英特尔® 体系结构内存加密技术

规格

2021 04月

修订版**1.3**

文件编号：336907-003US

目录

数字

桌子

修订历史记录

修订号	描述	日期
1.0	• 文件的首次发布。	2017年12月
1.2	• 添加了其他详细信息。	2019年4月
1.3	• 增加了对256b密钥和KeyID0/TME加密旁路的支持。	2021 04月

术语

TME（总内存加密）：这是使用单个临时密钥进行内存加密的基线功能。

MKTME（多密钥总内存加密）：添加对使用多个密钥进行页面粒度内存加密的支持，并额外支持软件配置的密钥。

1 介绍

本文档介绍从第三代Intel开始提供的内存加密支持® 至强® 处理器可扩展系列。请注意，Intel平台支持多种不同类型的内存，并非所有SOC都支持所有类型内存的这种功能。最初的实现集中在传统的DRAM上。

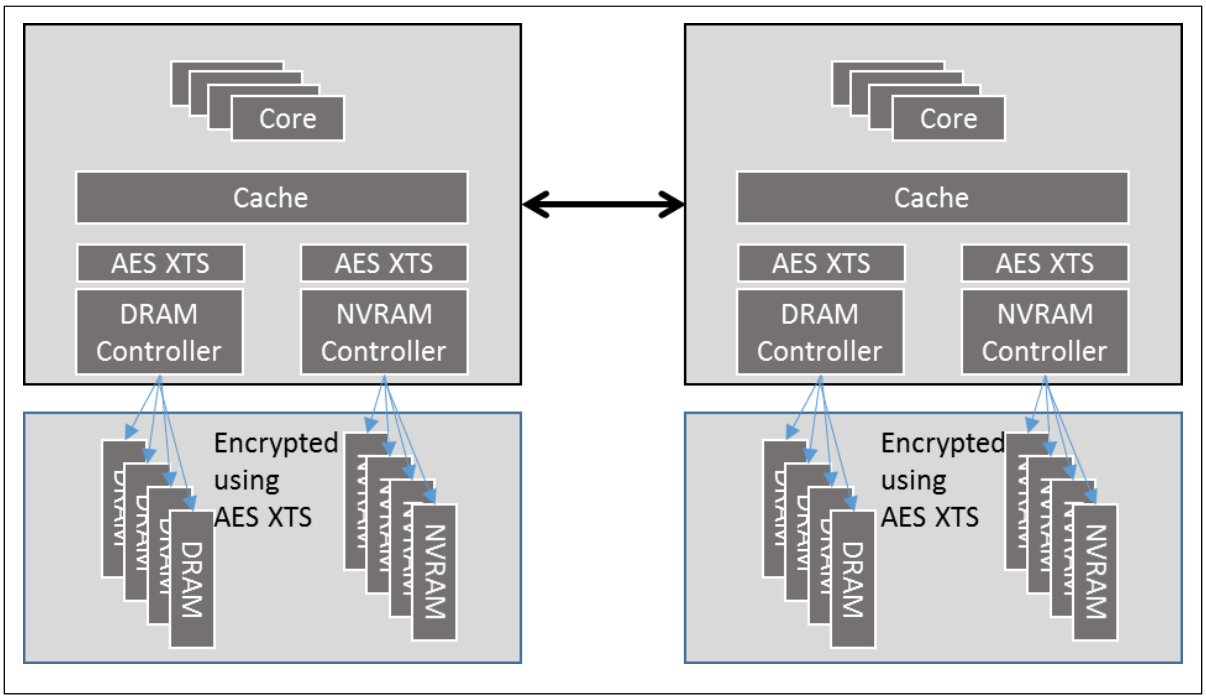
总内存加密（TME）–对系统的整个物理内存进行加密的能力。该功能通常在引导过程的早期阶段启用，只需对BIOS进行少量更改，一旦配置并锁定，将使用NIST标准AES-XTS算法加密SoC外部内存总线上的所有数据，该算法具有128位密钥或256位密钥，具体取决于算法可用性和选择。TME使用的加密密钥使用在Intel SoC中实现的硬件随机数生成器，软件或使用Intel SoC的外部接口无法访问密钥。TME功能旨在为外部内存总线和DIMM提供AES-XTS保护。该架构非常灵活，将来将支持额外的内存保护方案。此功能启用后，旨在支持（未修改）现有系统和应用软件。该功能对整体性能的影响可能相对较小，并且高度依赖于工作负载。

多密钥全内存加密（MKTME）建立在TME的基础上，增加了对多个加密密钥的支持。SoC实现支持固定数量的加密密钥，软件可以配置SoC以使用可用密钥的子集。软件管理密钥的使用，并可以使用每个可用密钥加密内存的任何页面。因此，MKTME允许对内存进行页面粒度加密。默认情况下，除非软件明确指定，否则MKTME使用TME加密密钥。除了支持CPU生成的临时密钥（软件或使用SoC的外部接口无法访问），MKTME还支持软件提供的密钥。
当与非易失性内存一起使用或与认证机制结合使用和/或与密钥提供服务一起使用时，软件提供的密钥特别有用。在虚拟化场景中，我们预计VMM或虚拟机监控程序将管理密钥的使用，以在不进行任何更改的情况下透明地支持遗留操作系统（因此，在这种部署场景中，MKTME也可以被视为TME虚拟化）。操作系统可以在本机和虚拟化环境中进一步利用MKTME功能。当正确启用时，虚拟化环境中的每个来宾操作系统都可以使用MKTME，并且来宾操作系统可以以与本机操作系统相同的方式利用MKTME。

2 全内存加密（TME）简介

下图概述了双插槽配置中的总内存加密。实际实施可能有所不同。

图2-1。TME的双插座配置



AES XTS加密引擎位于到外部内存总线的直接数据路径中，因此，在内存总线上进入和/或离开SoC的所有内存数据都使用AES XTS加密。SoC内的数据（缓存等）保持纯文本，并支持所有现有软件和输入/输出模型。

在典型部署中，加密密钥由CPU生成，因此软件不可见。当系统配置了NVRAM时，如果NVRAM被视为DRAM，那么它也可以使用CPU生成的密钥。但是，如果要将NVRAM视为非易失性内存，则可以选择跨平台电源循环/重新启动生成/重用相同的密钥。

3 多密钥全内存加密（MKTME）简介

3.1 高级架构

MKTME的高层架构如下图所示。

图3-1。MKTME的高层体系结构

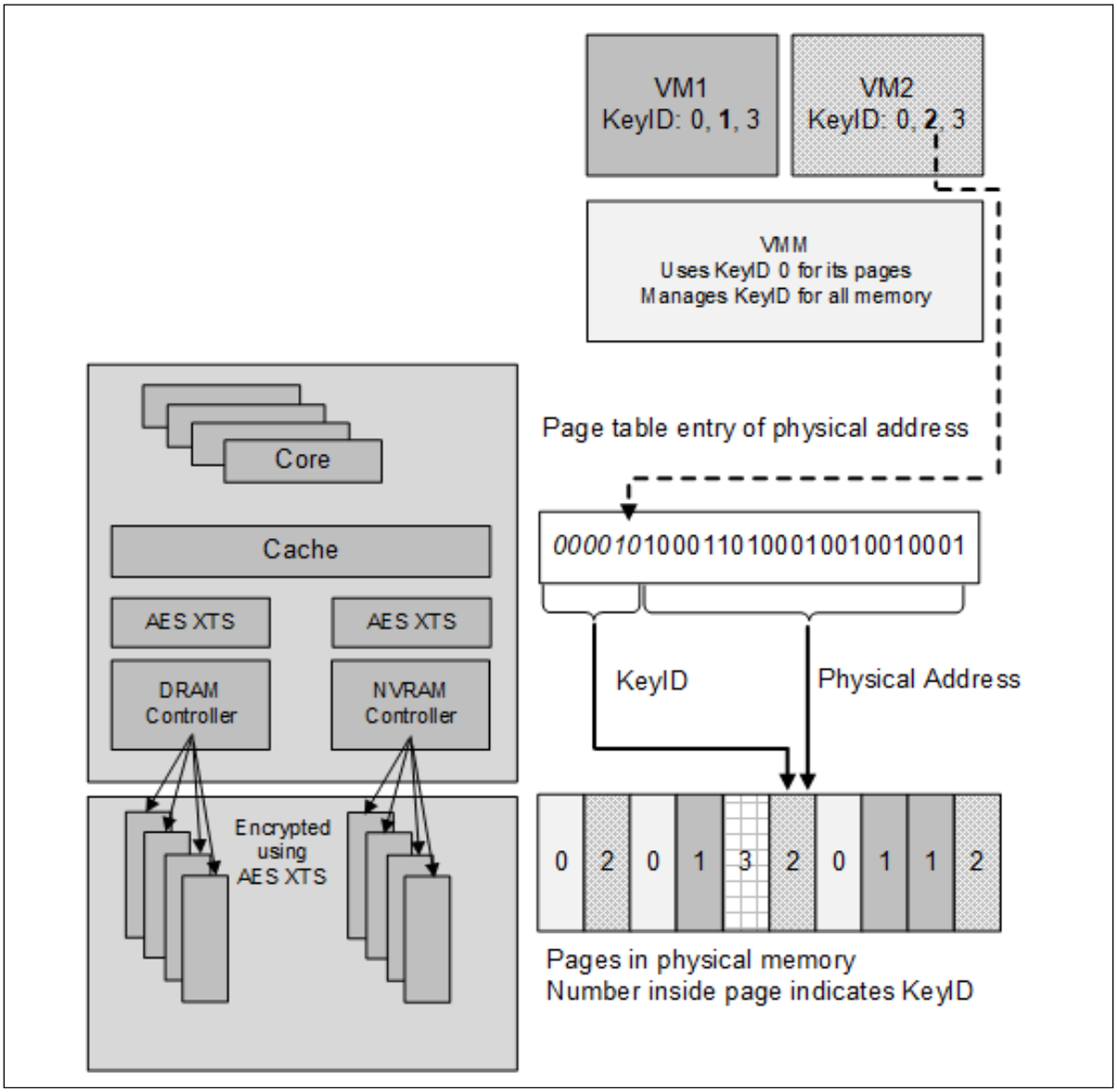


图3-1显示了MKTME的基本架构，该架构与TME共享基本硬件架构，但AES XTS现在支持多个密钥。图的右侧显示了在虚拟化环境中使用MKTME，尽管架构也支持在本机操作系统部署场景中使用MKTME。在本例中，我们展示了一个虚拟机监控程序/VMM和两个虚拟机。默认情况下，虚拟机监控程序使用KeyID 0（与TME相同），尽管它也可以为自己的内存使用不同的KeyID。VM1将KeyID1用于其自己的私有页面，VM2将keyid2用于其自己的私有页面。此外，VM1始终可以对任何页面使用KeyID 0（TME KeyID），并且还可以选择使用KeyID 3来实现自身和VM2之间的共享内存。键ID作为物理地址字段的高位包含在页表条目中。如本例所示，显示了KeyID 2。物理地址字段中的其余位用于实际寻址内存中的位。为了便于说明，该图显示了一个可能的页面分配和KeyID，不过在这种情况下，虚拟机监控程序可以完全自由地将任何KeyID与自己或其任何客户机vm的任何页面一起使用。注意，在页表中过度订阅物理地址位的想法也扩展到了其他页表，包括IA页表和IOMMU页表。除了AES XT和外部内存总线上的微调外，KeyID仍然是SoC中所有物理地址位的一部分。KeyID不在SoC之外或AES XTS的调整中使用。

4 TME和MKTME：枚举和控制寄存器

此信息仅适用于枚举TME和/或MKTME功能的CPU。

4.1 枚举

TME和MKTME功能通过本节所述的MSR暴露于BIOS/软件。列举了处理器中可用于/支持MKTME的最大密钥数。BIOS需要通过MSR（稍后描述）激活此功能，并且必须选择在早期引导过程中支持/用于MKTME的密钥数量。激活后，连接到CPU/SoC的所有内存（TME排除范围内的内存除外）使用AES-XTS进行加密，AES-XTS具有128位或256位临时密钥（平台密钥），该密钥在每次启动时由CPU生成。请注意，此行为仅适用于未绕过TME加密的情况（使用IA32\u TME\u ACTIVATE MSR中的位31）。如果绕过TME加密，则所有具有KeyID0的访问都将绕过加密/解密。

英特尔处理器支持外部内存控制器。这些内存控制器可以通过相干总线（如Intel）连接到处理器® 超路径互连（Intel®UPI）或Compute Express Link（CXL）。MKTME枚举可用于发现英特尔处理器的功能和连接到集成内存控制器的某些内存，但不一定表示外部内存控制器功能，也不一定表示连接到集成控制器的某些类型的内存。能够受到CPU加密能力保护的内存区域通过UEFI 2.8中引入的新UEFI内存属性EFI\u memory\u CPU\u CRYPTO与系统软件通信。如果设置了该标志，则内存区域能够受到CPU内存加密功能的保护。如果清除此标志，则内存区域无法使用CPU的内存加密功能进行保护，或者CPU不支持CPU内存加密功能。系统软件必须参考该属性，以确定可以使用MKTME加密的范围。

4.1.1 TME

CPUID。TME（CPUID。（EAX=07H，ECX=0H）：ECX[13]）列举了这四个架构MSR的存在及其MSR地址：

- IA32\U TME\U能力–地址981H
- IA32\U TME\U激活–地址982H
- IA32\u TME\u EXCLUDE\u掩码–地址983H
- IA32\u TME\u EXCLUDE\u BASE–地址984H

4.1.2 多键TME

CPUID。TME位表示TME_能力MSR的存在，MSR将进一步枚举TME特征以及MKTME可用性和特征。MKTME由BIOS使用IA32\u TME\u ACTIVATE MSR启用/配置。MKTME需要TME，因此在未启用TME的情况下无法启用。

4.1.3 内存加密能力MSR（IA32\U TME\U能力）

表4-1。IA32\U TME\U能力MSR–地址981H

注册地址	架构MSR名称和位字段	MSR/位描述	议论
981小时	IA32\U TME\U能力MSR公司	内存加密 能力MSR	一个用于TME和MKTME的MSR。
	0	支持AES-XTS 128位 加密算法。	NIST标准。
	1	保留	
	2	支持AES-XTS 256位 加密算法。	NIST标准。
	30:3	保留	
	31	TME加密旁路 支持。	
	35:32	MK\u TME\u MAX\u KEYID\u位 可以使用的位数 分配用作密钥 多键标识符 内存加密。 如果MKTME不是，则为零 支持。	4位允许最大值为15， 它可以处理32K个密钥。
	50:36	MK\u TME\u MAX\u键 表示最大值 钥匙数量 可供使用。 此值可能不是 2的幂。 如果MKTME不是，则为零 支持。	密钥ID 0是专门保留的，是 在该领域未考虑。 最大值为32K-1个键。
	63:51	保留	

4.1.4 最大PA_宽度的CPUID报告

最大PA_宽度（叶8000008.EAX）的CPUID枚举不受MKTME激活的影响，并将继续报告可供软件使用的物理地址位的最大数量，而不考虑密钥ID位的数量。

4.2 内存加密配置和状态寄存器

4.2.1 激活MSR（IA32\u TME\u ACTIVATE）

此MSR用于锁定以下MSR。锁定后，将忽略对以下MSR的任何写入。当CPU复位时，锁复位。

- IA32\U TME\U激活
- IA32\u TME\u EXCLUDE\u掩码
- IA32\u TME\u EXCLUDE\u BASE

注： IA32\u TME\u EXCLUDE\u掩码和IA32\u TME\u EXCLUDE\u基本MSR是预计在IA32\U TME\U激活MSR之前配置。

要启用MKTME，必须设置IA32\u TME\u ACTIVATE MSR中的硬件加密启用位，位35:32必须具有非零值（这将指定为MKTME配置的密钥ID位的数量）。

表4-2。IA32\U TME\U激活MSR–地址982H

注册地址	架构MSR名称和位字段	MSR/位描述	议论
982小时	IA32\U TME\U激活MSR	内存加密激活MSR	
	0	锁定RO–将在WRMSR成功（或第一次SMI）时设置；忽略写入值。	
	1	硬件加密启用 （TME启用取决于TME加密旁路启用（位31））	该位还启用MKTME；如果不启用加密硬件，则无法启用MKTME。
	2	按键选择： 0–创建新的TME密钥（预期冷/热启动）。 1–从存储器中恢复TME密钥（从待机状态恢复时需要）。	
	3	将TME密钥保存到待机状态： 将密钥保存到存储器中，以便从待机状态恢复时使用。	并非所有CPU都支持。

注册地址	架构MSR名称和位字段	MSR/位描述	议论
	7:4	TME策略/加密算法： 仅IA32\U TME\U功能中列举的算法 允许MSR。 例如： 0000–AES-XTS-128 0010–AES-XTS-256 其他值无效。	要使用的TME加密算法。
	30:8	保留	
	31	TME加密旁路启用 启用加密硬件时： • 在以下情况下，使用基于硬件随机数生成器的CPU生成的临时密钥启用总内存加密：该位设置为0。 • 当该位设置为1时，绕过总内存加密（KeyID0没有加密/解密）。在某些处理器上，绕过TME加密可以通过避免解密延迟或 加密和解密。 软件必须检查硬件加密启用（位1）和TME加密旁路启用（位31），以确定TME加密是否启用。	
	35:32	如果未枚举MKTME，则保留。	
		MK\u TME\u KEYID\u位 分配给MKTME使用的密钥标识符位数。与枚举类似，这是一个编码值。 写入大于MK\u TME\u MAX\u KEYID\u位的值将导致#GP。 如果EAX（硬件加密启用）的第1位也未设置为"1"，则向该字段写入非零值将#GP，因为必须启用加密硬件才能使用MKTME。	

注册地址	架构MSR名称和位字段	MSR/位描述	议论
		示例：为了支持255个键，该字段的值将设置为8。	
	47:36	保留	
	63:48	MK\u TME\u CRYPTO\u ALGS位48:AES-XTS 128 位49：保留 位50:AES-XTS-256 位63:51：保留（#GP） BIOS的位掩码用于设置MKTME允许的加密算法，稍后将由密钥加载ISA强制执行（“1=允许”）。	

4.2.2 IA32\U TME\U激活WRMSR响应和错误处理

表4-3。IA32\U TME\U激活WRMSR响应和错误处理

条件	回答
未枚举时的WRMSR。	#总成
锁定状态=1时的WRMSR。	#总成
WRMSR，63:8（保留）≠ 0	#总成
具有不支持的策略值的WRMSR（IA32\u TME\u能力[IA32\u TME\u激活[7:4]]=0）。	#总成
启用的WRMSR=0。	TME禁用，MSR锁定后续RDMSR返回x。。x01b。
WRMSR，启用=1，键选择=0（新键）；RNG成功。	TME已启用且MSR已锁定后续RDMSR返回x。。x011b。
WRMSR，enabled=1，key select=0；RNG故障	未启用后续RDMSR返回x。。x000b。
WRMSR，enabled=1，key select=1；从CPU还原的非零密钥。	TME已启用且MSR已锁定后续RDMSR返回x。。x111b。
WRMSR，enabled=1，key select=1；失败-从CPU恢复零密钥。	未启用后续RDMSR返回x。。x100b。
WRMSR具有任何其他法律价值。	后续RDMSR返回写入值+锁定状态=1。
如果MK\u TME\u KEYID\u位>MK\u TME\u MAX\u KEYID\u位	#总成
如果MK\u TME\u KEYID\u位>0&&（TME）启用==0（TME必须与MK-TME在同一点启用）。	#总成
如果MK\u TME\u KEYID\u位>0且TME未成功激活（未设置锁定）。	写入未提交。
如果设置了MK\u TME\u CRYPTO\u ALGS保留位。	#总成

4.2.3 核心地址屏蔽MSR（MK\u

TME\u Core\u ACTIVATE）

这是一个仅BIOS的MSR。

使用IA32\U TME\U ACTIVATE MSR成功激活后，应在每个物理核上写入该寄存器，EDX中的值为0:EAX；否则可能会导致不可预测的行为。如果不支持MKTME，则将#GP访问此MSR。

在激活MKTME后，BIOS将在每个内核上写入此MSR。每个核上的第一个SMI也将导致该值与包MSR值同步。

表4-4。MK\u TME\u CORE\u激活MSR–地址9FFH

注册地址	MSR名称和位字段	MSR/位描述	议论
9FFH	MK\u TME\u CORE\u激活MSR	如果不支持MKTME，则此MSR将#GP。	
	31:0	保留	
	35:32	MK\u TME\u KEYID\u位（只读） 分配给MKTME使用的密钥标识符位数。 与枚举类似，这是一个编码值。 这是一个只读字段#非零写入时的GP。	写入时将从包MSR值中隐藏。
	63:36	保留	

4.2.4 排除范围MSR

TME和MKTME（仅适用于KeyID=0）支持一个排除范围，用于特殊情况。（注意：对于除0以外的所有密钥ID，TME排除范围不适用于MKTME。）本MSR中指定的物理地址范围不适用本文档中描述的内存加密。该范围主要用于操作系统不可用的内存，通常由BIOS配置。然而，TME/MKTME（对于KeyID=0）架构对排除范围的使用没有任何限制。软件能够通过读取MSR来确定该范围。该范围的定义遵循英特尔处理器中实现的许多范围寄存器的定义。

表4-5。IA32\u TME\u EXCLUDE\u MASK MSR–地址983H

注册地址	MSR名称和位字段	MSR/位描述	议论
983小时	IA32\u TME\u EXCLUDE\uMask MSR		
	10:0	保留	
	11	启用-当设置为"1"时，则IA32\u TME\u EXCLUDE\u BASE和IA32\u TME\u EXCLUDE\u掩码 MSR用于定义TME/MKTME的排除区域（对于KeyID=0）。	
	MAXPHYSADDR-1:12	TMEEMASK-此字段指示必须与TMEEBASE匹配的位，以符合TME/MKTME（对于KeyID=0）排除内存范围访问的资格。	
	63:MAXPHYSADDR	保留；必须为零。	

表4-6。IA32\u TME\u EXCLUDE\u BASE MSR–地址984H

注册地址	MSR名称和位字段	MSR/位描述	议论
984小时	IA32\u TME\u EXCLUDE\uBase MSR		
	11:0	保留	
	MAXPHYSADDR-1:12	TMEEBASE-TME/MKTME 要排除的基址物理地址（对于KeyID=0）加密。	
	63:MAXPHYSADDR	保留；必须为零。	

注： 将"1"写入支持的最大物理大小以上的位将导致#GP。

IA32\u TME\u EXCLUDE\u掩码MSR必须定义一个连续区域。如果TMEEMASK字段未指定连续区域，WRMSR将#GP。

这些MSR由IA32\U TME\U激活MSR锁定。如果lock=1，则WRMSR到IA32\u TME\u EXCLUDE\u MASK/IA32\u TME\u EXCLUDE\u BASE MSR将导致#GP。

5 MKTME的运行时行为

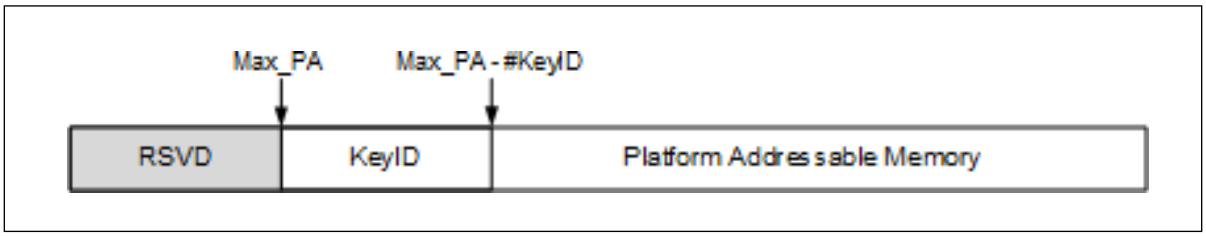
BIOS激活MKTME后，处理器的运行时行为会发生许多变化，本节对此进行了描述。

5.1 物理地址规范的更改

MKTME最重要的变化是重新调整物理地址位的用途，以将密钥ID传递给内存控制器中的加密引擎。为了保持正确的行为，这一变化需要许多其他硬件和软件的变化。

激活MKTME后，平台物理地址的高位（从CPUID MAX\U PA info枚举的可用最高阶位开始）被重新调整用途，用作密钥ID，如下所示。

图5-1。密钥ID使用



5.1.1 IA寻呼

当在没有EPT的情况下使用IA寻呼时，IA页表的每一级从MAX_PA开始的高位被重新用作密钥ID位。类似地，CR3中物理地址的高位将以相同的方式处理。

请注意，当EPT处于活动状态时，IA分页不会生成/使用平台物理地址，而是生成/使用来宾物理地址。来宾物理地址不会被MKTME修改，将继续索引到EPT页表遍历中，就像启用MKTME之前一样。

5.1.2 EPT寻呼

当在VMX非根操作期间启用EPT时，EPT页面遍历的每个级别的高位被重新调整用途，用作密钥ID位。类似地，将以相同的方式处理EPTP中的物理地址的高位。注意，来宾操作系统也可以在IA页面地址中使用KeyID，完整的来宾PA（包括KeyID）由EPT使用。

5.1.3 其他物理地址

其他物理寻址结构，如VMCS指针、物理寻址位图等，将在地址的高位开始时接受类似的处理

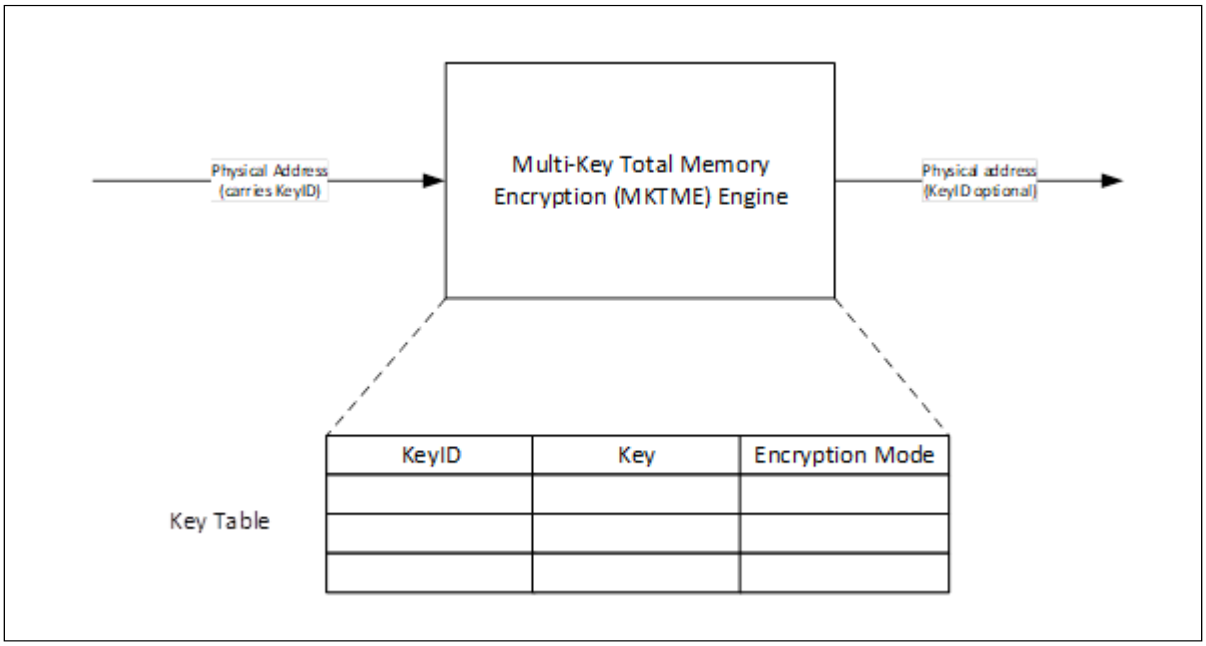
最大功率放大器被重新用作密钥ID位。请注意，任何保留位检查保持不变，这意味着这些地址的检查将仅基于CPUID MAX\u PA值。

6 MKTME键编程

6.1 概述

图6-1显示了MKTME引擎的高级概述，旨在介绍文档其余部分使用的术语，并不意味着实现。

图6-1。MKTME发动机概述



MKTME引擎维护软件无法访问的内部密钥表，以存储与每个密钥ID相关的信息（密钥和加密模式）。每个KeyID可以与三种加密模式相关联：使用指定的密钥加密、完全不加密（内存将是纯文本）或使用TME密钥加密。未来的实现可能支持其他加密模式。PCONFIG是一条新指令，用于为MKTME编程KeyID属性。虽然最初的实现可能只使用PCONFIG for MKTME，但将来可能会扩展它以支持其他用途。因此，PCONFIG与MKTME分开枚举。

6.2 PCONFIG指令

表6-1。PCONFIG指令详细信息

操作码	指示	描述
0F 01 C5	PCONFIG公司	此指令用于执行配置平台功能的功能。 EAX：要调用的叶函数。 RBX/RCX/RDX：叶特定用途。

6.2.1 PCONFIG说明

PCONFIG指令由软件调用，用于配置平台功能。PCONFIG支持多个叶，通过在EAX中设置适当的叶值来调用叶函数。RBX、RCX和RDX具有特定于叶子的用途。尝试执行未定义的叶会导致#GP（0）。PCONFIG是一个包范围的指令，同样，每个物理包需要执行一次，以配置所需的平台功能。

地址和操作数是64位模式外的32位（IA32\u EFER.LMA=0 | CS.L=0），在64位模式下是64位（IA32\u EFER.LMA=1和CS.L=1）。CS。D值对地址计算没有影响。DS段用于创建线性地址。

表6-2显示了PCONFIG的叶编码。

表6-2。PCONFIG叶编码

叶	编码	描述
MKTME_KEY_程序	0x00000000	此叶用于编程与密钥ID相关的密钥和加密模式。
保留	0x00000001-0xFFFFFF	保留供将来使用（如果使用，则为#GP（0））。

PCONFIG目标定义为平台上可以使用PCONFIG配置的任何硬件块。PCONFIG目前只支持一个目标，即MKTME。

表6-3显示了PCONFIG支持的目标。

表6-3。PCONFIG目标

目标标识符	价值	描述
无效的_目标	0x00000000	目标标识符无效。
MKTME公司	0x00000001	多密钥全内存加密引擎。
保留	0x00000002-0xFFFFFF	保留供将来使用。

6.2.1.1 MKTME_KEY_程序页

软件使用PCONFIG的MKTME\u KEY\u程序叶来管理与密钥ID相关的密钥。通过在EAX中设置叶值“0”和在RBX中设置MKTME\u KEY\u PROGRAM\u STRUCT的地址来调用叶函数。成功执行叶子清除RAX（设置为零），清除ZF、CF、PF、AF、of和SF。如果出现故障，则在RAX中指示故障原因，ZF设置为1，CF、PF、AF、of和SF被清除。MKTME\u KEY\u程序叶使用内存中的MKTME\u KEY\u PROGRAM\u结构工作，如表6-4所示。

表6-4。MKTME\u KEY\u PROGRAM\u结构格式

领域	偏移量（字节）	大小（字节）	评论
密钥ID	0	2	密钥标识
键ID\u CTRL	2	4	KeyID控件： • 位[7:0]：命令 • 位[23:8]：加密ALG • 位[31:24]：保留；必须为零
RSVD公司	6	58	保留；必须为零。
KEY_FIELD_1	64	64	软件提供的KeyID数据键或KeyID数据键的熵。
KEY_FIELD_2	128	64	软件提供的KeyID调整键或KeyID调整键的熵。

以下小节介绍了MKTME\u KEY\u PROGRAM\u STRUCT中的每个字段。

6.2.1.1.1 密钥ID

正在编程到MKTME引擎中的密钥标识符。

6.2.1.1.2 键ID\u CTRL

KEYID\u CTRL字段包含两个子字段，软件用于控制KEYID的行为：命令和KEYID加密算法。

使用的命令控制密钥ID的加密模式。表6-5总结了支持的命令。

表6-5。支持的按键编程命令

命令	编码	描述
KEYID\u SET\u KEY\u DIRECT	0	软件使用此模式直接编程密钥以与KeyID一起使用。
KEYID\u SET\u KEY\u RANDOM	1	CPU生成并分配一个临时密钥，用于密钥ID。每次执行指令时，CPU都会使用硬件随机数生成器生成一个新的密钥，并在重置时丢弃这些密钥。
KEYID\u CLEAR\u键	2	清除与钥匙ID相关的（软件编程）钥匙。执行此命令时，KeyID获取TME行为（使用平台TME密钥加密或绕过TME加密）。
KEYID\u NO\u加密	3	使用此密钥ID时不要加密内存。

加密算法字段（CRYPTO\u ALG）允许软件为密钥ID选择一个激活的加密算法。如前所述，BIOS可以激活一组算法，以便在使用IA32\u TME\u activate MSR编程密钥时使用（不适用于未绕过TME加密时使用TME策略的KeyID 0）。ISA检查以确保软件选择的算法是BIOS激活的算法之一。请注意

需要软件来提供该领域中激活的算法之一，包括KEYID\u CLEAR\u KEY和KEYID\u NO\u ENCRYPT命令。

6.2.1.1.3 KEY_FIELD_1

如果使用直接键编程选项（KeyID\u SET\u key\u direct），此字段包含软件提供的用于KeyID的数据键。当使用随机密钥编程选项（KEYID\u SET\u key\u random）时，该字段携带软件提供的熵，以混合在CPU生成的随机数据密钥中。软件有责任确保为直接编程选项提供的密钥或为随机编程选项提供的熵不会导致弱密钥。
指令中没有明确的检查来检测或防止弱键。当使用AES XTS-128时，上部48B被视为保留，必须在执行指令之前由软件清零。当使用AES XTS-256时，上部32B被视为保留，必须在执行指令之前由软件清零。

6.2.1.1.4 KEY_FIELD_2

如果使用直接键编程选项（KeyID\u SET\u key\u direct），此字段包含软件提供的用于KeyID的调整键。当使用随机键编程选项（KEYID\u SET\u key\u random）时，该字段携带软件提供的熵，将混合在CPU生成的随机调整键中。软件有责任确保为直接编程选项提供的密钥或为随机编程选项提供的熵不会导致弱密钥。
指令中没有明确的检查来检测或防止弱键。当使用AES XTS-128时，上部48B被视为保留，必须在执行指令之前由软件清零。当使用AES XTS-256时，上部32B被视为保留，必须在执行指令之前由软件清零。

在MKTME激活时，所有密钥ID默认为TME行为（使用TME密钥加密或绕过加密）。软件可以在任何时候使用PCONFIG指令决定更改密钥ID的密钥。更改密钥ID的密钥不会更改TLB、缓存或内存管道的状态。软件有责任采取适当的措施来确保正确的行为。第7节提供了软件流的示例。

表6-6显示了与PCONFIG的MKTME\u KEY\u程序叶相关的返回值。在指令执行时，RAX填充返回值。

表6-6。MKTME\u KEY\u程序的编程状态

返回值	编码	描述
PROG_成功	0	KeyID已成功编程。
无效的\u PROG\u CMD	1	KeyID编程命令无效。
熵_误差	2	熵不足。
无效的密钥ID	3	密钥ID无效。
无效的加密ALG	4	选择的加密算法无效（不支持）。
设备忙	5	无法访问密钥表（见第6.2.4节）。

6.2.2 PCONFIG虚拟化

VMX根模式下的软件可以使用为PCONFIG引入的以下执行控制来控制VMX非根模式下PCONFIG的执行：

- PCONFIG_ENABLE：此控件是一个单位控件，在VMX非根模式下启用PCONFIG指令。如果为0，则在VMX非根模式下执行PCONFIG会导致#UD。否则，PCONFIG的执行根据PCONFIG_退出而工作。辅助执行控制的VM退出控制27分配给PCONFIG_ENABLE。
- PCONFIG_退出：这是一个64b控件，允许VMX根模式为PCONFIG的各种叶函数导致VM退出。如果清除PCONFIG_ENABLE控件，则此控件没有任何效果。VMCS索引0x203E/0x203F（64b控制字段）分配给PCONFIG\u退出

6.2.3 PCONFIG枚举

PCONFIG在扩展特征（CPUID）中枚举。（EAX=07H，ECX=0H）：EDX[18]）。当为0时，PCONFIG将#UD。一个新的CPUID叶PCONFIG\u leaf（叶编码1BH）返回PCONFIG信息。更具体地说，子叶n（n□ 0）返回有关平台上支持的目标的信息。英特尔® 64和IA-32架构软件开发人员手册将定义子叶类型和返回的信息。预计软件将扫描所有子叶，以获取平台上支持的所有目标的信息。还应注意，同一目标的子叶不必是连续的。

6.2.3.1 CPUID。PCONFIG_叶。n（n□ 0)

返回有关平台上支持的目标的信息。返回的信息如下所示：

- EAX：子叶类型
  - 位11:0:0：无效子叶，1：目标标识符
- 如果EAX[11:0]==0
  - EAX:EBX:ECX:EDX=0
  - 子叶m>n返回所有0
- 如果EAX[11:0]==1
  - EAX[31:12]=0
  - EBX:Target\u ID\u 1
  - ECX:Target\u ID\u 2
  - EDX:Target\u ID\u 3

预计软件将扫描所有子叶，直到返回无效子叶。第一个无效子叶之后的所有子叶也无效。

6.2.4 PCONFIG并发

在PCONFIG的MKTME\u KEY\u程序叶在多个逻辑处理器上并发执行的场景中，只有一个逻辑处理器能够成功执行

更新键表。PCONFIG执行将在其他逻辑处理器上返回错误代码（DEVICE\u BUSY），软件必须重试。在指令执行失败且出现DEVICE\u BUSY错误代码的情况下，不会更新密钥表，从而确保要么使用密钥ID的信息整体更新密钥表，要么根本不更新密钥表。为了实现这一点，PCONFIG的MKTME\u KEY\u程序叶维护一个writer锁，用于更新密钥表。该锁称为键表锁，在指令流中表示为键表锁。当没有逻辑处理器持有锁（也是锁的初始状态）时，或者在逻辑处理器尝试更新密钥表的独占状态下，可以解锁锁。只能有一个逻辑处理器将锁保持在独占状态。锁是独占的，只能在锁处于解锁状态时获取。

PCONFIG使用以下语法以独占模式获取KEY\u TABLE\u锁并释放锁：

KEY\u TABLE\u锁。获取（写入）KEY\u TABLE\u锁。发布（）

6.2.5 PCONFIG操作

表6-7。变量定义

变量名称	类型	大小（字节）	描述
TMP\u KEY\u PROGRAM\u结构	MKTME_KEY_程序_结构	192	结构保持键编程结构。
TMP\u RND\u DATA\u键	UINT128	32	为随机密钥编程选项生成的随机数据密钥。
TMP\u RND\u TWEAK\u键	UINT128	32	为随机键编程选项生成的随机调整键。

```
（*#UD，如果未枚举PCONFIG或CPL>0*）
if（CPUID.7.0:EDX[18]==0或CPL>0）#UD;

if（在VMX非根模式下）
{
     if（VMCS.PCONFIG\u ENABLE==1）
     {
          如果（（EAX>62，VMCS.PCONFIG\u退出[63]==1）或（EAX<63，
             VMCS.PCONFIG\u退出[EAX]==1））
```

```
        }
        其他的
        {

        }
}
```

```
{

}

#环球开发商
```

```
设置VMCS。EXIT\u原因=PCONFIG; //无退出资格;

（*#GP（0）表示不受支持的叶*）
如果（EAX! =0）#GP（0）

（*KEY_程序叶流*）
if（EAX==0）
{
     （*#GP（0），如果TME_ACTIVATE MSR未锁定或未启用硬件加密或未启用多个密钥*）
     如果（IA32\u TME\u ACTIVATE.LOCK! =1或IA32\u TME\u ACTIVATE.ENABLE! =1或
     IA32\u TME\u ACTIVATE.MK\u TME\u KEYID\u位==0）#GP（0）

     （*检查MKTME\u KEY\u PROGRAM\u STRUCT是否与256B对齐*）
     if（DS:RBX未256B对齐）#GP（0）;

     （*检查MKTME\u KEY\u PROGRAM\u STRUCT是否可读取*）
     <<DS:RBX应可读取>>

     （*将MKTME\u KEY\u PROGRAM\u STRUCT复制到临时变量*）
     TMP\u KEY\u PROGRAM\u STRUCT=DS:RBX。*;

     （*RSVD现场检查*）
     if（TMP_KEY_PROGRAM_STRUCT.RSVD! =0）#GP（0）;

     if（TMP\u KEY\u PROGRAM\u STRUCT.KEYID\u CTRL.RSVD! =0）
     #GP（0）; if（TMP\u KEY\u PROGRAM\u STRUCT.KEYID\u
     CTRL.ENC\u ALG[0]==1）
     {

          if（TMP_KEY_PROGRAM_STRUCT.KEY_FIELD_1.BYTES[63:16]! =0）#GP（0）;

          if（TMP_KEY_PROGRAM_STRUCT.KEY_FIELD_2.BYTES[63:16]! =0）#GP（0）;
     }
     if（TMP\u KEY\u PROGRAM\u STRUCT.KEYID\u CTRL.ENC\u ALG[2]==1）
     {

          if（TMP_KEY_PROGRAM_STRUCT.KEY_FIELD_1.BYTES[63:32]! =0）#GP（0）;

          if（TMP_KEY_PROGRAM_STRUCT.KEY_FIELD_2.BYTES[63:32]! =0）#GP（0）;
     }
     （*检查有效命令*）
     if（TMP\u KEY\u PROGRAM\u STRUCT.KEYID\u CTRL.COMMAND不是有效命令）
     {
          RFLAGS。ZF=1;
          RAX=无效的PROG\u CMD;
          转到出口;
     }
     （*检查正在操作的密钥ID是否为有效的密钥ID*）
     if（TMP\u KEY\u PROGRAM\u STRUCT.KEYID>
                              2^IA32\U TME\U激活。MK\u TME\u KEYID\u
          BITS-1或TMP\u KEY\u PROGRAM\u STRUCT。密钥ID>
                              IA32\U TME\U能力。MK\u TME\u MAX\u
          KEYS或TMP\u KEY\u PROGRAM\u STRUCT。键ID==0）
```

```
{
    RFLAGS。ZF=1;
    RAX=无效的密钥ID;
    转到出口;
}
```

```
（*检查KeyID只请求一个算法，并且它是激活的算法之一*）
 if（NUM\u位（TMP\u KEY\u PROGRAM\u STRUCT.KEYID\u CTRL.CRYPTO\u
    ALG）！=1 ||（TMP_KEY_PROGRAM_STRUCT.KEYID_CTRL.CRYPTO_ALG&
      IA32\U TME\U激活。MK\u TME\u CRYPTO\u ALGS==0）
{
     RFLAGS。ZF=1;
     RAX=无效的加密ALG;
     转到出口;
}
（*尝试获取独占锁*）
if（非KEY\u TABLE\u LOCK.ACQUIRE（WRITE））
{
     //PCONFIG失败
     RFLAGS。ZF=1;
     RAX=设备忙;
     转到出口;
}

（*在此点之前，获取锁并根据命令更新密钥表，不更改密钥表*）

开关（TMP\u KEY\u PROGRAM\u STRUCT.KEYID\u CTRL.COMMAND）
{
case KEYID\u SET\u KEY\u DIRECT:
     <<写入
          DATA\u KEY=TMP\u KEY\u PROGRAM\u STRUCT。KEY\u FIELD\u
          1，TWEAK\u KEY=TMP\u KEY\u PROGRAM\u STRUCT。KEY\u
          FIELD\u 2，ENCRYPTION\u MODE=ENCRYPT\u WITH\u KEYID\u
          KEY，
          到索引TMP\u Key\u PROGRAM\u STRUCT的MKTME Key表。密钥ID
     >>
     打破

case KEYID\u SET\u KEY\u RANDOM:
     TMP\u RND\u DATA\u KEY=<<使用硬件RNG生成随机密钥>>如果（没有足够的熵）
     {
          RFLAGS。ZF=1;
          RAX=熵_误差;
          转到出口;
     }
     TMP\u RND\u TWEAK\u KEY=<<使用硬件RNG生成随机密钥>>如果（没有足够的熵）
     {
          RFLAGS。ZF=1;
          RAX=熵_误差;
          转到出口;
     }
     （*将用户提供的熵混合到数据键和调整键*）
     TMP\u RND\u DATA\u KEY=TMP\u RND\u KEY XOR TMP\u KEY\u PROGRAM\u
          STRUCT。KEY_FIELD_1。字节[15:0];
     TMP\u RND\u TWEAK\u KEY=TMP\u RND\u TWEAK\u KEY XOR TMP\u KEY\u
          PROGRAM\u STRUCT。KEY_FIELD_2。字节[15:0];

     <<写入
          DATA\u KEY=TMP\u RND\u DATA\u KEY，
```

```
            TWEAK\u KEY=TMP\u RND\u TWEAK\u KEY，ENCRYPTION\u MODE=ENCRYPT\u
            WITH\u KEYID\u KEY，
            到索引TMP\u KEY\u PROGRAM\u STRUCT的MKTME\u KEY\u表。密钥ID
        >>
        打破

    case KEYID\u CLEAR\u键：
        <<Write
        DATA\u KEY='0，
        TWEAK\u
        KEY='0，
        ENCRYPTION_MODE=ENCRYPTION_WITH_TME_KEY_或_BYPASS，
        到索引TMP\u KEY\u PROGRAM\u STRUCT的MKTME\u KEY\u表。密钥ID
        >>

    打破
    case KEYID\u NO\u加密：
        <<Write
        DATA\u KEY='0，
        TWEAK\u
        KEY='0，
        ENCRYPTION\u MODE=无加密，
        到索引TMP\u KEY\u PROGRAM\u STRUCT的MKTME\u KEY\u表。密钥ID
        >>
        打破
    }

RAX=0；
RFLAGS。ZF=0；
```

//释放锁

```
KEY_TABLE_锁（释放）；
```

出口：

```
RFLAGS。CF=0；
RFLAGS。PF=0；
RFLAGS。AF=0；
RFLAGS。OF=0；
RFLAGS。SF=0；
}
```

_流的结束

6.2.6 受影响的标志

采埃孚	如果指令成功完成，则清除。 如果发生错误，则设置。RAX设置为错误代码。
CF、PF、AF、OF、SF	变明朗。

6.2.7 前缀的使用

锁定	原因#UD
代表*	原因#UD（包括REPNE/REPNZ和REP/REPE/REPZ）
操作数大小	原因#UD
烦恼	原因#UD
段替代	忽略的
地址大小	忽略
雷克斯	忽略

6.2.8 保护模式异常

#环球开发商	如果使用了任何LOCK/REP/OSIZE/VEX前缀。 如果当前特权级别不是0。如果是 CPUID。7.0:EDX[18]=0。 如果处于VMX非根模式和VMCS。PCONFIG_ENABLE=0。
#GP（0）	如果EAX中的输入值编码了不受支持的叶。 如果IA32\U TME\U ACTIVATE MSR未锁定。 如果IA32\U TME\U中未启用硬件加密和MKTME功能，请激活MSR。 如果内存操作数不是256B对齐。 如果设置了MKTME\u KEY\u PROGRAM\u STRUCT中的任何保留位。 如果内存操作数有效地址超出DS段限制。
#PF（故障代码）	如果访问内存操作数时发生页面错误。

6.2.9 实地址模式异常

#环球开发商	如果使用了LOCK/REP/OSIZE/VEX前缀。 如果当前特权级别不是0。 如果是CPUID。7.0:EDX[PCONFIG\u位]=0。 如果处于VMX非根模式和VMCS。PCONFIG_ENABLE=0。
#GP（0）	如果EAX中的输入值编码了不受支持的叶。 如果IA32\U TME\U ACTIVE MSR未锁定。 如果IA32\u TME\u中未启用硬件加密和MKTME功能，请激活MSR。 如果内存操作数不是256B对齐。 如果设置了MKTME\u KEY\u PROGRAM\u STRUCT中的任何保留位

6.2.10 Virtual-8086模式异常

#环球开发商	在虚拟8086模式下无法识别PCONFIG指令

6.2.11 兼容模式异常

与保护模式中的异常相同。

6.2.12 64位模式异常

#环球开发商	如果使用了LOCK/REP/OSIZE/VEX前缀。 如果当前特权级别不是0。如果是 CPUID。7.0:EDX[18]=0 如果处于VMX非根模式和VMCS。PCONFIG_ENABLE=0。
#GP（0）	如果EAX中的输入值编码了不受支持的叶。 如果IA32\U TME\U ACTIVATE MSR未锁定。 如果IA32\U TME\U中未启用硬件加密和MKTME功能，请激活MSR。 如果内存操作数不是256B对齐。 如果设置了MKTME\u KEY\u PROGRAM\u STRUCT中的任何保留位。 如果内存操作数是非标准形式#PF（故障代码）如果访问内存

操作数时发生页面错误。

7 软件生命周期：使用KeyID管理页面

7.1 概述

如本文档前面所述，密钥ID是物理地址的一个组成部分，这意味着它不仅存在于页表中，而且还存在于TLB、缓存等中。因此，软件需要了解这一点，并且必须采取适当的步骤来维护操作的正确性和安全性。

请注意，虽然本节重点介绍虚拟化场景，但TME和MKTME体系结构适用于本机操作系统和虚拟化环境，以及DRAM和NVRAM类型的内存。

7.2 限制和缓存管理

硬件/CPU不强制具有不同密钥ID或加密密钥的同一物理页的映射之间的一致性。系统软件负责仔细管理有关密钥标识符（KeyID）使用的缓存，并在软件更改密钥id或与物理页面相关联的密钥时保持缓存一致性。具体来说，CPU将把除KeyID位之外相同的两个物理地址视为两个不同的物理地址，尽管这两个地址引用内存中的相同位置。软件必须采取必要的步骤，以确保这不会导致不可预测或不正确的行为，或违反所需的安全属性。MKTME保留整个物理地址（包括物理地址的KeyID部分）的缓存和

TLB的现有行为，并期望软件正确刷新缓存和/或执行TLB分解。

以下各节旨在给出软件不应使用的算法示例，以确保正确性和安全性。请检查本规范的最终版本，以了解该领域的任何更新算法或要求。

7.3 处理混叠地址映射的通用软件指南

以下列表详细介绍了操作系统/VMM软件供应商在使用具有多个默认单一密钥ID的MKTME时应考虑的一些一般准则。

1. 软件应避免使用多个密钥ID映射同一物理地址。
2. 如果软件必须用多个KeyID映射同一物理地址，则应将这些页面标记为只读，但一个KeyID除外。
3. 如果软件必须将同一物理地址与多个KeyID映射为读写，则软件必须确保所有写入都使用单个KeyID完成（这包括不修改数据的锁定和非锁定写入）。

7.4 AddPage：将密钥ID与页面关联

当向物理页分配新的密钥ID时，操作系统/VMM应该使用以下算法。

1. 如果尚未编程（使用PCONFIG指令），则为KeyID编程一个新密钥。
2. 如果尚未映射，则通过更新其分页结构条目（IA-PT）将物理页映射到VMM的地址空间（使用新的密钥ID）。
3. 确保步骤1已成功完成。
4. 通过新映射（使用新的KeyID）将页面内容归零，以避免KeyID域之间的数据泄漏。
5. 使用在EPT页面表条目中设置的新KeyID，使页面可供新VM使用。

这将确保当物理页的KeyID发生更改时（但数据在CPU缓存中处于清除状态），KeyID域（例如具有KeyID的虚拟机）之间不会发生数据泄漏。假设是，在使用此算法分配新的密钥ID之前，软件/VMM确保页面正确地从以前的密钥ID中退出（使用下一节中定义的算法）。

注： PCONFIG指令的使用指南：PCONFIG是包范围，因此软件预期在每个包/套接字的一个LP上执行PCONFIG。软件可以使用CPUID Leaf 0BH来确定系统的拓扑结构，该拓扑结构将指示系统上存在的物理包。

7.5 退出页面：解除键ID与页面的关联

当更改物理页面的密钥ID时，操作系统/VMM应使用以下算法，以便当前密钥ID不再与页面一起使用。

1. 更改密钥ID之前要完成的步骤：

   a) 使虚拟机无法访问物理页面（通过更新EPT页面表条目）。
   b) 从TLB（跨逻辑处理器，使用旧密钥ID）使所有页面映射/别名无效（如果页面映射为设备可访问，则INVEPT指令和IOMMU（VT-d）无效）。
   c) 如果尚未映射，则通过更新页面的分页结构条目（IA-PT）将页面映射到VMM地址空间（使用旧密钥ID）。
   d) 操作系统/VMM刷新脏缓存线（用于使用旧密钥ID的页面），以防止别名覆盖/数据损坏。
      — 选项：CLFLUSH、CLWB+fence、CLFLUSHOPT+fence或WBINVD。
      — 如果软件使用EPT页面修改日志或EPT（优化）的访问和脏标志跟踪页面修改，则可以选择性地避免进行这些刷新。

2. 该页面现在可以与新的KeyID一起使用（例如，使用上一节中的步骤）。

这将确保当物理页的键ID更改时，CPU缓存中不存在由物理地址混淆的缓存线。

注： WBINVD指令的使用指南：如果WBINVD指令使套接字上的所有相干缓存失效，则应在每个套接字上运行WBINVD指令。

7.6 按操作系统/VMM分页示例

下面是一个软件序列示例，其中操作系统/VMM正在将页面从VM2重新分配到VM3。VM2内存使用KeyID2，VM3内存使用KeyID3。

1. 使用第7.5节中描述的逐出页面算法，从VM2逐出具有KeyID2的页面。
2. 操作系统/VMM使用KeyID2读取退出的页面，使用完整磁盘加密密钥（可选）加密页面内容，并将页面写入交换文件（或在操作系统/VMM内存中，使用VMM KeyID=0）上的磁盘/存储。
3. 使用第7.4节中的AddPage算法将退出的页面添加到VM3 KeyID3。

7.7 操作系统/VMM对来宾内存的访问

操作系统/VMM可以通过在其分页结构条目（IA-PT）中设置来宾密钥ID位来访问来宾内存（以明文形式）以达到仿真目的（MMIO）。

7.8 输入/输出交互

操作系统/VMM可以使用TME密钥（KeyID=0）在来宾VM和VMM之间根据需要设置共享内存，以用于I/O目的。对于定向输入/输出（例如SR-IOV），操作系统/VMM应将密钥ID作为IOMMU（VT-d）页表中物理地址的一部分进行编程，对应于密钥ID作为EPT（用于来宾VM）中物理地址的一部分。这将允许DMA能够在不需要更改来宾VM或操作系统/VMM中的输入/输出设备和/或输入/输出驱动程序的情况下访问内存。