- TSX\_CPUID\_CLEAR (bit 1)

The IA32\_MISC\_PACKAGE\_CTRL MSR must be supported by the CPU, as enumerated by IA32\_ARCH\_CAPABILITIES.MISC\_PACKAGE\_CTRL (bit 11). IA32\_MISC\_PACKAGE\_CTL.ENERGY\_FILTERING\_ENABLE (bit 0) must be set to 1.

### Leaf Function Latency

- 10 TDH.SYS.LP.INIT execution time may be longer than most TDX module interface functions execution time. No interrupts (including NMI and SMI) are processed by the logical processor during that time.

### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 15 TDH.SYS.LP.INIT performs LP-scope initialization of the Intel TDX module. This function is intended to be executed during OS/VMM boot, and thus it has relaxed latency requirements.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.226: TDH.SYS.LP.INIT Operands Information**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Implicit	N/A	N/A	All Intel TDX module internal variables	N/A	RW	Hidden	N/A	Shared	N/A	N/A

20

In addition to the memory operand checks per the table above, the function checks the following conditions:

1. TDH.SYS.INIT has completed successfully (PL.SYS\_STATE is SYSINIT\_DONE).
2. This is the first invocation of TDH.SYS.LP.INIT on the current LP.

If successful, the function does the following:

- 25 3. Do a global EPT flush (INVEPT type 2).
- 4. Initialize the Intel TDX module's LP-scope variables.
- 5. Check the compatibility and uniformity of features and configuration. Once per LP, core or package, depending on the scope of the checked feature or configuration:
  - 5.1. Check features compatibility with the Intel TDX module. For example, the WBINVD scope must be the same as the scope the Intel TDX module was built to handle. In cases where the Intel TDX module supports several options, check that the features on the current LP are the same as sampled during TDH.SYS.INIT.
  - 5.2. Check configuration uniformity. For example, the SMRR and SMRR2 must be locked and configured in the same way as sampled during TDH.SYS.INIT.
- 30 6. Mark the current LP as initialized.

## Completion Status Codes

Table 5.227: TDH.SYS.LP.INIT Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_INCONSISTENT_CPUID_FIELD	Additional information is provided in RCX – R8
TDX_INCONSISTENT_MSR	
TDX_INCORRECT_MSR_VALUE	
TDX_INVALID_PKG_ID	
TDX_RND_NO_ENTROPY	Random number generation (e.g., RDRAND or RDSEED) failed because the hardware random number generator did not have enough entropy. The host VMM should retry the operation.
TDX_SUCCESS	TDH.SYS.LP.INIT is successful.
TDX_SYS_BUSY	The operation was invoked when another TDX module operation was in progress. The operation may be retried.
TDX_SYS_SHUTDOWN	
TDX_SYSINIT_NOT_DONE	
TDX_SYSINITLP_DONE	

DRAFT



**5.3.57. UPDATED: TDH.SYS.LP.SHUTDOWN Leaf (Deprecated)**

This function is deprecated; it is provided for backward compatibility.

**Table 5.228: TDH.SYS.LP.SHUTDOWN Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0

**Table 5.229: TDH.SYS.LP.SHUTDOWN Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

**TDH.SYS.LP.SHUTDOWN does nothing.**

**Completion Status Codes****Table 5.230: TDH.SYS.LP.SHUTDOWN Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_SUCCESS	TDH.SYS.LP.SHUTDOWN is successful.
TDX_SYS_BUSY	The operation was invoked when another TDX module operation was in progress. The operation may be retried.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	

**5.3.58. NEW: TDH.SYS.RD Leaf**

Read a TDX Module global-scope metadata field.

**Table 5.231: TDH.SYS.RD Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RDX	Field identifier – see 3.10 The LAST_ELEMENT_IN_FIELD and LAST_FIELD_IN_SEQUENCE components of the field identifier must be 0. WRITE_MASK_VALID, INC_SIZE, CONTEXT_CODE and ELEMENT_SIZE_CODE components of the field identifier are ignored. A value of -1 is a special case: it is not a valid field identifier; in this case the first readable field identifier is returned in RDX.		

**Table 5.232: TDH.SYS.RD Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
RDX	If the input field identifier was -1, RDX returns the first readable field identifier. Else, in there case of an error, RDX returns -1. On success, RDX returns the next readable field identifier. A value of -1 indicates no next field identifier is available.
R8	Contents of the field In case of no success, as indicated by RAX, R8 returns 0.
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.SYS.RD reads a TDX Module global-scope metadata field.

RDX returns the next host-side readable field identifier. This may be used by the host VMM to enumerate the TDX Module's capabilities and configuration. To read all the available fields, the host VMM can invoke TDH.SYS.RD in a loop, starting with field identifier -1 as an input, until RDX returns -1. Alternatively, the host VMM can use TDH.SYS.RDALL.

- 15 To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.233: TDH.SYS.RD Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
There are no relevant memory operands.										

The function checks the following conditions:

1. Global and LP-scope initialization has been done:
  - 1.1. TDH.SYS.INIT has been executed.
  - 1.2. TDH.SYS.LP.INIT has been executed on the current LP.

If successful, the function does the following:

2. Read the requested field using the algorithm described in 5.2.2.1.
3. Return the next readable field identifier, or a value of 0 if none exists.
4. Return the field value.

#### Completion Status Codes

**Table 5.234: TDH.SYS.RD Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_METADATA_FIRST_FIELD_ID_IN_CONTEXT	Indicates that the first field ID in context is returned
TDX_OPERAND_INVALID	
TDX_SUCCESS	TDH.SYS.INFO is successful.
TDX_SYS_SHUTDOWN	
TDX_SYSINITLP_NOT_DONE	

**5.3.59. NEW: TDH.SYS.RDALL Leaf**

Read all host-readable TDX Module global-scope metadata fields.

**Table 5.235: TDH.SYS.RDALL Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RDX	The physical address (including HKID bits) of a 4KB page where a metadata list will be returned In case of error, some field value entries might not contain valid data.		
R8	Initial field identifier – see 3.10  If R8's value is -1, then TDG.SYS.RDALL will start from the first global-scope metadata field identifier.  Else, LAST_ELEMENT_IN_FIELD, LAST_FIELD_IN_SEQUENCE, WRITE_MASK_VALID and CONTEXT_CODE are ignored.		

**Table 5.236: TDH.SYS.RDALL Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
R8	Next field identifier. A value of -1 means all applicable field identifiers have been returned in the metadata list.  In case of an error, as indicated by RAX, R8 returns -1.
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.SYS.RDALL reads all host-readable TDX Module global-scope metadata fields into a metadata list in the provided page.

If one or more applicable fields do not fit in the provided list buffer, the function can be invoked in a loop, each invocation providing an initial field identifier returned as the next field identifier of the previous invocation, as shown in the following example:

- 15 1. NEXT\_FIELD\_ID = -1  
 2. Repeat:  
   2.1. Set LIST\_BUFFER to the next 4K buffer  
   2.2. Invoke TDH.SYS.RDALL(RDX = LIST\_BUFFER, RDX = NEXT\_FIELD\_ID)  
   2.3. STATUS = RAX, NEXT\_FIELD\_ID = R8  
 20 Until ((STATUS is a non-recoverable error) or (NEXT\_FIELD\_ID is -1))

The function never returns an empty list if there's no error.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.237: TDH.SYS.RDALL Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RDX	HPA	Metadata list	MD_LIST	RW	Shared	4KB	None	None	None

5 In addition to the memory operand checks per the table above, the function checks the following conditions:

1. Global and LP-scope initialization has been done:
  - 1.1. TDH.SYS.INIT has been executed.
  - 1.2. TDH.SYS.LP.INIT has been executed on the current LP.

If successful, the function does the following:

- 10 2. Dump the list of next host-readable metadata fields into the provided page.

### Completion Status Codes

**Table 5.238: TDH.MNG.RDALL Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.MNG.RDALL is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

**5.3.60. NEW: TDH.SYS.SHUTDOWN Leaf**

Initiate Intel TDX module shutdown and generate handoff data for the next Intel TDX module.

**Table 5.239: TDH.SYS.SHUTDOWN Input Operands Definition**

Operand	Name	Description		
RAX	Leaf and Version	SEAMCALL instruction leaf number and version, see 5.3.1		
		Bits	Field	Description
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
		63:24	Reserved	Must be 0
RCX	REQ_HV	Requested handoff version		

**Table 5.240: TDH.SYS.SHUTDOWN Output Operands Definition**

Operand	Name	Description
RAX	Status	SEAMCALL instruction return code – see 5.3.1
Other		Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.SYS.SHUTDOWN initiates Intel TDX module shutdown and generates handoff data for the next Intel TDX module. Following this function, no further TDX module interface functions can be called.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.241: TDH.SYS.SHUTDOWN Operands Information**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Implicit	N/A	N/A	All Intel TDX module internal variables	N/A	RW	Hidden	N/A	Exclusive	N/A	N/A

In addition to the memory operand checks per the table above, the function checks the following conditions:

- The requested handoff version (REQ\_HV) is legal:
  - PL.MIN\_UPDATE\_HV ≤ HV ≤ PL.MODULE\_HV
  - If PL.NO\_DOWNGRADE == 1 then HV == PL.MODULE\_HV

If successful:

- Set TDX module's PL.STATE to SYS\_SHUTDOWN to fail further TDX module interface function calls.
- Check that all other LPs are not executing in SEAM mode.
- Prepare handoff data in handoff pages, according to REQ\_HV, from module's variables.
- Mark the handoff data as valid (ready for consumption).

**Completion Status Codes****Table 5.242: TDH.SYS.SHUTDOWN Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_OPERAND_INVALID	The requested handoff version is invalid.
TDX_SUCCESS	TDH.SYS.SHUTDOWN is successful.
TDX_SYS_BUSY	The operation was invoked when another TDX module operation was in progress. The operation may be retried.
TDX_SYS_NOT_READY	TDH.SYS.SHUTDOWN was called when TDX module's lifecycle state is not SYS_READY.

DRAFT

### 5.3.61. TDH.SYS.TDMR.INIT Leaf

Partially initialize a Trust Domain Memory Region (TDMR) and its associated PAMT.

**Table 5.243: TDH.SYS.TDMR.INIT Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	The physical base address of a TDMR (HKID bits must be 0)		

**Table 5.244: TDH.SYS.TDMR.INIT Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
RDX	On successful completion, RDX returns the TDMR next-to-initialize address. This is the physical address of the last byte that has been initialized so far, rounded down to 1GB. In all other cases, RDX returns 0.
Other	Unmodified

#### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.SYS.TDMR.INIT partially initializes the metadata (PAMT) associated with a Trust Domain Memory Region (TDMR), while adhering to latency considerations. It can run concurrently on multiple LPs as long as each concurrent flow initializes a different TDMR. After each 1GB range of a TDMR has been initialized, that 1GB range becomes available for use by any Intel TDX function that creates a private TD page or a control structure page – e.g., TDH.MEM.PAGE.ADD, TDH.VP.ADDCX, etc.
- 15 To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.245: TDH.SYS.TDMR.INIT Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDMR	Blob	None	None	1GB	Exclusive	N/A	N/A
Implicit	N/A	HPA	PAMT region associated with TDMR	Blob	RW	Hidden	N/A	Exclusive	N/A	N/A



In addition to the memory operand checks per the table above, the function checks the following conditions:

1. The provided TDMR start address belongs to one of the TDMRs set during TDH.SYS.INIT.
2. The TDMR has not been completely initialized yet.

5 If successful, the function does the following:

3. If the TDMR has been completely initialized, there is nothing to do.

Else, the function does the following:

4. Initialize the next implementation defined un-initialized number of PAMT entries. The maximum number of PAMT entries to be initialized is set to help avoid latency issues.
  - 4.1. PAMT\_4K entries associated with a physical address that is within a reserved range are marked with PT\_RSVD.
  - 4.2. Other PAMT\_4K entries are marked with PT\_NDA.
  - 4.3. PAMT\_2M and PAMT\_1G entries are marked with PT\_NDA.
5. If the PAMT for a 1GB block of TDMR has been fully initialized, mark that 1GB block as ready for use. This means that 4KB pages in this 1GB block may be converted to private pages – e.g., by SEAMCALL(TDH.MEM.PAGE.ADD). This can be done concurrently with initializing other TDMRs.
- 15 6. Return the next-to-initialize address rounded down to 1GB. This is done so the host VMM will not attempt to use a 1GB block that is not fully initialized.

### Completion Status Codes

**Table 5.246: TDH.SYS.TDMR.INIT Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_SUCCESS	TDH.SYS.TDMR.INIT is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TDMR_ALREADY_INITIALIZED	

**5.3.62. NEW: TDH.SYS.UPDATE Leaf**

Populate Intel TDX module internal variables from the handoff data prepared by the previous Intel TDX module.

**Table 5.247: TDH.SYS.UPDATE Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0

**Table 5.248: TDH.SYS.UPDATE Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.SYS.UPDATE reads the handoff data prepared by the previous Intel TDX module. The operation may fail in the following cases:
- No valid handoff data
  - The old module's handoff data's version is too old for the current TDX module
  - The old module's handoff data's version is newer than the current TDX module's handoff data version
- 15 On such failures the host VMM is expected to request a non-updating TDX module installation (LOAD scenario) from the Persistent SEAMLDR.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.249: TDH.SYS.UPDATE Operands Information**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Implicit	N/A	N/A	All Intel TDX module internal variables	N/A	RW	Hidden	N/A	Exclusive	N/A	N/A

In addition to the memory operand checks per the table above, the function checks the following conditions:

1. The TDX module's PL.STATE is SYSINIT\_DONE.
2. All LPs have been initialized.
3. The handoff data in memory is valid and its handoff version (HV) is legal.

If successful:

4. Populate HV-specific variables within SEAM range from the handoff data.
5. Mark the handoff data as invalid (consumed).
6. Set the TDX module's PL.STATE to SYS\_READY.

## 5 Completion Status Codes

**Table 5.250: TDH.SYS.UPDATE Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_SUCCESS	TDH.SYS.UPDATE is successful.
TDX_SYS_BUSY	The operation was invoked when another TDX module operation was in progress. The operation may be retried.
TDX_SYS_STATE_INCORRECT	TDH.SYS.SHUTDOWN was called when the TDX module's lifecycle state is not SYSINIT_DONE or some LPs have not yet been initialized by TDH.SYS.LP.INIT.
TDX_SYS_INVALID_HANDOFF	The handoff data in SEAM range is invalid.

DRAFT

**5.3.63. UPDATED: TDH.VP.ADDCX Leaf**

Add a **TDCX** page to memory as a child of a given TDVPR.

**Table 5.251: TDH.VP.ADDCX Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	The physical address of a page where the <b>TDCX</b> page will be added (HKID bits must be 0)		
RDX	The physical address of a TDVPR page (HKID bits must be 0)		

**Table 5.252: TDH.VP.ADDCX Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.VP.ADDCX adds a **TDCX** page as a child of a given TDVPR.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.253: TDH.VP.ADDCX Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	<b>TDCX</b> page	Blob	RW	Opaque	4KB	Exclusive	Shared	Shared
Explicit	RDX	HPA	TDVPR page	Blob	RW	Opaque	4KB	Exclusive	Shared	Shared
Implicit	N/A	HPA	TDR page	TDR	RW	Opaque	N/A	Shared	None	None
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	<b>TDCS.OP_STATE</b>	<b>OP_STATE</b>	<b>RW</b>	<b>Opaque</b>	<b>N/A</b>	<b>Shared</b>	<b>N/A</b>	<b>N/A</b>

- 15 In addition to the explicit memory operand checks per the table above, the function checks the following conditions:

1. The TDVPR page metadata in PAMT must be correct (PT must be PT\_TDVPR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).

4. The TD has been initialized (by TDH.MNG.INIT).
5. The TD build and measurement must not have been finalized (by TDH.MR.FINALIZE).
6. The TD VCPU has not been initialized (by TDH.VP.INIT) and is not being torn down (TDVPS.VCPU\_STATE is VCPU\_UNINITIALIZED).
- 5 7. The new **TDCX** page metadata in PAMT must be correct (PT must be PT\_NDA).
8. The maximum number of TDCX pages per TDVPS (as enumerated by **TDH.SYS.RD\*** or TDH.SYS.INFO) has not been exceeded.

If successful, the function does the following:

9. Zero out the **TDCX** page contents using direct writes (MOVDIR64B).
- 10 10. Increment the VCPU's **TDCX** counter, and set a pointer in the parent TDVPR page to the new **TDCX** page.
11. Increment TDR.CHLDCNT.
12. Initialize the **TDCX** page metadata in PAMT.

### Completion Status Codes

**Table 5.254: TDH.VP.ADDCX Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.VP.ADDCX is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_FINALIZED	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	
TDX_TDCX_NUM_INCORRECT	
TDX_VCPU_STATE_INCORRECT	

15

**5.3.64. UPDATED: TDH.VP.CREATE Leaf**

Create a guest TD VCPU and its root TDVPR page.

**Table 5.255: TDH.VP.CREATE Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	The physical address of a page where TDVPR will be added (HKID bits must be 0)		
RDX	The physical address of the owner TDR page (HKID bits must be 0)		

**Table 5.256: TDH.VP.CREATE Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.VP.CREATE begins the build of a new guest TD VCPU. It adds a TDVPR page as a child of a TDR page.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.257: TDH.VP.CREATE Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDVPR page	Blob	RW	Opaque	4KB	Exclusive	Shared	Shared
Explicit	RDX	HPA	TDR page	TDR	RW	Opaque	4KB	Shared	Shared	Shared
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	4KB	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Shared	N/A	N/A

- 15 In addition to the explicit memory operand checks per the table above, the function checks the following conditions:

1. The TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).

4. The TD must **either** have been initialized but not finalized (TDR.NUM\_TDCX is the required number and TDCS.OP\_STATE is INITIALIZED), **or a migration session is in progress as state migration has begun by TDH.IMPORT.STATE.TD (OP\_STATE is STATE\_IMPORT).**
  5. The TDVPR page metadata in PAMT must be correct (PT must be PT\_NDA).
- 5 If successful, the function does the following:
6. Zero out the TDVPR page contents using direct write (MOVDIR64B).
  7. Increment TDR.CHLDCNT.
  8. Initialize the TDVPS management fields, which all reside in the TDVPR page.
  9. Initialize the TDVPR page metadata in PAMT.

## 10 Completion Status Codes

**Table 5.258: TDH.VP.CREATE Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.VP.CREATE is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_FINALIZED	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

**5.3.65. UPDATED: TDH.VP.ENTER Leaf**

Enter TDX non-root operation.

From the host VMM's point of view, TDH.VP.ENTER is a complex operation that normally involves TD entry followed by a TD exit. Therefore, input and output operands are specified by multiple tables below.

**Inputs**

TDH.VP.ENTER output format depends on how the previous TDH.VP.ENTER was terminated. There are two cases:

- Initial entry or following a previous asynchronous TD exit
- Following a previous TDCALL(TDG.VP.VMCALL)

The following table details TDH.VP.ENTER input operands for **initial entry** or following a **previous asynchronous TD exit**.

**Table 5.259: TDH.VP.ENTER Input Operands Definition for Initial Entry or Following a Previous Asynchronous TD Exit**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	VCPU handle and flags		
	Bit(s)	Name	Description
	11:0	RESERVED	Must be 0
	51:12	TDVPR_HPA	Bits 51:12 of the physical address of the TD VCPU's TDVPR page (HKID bits must be 0)
	52	HOST_RECOVERABILITY_HINT	Applicable only following a <b>previous trap-like asynchronous TD exit</b> , where bit 60 (HOST_RECOVERABILITY_HINT) of the previous TDH.VP.ENTER completion status (returned in RAX) was set to 1. In all other cases this bit must be 0.  0: The host VMM hints that the guest-side function may possibly be retried (e.g., the host may have corrected some conditions).  1: The host VMM hints that the error is probably not recoverable.  This bit is reflected to the guest TD in bit 60 of RAX.
	53	RESUME_L1	For partitioned TDs, indicates that the L1 VMM should be resumed. Applicable after TD exits from an L2 VM.  0: TDH.VP.ENTER resumes the L2 VM it last exited from.  1: TDH.VP.ENTER resumes L1 VMM, even if the previous TD exit was from an L2 VM.
	63:54	RESERVED	Must be 0



The following table details TDH.VP.ENTER input operands for following a **previous synchronous TD exit (TDG.VP.VMCALL)**.

**Table 5.260: TDH.VP.ENTER Input Operands Definition Following a Previous TDCALL(TDG.VP.VMCALL)**

Operand	Description		
RAX	SEAMCALL instruction leaf and version numbers – see 5.3.1		
RCX	VCPU handle and flags		
	Bit(s)	Name	Description
	11:0	RESERVED	Must be 0
	51:12	TDVPR_HPA	Bits 51:12 of the physical address of the TD VCPU's TDVPR page (HKID bits must be 0)
	52	RESERVED	Must be 0
	53	RESUME_L1	For partitioned TDs, indicates that the L1 VMM should be resumed. Applicable after TD exits from an L2 VM. 0: TDH.VP.ENTER resumes the L2 VM it last exited from. 1: TDH.VP.ENTER resumes L1 VMM, even if the previous TD exit was from an L2 VM.
	63:54	RESERVED	Must be 0
RBX, RDX, RBP, RSI, RDI, R8 – R15	If the corresponding bit of RCX at the previous TD exit (i.e., previous TDH.VP.ENTER termination) was set to 1, the register value is passed as-is to the guest TD – see the description of TDG.VP.VMCALL in 5.4.21 for details. Else, the register value is not used as an input.		
XMM0 – XMM15	If the corresponding bit of RCX at the previous TD exit (i.e., previous TDH.VP.ENTER termination) was set to 1, the register value is passed as-is to the guest TD – see the description of TDG.VP.VMCALL in 5.4.21 for details. Else, the register value is not used as an input.		

## Outputs

TDH.VP.ENTER output format depends on how the function was terminated. There are multiple cases:

- Error (No TD Entry)
- Asynchronous TD exit following a TD entry (with a VMX architectural exit reason)
- Asynchronous TD exit following a TD entry (with a non-VMX TD exit status)
- Asynchronous TD exit following a TD entry (with cross-TD exit details)
- TDCALL(TDG.VP.VMCALL) following a TD entry

The following table details TDH.VP.ENTER output operands when an error occurs, and the interface function returns **without entering the TD**.

**Table 5.261: TDH.VP.ENTER Output Operands Definition on Error (No TD Entry)**

Operand	Description		
RAX	Status	SEAMCALL instruction return code	
		Bit(s)	Name
		63	ERROR
		47:32	CLASS and DETAILS_L1
		Other	See the function completion status definition in 5.3.1.
		Description	
		Set to 1	
		None of the values detailed in the table below	

Operand	Description
Other GPRs	Unmodified
Extended State	Any extended state that the TD is allowed to use (per TDCS.XFAM) may be cleared to its architectural INIT state.

The following table details TDH.VP.ENTER output operands when TD entry succeeds, and later an **asynchronous TD exit** occurs due to a **VMX architectural exit reason**.

**Table 5.262: TDH.VP.ENTER Output Operands Definition on Asynchronous TD Exits Following a TD Entry (with a VMX Architectural Exit Reason)**

Operand	Name	Description		
RAX	Status	SEAMCALL instruction return code		
		Bit(s)	Name	Description
		31:0	DETAILS_L2: Exit Reason	VMCS exit reason <b>Note:</b> Exit reason TDCALL (77) is a special case, indicating a synchronous TD exit initiated by TDG.VP.VMCALL; see below.
		47:32	CLASS and DETAILS_L1	May have the following values: <ul style="list-style-type: none"><li>TDX_SUCCESS, indicating a normal TD exit</li><li>TDX_NON_RECOVERABLE_VCPU, indicating that the VCPU is disabled</li><li>TDX_NON_RECOVERABLE_TD, indicating that the TD is disabled</li><li>TDX_NON_RECOVERABLE_TD_NON_ACCESSIBLE, indicating that the TD is disabled, and its private memory can't be accessed</li></ul>
		Other		See the function completion status definition in 5.3.1.
RCX	Exit Information	Index of the VM from which the TD exit occurred		
		Bit(s)	Name	Description
		31:0	EXIT_QUALIFICATION	VMCS exit qualification bits 31:0 <b>Note:</b> VMCS exit qualification bits 63:32 are always 0.  When exit is due to an EPT violation, bits 12:7 are cleared to 0.
		33:32	VM	Index of the VM that was running at the time of TD exit
		63:34	RESERVED	Reserved, set to 0
RDX	Extended Exit Qualification	Additional non-VMX, TDX-specific information – see 3.7.1		
R8	Guest Physical Address	When exit is due to EPT violation or EPT misconfiguration, format is similar to the VMCS guest-physical address, except that bits 11:0 are cleared to 0.  In other cases, R8 is cleared to 0.		

Operand	Name	Description
R9	VM-Exit Interruption Information	When exit is due to a vectored event, format of bits 31:0 is similar to the VMCS VM-exit interruption information. Bits 63:32 are cleared to 0. In other cases, R9 is cleared to 0.
RBX, RSI, RDI, R10 – R15	None	Cleared to 0
Extended State	Any extended state that the TD is allowed to use (per TDCS.XFAM) is cleared to its architectural INIT state.	

The following table details TDH.VP.ENTER output operands on when TD entry succeeds, and later an asynchronous TD exit occurs with a non-VMX TD exit status as described below.

**Table 5.263: TDH.VP.ENTER Output Operands Definition on Asynchronous TD Exits Following a TD Entry (with a non-VMX TD Exit Status)**

Operand	Name	Description		
RAX	Status	SEAMCALL instruction return code		
		Bit(s)	Name	Description
		47:32	CLASS and DETAILS_L1	May have the following values: <ul style="list-style-type: none"><li>• TDX_NON_RECOVERABLE_TD_CORRUPTED_MD</li><li>• TDX_HOST_PRIORITY_BUSY_TIMEOUT</li><li>• TDX_TD_EXIT_BEFORE_L2_ENTRY</li><li>• TDX I/O: TDX_DEVIF_HANDLE_ERROR</li></ul>
		Other		See the function completion status definition in [ref]
RCX	Exit Information	TD exit information		
		Bit(s)	Name	Description
		31:0	RESERVED	Reserved, set to 0
		33:32	VM	Index of the VM that was running at the time of TD exit
		63:34	RESERVED	Reserved, set to 0
RDX, RBX, RBP, RDI, RSI, R8 – R15	Reserved	Cleared to 0		
Extended State	Any extended state that the TD is allowed to use (per TDCS.XFAM) is cleared to its architectural INIT state.			

The following table details TDH.VP.ENTER output operands on when TD entry succeeds, and later an asynchronous TD exit occurs due to a **cross-TD operation**, i.e., the current TD operating on another TD.

**Table 5.264: TDH.VP.ENTER Output Operands Definition on Asynchronous TD Exits Following a TD Entry (with Cross-TD Exit Details)**

Operand	Name	Description	
RAX	Status	SEAMCALL instruction return code	
		<table> <tr> <th>Bit(s)</th><th>Name</th><th>Description</th></tr> </table>	Bit(s)
Bit(s)	Name	Description	

Operand	Name	Description		
		47:32	CLASS and DETAILS_L1	May have the following values: <ul style="list-style-type: none"><li>TDX_CROSS_TD_FAULT, indicating a fault-like asynchronous TD exit, with non-VMX cross-TD status.</li><li>TDX_CROSS_TD_TRAP, indicating a trap-like asynchronous TD exit, with non-VMX cross-TD status.</li></ul>
		Other		See the function completion status definition in [ref]
RCX	Exit Information	TD exit information		
		Bit(s)	Name	Description
		32:0	RESERVED	Reserved, set to 0
		33:32	VM	Index of the VM that was running at the time of TD exit
		63:34	RESERVED	Reserved, set to 0
RDX	Cross-TD Status	Status code of the error which caused the TD exit, using the same format as TDCALL instruction return code		
R8	Target TD	HPA of the TDR page of the TD which was the target of the cross-TD operation		
RBX, RBP, RDI, RSI, R9 – R15	Reserved	Cleared to 0		
Extended State	Any extended state that the TD is allowed to use (per TDCS.XFAM) is cleared to its architectural INIT state.			

The following table details TDH.VP.ENTER output operands on when TD entry succeeds, and later a **synchronous TD exit**, triggered by **TDG.VP.VMCALL**, occurs.

**Table 5.265: TDH.VP.ENTER Output Operands Definition on TDCALL(TDG.VP.VMCALL) Following a TD Entry**

Operand	Name	Description		
RAX	Status	SEAMCALL instruction return code		
		Bit(s)	Name	Description
		31:0	DETAILS_L2: Exit Reason	VMCS exit reason, indicating TDCALL (77)
		47:32	CLASS and DETAILS_L1	Indicating TDX_SUCCESS
		Other		See the function completion status definition in [ref]
RCX	Exit Information	TD exit information		
		Bit(s)	Name	Description
		31:0	PARAMS_MASK	Value as passed in to TDCALL(TDG.VP.VMCALL) by the guest TD: indicates which part of the guest TD GPR and XMM state is passed as-is to the VMM and back. For details, see the description of TDG.VP.VMCALL in [ref].
		33:32	VM	Index of the VM that was running at the time of TD exit
		63:34	RESERVED	Reserved, set to 0

Operand	Name	Description
RBX, RDX, RBP, RDI, RSI, R8 – R15	GPRs	If the corresponding bit in RCX is set to 1, the register value is passed as-is from the guest TD's input to TDG.VP.VMCALL. Else, the register value cleared to 0.
XMM0 – XMM15	XMMs	If the corresponding bit in RCX is set to 1, the register value is passed as-is from the guest TD's input to TDG.VP.VMCALL. Else, the register value cleared to 0.
Extended State except XMM	Any extended state, except XMM, that the TD is allowed to use (per TDCS.XFAM) is cleared to its architectural INIT state.	

### CPU State Preservation Following a Successful TD Entry and a TD Exit

Following a successful TD entry and a TD exit, some CPU state is modified:

- Registers DR0, DR1, DR2, DR3, DR6 and DR7 are set to their architectural INIT value.
- XCR0 is set to the TD's user-mode feature bits of XFAM (bits 7:0, 9).
- Multiple MSRs are set as described below. In this table, Init(condition) means that the MSR is set to its INIT value if the condition is true, else the MSR is unmodified.

**Table 5.266: MSR State Preservation Notation**

Notation	Meaning
INIT	TDX Module sets the MSR to its RESET value.
Init(condition)	TDX Module sets the MSR to its RESET value if condition is true, else the value is unmodified.

10

**Table 5.267: MSRs that may be Modified by TDH.VP.ENTER**

MSR Index Range (Hex)			MSR Architectural Name	MSR Preservation across TDH.VP.ENTER
First (H)	Last (H)	Size (H)		
0x00C1	0x00C8	0x8	IA32_PMCx	Implicit (via IA32_A_PMCx): Init(PERFMON)
0x00E1	0x00E1	0x1	IA32_UMWAIT_CONTROL	Init(virt. CPUID(7,0).ECX[5])
0x0186	0x018D	0x8	IA32_PERFEVTSELx	Init(PERFMON)
0x01A6	0x01A7	0x2	MSR_OFFCORE_RSPx	Init(PERFMON)
0x01C4	0x01C4	0x1	IA32_XFD	Init(virt. CPUID(0xD,0x1).EAX[4])
0x01C5	0x01C5	0x1	IA32_XFD_ERR	Init(virt. CPUID(0xD,0x1).EAX[4])
0x01D9	0x01D9	0x1	IA32_DEBUGCTL	INIT, except for the following bits which are preserved: Bit 1 (BTF) Bit 12 (FREEZE_PERFMON_ON_PMI) Bit 14 (FREEZE_WHILE_SMM)
0x0309	0x030C	0x4	IA32_FIXED_CTRx	Init(PERFMON)
0x0329	0x0329	0x1	IA32_PERF_METRICS	Init(PERFMON)
0x038D	0x038D	0x1	IA32_FIXED_CTR_CTRL	Init(PERFMON)
0x038E	0x038E	0x1	IA32_PERF_GLOBAL_STATUS	Init(PERFMON)
0x038F	0x038F	0x1	IA32_PERF_GLOBAL_CTRL	Init(PERFMON)

MSR Index Range (Hex)			MSR Architectural Name	MSR Preservation across TDH.VP.ENTER
First (H)	Last (H)	Size (H)		
0x03F1	0x03F1	0x1	IA32_PEBs_ENABLE	Init(PERFMON)
0x03F2	0x03F2	0x1	MSR_PEBs_DATA_CFG	Init(PERFMON)
0x03F6	0x03F6	0x1	MSR_PEBs_LD_LAT	Init(PERFMON)
0x03F7	0x03F7	0x1	MSR_PEBs_FRONTEND	Init(PERFMON)
0x04C1	0x04C8	0x8	IA32_A_PMCx	Init(PERFMON)
0x0560	0x0560	0x1	IA32_RTIT_OUTPUT_BASE	Init(XFAM(8))
0x0561	0x0561	0x1	IA32_RTIT_OUTPUT_MASK_PTRS	Init(XFAM(8))
0x0570	0x0570	0x1	IA32_RTIT_CTL	Init(XFAM(8))
0x0571	0x0571	0x1	IA32_RTIT_STATUS	Init(XFAM(8))
0x0572	0x0572	0x1	IA32_RTIT_CR3_MATCH	Init(XFAM(8))
0x0580	0x0580	0x1	IA32_RTIT_ADDR0_A	Init(XFAM(8))
0x0581	0x0581	0x1	IA32_RTIT_ADDR0_B	Init(XFAM(8))
0x0582	0x0582	0x1	IA32_RTIT_ADDR1_A	Init(XFAM(8))
0x0583	0x0583	0x1	IA32_RTIT_ADDR1_B	Init(XFAM(8))
0x0584	0x0584	0x1	IA32_RTIT_ADDR2_A	Init(XFAM(8))
0x0585	0x0585	0x1	IA32_RTIT_ADDR2_B	Init(XFAM(8))
0x0586	0x0586	0x1	IA32_RTIT_ADDR3_A	Init(XFAM(8))
0x0587	0x0587	0x1	IA32_RTIT_ADDR3_B	Init(XFAM(8))
0x06A0	0x06A0	0x1	IA32_U_CET	Init(XFAM[11]   XFAM[12])
0x06A4	0x06A4	0x1	IA32_PL0_SSP	Init(XFAM[11]   XFAM[12])
0x06A5	0x06A5	0x1	IA32_PL1_SSP	Init(XFAM[11]   XFAM[12])
0x06A6	0x06A6	0x1	IA32_PL2_SSP	Init(XFAM[11]   XFAM[12])
0x06A7	0x06A7	0x1	IA32_PL3_SSP	Init(XFAM[11]   XFAM[12])
0x0985	0x0985	0x1	IA32_UINTR_RR	Init(XFAM[14])
0x0986	0x0986	0x1	IA32_UINTR_HANDLER	Init(XFAM[14])
0x0987	0x0987	0x1	IA32_UINTR_STACKADJUST	Init(XFAM[14])
0x0988	0x0988	0x1	IA32_UINTR_MISC	Init(XFAM[14])
0x0989	0x0989	0x1	IA32_UINTR_PD	Init(XFAM[14])
0x098A	0x098A	0x1	IA32_UINTR_TT	Init(XFAM[14])
0x0DA0	0x0DA0	0x1	IA32_XSS	Supervisor-mode feature bits of XFAM (bits 8, 16:10)
0x1200	0x12FF	0x100	IA32_LBR_INFO	Init(XFAM[15])
0x14CE	0x14CE	0x1	IA32_LBR_CTL	Init(XFAM[15])
0x14CF	0x14CF	0x1	IA32_LBR_DEPTH	Init(XFAM[15])
0x1500	0x15FF	0x100	IA32_LBR_FROM_IP	Init(XFAM[15])
0x1600	0x16FF	0x100	IA32_LBR_TO_IP	Init(XFAM[15])
0x1B01	0x1B01	0x1	IA32_UARCH_MISC_CTL	INIT
0xC0000081	0xC0000081	0x1	IA32_STAR	INIT
0xC0000082	0xC0000082	0x1	IA32_LSTAR	INIT
0xC0000084	0xC0000084	0x1	IA32_FMASK	INIT
0xC0000102	0xC0000102	0x1	IA32_KERNEL_GS_BASE	INIT
0xC0000103	0xC0000103	0x1	IA32_TSC_AUX	INIT

## Special Environment Requirements

### Guest TD State Loading or VM Entry Failure

TDH.VP.ENTER may fail loading guest TD state in the cases shown in the table below. TDH.VP.ENTER returns with information detailing the failure case. Such failures may happen due to the following reasons:

- The TD is being debugged (its ATTRIBUTES.DEBUG bit is set) and the debugger set some wrong guest state value using TDH.VP.WR. For a debuggable TD, the completion status (in RAX[63:32]) is set in such cases to TDX\_SUCCESS, and the details are provided as described below. The debugger may update the VCPU state using TDH.VP.WR and invoke TDH.VP.ENTER again.

- The TD has been migrated, and some of its state is not compatible with the destination platform. The TDX module does its best effort to check guest state values during import, but there might still be cases where incompatible guest TD state gets migrated. For a non-debuggable TD, the completion status (in RAX[63:32]) is set in such cases to TDX\_NON\_RECOVERABLE\_TD, and the details are provided as described below. The host VMM should tear down the TD.

Table 5.268: Guest State Loading Errors

Guest State Loading Error	VM Exit Reason in RAX[31:0]	Extended Exit Qualification in RDX
Error while loading guest MSR values from TDVPS	34: VM-entry failure due to MSR loading	TD_ENTRY_MSR_LOAD_FAILURE with the MSR index
Error while loading CPU extended state from TDVPS	33: VM-entry failure due to invalid guest state	TD_ENTRY_XSTATE_LOAD_FAILURE
VM entry (VMLAUNCH or VMRESUME) which loads guest state from VMCS	33: VM-entry failure due to invalid guest state	NONE

### Leaf Function Latency

In some cases (e.g., suspected single/zero step attack mitigation), TDH.VP.ENTER execution time may be longer than most TDX module interface functions execution time. No interrupts (including NMI and SMI) are processed by the logical processor during that time.

### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.VP.ENTER enters TDX non-root operation. It returns immediately if TD entry failed. If TD entry succeeded, TDH.VP.ENTER returns when TD exit is initiated.

For partitioned TDs, the TD VCPU may operate in the L1 VM or one of the L2 VMs, if any. TD exit may be initiated from each of the TD's VMs. If last TD exit was from an L2 VM, TDH.VP.ENTER resumes the same L2 VM, unless the RESUME\_L1 input flag is set to 1, instructing TDH.VP.ENTER to resume the L1 VM.

**VCPU Association:** TDH.VP.ENTER associates the target TD VCPU with the current LP. This requires that the VCPU will not be associated with another LP. For details, see the [TDX Module Base Spec].

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

Table 5.269: TDH.VP.ENTER Operands Information Definition

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDVPR page	TDVPS	RW	Opaque	4KB	Shared(c) <sup>7</sup>	Shared(c) <sup>7</sup>	Shared(c) <sup>7</sup>
Implicit	N/A	HPA	TDR page	TDR	RW	Opaque	N/A	Shared(c) <sup>7</sup>	N/A	N/A
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i,c) <sup>7</sup>	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Shared(t) <sup>8</sup>	N/A	N/A

<sup>7</sup> The shared locking of TDVPS, TDR, TDCS, TDCS.OP\_STATE is for the whole duration of running in TDX non-root mode; the locks are released on TD exit.

<sup>8</sup> The locking of OP\_STATE, SEPT tree and the TLB tracking fields is until before entering TDX non-root mode; the locks are released before VM entry into the TD VCPU.

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Implicit	N/A	N/A	TDCS TLB Tracking Fields	N/A	RW	Opaque	N/A	Shared(t) <sup>8</sup>	N/A	N/A
Implicit	N/A	N/A	SEPT tree	N/A	R	Opaque	N/A	Exclusive(t) <sup>8</sup>	N/A	N/A

In addition to the explicit memory operand checks per the table above, the function checks the following conditions:

**TODO: update with L2 details**

1. The TDVPR page metadata in PAMT must be correct (PT must be PT\_TDVPR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
4. **The TD must have been finalized and is allowed to run (TDR.NUM\_TDCX is the required number, and TDCS.OP\_STATE is either RUNNING, LIVE\_EXPORT or LIVE\_IMPORT).**

If successful, the function does the following:

5. Associate the VCPU with the current LP, and update TD VMCS using the algorithm described in 5.2.1.

If passed:

6. Update the TLB tracking state. This is done as a critical section allowing concurrent TDH.VP.ENTERS but no concurrent TDH.MEM.TRACK. A concurrent TDH.MEM.TRACK may cause this locking to fail; in this case, the caller is expected to retry TDH.VP.ENTER.
  - 6.1. Lock the TDCS epoch tracking fields in shared mode.
  - 6.2. Sample the TD's epoch counter (TDCS.TD\_EPOCH) into the VCPU's TDVPS.VCPU\_EPOCH.
  - 6.3. Atomically increment the TD's REFCOUNT that is associated with the sampled epoch (TDCS.REFCOUNT[TD\_EPOCH % 2]).
  - 6.4. Release the shared mode locking of the epoch tracking fields.

If successful:

7. If TDVPS.VCPU\_EPOCH was updated above, and this is not a new VCPU association:
  - 7.1. Execute single-context (type 1) INVEPT.
  - 7.2. **Invalidate all soft-translated GPAs.**
8. **Translate GPAs:**
  - 8.1. **Translate TDG.VP.ENTER memory output operands GPAs and GPAs of L2 VMCS fields. Translation needs to be done in one of the following cases:**
    - The GPA has been blocked since last translated.
    - HPA shadow for this GPA is NULL\_PA.

**Translation failure leads to an EPT violation TD exit emulation.**
  - 8.2. **Set the shadow HPA fields of all L2 VMs' soft-translated GPA fields to NULL\_PA (-1).**

If passed:

9. If the TD VCPU to be entered is different than the last TD VCPU entered on the current LP, issue an indirect branch prediction barrier command to the CPU by writing to the IA32\_PRED\_CMD MSR with the IBPB bit set.
10. Set TDVPS.VCPU\_STATE to VCPU\_ACTIVE.
11. Restore guest TD state:
  - 11.1. If previous TD exit was due to a TDG.VP.VMCALL:
    - 11.1.1. Restore guest XMM and GPR state that is not passed as-is from the host VMM, as controlled by the value of guest TD RCX input to TDG.VP.VMCALL.
    - 11.1.2. Set guest RAX to 0.
  - 11.2. Else (TD exit was an asynchronous exit):
    - 11.2.1. Restore CPU extended state from TDVPS (per TDCS.XFAM).
  - 11.3. Restore other guest state from TDVPS.
12. Execute VMLAUNCH or VMRESUME depending on whether the **entered VCPU and VM (i.e., the current VMCS) has been launched on this LP since the VCPU's last association with the LP (TDVPS.LAUNCHED[VM]).**



**Note:** Logically, from the point of view of the host VMM, a successful TDH.VP.ENTER is terminated by the next TD exit.

#### Completion Status Codes

**Table 5.270: TDH.VP.ENTER Completion Status Codes (Returned in RAX) Definition [TO BE COMPLETED]**

Completion Status Code	Description
TDX_NON_RECOVERABLE_TD	TDH.VP.ENTER launched or resumed TD VCPU operation (TDX non-root mode) – followed later by a TD exit. The TD state is non-recoverable – further TD entry is prohibited. Exit reason is in RAX bits 31:0.
TDX_NON_RECOVERABLE_VCPU	TDH.VP.ENTER launched or resumed TD VCPU operation (TDX non-root mode) – followed later by a TD exit. The TD VCPU state is non-recoverable – further TD entry to this VCPU is prohibited. Exit reason is in RAX bits 31:0.
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	<p>Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.</p> <p>Note the special case where the indicated operand is TLB_EPOCH. This may happen due to a conflict with TDH.MEM.TRACK or TDH.EXPORT.PAUSE. The host VMM may retry TDH.VP.ENTER.</p> <p>Another special case is where the indicated operand is SEPT_TREE. In some cases, TDH.VP.ENTER may acquire exclusive access on the SEPT tree for a short period of time, and may fail due to a concurrent operation. The host VMM should retry TDH.VP.ENTER.</p>
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.VP.ENTER launched or resumed TD VCPU operation (TDX non-root mode) – followed later by a TD exit. Exit reason is in RAX bits 31:0.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_FINALIZED	
TDX_TD_NOT_INITIALIZED	
TDX_VCPU_ASSOCIATED	
TDX_VCPU_STATE_INCORRECT	

**5.3.66. UPDATED: TDH.VP.FLUSH Leaf**

Flush the address translation caches and cached TD VMCS associated with a TD VCPU on the current logical processor.

**Table 5.271: TDH.VP.FLUSH Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	The physical address of a TDVPR page (HKID bits must be 0)		

**Table 5.272: TDH.VP.FLUSH Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.VP.FLUSH flushes the address translation caches and cached TD VMCS associated with a TD VCPU on the current LP. It then marks the VCPU as not associated with any LP.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.273: TDH.VP.FLUSH Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDVPR page	TDVPS	RW	Opaque	4KB	Exclusive	Shared	Shared
Implicit	N/A	HPA	TDR page	TDR	R	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A

In addition to the memory operand checks per the table above, the function checks the following:

1. The TDVPR page metadata in PAMT must be correct (PT must be PT\_TDVPR).
2. TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED.

3. The current VCPU must be currently associated with the current LP.

If the above checks pass, the function does the following:

4. For each L2 VM:
  - 4.1. Flush the TLB context and extended paging structure (EPxE) caches associated with the L2 VM using INVEPT single-context invalidation (type 1).
  - 4.2. Flush the cached L2 VMCS content to TDVPS using VMCLEAR.
5. Flush the TLB context and extended paging structure (EPxE) caches associated with the TD using INVEPT single-context invalidation (type 1).
6. Flush the cached TD VMCS content to TDVPS using VMCLEAR.
7. Mark the current VCPU as not associated with any LP.
8. Atomically decrement (using LOCK XADD) the associated VCPUs counter (TDCS.NUM\_ASSOC\_VCPUS).

### Completion Status Codes

**Table 5.274: TDH.VP.FLUSH Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_LIFECYCLE_STATE_INCORRECT	
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.VP.FLUSH is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_VCPU_NOT_ASSOCIATED	

**5.3.67. UPDATED: TDH.VP.INIT Leaf**

Initialize the saved state of a TD VCPU.

**Operands****Table 5.275: TDH.VP.INIT Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	The physical address of a TDVPR page (HKID bits must be 0)		
RDX	Initial value of the guest TD VCPU RCX		

**Table 5.276: TDH.VP.INIT Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
Other	Unmodified

**Leaf Function Latency**

TDH.VP.INIT execution time may be longer than most TDX module interface functions execution time. No interrupts (including NMI and SMI) are processed by the logical processor during that time.

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.VP.INIT initialized the saved state of a VCPU in the TDVPR and TDPX pages.

**VCPU Association:** TDH.VP.INIT associates the target TD VCPU with the current LP – for details, see the [TDX Module Base Spec].

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.277: TDH.VP.INIT Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDVPR page	TDVPS	RW	Opaque	4KB	Exclusive	Shared	Shared
Implicit	N/A	HPA	TDR page	TDR	R	Opaque	4KB	Shared	N/A	N/A
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	4KB	Shared(i)	N/A	N/A

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Shared	N/A	N/A

In addition to the memory operand checks per the table above, the function checks the following:

1. The TDVPR page metadata in PAMT must be correct (PT must be PT\_TDVPR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
4. **The TD must have been initialized but not finalized (TDR.NUM\_TDCX is the required number and TDCS.OP\_STATE is INITIALIZED).**
5. The number of pages allocated to this TDVPS is correct.
6. The TD VCPU has not been initialized (by TDH.VP.INIT) and is not being torn down (TDVPS.VCPU\_STATE is VCPU\_UNINITIALIZED).

If successful, the function does the following:

7. Atomically increment the TD's VCPU counter (TDCS.NUM\_VCPUS), and check that maximum number of VCPUs (TDCS.MAX\_VCPUS) has not been exceeded.

If passed:

8. Assign a unique sequential identifier to the VCPU.
9. Initialize the VCPU state fields in the logical TDVPS structure (TDVPR and TDCX pages).
10. Associate the VCPU with the current LP and update the VMCS physical pointers and HKID execution control with the TD's HKID.
11. Set the TDVPS.LAST\_TD\_EXIT to ASYNC\_FAULT since the first TD entry is the same as TD entry following an asynchronous fault-like TD exit.

#### Completion Status Codes

Table 5.278: TDH.VP.INIT Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_MAX_VCPUS_EXCEEDED	
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.VP.INIT is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_FINALIZED	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TDCX_NUM_INCORRECT	
TDX_VCPU_ASSOCIATED	

Completion Status Code	Description
TDX_VCPU_STATE_INCORRECT	
TDX_VCPU_STATE_INCORRECT	

DRAFT

**5.3.68. UPDATED: TDH.VP.RD Leaf**

Read a VCPU-scope metadata fields (control structure field) of a TD.

**Table 5.279: TDH.VP.RD Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version May be 0 or 1
	63:24	Reserved	Must be 0
RCX	The physical address of a TDVPR page (HKID bits must be 0)		
RDX	Field <b>identifier</b> – see 3.10 The <b>LAST_ELEMENT_IN_FIELD</b> and <b>LAST_FIELD_IN_SEQUENCE</b> components of the field identifier must be 0. <b>WRITE_MASK_VALID</b> , <b>INC_SIZE</b> , <b>CONTEXT_CODE</b> and <b>ELEMENT_SIZE_CODE</b> components of the field identifier are ignored. For TDH.VP.RD version 1 or higher, a value of -1 is a special case: it is not a valid field identifier; in this case the first readable field identifier is returned in RDX.		

**Table 5.280: TDH.VP.RD Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
RDX	For TDH.VP.RD version 0, RDX is unmodified. For TDH.VP.RD version 1 or higher: <ul style="list-style-type: none"> <li>If the input field identifier was -1, RDX returns the first readable field identifier.</li> </ul> Else, in there case of an error, RDX returns -1. On success, RDX returns the next readable field identifier. A value of -1 indicates no next field identifier is available.
R8	Field content In case of no success, as indicated by RAX, R8 returns 0.
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.VP.RD reads a TDVPS field, given its field code. Reading is subject to the field's readability (per the TD's ATTRIBUTES.DEBUG bit).

If version 1 or higher is specified in RAX, RDX returns the next host-side readable field identifier. This may be used by the host VMM to dump the host readable VCPU metadata. To read all the available fields, the host VMM can invoke TDH.VP.RD in a loop, starting with field identifier -1 as an input, until RDX returns -1.

- 15 **VCPU Association:** TDH.VP.RD associates the target TD VCPU with the current LP. This requires that the VCPU will not be associated with another LP – for details, see the [TDX Module Base Spec].

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.281: TDH.VP.RD Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDVPR page	TDVPS	RW	Opaque	4KB	Shared	Shared	Shared
Implicit	N/A	HPA	TDR page	TDR	R	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Shared	N/A	N/A

5 In addition to the memory operand checks per the table above, the function checks the following:

1. The TDVPR page metadata in PAMT must be correct (PT must be PT\_TDVPR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
4. The TD must have been initialized (TDR.NUM\_TDCX is the required number and TDCS.OP\_STATE is not UNALLOCATED nor UNINITIALIZED).
5. The provided field code is valid.
6. The provided TDVPS field is readable per the TD's debug attribute (TDCS.ATTRIBUTES.DEBUG).

If successful, the function does the following:

7. Associate the VCPU with the current LP, and update TD VMCS using the algorithm described in 5.2.1.

15 If passed:

8. Read the control structure field using the algorithm described in 5.2.2.1.

#### Completion Status Codes

**Table 5.282: TDH.VP.RD Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_METADATA_FIRST_FIELD_ID_IN_CONTEXT	Indicates that the first field ID in context is returned
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.VP.RD is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	



Completion Status Code	Description
TDX_VCPU_ASSOCIATED	
TDX_VCPU_STATE_INCORRECT	

DRAFT

**5.3.69. UPDATED: TDH.VP.WR Leaf**

Write a VCPU-scope metadata field (control structure field) of a TD.

**Table 5.283: TDH.VP.WR Input Operands Definition**

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	The physical address of a TDVPR page (HKID bits must be 0)		
RDX	Field <b>identifier</b> – see 3.10  The <b>LAST_ELEMENT_IN_FIELD</b> and <b>LAST_FIELD_IN_SEQUENCE</b> components of the field identifier must be 0.  The <b>WRITE_MASK_VALID</b> , <b>INC_SIZE</b> , <b>CONTEXT_CODE</b> and <b>ELEMENT_SIZE_CODE</b> components of the field identifier are ignored.		
R8	64b value to write to the field		
R9	A 64b write mask to indicate which bits of the value in R8 are to be written to the field		

**Table 5.284: TDH.VP.WR Output Operands Definition**

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
R8	Previous content of the field In case of an error, as indicated by RAX, R8 returns 0.
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDH.VP.WR writes a TDVPS field, given its field code. The specific bits of the value (R8) are written as specified by the write mask (R9). Writing is subject to the field's writability (per the TD's ATTRIBUTES.DEBUG bit). Writing of specific fields is also subject to additional rules as detailed in 4.2.

TDH.VP.WR returns the previous content of the field masked by the field's readability (per the TD's ATTRIBUTES.DEBUG bit).

- 15 **VCPU Association:** TDH.VP.WR associates the target TD VCPU with the current LP. This requires that the VCPU will not be associated with another LP – for details, see the [TDX Module Base Spec].

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.285: TDH.VP.WR Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	TDVPR page	TDVPS	RW	Opaque	4KB	Shared	Shared	Shared
Implicit	N/A	HPA	TDR page	TDR	R	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	N/A	N/A
Implicit	N/A	N/A	TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Shared	N/A	N/A

In addition to the memory operand checks per the table above, the function checks the following:

1. The TDVPR page metadata in PAMT must be correct (PT must be PT\_TDVPR).
2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
3. The TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
4. The TD must have been initialized (TDR.NUM\_TDCX is the required number and TDCS.OP\_STATE is not UNALLOCATED nor UNINITIALIZED).
5. The provided field code is valid.
6. The provided TDVPS field is writable per the TD's debug attribute (TDCS.ATTRIBUTES.DEBUG).

If successful, the function does the following:

7. Associate the VCPU with the current LP, and update TD VMCS using the algorithm described in 5.2.1.

If passed:

8. Write the control structure field and return its old value, using the algorithm described in 5.2.2.2.
  - 8.1. Writes of some fields are subject to rules, as detailed per field in 4.2 – e.g., the value of fields that contain Shared physical address, such as the Shared EPT Pointer, must have a Shared HKID value and must comply with some alignment rules.
  - 8.2. In most cases, writes of guest state fields are subject to the same rules as if the write is done by the guest itself – e.g., writing to guest CR4 is subject to the rules described in the [TDX Module Base Spec]. If the write operation is invalid, TDH.VP.WR fails and returns a proper error code.
  - 8.3. In debug mode (ATTRIBUTES.DEBUG == 1), there are some TDVPS fields where the TDH.VP.WR does not check whether the written values are architecturally valid. It is the responsibility of the host VMM, and failing to do so will later cause a VM entry failure leading to a fatal shutdown of the Intel TDX module. The security of any guest TD is not impacted.
  - 8.4. In other cases, in debug mode (ATTRIBUTES.DEBUG == 1), TDH.VP.WR allows setting of TDVPS fields to values that may impact the correct operation of the TD under debug. It is the responsibility of the host VMM to take this into consideration.
    - TDH.VP.WR is allowed to enable BTM by setting guest IA32\_DEBUGCTL[7:6] to 0x1.
    - TDH.VP.WR is allowed to modify the state of IA32\_DEBUGCTL[13] (ENABLE\_UNCORE\_PMI).
    - TDH.VP.WR is allowed to enable VM exits on exceptions other than MCE by setting the TD VMCS exception bitmap execution control. The Intel TDX module does not take this into account when handling VM exits that occur during event delivery.

### Completion Status Codes

**Table 5.286: TDH.VP.WR Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	TDH.VP.WR is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	
TDX_TD_VMCS_FIELD_NOT_INITIALIZED	
TDX_TDVPS_FIELD_NOT_WRITABLE	
TDX_VCPU_ASSOCIATED	
TDX_VCPU_STATE_INCORRECT	
TO BE COMPLETED	

DRAFT

#### 5.4. **UPDATED:** Guest-Side (TDCALL) Interface Functions

The TDCALL instruction causes a VM exit to the Intel TDX module. It is used to call guest-side Intel TDX functions, either local or a TD exit to the host VMM, as selected by RAX.

##### 5.4.1. TDCALL Instruction (Common)

- 5 This section describes the common functionality of TDCALL. Leaf functions are described in the following sections. As used by the Intel TDX module, TDCALL is allowed only in 64b mode.

**Table 5.287: TDCALL Input Operands Definition**

Operand	Description		
RAX	Leaf and version numbers, as defined in the [TDX Module Base Spec]. See Table 5.289 below for TDCALL leaf numbers.		
	Bits	Field	Description
	15:0	Leaf Number	Selects the TDCALL interface function
	23:16	Version Number	Selects the TDCALL interface function version
	63:24	Reserved	Must be 0
Other	See individual TDCALL leaf functions.		

**Table 5.288: TDCALL Output Operands Definition**

Operand	Description
RAX	Instruction return code, indicating the outcome of execution of the instruction – see the [TDX Module Base Spec] for details.
Other	See individual TDCALL leaf functions.

**Table 5.289: TDCALL Instruction Leaf Numbers Definition**

Leaf #	Interface Function Name	Description
0	TDG.VP.VMCALL	Call a host VM service
1	TDG.VP.INFO	Get TD execution environment information
2	TDG.MR.RTMR.EXTEND	Extend a TD run-time measurement register
3	TDG.VP.VEINFO.GET	Get Virtualization Exception Information for the recent #VE exception
4	TDG.MR.REPORT	Creates a cryptographic report of the TD
5	TDG.VP.CPUIDVE.SET	Control delivery of #VE on CPUID instruction execution
6	TDG.MEM.PAGE.ACCEPT	Accept a pending private page into the TD
7	TDG.VM.RD	Read a TD-scope metadata field
8	TDG.VM.WR	Write a TD-scope metadata field
9	TDG.VP.RD	Read a VCPU-scope metadata field
10	TDG.VP.WR	Write a VCPU-scope metadata field
11	TDG.SYS.RD	Read a TDX Module global-scope metadata field
12	TDG.SYS.RDALL	Read all guest-readable TDX Module global-scope metadata fields
18	TDG.SERVTD.RD	Read a target TD metadata field
20	TDG.SERVTD.WR	Write a target TD metadata field
22	TDG.MR.VERIFYREPORT	Verify a cryptographic report of a TD, generated on the current platform
23	TDG.MEM.PAGE.ATTR.RD	Read the GPA mapping and attributes of a TD private page
24	TDG.MEM.PAGE.ATTR.WR	Write the attributes of a private page
25	TDG.VP.ENTER	Enter L2 VCPU operation

Leaf #	Interface Function Name	Description
26	TDG.VP.INVEPT	Invalidate cached EPT translations for selected L2 VMs
27	TDG.VP.INVVPID	Invalidate cached translations for selected pages in an L2 VM

### Instruction Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

5 This section describes how TDCALL leaf functions are implemented by the Intel TDX module.

On VM exit, the Intel TDX module performs the following checks:

1. If the CPU mode is not 64b ((IA32\_EFER.LMA == 1) && (CS.L == 1)), the Intel TDX module injects a #GP(0) fault into the guest TD.
  2. If the leaf number in RAX is not supported by the Intel TDX module, it returns a TDX\_OPERAND\_INVALID(0) status code in RAX.
- 10

If all checks pass, the Intel TDX module calls the leaf function according to the leaf number in RAX – see the following sections for individual leaf function details.

### Completion Status Codes

**Table 5.290: TDCALL Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_SUCCESS	TDCALL is successful.
TDX_OPERAND_INVALID	Invalid leaf number
Other	See individual leaf functions

### 5.4.2. TDG.MEM.PAGE.ACCEPT Leaf

Accept a pending private page and initialize it to all-0 using the TD ephemeral private key.

**Table 5.291: TDG.MEM.PAGE.ACCEPT Input Operands Definition**

Operand	Description		
RAX	TDCALL instruction leaf number and version, see 5.4.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the TDCALL interface function
	23:16	Version Number	Selects the TDCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	EPT mapping information:		
	Bits	Name	Description
	2:0	Level	Level of the Secure EPT leaf entry that maps the private page to be accepted: either 0 (4KB) or 1 (2MB) – see 3.6.1.
	11:3	Reserved	Reserved: must be 0
	51:12	GPA	Bits 51:12 of the guest physical address of the private page to be accepted
	63:52	Reserved	Reserved: must be 0

**Table 5.292: TDG.MEM.PAGE.ACCEPT Output Operands Definition**

Operand	Description
RAX	TDCALL instruction return code – see 5.4.1
Other	Unmodified

#### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDG.MEM.PAGE.ACCEPT accepts a pending private page, previously added by TDH.MEM.PAGE.AUG, into the TD. It initializes the page to 0. **If page attributes have been set by the guest TD, using TDG.MEM.PAGE.ATTR.WR, while the page was pending, they become effective when the page is accepted.**

#### SEPT Mapping Size Considerations

- 15 In most cases, the guest TD is unaware of how TD private pages are mapped by the host VMM in SEPT. However, TDG.MEM.PAGE.ACCEPT operation specifies a page mapping size and may fail if the specified size is different than the actual mapping size.

- If the page is mapped at a lower level than requested, the function returns TDX\_PAGE\_SIZE\_MISMATCH. The guest may re-invoke TDG.MEM.PAGE.ACCEPT specifying a 4KB page size.
  - If the page is mapped at a higher level than requested, this results in an EPT violation TD exit, with extended exit qualification indicating the error SEPT entry level and state, and the guest-requested mapping level. The host VMM is expected to demote the page, then re-enter the guest TD so TDG.MEM.PAGE.ACCEPT is re-invoked.
- 20

### Other Conditions that Prevent Page Acceptance

- If the page has already been accepted, the function returns `TDX_PAGE_ALREADY_ACCEPTED`.
  - If the page is not `PENDING` nor `PENDING_EXPORTED_DIRTY`, this results in an EPT violation TD exit, with extended exit qualification indicating the error SEPT entry level and state, and the guest-requested accept level.
- 5 **Interruptibility** If, during its execution, `TDG.MEM.PAGE.ACCEPT` detects that an external interrupt is pending, it may resume the guest TD with the CPU state unmodified. The progress so far is recorded in the page's Secure EPT entry. This allows the external interrupt to be recognized, causing a TD exit or a posted interrupt delivery. Typically, `TDG.MEM.PAGE.ACCEPT` will be re-invoked and continue its work. Guest TD software is not directly involved.
- 10 To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.293 TDG.MEM.PAGE.ACCEPT Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions
Explicit	RCX	GPA	TD private page	Blob	RW	Private	$2^{12+9*\text{Level}}$ Bytes	None
Implicit	N/A	N/A	TDR page	TDR	None	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDCS structure	TDCS	R	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDVPS structure	TDVPS	RW	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	Secure EPT tree	N/A	RW	Private	N/A	None
Implicit	N/A	GPA	Secure EPT entry	SEPT Entry	RW	Private	N/A	Exclusive(h)

15 `TDG.MEM.PAGE.ACCEPT` checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

In addition to the memory operand checks per the table above, the function does the following (no specific order is implied):

1. Walk the Secure EPT based on the GPA operand and requested level. The walk is successful if arrived at a leaf entry whose state is `PENDING`. In case of error, return a status code or TD exit as described in the [TDX Module Base Spec].
- 20 If successful, do the following:
  2. Loop until the whole page has been initialized, or until interrupted:
    - 2.1. Initialize the next 4KB chunk to 0 using the TD's ephemeral private HKID and direct writes (`MOVDIR64B`).
    - 2.2. If not done and there is a pending interrupt, abort `TDG.MEM.PAGE.ACCEPT` and resume the guest TD without updating RIP and any GPR.
  - 25 If done initializing the page, do the following:
    3. Set the SEPT entry to `MAPPED`.
    4. For each L2 VM where the page is mapped:
      - 4.1. Walk the L2 Secure EPT based on the GPA operand and find the L2 SEPT entry for the page to be accepted.
      - 4.2. Restore the L2 SEPT entry attributes.
      - 30 4.3. Set the L2 SEPT entry state to `L2_MAPPED`.

### Completion Status Codes

**Table 5.294: TDG.MEM.PAGE.ACCEPT Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
<code>TDX_OPERAND_INVALID</code>	



Completion Status Code	Description
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation. Specifically, it may indicate that a concurrent TDG.MEM.PAGE.ACCEPT is using the same Secure EPT entry
TDX_PAGE_ALREADY_ACCEPTED	
TDX_PAGE_SIZE_MISMATCH	Requested page size is 2MB, but the page GPA is not mapped at 2MB size
TDX_SUCCESS	TDG.MEM.PAGE.ACCEPT is successful.

DRAFT

**5.4.3. NEW: TDG.MEM.PAGE.ATTR.RD Leaf**

Read the GPA mapping and attributes of a TD private page.

**Table 5.295: TDG.MEM.PAGE.ATTR.RD Input Operands Definition**

Operand	Name	Description		
RAX	Leaf and Version	TDCALL instruction leaf number and version, see 5.4.1		
		Bits	Field	Description
		15:0	Leaf Number	Selects the TDCALL interface function
		23:16	Version Number	Selects the TDCALL interface function version Must be 0
		63:24	Reserved	Must be 0
RCX	GPA	Guest physical address		

5

**Table 5.296: TDG.MEM.PAGE.ATTR.RD Output Operands Definition**

Operand	Name	Description		
RAX	STATUS	TDCALL instruction return code – see 5.4.1		
RCX	GPA_MAPPING	Actual GPA mapping of the page:		
		Bits	Name	Description
		2:0	LEVEL	Level of the Secure EPT leaf entry that maps the private page: either 0 (4KB), 1 (2MB) or 2 (1GB) – see 3.6.1.
		11:3	RESERVED	Reserved: set to 0
		51:12	GPA	Bits 51:12 of the guest physical start address of the private page  Depending on the level, the following least significant bits are always 0:  Level 0 (EPTE):      None Level 1 (EPDE):      Bits 20:12 Level 2 (EPDPTE):   Bits 29:12
		61:52	RESERVED	Reserved: set to 0
		62	PENDING	Flags that the page is PENDING
		63	RESERVED	Reserved: set to 0
RDX	GPA_ATTR	Guest-visible page attributes. See the GPA_ATTR definition in 3.6.3.		
Other		Unmodified		

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.MEM.PAGE.ATTR.RD reads the GPA mapping and attributes of a TD private page. Given a GPA (which can be anywhere within a page) it returns the actual mapping level – either 0 (4KB), 1 (2MB) or 2 (1GB) – and the guest-readable attributes. TDH.MEM.PAGE.ATTR.RD can read the attributes of a PENDING page.

GPA mapping level is exposed to the guest TD since page acceptance (TDH.MEM.PAGE.ACCEPT) and page attributes modifications and L2 page aliases management (TDH.MEM.PAGE.ATTR.WR) are done at mapping granularity.

**EPT Violation:** If the requested GPA is not guest-readable and not pending acceptance, TDH.MEM.PAGE.ATTR.RD causes an EPT violation TD exit.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.297 TDG.MEM.PAGE.ATTR.RD Memory Operands Information Definition**

Explicit/ Implicit	Reg.	Addr. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions
Explicit	RCX	GPA	TD private page	Blob	RW	Private	None	None
Implicit	N/A	N/A	TDR page	TDR	None	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	Secure EPT tree	N/A	R	Private	N/A	None
Implicit	N/A	N/A	L2 Secure EPT trees	N/A	RW	Private	N/A	None
Implicit	N/A	GPA	Secure EPT entry	SEPT Entry	R	Private	N/A	Exclusive(h)
Implicit	N/A	GPA	L2 Secure EPT entries	SEPT Entry	RW	Private	N/A	Exclusive(i)

TDG.MEM.PAGE.ATTR.RD checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

In addition to the memory operand checks per the table above, the function does the following (no specific order is implied):

1. Check that the requested GPA is valid.
2. Walk the L1 Secure EPT based on the GPA operand.
  - 2.1. The walk is successful if arrived at a leaf entry whose state is either guest-accessible (MAPPED, EXPORTED\_DIRTY, \*BLOCKEDW\*) or pending but not blocked (PENDING, or PENDING\_EXPORTED\_\*).
  - 2.2. Else, do an EPT violation TD exit.

If successful, do the following:

3. For each L2 page alias to the L1 SEPT entry:
  - 3.1. Walk that L2 VM's SEPT and locate the page alias L2 SEPT leaf entry.
  - 3.2. Read the L2 SEPT entry and assemble the returned attributes. If the page is pending or blocked for writing, the L2 SEPT entry's original access permission bits are read from their saved locations in the L2 SEPT entry.
4. Return the page mapping and attributes.

#### Completion Status Codes

**Table 5.298: TDG.MEM.PAGE.ATTR.RD Completion Status Codes (Returned in RAX) Definition** **(TO BE COMPLETED)**

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_SUCCESS	TDG.MEM.PAGE.ATTR.RD is successful.

**5.4.4. NEW: TDG.MEM.PAGE.ATTR.WR Leaf**

Write the attributes of a private page. Create or remove L2 page aliases as required.

**Table 5.299: TDG.MEM.PAGE.ATTR.WR Input Operands Definition**

Operand	Name	Description		
RAX	Leaf and Version	TDCALL instruction leaf number and version, see 5.4.1		
		Bits	Field	Description
		15:0	Leaf Number	Selects the TDCALL interface function
		23:16	Version Number	Selects the TDCALL interface function version Must be 0
		63:24	Reserved	Must be 0
RCX	GPA_MAPPING	GPA mapping information:		
		Bits	Name	Description
		2:0	LEVEL	Level of the Secure EPT leaf entry that maps the private page: either 0 (4KB), 1 (2MB) or 2 (1GB) – see 3.6.1.
		11:3	RESERVED	Reserved: must be 0
		51:12	GPA	Bits 51:12 of the guest physical address of the private page Depending on the level, the following least significant bits must be 0: Level 0 (EPTE):       None Level 1 (EPDE):       Bits 20:12 Level 2 (EPDPTE):    Bits 29:12
		63:52	RESERVED	Reserved: must be 0
RDX	GPA_ATTR	Guest-visible page attributes. See the GPA_ATTR definition in 3.6.3. To avoid writing the attributes of a certain VM, all 16 attribute bits (GPA_ATTR_SINGLE_VM) for that VM should be set to 0. Attribute bits for non-existent VMs must be 0.		
R8	ATTR_FLAGS	Attributes masks and invalidate EPT flags		
		Bits	Field	Description
		15:0	RESERVED	Must be 0
		30:16	ATTR_MASK1	A bit value of 1 indicates that the applicable attributes bit is to be written. Otherwise, the attributes bit is unmodified. Must be 0 if the TD has no VM #1.
		31	INVEPT1	Invalidate EPT for L2 VM #1 Must be 0 if the TD has no VM #1.
		46:32	ATTR_MASK2	A bit value of 1 indicates that the applicable attributes bit is to be written. Otherwise, the attributes bit is unmodified. Must be 0 if the TD has no VM #2.
		47	INVEPT2	Invalidate EPT for L2 VM #2

Operand	Name	Description		
				Must be 0 if the TD has no VM #2.
		62:48	ATTR_MASK3	A bit value of 1 indicates that the applicable attributes bit is to be written. Otherwise, the attributes bit is unmodified. Must be 0 if the TD has no VM #3.
		63	INVEPT3	Invalidate EPT for L2 VM #3 Must be 0 if the TD has no VM #3.

Table 5.300: TDG.MEM.PAGE.ATTR.WR Output Operands Definition

Operand	Name	Description		
RAX	STATUS	TDCALL instruction return code – see 5.4.1		
RCX	GPA_MAPPING	Actual GPA mapping of the page:		
		<b>Bits</b>	<b>Name</b>	<b>Description</b>
		2:0	LEVEL	Level of the Secure EPT leaf entry that maps the private page: either 0 (4KB), 1 (2MB) or 2 (1GB) – see 3.6.1.
		11:3	RESERVED	Reserved: set to 0
		51:12	GPA	Bits 51:12 of the guest physical start address of the private page Depending on the level, the following least significant bits are always 0: Level 0 (EPTE):      None Level 1 (EPDE):      Bits 20:12 Level 2 (EPDPTE):    Bits 29:12
		61:52	RESERVED	Reserved: set to 0
		62	PENDING	Flags that the page is PENDING
		63	RESERVED	Reserved: set to 0
RDX	GPA_ATTR	On success, if the attribute bits (GPA_ATTR_SINGLE_VM) for a specific VM were 0 on input, they remain unmodified. For other VMs, RDX returns the updated guest-visible page attributes.  In case of an error, RDX returns the current value of page attributes when possible. If the current attributes for a certain VM have not been read, that VM's attributes VALID bit returns 0.  See the GPA_ATTR definition in 3.6.3.		
Other		Unmodified		

#### Leaf Function Description

- 5 **Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.MEM.PAGE.ATTR.WR writes the specified set of attributes of a TD private page, including L2 page alias attributes. Only the bits selected by the attributes mask are updated. The private page can be either writable by the TD (MAPPED or EXPORTED\_DIRTY) or pending acceptance (PENDING or PENDING\_EXPORTED\_DIRTY). If the page is pending

acceptance, the written attributes will become effective when the page is later accepted by the guest TD, using TDG.MEM.PAGE.ACCEPT.

TDH.MEM.PAGE.ATTR.WR ignores any VM's GPA attributes set whose VALID bit is 0.

TDH.MEM.PAGE.ATTR.WR creates or removes L2 page aliases as required:

- 5 • If any of the requested L2 attributes VALID bit is set, and the R, W, Xs, Xu and PWA bits combination has a legal, non-0 value, then if the L2 page alias does not exist, it is created. The rules for checking the legal combination of attributes bits are described in 3.6.3.1.
  - If any of the requested L2 attributes VALID bit is set, and the R, W, Xs, Xu and PWA bits are all 0, then if the L2 page alias exists, it is removed.
- 10 **Note:** For the above operations, the Xu and PWA bits are always considered, regardless of the L2 VMCS setting of the “mode-based execute control for EPT” (MBEC) and “EPT paging-write control” VM-execution controls.

### SEPT Mapping Size Considerations

In most cases, the guest TD is unaware of how TD private pages are mapped by the host VMM in SEPT. However, TDG.MEM.PAGE.ATTR.WR operation specifies a page mapping size and may fail if the specified size is different than the

- 15 actual mapping size.
- If the page is mapped at a lower level than requested, the function returns TDX\_PAGE\_SIZE\_MISMATCH. The guest may re-invoke TDG.MEM.PAGE.ATTR.WR specifying the actual mapping size as returned in RCX.
  - If the page is mapped at a higher level than requested, this results in an EPT violation TD exit, with extended exit qualification indicating the error SEPT entry level and state, and the guest-requested mapping level. The host VMM
- 20 is expected to demote the page, then re-enter the guest TD so TDG.MEM.PAGE.ATTR.WR is re-invoked.

### Other Conditions that Prevent Page Attributes Modifications

- If the page is not guest-writable and is not pending, this results in an EPT violation TD exit, indicating a failed write operation.
  - 25 • If an L2 SEPT walk fails, meaning there's a missing non-leaf L2 SEPT page, the operation depends on the setting of the host writable TDCS field VM\_CTLs, which is an array of 4 bitmaps, one per VM (only L2 VMs are applicable). Bit 0 controls the operation on L2 SEPT walk fails in TDCALL flows:
    - The default value of 0 means that a TDX\_L2\_SEPT\_WALK\_FAILED status is returned to the L1 VMM.
    - If the value is 1, the TDX module does an EPT violation TD exit, indicating a failed write operation exit, with extended exit qualification indicating the error L2 SEPT level and VM index. The host VMM may then add the
- 30 missing L2 SEPT page using TDH.MEM.SEPT.ADD.

In any of the above cases, the page attributes are not modified.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.301 TDG.MEM.PAGE.ATTR.WR Memory Operands Information Definition**

Explicit/ Implicit	Reg.	Addr. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions
Explicit	RCX	GPA	TD private page	Blob	RW	Private	$2^{12+9*Level}$ Bytes	None
Implicit	N/A	N/A	TDR page	TDR	None	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	Secure EPT tree	N/A	R	Private	N/A	None
Implicit	N/A	N/A	L2 Secure EPT tree	N/A	RW	Private	N/A	None
Implicit	N/A	GPA	Secure EPT entry	SEPT Entry	RW	Private	N/A	Exclusive(h)
Implicit	N/A	GPA	L2 Secure EPT entry	SEPT Entry	RW	Private	N/A	Exclusive(i)

TDG.MEM.PAGE.ATTR.WR checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

In addition to the memory operand checks per the table above, the function does the following (no specific order is implied):

- 5    1. Check that the requested GPA and level are valid.
2. Check that the requested page attributes and attributes mask operands are valid.
3. Walk the L1 Secure EPT based on the GPA operand and requested level.
  - 3.1. The walk is successful if arrived at a leaf entry at the request level, whose state is MAPPED, EXPORTED\_DIRTY, PENDING or PENDING\_EXPORTED\_DIRTY.
  - 10    3.2. Else, if the page is not mapped at the requested level, return a TDX\_PAGE\_SIZE\_MISMATCH status.
  - 3.3. Else, do an EPT violation TD exit, indicating a failed write operation.

If successful, do the following:

4. For each L2 VM:
  - 4.1. If an L2 page alias exists:
    - 15    4.1.1. Walk that L2 VM's SEPT and locate the page alias L2 SEPT leaf entry.
    - 4.1.2. If the GPA\_ATTR's VALID bit is 1, and RWXsXu bits combined value is a valid, non-0 value, update the L2 SEPT entry. If the page is pending, then the L2 SEPT entry's access permission bits remain 0; their original values are stored in their saved locations in the L2 SEPT entry.
    - 4.1.3. If the GPA\_ATTR's VALID bit is 1, and RWXsXu bits combined value is 0, remove the alias by setting the L2 SEPT entry state to L2\_FREE. Update the L1 SEPT entry to indicate a page alias exists to this page.
    - 20    4.2. Else (L2 page alias does not exist), if the GPA\_ATTR's VALID bit is 1, and RWXsXu bits combined value is a valid, non-0 value:
      - 4.2.1. Walk that L2 VM's SEPT and locate the page alias L2 SEPT leaf entry.
      - 4.2.2. If failed, do an EPT violation TD exit, indicating a failed write operation.
      - 25    If passed:
        - 4.2.3. Create the page alias by writing the SEPT entry with the page HPA and the requested set of attributes masked by the attributes mask and setting its state to L2\_MAPPED or L2\_BLOCKED, depending on the L1 SEPT entry state. If the page is pending, then the L2 SEPT entry's access permission bits remain 0; their original values are stored in their saved locations in the L2 SEPT entry.
        - 30    4.2.4. Update the L1 SEPT entry to indicate a page alias exists to this page.

## Completion Status Codes

**Table 5.302: TDG.MEM.PAGE.ATTR.WR Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_OPERAND_BUSY	Concurrent TDG.MEM.PAGE.ACCEPT is using the same Secure EPT entry
TDX_PAGE_ATTR_INVALID	The combination of page attributes to be set, after taking into account the requested attributes, the requested attributes mask and the current page attributes, is invalid.
TDX_PAGE_SIZE_MISMATCH	Requested page size does not match its GPA mapping size
TDX_SUCCESS	TDG.MEM.PAGE.ATTR.WR is successful.

#### 5.4.5. TDG.MR.REPORT Leaf

TDG.MR.REPORT creates a TDREPORT\_STRUCT structure that contains the measurements/configuration information of the guest TD that called the function, measurements/configuration information of the Intel TDX module and a REPORTMACSTRUCT.

**Table 5.303: TDG.MR.REPORT Input Operands Definition**

Operand	Description		
RAX	TDCALL instruction leaf number and version, see 5.4.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the TDCALL interface function
	23:16	Version Number	Selects the TDCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	1024B-aligned guest physical address of newly created report structure		
RDX	64B-aligned guest physical address of additional data to be signed		
R8	Bits	Name	Description
	7:0	Report sub type	Must be 0
	63:8	Reserved	Reserved: must be 0

**Table 5.304: TDG.MR.REPORT Output Operands Definition**

Operand	Description
RAX	TDCALL instruction return code – see 5.4.1
Other	Unmodified

#### Leaf Function Description

- Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

This function creates a TDREPORT\_STRUCT structure that contains the measurements/configuration information of the guest TD that called the function, measurements/configuration information of the Intel TDX module and a REPORTMACSTRUCT. The REPORTMACSTRUCT is integrity-protected with a MAC, and it contains the hash of the measurements and configuration as well as additional REPORTDATA provided by the TD software.

Additional REPORTDATA, a 64-byte value, is provided by the guest TD to be included in the TDG.MR.REPORT.

**Note:** Although not enforced by TDG.MR.REPORT, the guest TD should normally place REPORTDATA in private memory to help ensure secure report generation.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.305: TDG.MR.REPORT Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions
Explicit	RCX	GPA	Output report	TDREPORT_STRUCT	RW	Private/ Shared	1024B	None



Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions
Explicit	RDX	GPA	Input report data	REPORTDATA	R	Private/ Shared	64B	None
Implicit	N/A	N/A	TDR page	TDR	None	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDCS structure	TDCS	R	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDCS.RTMR	SHA384_HASH	N/A	Opaque	N/A	Shared
Implicit	N/A	N/A	TDVPS structure	TDVPS	None	Opaque	N/A	Shared(i)

In addition to the memory operand checks per the table above, the function checks the following conditions (no specific order is implied):

1. R8 must specify report sub type 0.
- 5 If passed, the function does the following:
  2. Assemble a report type structure based on the report sub type provided in R8.
  3. Assemble the output report's TDINFO fields from the TDCS reported fields (ATTRIBUTES, XFAM, MRTD, MRCONFIGID, MROWNER, MROWNERCONFIG and RTMRs).
  4. Calculate a SHA384 hash over TDINFO.
  - 10 5. If the CPU supports TD preserving updates:
    - 5.1. Execute SEAMDB\_REPORT to complete the output report, based on the input report data, the TDINFO hash calculated above, the report type structure and the SEAMDB entry's index/nonce pair of the TDR.
    - 5.2. If SEAMDB\_REPORT returns an error (unrecognized index/nonce pair), then mark the TD state as FATAL and do a TD exit with a TDX\_NON\_RECOVERABLE\_TD\_CORRUPTED\_MD status code.
  - 15 6. Else (the CPU does not support TD preserving updates):  
Execute SEAMREPORT to complete the output report, based on the input report data, the TDINFO hash calculated above and the report type structure.

If successful:

7. Write the output report to memory.

## 20 Completion Status Codes

**Table 5.306: TDG.MR.REPORT Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_SUCCESS	TDG.MR.REPORT is successful.

#### 5.4.6. TDG.MR.RTMR.EXTEND Leaf

Extend a TDCS.RTMR measurement register.

**Table 5.307: TDG.MR.RTMR.EXTEND Input Operands Definition**

Operand	Description		
RAX	TDCALL instruction leaf number and version, see 5.4.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the TDCALL interface function
	23:16	Version Number	Selects the TDCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	64B-aligned guest physical address of a 48B extension data buffer		
RDX	Index of the measurement register to be extended		

**Table 5.308: TDG.MR.RTMR.EXTEND Output Operands Definition**

Operand	Description
RAX	TDCALL instruction return code – see 5.4.1
Other	Unmodified

#### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 This function extends one of the RTMR measurement registers in TDCS with the provided extension data in memory. To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.309 TDG.MR.RTMR.EXTEND Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions
Explicit	RCX	GPA	EXTEND_DATA	Blob	R	Private	64B	None
Implicit	N/A	N/A	TDR page	TDR	None	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDCS.RTMR	SHA384_HASH	N/A	Opaque	N/A	Exclusive
Implicit	N/A	N/A	TDVPR page	TDVPS	None	Opaque	N/A	Shared(i)

- 15 In addition to the memory operand checks per the table above, the function checks the following conditions (no specific order is implied):
1. RDX must contain a valid RTMR index.

If successful, the function does the following:

2. Extend the RTMR indexed by RDX with the extension data. Extension is done by calculating SHA384 hash over a 96B buffer, composed as follows:
  - Bytes 0 through 47 contain the current RTMR value.
  - Bytes 48 through 95 contain the extension data.

#### Completion Status Codes

**Table 5.310: TDG.MR.RTMR.EXTEND Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_SUCCESS	TDG.MR.RTMR.EXTEND is successful.

DRAFT

### 5.4.7. TDG.MR.VERIFYREPORT

Verify a cryptographic REPORTMACSTRUCT that describes the contents of a TD, to determine that it was created on the current TEE on the current platform.

**Table 5.311: TDG.MR.VERIFYREPORT Input Operands Definition**

Operand	Description		
RAX	TDCALL instruction leaf number and version, see 5.4.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the TDCALL interface function
	23:16	Version Number	Selects the TDCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	256B-aligned guest physical address of the REPORTMACSTRUCT to be verified.		

**Table 5.312: TDG.MR.VERIFYREPORT Output Operands Definition**

Operand	Description
RAX	TDCALL instruction return code – see 5.4.1
Other	Unmodified

#### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDG.MR.VERIFYREPORT computes a MAC over the provided REPORTMACSTRUCT structure; it then checks that the computed value is the same as the MAC field of that structure.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.313: TDG.MR.VERIFYREPORT Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions
Explicit	RCX	GPA	Input report	REPORTMACSTRUCT	RW	Private	256B	None
Implicit	N/A	N/A	TDR page	TDR	None	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDCS structure	TDCS	R	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDCS.RTMR	SHA384_HASH	N/A	Opaque	N/A	Shared
Implicit	N/A	N/A	TDVPS structure	TDVPS	None	Opaque	N/A	Shared(i)

The function performs the memory operand checks per the table above.

If passed, the function does the following:

1. Calculate MAC over the input REPORTMACSTRUCT fields that are included in the MAC calculation.
2. Compare the calculated MAC to the REPORTMACSTRUCT.MAC field and return a proper status.

**Completion Status Codes****Table 5.314: TDG.MR.VERIFYREPORT Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_INVALID_CPUSVN	
TDX_INVALID_REPORTMACSTRUCT	
TDX_OPERAND_INVALID	
TDX_SUCCESS	TDG.MR.VERIFYREPORT is successful.

DRAFT

**5.4.8. NEW: TDG.SERVTD.RD Leaf**

As a service TD, read a metadata field (control structure field) of a target TD.

**Table 5.315: TDG.SERVTD.RD Input Operands Definition**

Operand	Description		
RAX	TDCALL instruction leaf number and version, see 5.4.1		
	<b>Bits</b>	<b>Field</b>	<b>Description</b>
	15:0	Leaf Number	Selects the TDCALL interface function
	23:16	Version Number	Selects the TDCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	Binding handle		
RDX	Field identifier – see 3.10 The LAST_ELEMENT_IN_FIELD and LAST_FIELD_IN_SEQUENCE components of the field identifier must be 0. WRITE_MASK_VALID, INC_SIZE, CONTEXT_CODE and ELEMENT_SIZE_CODE components of the field identifier are ignored. A value of -1 is a special case: it is not a valid field identifier; in this case the first readable field identifier is returned in RDX.		
R10	Target TD's TD_UUID bits 63:0		
R11	Target TD's TD_UUID bits 127:64		
R12	Target TD's TD_UUID bits 191:128		
R13	Target TD's TD_UUID bits 255:192		

5

**Table 5.316: TDG.SERVTD.RD Output Operands Definition**

Operand	Description
RAX	TDCALL instruction return code – see 5.4.1
RDX	RDX returns the next readable field identifier. A value of -1 indicates no next field identifier is available. If the input field identifier was -1, RDX returns the first readable field identifier. In case of another error, RDX returns -1.
R8	Contents of the field In case of an error, as indicated by RAX, R8 returns 0.
R10	Updated target TD's TD_UUID bits 63:0 – see the description below.
R11	Updated target TD's TD_UUID bits 127:64 – see the description below.
R12	Updated target TD's TD_UUID bits 191:128 – see the description below.
R13	Updated target TD's TD_UUID bits 255:192 – see the description below.
Other	Unmodified

## Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDG.SERVTD.RD reads a metadata field (control structure field) of a target TD.

- 5 **TD\_UUID Update:** TD\_UUID is updated when the target TD is imported. If the service TD binding to the target TD happened before the target TD was imported, the TD\_UUID provided in R13:R10 may no longer be correct. In this case, if the TD\_UUID provided in R13:R10 is equal to the pre-import TD\_UUID of the target TD, TDG.SERVTD.RD returns TDX\_TARGET\_UUID\_MISMATCH status in RAX, and updates R13:R10 with the imported value of TD\_UUID. The caller should retry the operation with the new TD\_UUID.

10

**Cross-TD Traps:** Failure to access the metadata of the target TD may result in a cross-TD trap TD exit to the host VMM. This TD exit is trap like, meaning it happens after TDG.SERVTD.RD has completed its operation. On the following TDH.VP.ENTER, the host VMM may set a HOST\_RECOVERABILITY\_HINT flag, indicating that TDG.SERVTD.RD may be retried. From the guest TD's perspective, this flag appears in bit 60 of the status code returned in RAX. See the [TDX Module Base Spec] for details.

15

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.317 TDG.SERVTD.RD Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	Target TD's TDR page (from binding handle)	TDR	None	Opaque	N/A	Shared(h)	Shared(h)	Shared(h)
Implicit	N/A	N/A	Service (this) TD's TDR page	TDR	None	Opaque	N/A	Shared(i)	None	None
Implicit	N/A	N/A	Service (this) TD's TDCS structure	TDCS	R	Opaque	N/A	Shared(i)	None	None
Implicit	N/A	N/A	Service (this) TD's TDCS.RTMR	SHA384_HASH	N/A	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	N/A	Service (this) TD's TDVPS structure	TDVPS	None	Opaque	N/A	Shared(i)	Shared(i)	Shared(i)
Implicit	N/A	N/A	Target TD's TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	None	None
Implicit	N/A	N/A	Target TD's TDCS.OP_STATE	OP_STATE	RW	Opaque	N/A	Shared(h)	N/A	N/A
Implicit	N/A	N/A	Target TD's Binding table		R	Opaque	N/A	Shared(h)	None	None
Implicit	N/A	N/A	Target TD's TD metadata	N/A	R	Opaque	N/A	None	None	None

- 20 If the memory operand checks per the table above pass:

1. Based on the provided binding handle and the current (service) TD's TD\_UUID, calculate the target TD's TDR HPA and binding slot number.

2. Check that the calculated binding slot number does not exceed target TD's the number of available slots<sup>9</sup>.
3. Acquire access to the target TD's TDR in a shared mode.
  - 3.1. If failed due to HOST\_PRIORITY, do a TD exit.
4. Check the target TD state:
  - 4.1. The target TD's TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
  - 4.2. The target TD is not in a FATAL state (TDR.FATAL is FALSE).
  - 4.3. The target TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
  - 4.4. The target TD's TDCS pages must have been allocated (TDR.NUM\_TDCX is the required number).

If passed:

5. Check that the target TD's TD\_UUID is the same as specified.
  - 5.1. If failed, and the target TD's PRE\_IMPORT\_UUID is the same as the specified TD\_UUID, abort and return the current target TD's TD\_UUID.

If passed:

6. Check that the target TD's binding slot's SERVTD\_BINDING\_STATE is BOUND.
7. Calculate the current (service) TD's TD\_UUID and check it is equal to the target TD's binding slot's SERVTD\_UUID.
8. Calculate the current (service) TD's TDINFO\_HASH and check it is equal to the target TD's binding slot's SERVTD\_TDINFO\_HASH.

If passed:

9. Read the control structure field using the algorithm described in 5.2.2.1.

## Completion Status Codes

**Table 5.318: TDG.SERVTD.RD Completion Status Codes (Returned in RAX) Definition** [TO BE COMPLETED]

Completion Status Code	Description
TDX_METADATA_FIELD_ID_INCORRECT	
TDX_METADATA_FIELD_NOT_READABLE	
TDX_METADATA_FIELD_VALUE_NOT_VALID	
TDX_OP_STATE_INCORRECT	This status code refers to the target TD. Bit 60 (HOST_RECOVERABILITY_HINT) contains a hint from the host VMM that the error condition has been resolved and the service TD can retry the operation.
TDX_OPERAND_ADDR_RANGE_ERROR	This status code refers to the target TD. Bit 60 (HOST_RECOVERABILITY_HINT) contains a hint from the host VMM that the error condition has been resolved and the service TD can retry the operation.
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_PAGE_METADATA_INCORRECT	This status code refers to the target TD. Bit 60 (HOST_RECOVERABILITY_HINT) contains a hint from the host VMM that the error condition has been resolved and the service TD can retry the operation.

<sup>9</sup> This value is a property of the TDX module and is the same for all TDs.



Completion Status Code	Description
TDX_SERVTD_INFO_HASH_MISMATCH	This service TD's info hash doesn't match the service TD info hash in the target TD's binding information. Bit 60 (HOST_RECOVERABILITY_HINT) contains a hint from the host VMM that the error condition has been resolved and the service TD can retry the operation.
TDX_SERVTD_NOT_BOUND	This service TD is not bound to the target TD. Bit 60 (HOST_RECOVERABILITY_HINT) contains a hint from the host VMM that the error condition has been resolved and the service TD can retry the operation.
TDX_SERVTD_UUID_MISMATCH	This service TD's TD_UUID doesn't match the service TD UUID in the target TD's binding information. Bit 60 (HOST_RECOVERABILITY_HINT) contains a hint from the host VMM that the error condition has been resolved and the service TD can retry the operation.
TDX_SUCCESS	TDG.SERVTD.RD is successful.
TDX_TARGET_UUID_MISMATCH	The target TD's TD_UUID value provided in R13:R10 doesn't match the actual value.
TDX_TARGET_UUID_UPDATED	The target TD's TD_UUID value provided in R13:R10 doesn't match the current actual value, but it does match the TD_UUID that target TD had before it was imported. In this case, the current TD_UUID value is provided in R13:R10, and the operation can be retried.
TDX_TD_FATAL	This status code refers to the target TD. Bit 60 (HOST_RECOVERABILITY_HINT) contains a hint from the host VMM that the error condition has been resolved and the service TD can retry the operation.
TDX_TD_KEYS_NOT_CONFIGURED	This status code refers to the target TD. Bit 60 (HOST_RECOVERABILITY_HINT) contains a hint from the host VMM that the error condition has been resolved and the service TD can retry the operation.
TDX_TDCS_NOT_ALLOCATED	This status code refers to the target TD. Bit 60 (HOST_RECOVERABILITY_HINT) contains a hint from the host VMM that the error condition has been resolved and the service TD can retry the operation.

Completion Status Code	Description
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_SUCCESS	TDG.SERVTD.RD is successful.

**5.4.9. NEW: TDG.SERVTD.WR Leaf**

As a service TD, write a metadata field (control structure field) of a target TD.

**Table 5.319: TDG.SERVTD.WR Input Operands Definition**

Operand	Description		
RAX	TDCALL instruction leaf number and version, see 5.4.1		
	<b>Bits</b>	<b>Field</b>	<b>Description</b>
	15:0	Leaf Number	Selects the TDCALL interface function
	23:16	Version Number	Selects the TDCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	Binding handle		
RDX	Field identifier – see 3.10 The LAST_ELEMENT_IN_FIELD and LAST_FIELD_IN_SEQUENCE components of the field identifier must be 0. WRITE_MASK_VALID, INC_SIZE, CONTEXT_CODE and ELEMENT_SIZE_CODE components of the field identifier are ignored. A value of -1 is a special case: it is not a valid field identifier; in this case the first readable field identifier is returned in RDX.		
R8	Data to write to the field		
R9	A 64b write mask to indicate which bits of the value in R8 are to be written to the field		
R10	Target TD's TD_UUID bits 63:0		
R11	Target TD's TD_UUID bits 127:64		
R12	Target TD's TD_UUID bits 191:128		
R13	Target TD's TD_UUID bits 255:192		

**Table 5.320: TDG.SERVTD.WR Output Operands Definition**

Operand	Description
RAX	TDCALL instruction return code – see 5.4.1
R8	Previous contents of the field In case of an error, R8 returns 0.
R10	Updated target TD's TD_UUID bits 63:0 – see the description below.
R11	Updated target TD's TD_UUID bits 127:64 – see the description below.
R12	Updated target TD's TD_UUID bits 191:128 – see the description below.
R13	Updated target TD's TD_UUID bits 255:192 – see the description below.

## Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDG.SERVTD.WR writes a metadata field (control structure field) of a target TD.

- 5 **TD\_UUID Update:** TD\_UUID is updated when the target TD is imported. If the service TD binding to the target TD happened before the target TD was imported, the TD\_UUID provided in R13:R10 may no longer be correct. In this case, if the TD\_UUID provided in R13:R10 is equal to the pre-import TD\_UUID of the target TD, TDG.SERVTD.WR returns TDX\_TARGET\_UUID\_MISMATCH status in RAX, and updates R13:R10 with the imported value of TD\_UUID. The caller should retry the operation with the new TD\_UUID.

10

**Cross-TD Traps:** Failure to access the metadata of the target TD may result in a cross-TD trap TD exit to the host VMM. This TD exit is trap like, meaning it happens after TDG.SERVTD.WR has completed its operation. On the following TDH.VP.ENTER, the host VMM may set a HOST\_RECOVERABILITY\_HINT flag, indicating that TDG.SERVTD.WR may be retried. From the guest TD's perspective, this flag appears in bit 60 of the status code returned in RAX. See the [TDX Module Base Spec] for details.

15

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.321 TDG.SERVTD.WR Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	HPA	Target TD's TDR page (from binding handle)	TDR	None	Opaque	N/A	Shared(h)	Shared(h)	Shared(h)
Implicit	N/A	N/A	Service (this) TD's TDR page	TDR	None	Opaque	N/A	Shared(i)	None	None
Implicit	N/A	N/A	Service (this) TD's TDCS structure	TDCS	R	Opaque	N/A	Shared(i)	None	None
Implicit	N/A	N/A	Service (this) TD's TDCS.RTMR	SHA384_HASH	N/A	Opaque	N/A	Shared	N/A	N/A
Implicit	N/A	N/A	Service (this) TD's TDVPS structure	TDVPS	None	Opaque	N/A	Shared(i)	Shared(i)	Shared(i)
Implicit	N/A	N/A	Target TD's TDCS structure	TDCS	RW	Opaque	N/A	Shared(i)	None	None
Implicit	N/A	N/A	Service (this) TD's TDCS.OP_STATE	OP_STATE	R	Opaque	N/A	Shared(h)	N/A	N/A
Implicit	N/A	N/A	Target TD's Binding table		R	Opaque	N/A	Shared(h)	None	None
Implicit	N/A	N/A	Target TD's TD metadata	N/A	RW	Opaque	N/A	None	None	None

- 20 If the memory operand checks per the table above pass:

1. Based on the provided binding handle and the current (service) TD's TD\_UUID, calculate the target TD's TDR HPA and binding slot number.

2. Check that the calculated binding slot number does not exceed target TD's the number of available slots<sup>10</sup>.
3. Acquire access to the target TD's TDR in a shared mode.
  - 3.1. If failed due to HOST\_PRIORITY, do a TD exit.
4. Check the target TD state:
  - 4.1. The target TD's TDR page metadata in PAMT must be correct (PT must be PT\_TDR).
  - 4.2. The target TD is not in a FATAL state (TDR.FATAL is FALSE).
  - 4.3. The target TD keys are configured on the hardware (TDR.LIFECYCLE\_STATE is TD\_KEYS\_CONFIGURED).
  - 4.4. The target TD's TDCS pages must have been allocated (TDR.NUM\_TDCX is the required number).
  - 4.5. The target TD has not been paused for export.
- 10 If passed:
  5. Check that the target TD's TD\_UUID is the same as specified.
    - 5.1. If failed, and the target TD's PRE\_IMPORT\_UUID is the same as the specified TD\_UUID, abort and return the current target TD's TD\_UUID.
 If passed:
- 15 6. Check that the target TD's binding slot's SERVTD\_BINDING\_STATE is BOUND.
7. Calculate the current (service) TD's TD\_UUID and check it is equal to the target TD's binding slot's SERVTD\_UUID.
8. Calculate the current (service) TD's TDINFO\_HASH and check it is equal to the target TD's binding slot's SERVTD\_TDINFO\_HASH.
- If passed:
- 20 9. Write the control structure field and return its old value, using the algorithm described in 5.2.2.2.

### Completion Status Codes

**Table 5.322: TDG.SERVTD.WR Completion Status Codes (Returned in RAX) Definition [TO BE COMPLETED]**

Completion Status Code	Description
TDX_METADATA_FIELD_ID_INCORRECT	
TDX_METADATA_FIELD_NOT_WRITABLE	
TDX_METADATA_FIELD_VALUE_NOT_VALID	
TDX_OP_STATE_INCORRECT	This status code refers to the target TD. Bit 60 (HOST_RECOVERABILITY_HINT) contains a hint from the host VMM that the error condition has been resolved and the service TD can retry the operation.
TDX_OPERAND_ADDR_RANGE_ERROR	This status code refers to the target TD. Bit 60 (HOST_RECOVERABILITY_HINT) contains a hint from the host VMM that the error condition has been resolved and the service TD can retry the operation.
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_PAGE_METADATA_INCORRECT	This status code refers to the target TD. Bit 60 (HOST_RECOVERABILITY_HINT) contains a hint from the host VMM that the error condition has been resolved and the service TD can retry the operation.

<sup>10</sup> This value is a property of the TDX module and is the same for all TDs.

Completion Status Code	Description
TDX_SERVTD_INFO_HASH_MISMATCH	This service TD's info hash doesn't match the service TD info hash in the target TD's binding information. Bit 60 (HOST_RECOVERABILITY_HINT) contains a hint from the host VMM that the error condition has been resolved and the service TD can retry the operation.
TDX_SERVTD_NOT_BOUND	This service TD is not bound to the target TD. Bit 60 (HOST_RECOVERABILITY_HINT) contains a hint from the host VMM that the error condition has been resolved and the service TD can retry the operation.
TDX_SERVTD_UUID_MISMATCH	This service TD's TD_UUID doesn't match the service TD UUID in the target TD's binding information. Bit 60 (HOST_RECOVERABILITY_HINT) contains a hint from the host VMM that the error condition has been resolved and the service TD can retry the operation.
TDX_SUCCESS	TDG.SERVTD.WR is successful.
TDX_TARGET_UUID_MISMATCH	The target TD's TD_UUID value provided in R13:R10 doesn't match the actual value.
TDX_TARGET_UUID_UPDATED	The target TD's TD_UUID value provided in R13:R10 doesn't match the current actual value, but it does match the TD_UUID that target TD had before it was imported. In this case, the current TD_UUID value is provided in R13:R10, and the operation can be retried.
TDX_TD_FATAL	This status code refers to the target TD. Bit 60 (HOST_RECOVERABILITY_HINT) contains a hint from the host VMM that the error condition has been resolved and the service TD can retry the operation.
TDX_TD_KEYS_NOT_CONFIGURED	This status code refers to the target TD. Bit 60 (HOST_RECOVERABILITY_HINT) contains a hint from the host VMM that the error condition has been resolved and the service TD can retry the operation.
TDX_TDCS_NOT_ALLOCATED	This status code refers to the target TD. Bit 60 (HOST_RECOVERABILITY_HINT) contains a hint from the host VMM that the error condition has been resolved and the service TD can retry the operation.

**5.4.10. NEW: TDG.SYS.RD Leaf**

Read a TDX Module global-scope metadata field.

**Table 5.323: TDG.SYS.RD Input Operands Definition**

Operand	Description		
RAX	TDCALL instruction leaf number and version, see 5.4.1		
	<b>Bits</b>	<b>Field</b>	<b>Description</b>
	15:0	Leaf Number	Selects the TDCALL interface function
	23:16	Version Number	Selects the TDCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RDX	Field identifier – see 3.10 The LAST_ELEMENT_IN_FIELD and LAST_FIELD_IN_SEQUENCE components of the field identifier must be 0. WRITE_MASK_VALID, INC_SIZE, CONTEXT_CODE and ELEMENT_SIZE_CODE components of the field identifier are ignored. A value of -1 is a special case: it is not a valid field identifier; in this case the first readable field identifier is returned in RDX.		

**Table 5.324: TDG.SYS.RD Output Operands Definition**

Operand	Description
RAX	TDCALL instruction return code – see 5.4.1
RDX	If the input field identifier was -1, RDX returns the first readable field identifier. Else, in there case of an error, RDX returns -1. On success, RDX returns the next readable field identifier. A value of -1 indicates no next field identifier is available.
R8	Contents of the field In case of no success, as indicated by RAX, R8 returns 0.
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDG.SYS.RD reads a TDX Module global-scope metadata field.

RDX returns the next guest-side readable field identifier. This may be used by the guest TD to enumerate the TDX Module's capabilities and configuration. To read all the available fields, the guest TD can invoke TDG.SYS.RD in a loop, starting with field identifier -1 as an input, until RDX returns -1. Alternatively, the guest TD can use TDG.SYS.RDALL.

- 15 To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

Table 5.325 TDG.SYS.RD Operands Information Definition

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions
Implicit	N/A	N/A	TDR page	TDR	None	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDCS structure	TDCS	R	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDVPS structure	TDVPS	None	Opaque	N/A	Shared(i)

1. Read the requested field using the algorithm described in 5.2.2.1.
2. Return the next readable field identifier, or a value of 0 if none exists.
3. Return the field value.

5

### Completion Status Codes

Table 5.326: TDG.SYS.RD Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_METADATA_FIRST_FIELD_ID_IN_CONTEXT	Indicates that the first field ID in context is returned
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_SUCCESS	TDG.SYS.RD is successful.

DRAFT

**5.4.11. NEW: TDG.SYS.RDALL Leaf**

Read all guest-readable TDX module global-scope metadata fields.

**Table 5.327: TDG.SYS.RDALL Input Operands Definition**

Operand	Description		
RAX	TDCALL instruction leaf number and version, see 5.4.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the TDCALL interface function
	23:16	Version Number	Selects the TDCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RDX	The GPA of a 4KB buffer where a metadata list will be returned The buffer must be aligned on 4KB. In case of error, some field value entries might not contain valid data.		
R8	Initial field identifier – see 3.10 If R8's value is -1, then TDG.SYS.RDALL will start from the first global-scope metadata field identifier. Else, LAST_ELEMENT_IN_FIELD, LAST_FIELD_IN_SEQUENCE, WRITE_MASK_VALID and CONTEXT_CODE are ignored.		

**Table 5.328: TDG.SYS.RDALL Output Operands Definition**

Operand	Description
RAX	TDCALL instruction return code – see 5.4.1
R8	Next field identifier. A value of -1 means all applicable field identifiers have been returned in the metadata list. In case of an error, as indicated by RAX, R8 returns -1.
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDG.SYS.RDALL reads all host-readable TDX Module global-scope metadata fields into a metadata list in the provided page.

If one or more applicable fields do not fit in the provided list buffer, the function can be invoked in a loop, each invocation providing an initial field identifier returned as the next field identifier of the previous invocation, as shown in the following example:

- 15 1. NEXT\_FIELD\_ID = -1  
 2. Repeat:  
   2.1. Set LIST\_BUFFER to the next 4K buffer  
   2.2. Invoke TDG.SYS.RDALL(RDX = LIST\_BUFFER, RDX = NEXT\_FIELD\_ID)  
   2.3. STATUS = RAX, NEXT\_FIELD\_ID = R8  
 20 Until ((STATUS is a non-recoverable error) or (NEXT\_FIELD\_ID is -1))

The function never returns an empty list if there's no error.



To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.329: TDG.SYS.RDALL Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions
Explicit	RDX	GPA	Metadata List	MD_LIST	RW	Private	4096	None
Implicit	N/A	N/A	TDR page	TDR	None	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDCS structure	TDCS	R	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDVPS structure	TDVPS	None	Opaque	N/A	Shared(i)

- 5 If the memory operand checks per the table above pass:
  1. Dump all guest-readable metadata fields into the provided list buffer.

#### Completion Status Codes

**Table 5.330: TDG.SYS.RDALL Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_SUCCESS	TDG.SYS.RDALL is successful.

**5.4.12. UPDATED: TDG.VM.RD Leaf**

Read a TD-scope metadata field (control structure field) of a TD.

**Table 5.331: TDG.VM.RD Input Operands Definition**

Operand	Description		
RAX	TDCALL instruction leaf number and version, see 5.4.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the TDCALL interface function
	23:16	Version Number	Selects the TDCALL interface function version May be 0 or 1
	63:24	Reserved	Must be 0
RCX	Reserved, must be 0		
RDX	Field identifier – see 3.10 The <code>LAST_ELEMENT_IN_FIELD</code> and <code>LAST_FIELD_IN_SEQUENCE</code> components of the field identifier must be 0. <code>WRITE_MASK_VALID</code> , <code>INC_SIZE</code> , <code>CONTEXT_CODE</code> and <code>ELEMENT_SIZE_CODE</code> components of the field identifier are ignored. For TDG.VM.RD version 1 or higher, a value of -1 is a special case: it is not a valid field identifier; in this case the first readable field identifier is returned in RDX.		

**Table 5.332: TDG.VM.RD Output Operands Definition**

Operand	Description
RAX	TDCALL instruction return code – see 5.4.1
RDX	For TDG.VM.RD version 0, RDX is unmodified. For TDG.VM.RD version 1 or higher: <ul style="list-style-type: none"> <li>If the input field identifier was -1, RDX returns the first readable field identifier.</li> <li>Else, in there case of an error, RDX returns -1. On success, RDX returns the next readable field identifier. A value of -1 indicates no next field identifier is available.</li> </ul>
R8	Contents of the field In case of no success, as indicated by RAX, R8 returns 0.
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDG.VM.RD reads a VM-scope metadata field (control structure field) of a TD.

If version 1 or higher is specified in RAX, RDX returns the next host-side readable field identifier. This may be used by the guest TD to dump the guest readable TD metadata. To read all the available fields, the guest TD can invoke TDG.VM.RD in a loop, starting with field identifier -1 as an input, until RDX returns -1.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.333 TDG.VM.RD Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions
Implicit	N/A	N/A	TDR page	TDR	None	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDCS structure	TDCS	R	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDVPS structure	TDVPS	None	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TD metadata (guest-side access)	N/A	R	Opaque	N/A	Shared

- 5 If the memory operand checks per the table above pass:
10. Read the control structure field using the algorithm described in 5.2.2.1.

#### Completion Status Codes

**Table 5.334: TDG.VM.RD Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_METADATA_FIRST_FIELD_ID_IN_CONTEXT	Indicates that the first field ID in context is returned
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_SUCCESS	TDG.VM.RD is successful.

**5.4.13. UPDATED: TDG.VM.WR Leaf**

Write a TD-scope metadata field (control structure field) of a TD.

**Table 5.335: TDG.VM.WR Input Operands Definition**

Operand	Description		
RAX	TDCALL instruction leaf number and version, see 5.4.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the TDCALL interface function
	23:16	Version Number	Selects the TDCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	Reserved, must be 0		
RDX	Field identifier – see 3.10  The <code>LAST_ELEMENT_IN_FIELD</code> and <code>LAST_FIELD_IN_SEQUENCE</code> components of the field identifier must be 0.  <code>WRITE_MASK_VALID</code> , <code>INC_SIZE</code> , <code>CONTEXT_CODE</code> and <code>ELEMENT_SIZE_CODE</code> components of the field identifier are ignored.		
R8	Data to write to the field		
R9	A 64b write mask to indicate which bits of the value in R8 are to be written to the field		

**Table 5.336: TDG.VM.WR Output Operands Definition**

Operand	Description
RAX	TDCALL instruction return code – see 5.4.1
R8	Previous contents of the field In case of an error, as indicated by RAX, R8 returns 0.
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDG.VM.WR writes a VM-scope metadata field (control structure field) of a TD.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.337 TDG.VM.WR Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions
Implicit	N/A	N/A	TDR page	TDR	None	Opaque	N/A	Shared
Implicit	N/A	N/A	TDCS structure	TDCS	R	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDVPS structure	TDVPS	None	Opaque	N/A	Shared(i)

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions
Implicit	N/A	N/A	TD metadata (guest-side access)	N/A	R	Opaque	N/A	Shared

If the memory operand checks per the table above pass:

1. Write the control structure field and return its old value, using the algorithm described in 5.2.2.2.

#### Completion Status Codes

**Table 5.338: TDG.VM.WR Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_SUCCESS	TDG.VM.WR is successful.

DRAFT

**5.4.14. UPDATED: TDG.VP.CPUIDVE.SET Leaf**

TDG.VP.CPUIDVE.SET controls unconditional #VE on CPUID execution by the guest TD.

**Note:** TDG.VP.CPUIDVE.SET is provided for backward compatibility. The guest TD may control the same settings by writing to the VCPU-scope metadata fields CPUID\_SUPERVISOR\_VE and CPUID\_USER\_VE using TDG.VP.WR.

**Table 5.339: TDG.VP.CPUIDVE.SET Input Operands Definition**

Operand	Description		
RAX	TDCALL instruction leaf number and version, see 5.4.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the TDCALL interface function
	23:16	Version Number	Selects the TDCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	Controls whether CPUID executed by the guest TD will cause #VE unconditionally		
	Bits	Name	Description
	0	SUPERVISOR	Flags that when CPL is 0, a CPUID executed by the guest TD will cause a #VE unconditionally
	1	USER	Flags that when CPL > 0, a CPUID executed by the guest TD will cause a #VE unconditionally
	63:2	RESERVED	Reserved: must be 0

**Table 5.340: TDG.VP.CPUIDVE.SET Output Operands Definition**

Operand	Description
RAX	TDCALL instruction return code – see 5.4.1
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

This function controls whether execution of CPUID by the guest TD, when running in supervisor mode and/or in user mode, will unconditionally result in a #VE.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.341 TDG.VP.CPUIDVE.SET Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions
Implicit	N/A	N/A	TDR page	TDR	None	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDCS structure	TDCS	R	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDVPS structure	TDVPS	RW	Opaque	N/A	Shared(i)

In addition to the memory operand checks per the table above, the function checks the following conditions (no specific order is implied):

1. Reserved bits of RCX must be 0.

5 If successful, the function does the following:

2. Update the TDVPS.CPUID\_VE flags which control unconditional #VE injection for CPUID for the current VCPU.

#### Completion Status Codes

**Table 5.342: TDG.VP.CPUIDVE.SET Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_SUCCESS	TDG.VP.CPUIDVE.SET is successful.

10

DRAFT

#### 5.4.15. NEW: TDG.VP.ENTER Leaf

Enter L2 VCPU operation.

From the L1 VMM's point of view, TDG.VP.ENTER is a complex operation that normally involves L1→L2 VM entry and L2→L1 VM exit, but may fail before L2 VM entry and may also involve TD exits and entries. Therefore, output operands

5 are specified by multiple tables below.

**Table 5.343: TDG.VP.ENTER Input Operands Definition**

Operand	Name	Description		
RAX	Leaf and Version	TDCALL instruction leaf number and version, see 5.4.1		
		Bits	Field	Description
		15:0	Leaf Number	Selects the TDCALL interface function
		23:16	Version Number	Selects the TDCALL interface function version Must be 0
		63:24	Reserved	Must be 0
RCX	VM_FLAGS	VM identifier and flags		
		Bits	Name	Description
		0	DO_INVEPT	Indicates that TDG.VP.ENTER should flush the TLB context and extended paging structure (EPxE) caches associated with the L2 VM, using INVEPT single-context invalidation (type 1), before entering the L2 VCPU
		51:1	Reserved	Reserved: must be 0
		53:52	VM	L2 virtual machine index (must be 1 or higher)
		63:54	Reserved	Reserved: must be 0
RDX	GUEST_STATE_GPA	The GPA of a 256-bytes aligned L2_ENTER_GUEST_STATE structure - see 3.8.1 for details.		

TDG.VP.ENTER output format depends on how the function was terminated.

The following table details TDG.VP.ENTER output operands when the interface function returns **without entering the L2 VCPU** due to an error or some other condition.

10

**Table 5.344: TDG.VP.ENTER Output Operands Definition on No L2 VM Entry**

Operand	Name	Description		
RAX	Status	SEAMCALL instruction return code		
		Bit(s)	Name	Description
		47:32	CLASS and DETAILS_L1	May have the following values: <ul style="list-style-type: none"> <li>• TDX_PENDING_INTERRUPT, indicating that an interrupt is pending for L1</li> <li>• Any other value not in the table below</li> </ul>
		Other		See the function completion status definition in 3.1
Other		Unmodified		



The following table details TDG.VP.ENTER output operands when L2 VM entry succeeds, and later an L2 VM exit occurs due to a **VMX architectural exit reason**.

**Table 5.345: TDG.VP.ENTER Output Operands Definition on an L2→L1 Exits Following an L1→L2 Entry**

Operand	Name	Description		
RAX	Status	SEAMCALL instruction return code		
		Bit(s)	Name	Description
		31:0	DETAILS_L2: Exit Reason	L2 VMCS exit reason
		47:32	CLASS and DETAILS_L1	May have the following values: <ul style="list-style-type: none"><li>TDX_SUCCESS, indicating a normal L2→L1 exit</li><li>TDX_L2_EXIT_PENDING_INTERRUPT, indicating an L2→L1 exit due to an interrupt posted to L1</li><li>TDX_L2_EXIT_HOST_ROUTED_*, indicating a TD exit from L2 where the host VMM requested resumption of L1</li></ul> Other values in the range 0x1100 through 0x111F are reserved for future additional status codes that indicate an L2→L1 exit following an L1→L2 entry.
		Other		See the function completion status definition in 3.1
RCX	Exit Qualification	exit qualification from L2 VMCS		
RDX	Guest Linear Address	guest-linear address from L2 VMCS		
RSI	CS Info	CS selector, AR and limit		
		Bits	Details	
		15:0	CS Selector	
		31:16	CS AR bit 15:0	
		63:32	CS Limit	
RDI	CS Base	CS base address		
R8	Guest Physical Address	guest-physical address from L2 VMCS		
R9	VM-Exit Interruption Information	The following information is provided for L2 VM exits due to vectored events. In other cases, R9 content should be ignored.		
		Bits	Details	
		31:0	VM-Exit Interruption Information	
		63:32	VM-Exit Interruption Error Code	
R10	IDT-Vectoring Information	The following information is provided for L2 VM exits that occur during event delivery. In other cases, R10 content should be ignored.		
		Bits	Details	
		31:0	IDT-Vectoring Information	
		63:32	IDT-Vectoring Error Code	

Operand	Name	Description	
R11	VM-Exit Instruction Information	The following information is provided for L2 VM exits due to instruction execution. In other cases, R11 content should be ignored.	
		Bits	Details
		31:0	VM-Exit Instruction Information
		63:32	VM-Exit Instruction Length
R12	Additional Exit Information	Additional exit information	
		Bits	Details
		1:0	CPL (from GuestSS.AR.DPL)
		63:2	Reserved, cleared to 0
RBX, R13 – R15	None	Cleared to 0	
Other state	Any state that the L2 VM is allowed to use may be modified.		

#### CPU State Preservation Following a Successful L1→L2 VM Entry and an L2→L1 VM Exit

Following a successful L1→L2 VM entry and an L2→L1 VM exit, some CPU state is modified:

- General purpose register (GPR) values are not preserved.

[Others TBD]

#### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDG.VP.ENTER transitions the VCPU into L2 VM operation. The function returns either if failed to enter L2 VM or after successful entry to L2 VM and then exit from L2 VM.

The following table lists non-error TDG.VP.ENTER termination conditions:

**Table 5.346: TDG.VP.ENTER Termination Cases**

Case	Status in RAX[63:32]	Description
<b>Normal</b>	TDX_SUCCESS	L1→L2 entry was successful, followed by an L2→L1 exit. L2 VM exit information is provided in output GPRs.
<b>Host Requested L2 Exit</b>	TDX_L2_EXIT_HOST_ROUTED_ASYNC	<p>L1→L2 entry was successful. Later, following direct TD exit from L2, the host VMM requested resumption of L1. L2 CPU state is updated in the register list. L2 VM exit information is provided in output GPRs. This information was provided to the host VMM on TD exit; it may or may not be meaningful to the L1 VMM.</p> <p>In case of TDG.VP.VMCALL, the L2 CPU state is the state after completion of that function, e.g., GPR values are as returned by the host VMM as inputs to TDH.VP.ENTER.</p> <p>This condition is sticky. I.e., if resumption of L1 encountered a problem that required a TD exit (e.g., an EPT violation) the following TD entry resumes L1 and provides the same TDX_L2_EXIT_HOST_ROUTED status.</p>

Case	Status in RAX[63:32]	Description
<b>Host Requested L2 Exit following TDG.VP.VMCALL</b>	TDX_L2_EXIT_HOST_ROUTED_TDVMCALL	<p>L1→L2 entry was successful. Later, following TDG.VP.VMCALL that caused a direct TD exit from L2, the host VMM requested resumption of L1.</p> <p>L2 CPU state is updated in the L2_ENTER_GUEST_STATE structure. The L2 CPU state is the state after completion of TDG.VP.VMCALL, e.g., GPR values are as returned by the host VMM as inputs to TDH.VP.ENTER.</p> <p>L2 VM exit information is provided in output GPRs. This information was provided to the host VMM on the last TD exit; it may or may not be meaningful to the L1 VMM.</p> <p>This condition is sticky. I.e., if resumption of L1 encountered a problem that required a TD exit (e.g., an EPT violation) the following TD entry resumes L1 and provides the same TDX_L2_EXIT_HOST_ROUTED_TDVMCALL status. Note that in such case the L2 VM exit information reflects, e.g., the EPT violation, not the original TDG.VP.VMCALL exit.</p>
<b>Pending Interrupt L2 Exit</b>	TDX_L2_EXIT_PENDING_INTERRUPT	<p>L1→L2 entry was successful. Later, an L2→L1 exit happened due to an interrupt that was posted to L1. L2 CPU state is updated in the register list. L2 VM exit information is provided in output GPRs but is not necessarily meaningful to the L1 VMM (e.g., VM-exit interruption information is for the external interrupt that triggered the L2→L1 exit).</p>
<b>No L2 Entry</b>	Other	<p>L1→L2 entry was aborted because of some condition, which may or may not be an error, as indicated by RAX bit 63.</p> <p>E.g., entry may have been aborted due to a pending virtual interrupt. In this case, the L1 VMM typically sets RFLAGS.IF to 1, handles the interrupt and then invokes TDG.VP.ENTER again.</p>

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.347: TDG.VP.ENTER Memory Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions
Explicit	RDX	GPA	Guest state	L2_ENTER_GUEST_STATE	RW	Private	256B	None
Implicit	N/A	N/A	TDR page	TDR	None	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDCS structure	TDCS	R	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDVPS structure	TDVPS	None	Opaque	N/A	Shared(i)

In addition to the explicit memory operand checks per the table above, the function checks the following conditions:

1. VM\_FLAGS:
  - 1.1. VM index must be between 1 and TDCS.NUM\_L2\_VMS.
  - 1.2. The reserved fields must be 0.
2. GUEST\_STATE\_GPA is a valid private GPA.

If passed:

3. Check if there is a pending virtual interrupt to L1. This is indicated by RVI[7:4] > VPPR[7:4]. If so, terminate with a TDX\_PENDING\_INTERRUPT status.

If no interrupt is pending:

4. Translate GPAs:

4.1. Translate TDG.VP.ENTER memory output operands GPAs. Translation needs to be done in one of the following cases:

- The GPA is different than stored in TDVPS from last time TDG.VP.ENTER was called for this VM.
- The GPA has been blocked since last translated.
- HPA shadow for this GPA is NULL\_PA.

Translation failure leads to an EPT violation TD exit.

If passed:

4.2. Translate GPAs of L2 VMCS fields. Translation needs to be done in one of the following cases:

- The GPA has been blocked since last translated.
- HPA shadow for this GPA is NULL\_PA.

If passed:

5. If DO\_INVEPT is set:

5.1. Execute INVEPT type 1 to flush address translations of the L2 VM.

6. Read the provided register list

6.1. Write VMCS-stored register values (e.g., RSP) to the L2 VMCS.

6.2. Load GPR values to the CPU's GPRs.

7. Execute VMLAUNCH or VMRESUME depending on whether the entered VCPU and L2 VM (i.e., the current L2 VMCS) has been launched on this LP since the VCPU's last association with the LP (TDVPS.LAUNCHED[VM]).

## Completion Status Codes

**Table 5.348: TDG.VP.ENTER Completion Status Codes (Returned in RAX) Definition [TO BE COMPLETED]**

Completion Status Code	Description
TDX_L2_EXIT_HOST_ROUTED_ASYNC	L1→L2 entry succeeded. Later, following an asynchronous TD exit from L2, the host VMM requested resumption of L1.
TDX_L2_EXIT_HOST_ROUTED_TDVMCALL	L1→L2 entry succeeded. Later, following a TDG.VP.VMCALL TD exit from L2, the host VMM requested resumption of L1.
TDX_L2_EXIT_PENDING_INTERRUPT	L1→L2 entry succeeded, and later L2→L1 exit happened due to an interrupt that was posted to L1 and is pending.
TDX_PENDING_INTERRUPT	L1→L2 entry was aborted because an interrupt is pending for the L1 VMM. This indication is returned even if the L1 VMs cleared RFLAGS.IF.
TDX_SUCCESS	

**5.4.16. UPDATED: TDG.VP.INFO Leaf**

Get guest TD execution environment information.

**Table 5.349: TDG.VP.INFO Input Operands Definition**

Operand	Description		
RAX	TDCALL instruction leaf number and version, see 5.4.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the TDCALL interface function
	23:16	Version Number	Selects the TDCALL interface function version Must be 0
	63:24	Reserved	Must be 0

5

**Table 5.350: TDG.VP.INFO Output Operands Definition**

Operand	Description		
RAX	TDCALL instruction return code – see 5.4.1 – returns a constant value of TDX_SUCCESS (0)		
RCX	Bits	Name	Description
	5:0	GPAW	The effective GPA width (in bits) for this TD (do not confuse with MAXPA). SHARED bit is at GPA bit GPAW-1. Only GPAW values 48 and 52 are possible.
	63:6	RESERVED	Reserved: 0
RDX	The TD's ATTRIBUTES (provided as input to TDH.MNG.INIT)		
R8	Bits	Name	Description
	31:0	NUM_VCPUS	Number of Virtual CPUs that are usable (i.e. either active or ready)
	63:32	MAX_VCPUS	TD's maximum number of Virtual CPUs (provided as input to TDH.MNG.INIT)
R9	Bits	Name	Description
	31:0	VCPU_INDEX	Virtual CPU index, starting from 0 and allocated sequentially on each successful TDH.VP.INIT
	63:32	RESERVED	Reserved for enumerating future Intel TDX module capabilities, etc.: set to 0
R10	Bits	Name	Description
	0	SYS_RD	Indicates that the TDG.SYS.RD/RDM/RDALL functions are available. Further enumeration can be done using these functions.
	63:1	RESERVED	Reserved – set to 0
R11	Reserved for enumerating future Intel TDX module capabilities, etc.: set to 0		
Other	Unmodified		

### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDG.VP.INFO provides the TD software with execution environment information – beyond information that is provided by CPUID.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.351: TDG.VP.INFO Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions
Implicit	N/A	N/A	TDR page	TDR	None	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDCS structure	TDCS	R	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDVPS structure	TDVPS	R	Opaque	N/A	Shared(i)

### Completion Status Codes

**Table 5.352: TDG.VP.INFO Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_SUCCESS	TDG.VP.INFO is successful.

DRAFT

**5.4.17. NEW: TDG.VP.INVEPT Leaf**

Invalidate cached EPT translations for selected L2 VMs.

**Table 5.353: TDG.VP.INVEPT Input Operands Definition**

Operand	Description		
RAX	TDCALL instruction leaf number and version, see 5.4.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the TDCALL interface function
	23:16	Version Number	Selects the TDCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	VM index bitmap Bit N value of 1 indicates a request to invalidate EPT for VM index N. N must be between 1 and the number of L2 VMs in this TD.		
	Bits	Name	Description
	0	Reserved	Reserved: must be 0
	1	L2_VM_1	Invalidate EPT for L2 VM #1
	2	L2_VM_2	Invalidate EPT for L2 VM #2
	3	L2_VM_3	Invalidate EPT for L2 VM #3
	63:4	Reserved	Reserved: must be 0

**Table 5.354: TDG.VP.INVEPT Output Operands Definition**

Operand	Description
RAX	TDCALL instruction return code
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDG.VP.INVEPT executes INVEPT to invalidate the EPT translations of the specified L2 VMs.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.355 TDG.VP.RD Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions
Implicit	N/A	N/A	TDR page	TDR	None	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDCS structure	TDCS	R	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDVPS structure	TDVPS	R	Opaque	N/A	Shared(i)

If the memory operand checks per the table above pass:

1. Check the validity of the VM index bit mask
2. Execute INVEPT type 1 (single context invalidation) for the specified L2 VMs' EPTP.

#### Completion Status Codes

5

**Table 5.356: TDG.VP.INVEPT Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_SUCCESS	TDG.VP.INVEPT is successful.

DRAFT



**5.4.18. NEW: TDG.VP.INVVPID Leaf**

Invalidate mappings in the translation lookaside buffers (TLBs) and paging-structure caches for a specified L2 VM and a specified list of 4KB page linear addresses.

**Table 5.357: TDG.VP.INVVPID Input Operands Definition**

Operand	Name	Description		
RAX	Leaf and Version	TDCALL instruction leaf number and version, see 5.4.1		
		Bits	Field	Description
		15:0	Leaf Number	Selects the TDCALL interface function
		23:16	Version Number	Selects the TDCALL interface function version Must be 0
		63:24	Reserved	Must be 0
RCX	VM_AND_FLAGS	VM identifier and flags		
		Bits	Name	Description
		0	LIST	If 0, then RDX contains a single page GLA list entry Else, RDX contains the GPA and other information of a page GLA list in memory.
		51:1	Reserved	Reserved: must be 0
		53:52	VM	L2 virtual machine index (must be 1 or higher)
		63:54	Reserved	Reserved: must be 0
RDX	GLA_LIST_ENTRY or GLA_LIST_INFO	Depending on the LIST flag, RDX contains either of the following: <ul style="list-style-type: none"> <li>A single GLA_LIST_ENTRY</li> <li>GLA_LIST_INFO, containing the GPA of a Guest Linear Address (GLA) list page in private memory, and first and last entries to process. See 3.6.4 for details.</li> </ul>		

5

**Table 5.358: TDG.VP.INVVPID Output Operands Definition**

Operand		Description
RAX	Status	TDCALL instruction return code
RDX	GLA_LIST_ENTRY or GLA_LIST_INFO	Depending on the LIST flag, RDX contains either of the following: <ul style="list-style-type: none"> <li>If LIST was 0, RDX contains the single GLA_LIST_ENTRY provided as an input, unmodified</li> <li>If LIST was 1, RDX contains the GLA_LIST_INFO provided as input, but with the FIRST_ENTRY and NUM_ENTRIES fields updated to reflect the number of entries processed so far. If all entries have been processed successfully, NUM_ENTRIES is set to 0.</li> </ul>
Other		Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

10

TDG.VP.INVVPID executes INVVPID type 0 to invalidate the cached translations for the specified list of 4KB page Guest Linear Addresses (GLA) of the specified L2 VM.

**Interruptibility** If, during its execution, TDG.VP.INVVPID detects that an external interrupt is pending, it may resume the guest TD with the CPU state unmodified, except for the following:

- GLA\_LIST\_INFO in RDX is updated to reflect the GLAs processed so far.

This allows the external interrupt to be recognized, causing a VM exit or a posted interrupt delivery. Typically, TDG.VP.INVVPID will be re-invoked (since RIP has not changed) and continue its work. Guest TD software is not directly involved.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.359 TDG.VP.INVVPID Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions
Explicit	RDX	GPA	GLA list page	PAGE_GLA_LIST	RW	Private	4096	None
Implicit	N/A	N/A	TDR page	TDR	None	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDCS structure	TDCS	R	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDVPS structure	TDVPS	R	Opaque	N/A	Shared(i)

If the memory operand checks per the table above pass, the function checks the following conditions (no specific order is implied):

- VM is the index of an existing L2 VM

If passed:

- If the LIST flag in RCX is 0, process a single GLA\_LIST\_ENTRY in RDX:
  - With current GLA starting from BASE\_GLA, repeat for each page through LAST\_PAGE:
    - Execute INVVPID type 0, providing the current GLA and the VM's VPID
    - Advance the current GLA by 4KB

- Else (LIST flag in RCX is 1), process a GLA list:

- Check the validity of GLA\_LIST\_INFO.

If passed:

- Translate the GPA list's GPA.

- On translation error, do a TD exit with an EPT violation indication.

- Repeat for NUM\_ENTRIES from FIRST\_ENTRY:

- Read the current GLA\_LIST\_ENTRY.

- Process the current entry as described in the single-entry case above.

- If there is a pending interrupt and this was not the last entry:

- Update RDX to reflect the work done so far: set FIRST\_ENTRY to the index of the next entry and NUM\_ENTRIES to the remaining number of entries.

- Resume the guest TD without updating RIP or any other state except for RDX.

## Completion Status Codes

**Table 5.360: TDG.VP.INVVPID Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_SUCCESS	TDG.VP.INVVPID is successful.

**5.4.19. NEW: TDG.VP.RD Leaf**

Read a VCPU-scope metadata field (control structure field) of a TD.

**Table 5.361: TDG.VP.RD Input Operands**

Operand	Description		
RAX	TDCALL instruction leaf number and version, see 5.4.1		
	<b>Bits</b>	<b>Field</b>	<b>Description</b>
	15:0	Leaf Number	Selects the TDCALL interface function
	23:16	Version Number	Selects the TDCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	Reserved, must be 0		
RDX	Field identifier – see 3.10 The LAST_ELEMENT_IN_FIELD and LAST_FIELD_IN_SEQUENCE components of the field identifier must be 0. WRITE_MASK_VALID, INC_SIZE, CONTEXT_CODE and ELEMENT_SIZE_CODE components of the field identifier are ignored. A value of -1 is a special case: it is not a valid field identifier; in this case the first readable field identifier is returned in RDX.		

**Table 5.362: TDG.VP.RD Output Operands Definition**

Operand	Description
RAX	TDCALL instruction return code – see 5.4.1
RDX	<ul style="list-style-type: none"> <li>If the input field identifier was -1, RDX returns the first readable field identifier.</li> </ul> Else, in case of an error, RDX returns -1. On success, RDX returns the next readable field identifier. A value of -1 indicates no next field identifier is available.
R8	Contents of the field In case of no success, as indicated by RAX, R8 returns 0.
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

10 TDG.VP.RD reads a VCPU-scope metadata field (control structure field) of a TD.

RDX returns the next host-side readable field identifier. This may be used by the guest TD to dump the guest readable VCPU metadata. To read all the available fields, the guest TD can invoke TDG.VP.RD in a loop, starting with field identifier -1 as an input, until RDX returns -1.

15 To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.363 TDG.VP.RD Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions
Implicit	N/A	N/A	TDR page	TDR	None	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDCS structure	TDCS	R	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDVPS structure	TDVPS	R	Opaque	N/A	Shared(i)

If the memory operand checks per the table above pass:

- Read the control structure field using the algorithm described in 5.2.2.1.

## 5 Completion Status Codes

**Table 5.364: TDG.VP.RD Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_METADATA_FIRST_FIELD_ID_IN_CONTEXT	Indicates that the first field ID in context is returned
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_SUCCESS	TDG.VP.RD is successful.

DRAFT

**5.4.20. TDG.VP.VEINFO.GET Leaf**

Intel SDM, Vol. 3, 24.9.4 Information for VM Exits Due to Instruction Execution  
 Intel SDM, Vol. 3, 25.5.6 Virtualization Exceptions  
 Intel SDM, Vol. 3, 27.2.5 Information for VM Exits Due to Instruction Execution

- 5 Get Virtualization Exception Information for the recent #VE exception.

**Table 5.365: TDG.VP.VEINFO.GET Input Operands Definition**

Operand	Description		
RAX	TDCALL instruction leaf number and version, see 5.4.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the TDCALL interface function
	23:16	Version Number	Selects the TDCALL interface function version Must be 0
	63:24	Reserved	Must be 0

**Table 5.366: TDG.VP.VEINFO.GET Output Operands Definition**

Operand	Description		
RAX	TDCALL instruction return code – see 5.4.1		
RCX	Bits	Name	Description
	31:0	Exit Reason	The 32-bit value that would have been saved into the VMCS as an exit reason if a VM-exit had occurred instead of the virtualization exception
	63:32	Reserved	Reserved: 0
	In case of an error, RCX returns 0.		
RDX	Exit Qualification: the 64-bit value that would have been saved into the VMCS as an exit qualification if a legacy VM exit had occurred instead of the virtualization exception In case of an error, RDX returns 0.		
R8	Guest Linear Address: the 64-bit value that would have been saved into the VMCS as a guest-linear address if a legacy VM exit had occurred instead of the virtualization exception In case of an error, R8 returns 0.		
R9	Guest Physical Address: the 64-bit value that would have been saved into the VMCS as a guest-physical address if a legacy VM exit had occurred instead of the virtualization exception In case of an error, R9 returns 0.		
R10	Bits	Name	Description
	31:0	VM-exit instruction length	The 32-bit value that would have been saved into the VMCS as VM-exit instruction length if a legacy VM exit had occurred instead of the virtualization exception
	63:32	VM-exit instruction information	The 32-bit value that would have been saved into the VMCS as VM-exit instruction information if a legacy VM exit had occurred instead of the virtualization exception

Operand	Description
	The content of R10 is only applicable for TDX-extended #VE (injected by the TDX module), where Exit Reason is not EPT violation (48). It should be ignored for EPT violations converted by the CPU to #VE. In case of an error, R10 returns 0.
Other	Unmodified

### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 5 TDG.VP.VEINFO.GET returns the virtualization exception information of a #VE exception that was previously delivered to the guest TD.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.367: TDG.VP.VEINFO.GET Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions
Implicit	N/A	N/A	TDR page	TDR	None	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDCS structure	TDCS	None	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDVPS structure	TDVPS	RW	Opaque	N/A	Shared(i)

10

The function checks the following conditions (no specific order is implied):

- The VALID field in TDVPS.VE\_INFO must non-0 to indicate that a valid virtualization information is available.

If successful, the function does the following:

- 15
1. Return the EXIT\_REASON, EXIT\_QUALIFICATION, GLA, GPA, INSTRUCTION\_LENGTH and INSTRUCTION\_INFORMATION from TDVPS.VE\_INFO in GPRs.
  2. Clear the VALID field in TDVPS.VE\_INFO to 0 to indicate that the virtualization information has been read.

### Completion Status Codes

**Table 5.368: TDG.VP.VEINFO.GET Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_NO_VE_INFO	There is no Virtualization Exception information.
TDX_SUCCESS	TDG.VP.VEINFO.GET is successful.

### 5.4.21. TDG.VP.VMCALL Leaf

Perform a TD Exit to the host VMM.

**Table 5.369: TDG.VP.VMCALL Input Operands Definition**

Operand	Description		
RAX	TDCALL instruction leaf number and version, see 5.4.1		
	Bits	Field	Description
	15:0	Leaf Number	Selects the TDCALL interface function
	23:16	Version Number	Selects the TDCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	<p>A bitmap that controls which part of the guest TD GPR and XMM state is passed as-is to the VMM and back</p> <p>A bit value of 0 indicates that the corresponding register is saved by the Intel TDX module, scrubbed to 0 before SEAMRET to the host VMM, and restored by the Intel TDX module on the following TDH.VP.ENTER.</p> <p>A bit value of 1 indicates that the corresponding register is passed as-is to the host VMM, and on the following TDH.VP.ENTER, the register value is used as input from the host VMM and passed as-is to the guest TD.</p> <p>The value of RCX is passed to the host VMM.</p>		
	Bits	Name	Description
	15:0	GPR Mask	Controls the transfer of GPR values: Bit 0: RAX (must be 0) Bit 1: RCX (must be 0) Bit 2: RDX Bit 3: RBX Bit 4: RSP (must be 0) Bit 5: RBP Bit 6: RSI Bit 7: RDI Bits 15:8: R15 – R8
	31:16	XMM Mask	Controls the transfer of XMM15 – XMM0 register values
	63:32	Reserved	Reserved: must be 0
RBX, RDX, RBP, RSI, RDI, R8 – R15	<p>If the corresponding bit in RCX is set to 1, the register value passed as-is to the host VMM on SEAMRET.</p> <p>Else, the register value is not used as an input and is preserved.</p>		
XMM0 – XMM15	<p>If the corresponding bit in RCX is set to 1, the register value passed as-is to the host VMM on SEAMRET.</p> <p>Else, the register value is not used as an input and is preserved.</p>		

**Table 5.370: TDG.VP.VMCALL Output Operands Definition**

Operand	Description
RAX	TDCALL instruction return code: returns a constant value of TDX_SUCCESS (0)

Operand	Description
RCX	Unmodified
RBX, RDX, RBP, RDI, RSI, R8 – R15	If the corresponding bit in RCX is set to 1, the register value passed as-is from the host VMM's SEAMCALL(TDH.VP.ENTER) input. Else, the register value is unmodified.
XMM0 – XMM15	If the corresponding bit in RCX is set to 1, the register value passed as-is from the host VMM's SEAMCALL(TDH.VP.ENTER) input. Else, the register value is unmodified.
Other	Unmodified

### Leaf Function Description

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDG.VP.VMCALL performs a TD exit to the host VMM. From the VMM's point of view, this is the termination of a previous SEAMCALL(TDH.VP.ENTER). Selected GPR and XMM state is passed to the VMM host, controlled by RCX as shown above. The rest of the CPU state is saved in TDVPS and replaced with a synthetic state.

From the guest TD's point of view, a subsequent SEAMCALL(TDH.VP.ENTER) from the host VMM terminates the TDG.VP.VMCALL function. Most GPR state, and if the value of RCX bit 1 is set, all XMM state, is passed to the TD guest as shown above.

**L2 VM Details:** TDG.VP.VMCALL may be invoked by an L2 VM, if enabled by the L1 VMM for the current VCPU (by setting TDVPS.L2\_CTL.S\_ENABLE\_TDVMCALL). If not enabled, then TDG.VP.VMCALL results in an L2→L1 exit.

On subsequent TD resumption, the host VMM may request resumption into L1 by setting TDH.VP.ENTER's RESUME\_L1 flag. In this case, L1 is resumed (i.e., the TDG.VP.ENTER it has invoked is terminated) with a TDX\_L2\_EXIT\_HOST\_ROUTED\_TDVMCALL status. The L2 VCPU state reflects the successful completion of TDG.VP.VMCALL.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.371: TDG.VP.VMCALL Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions
Implicit	N/A	N/A	TDR page	TDR	None	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDCS structure	TDCS	None	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDVPS structure	TDVPS	RW	Opaque	N/A	Shared(i)

- If invoked from an L2 VM, and TDVPS.L2\_CTL.S\_ENABLE\_TDVMCALL is 0, do an L2→L1 exit.
- Save guest TD CPU state to TDVPS (including TD VMCS):
  - Save extended state per TDCS.XFAM. There is no strict requirement to save XMM state that will be passed to the host VMM as controlled by RCX. This state will be overwritten on the next TD entry.
  - Save GPR state. There is no strict requirement to save GPR state that will be passed to the host VMM as controlled by RCX (but RCX itself must be saved). This state will be overwritten on the next TD entry.
  - Advance the saved RIP to the instruction following TDCALL.
- Adjust the TDCS TLB tracking counters.
- Release the shared locking – acquired on TDH.VP.ENTER of TDR, TDCS and TDVPS.
- Load host VMM state:
  - Clear the extended state except XMM (per TDCS.XFAM) to synthetic INIT values.
  - As controlled by RCX, either clear or set to the guest TD's value the state of XMM0 – XMM15.



- 5.3. As controlled by RCX, either clear or set to the guest TD's value the state of RBX, RDX, RBP, RDI, RSI and R8 – R15.
- 5.4. Set RCX to the guest TD's value.
- 5.5. Set RAX to the TDCALL exit reason.
- 5.6. Restore other host VMM state – saved during TDH.VP.ENTER.
- 5    6. Execute SEAMRET to return to the host VMM.

**Note:** Logically, from the point of view of the guest TD, TDG.VP.VMCALL is terminated by the next TDH.VP.ENTER.

#### Completion Status Codes

**Table 5.372: TDG.VP.VMCALL Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_SUCCESS	TDG.VP.VMCALL is successful. TD exit was done, resulting a in a completion of SEAMCALL(TDH.VP.ENTER) on the host VMM side. Later, the host VMM executed SEAMCALL(TDH.VP.ENTER) again, and execution returned to the guest TD VCPU (in TDX non-root mode) completing TDG.VP.VMCALL.

10

DRAFT

**5.4.22. NEW: TDG.VP.WR Leaf**

Write a VCPU-scope metadata field (control structure field) of a TD.

**Table 5.373: TDG.VP.WR Input Operands**

Operand	Description		
RAX	TDCALL instruction leaf number and version, see 5.4.1		
	<b>Bits</b>	<b>Field</b>	<b>Description</b>
	15:0	Leaf Number	Selects the TDCALL interface function
	23:16	Version Number	Selects the TDCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	Reserved, must be 0		
RDX	Field identifier – see 3.10 The LAST_ELEMENT_IN_FIELD and LAST_FIELD_IN_SEQUENCE components of the field identifier must be 0. WRITE_MASK_VALID, INC_SIZE, CONTEXT_CODE and FIELD_SIZE components of the field identifier are ignored.		
R8	Data to write to the field		
R9	A 64b write mask to indicate which bits of the value in R8 are to be written to the field		

**Table 5.374: TDG.VP.WR Output Operands Definition**

Operand	Description
RAX	TDCALL instruction return code – see 5.4.1
R8	Previous contents of the field In case of an error, as indicated by RAX, R8 returns 0.
Other	Unmodified

**Leaf Function Description**

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

- 10 TDG.VP.WR writes a VCPU-scope metadata field (control structure field) of a TD.

To understand the table and text below, please refer to the [TDX Module Base Spec] chapter discussing general aspects of the Intel TDX Module API.

**Table 5.375 TDG.VM.WR Operands Information Definition**

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions
Implicit	N/A	N/A	TDR page	TDR	None	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDCS structure	TDCS	R	Opaque	N/A	Shared(i)
Implicit	N/A	N/A	TDVPS structure	TDVPS	RW	Opaque	N/A	Shared(i)

If the memory operand checks per the table above pass:

1. Write the control structure field and return its old value, using the algorithm described in 5.2.2.2.

#### Completion Status Codes

5

**Table 5.376: TDG.VP.WR Completion Status Codes (Returned in RAX) Definition**

Completion Status Code	Description
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_SUCCESS	TDG.VP.WR is successful.

DRAFT