云基础结构中可信计算性能的性能研究与 IntelSGX 一起使用

安德斯，罗伯特佩特森，哈佛约翰森和大约翰森
挪威北极大学，特罗姆斯 0，挪威

关键词： 隐私，安全，云计算，可信计算，性能。

摘要： 敏感的个人数据越来越多地托管在第三方云提供商上。这就产生了强大的力量
随着可信计算基础的扩展，对数据安全和隐私的担忧被扩展到包括不受负责数据的管理实体直接监督的硬件组件。幸运的是，主要的硬件制造商现在包括了促进安全远程执行的机制。本文研究了 InteFs 软件保护扩展(SGX)，并通过实验量化了该指令集扩展的基本使用将如何影响如何构建云托管服务。我们的实验表明，服务的功能组件的正确分区将对性能至关重要。

1 简介

传感器和移动设备记录了我们日常生活的更多方面。这导致了数据流流入各种云提供商远程托管的潜在复杂分析管道。生成的数据不仅难以大规模存储和分析；数据还可能伴随着严格的隐私要求，如智能家居和健康监控设备（Gjerdrum 等人，2016）。

在云中处理敏感和个人数据需要设计新的软件即服务(SaaS)架构，能够在整个硬件和软件栈中执行严格的隐私和安全政策(Johansen 等人，2015)，包括底层云提供的基础设施即服务(IaaS)组件。尽管用于可信计算的商品硬件机制已经可用了一段时间(TCG 发布，2011 年；奥斯本和查勒纳，2013 年)，但这些机制通常都有性能和功能限制。Intel 以前的实现，如信任平台模块(TPM)和信任执行技术(TXT)，能够建立信任和保证软件的完整性，后者也支持基本的安全代码执行。

软件保护扩展(SGX)(Anati 等人，2013)是英特尔新的可信计算平台，连同 ARM 和 AMD 的类似努力，正在迅速使一般可信计算成为 com-模型。从根本上说，SGX 是一个指令集扩展，由 Skylake 生成的入侵核心架构一起引入，支持对在不信任平台上运行的可信代码的机密性、完整性和认证。SGX 能够通过构建由受信任的代码和数据段组成的安全飞地来对抗许多不同的软件和物理攻击。虽然 SGX 应该被认为是一种建立在以前的努力基础上的迭代技术，但它在性能和功能方面都超过了以前的迭代。SGX 旨在提供一般安全计算设施，允许开发人员轻松地将现有遗留应用程序移植到启用 SGX 的飞地。这些特性使 SGX 成为处理人员敏感数据的基于云的 SaaS 架构的一项很有吸引力的技术。

SGX 是一个专有平台，先验知识是基于描述其体系结构的有限文档。此外，对于 SGX 平台提供的原语的性能以及如何在最大化性能的同时利用这些原语编写软件，我们也知之甚少。

在本文中，我们分析了目前 SGX 技术的性能特征，以更好地理解这些技术如何用于在云托管 SaaS 体系架构中执行隐私策略。我们在细粒度级别上分析了 SGX 原语，并提供了 SGX 中核心机制的详细性能评估。本文的结构如下：第 2 节概述了 SGX 微观体系结构的相关部分，而第 3 节概述了

气鼓，佩特森，约翰森，约翰森和约翰森，D.
利用 IntelSGX 在云基础结构中的可信计算的性能。
部门：10.5220/0006373706960703

在第七届云计算与服务科学国际会议论文集上（2017 年即将出版）中，第 668-675 页
编号：978-989-758-243-1

我们的微基准测试的细节。第 4 节对我们的调查结果进行了知情的讨论，第 5 节在结束发言前详细介绍了相关的工作。

2 英特尔软件保护装置扩展名(sgx)

SGX 通过向逻辑处理器发出特殊的 EENTER 特殊指令，允许常规的应用程序线程转换到安全的飞地。输入是通过对飞地代码执行控制跳转来启动的，类似于进入虚拟机上下文的方式。进程只能从环 3 进入飞地，即用户级别，在飞地模式下运行的线程不允许触发软件中断，也禁止使用系统调用。一个需要访问公共操作系统(OS)提供的服务的应用程序，如文件系统，必须仔细设计，以便其线程在调用任何系统调用之前通过应用程序定义的接口退出飞地模式。由于 SGX 的可信计算基础(TCB)不包括底层操作系统，因此所有这些转换、参数和响应都必须由应用程序设计器仔细验证。

SGX 允许多个线程在同一飞地内执行。对于在飞地内执行的每个逻辑处理器，都需要一个线程控制结构(TCS)。必须在飞地启动之前设置这些数据结构，并存储在为飞地预留的飞地页面缓存(EPC)主内存页面中。除此之外，TCS 包含 OENTRY 字段，它在进入飞地时被加载到指令指针中。在此之前，SGX 通过使用 XSAVE 指令将不受信任代码的执行上下文存储到常规内存中，然后在离开飞地时再次恢复。然而，进入飞地时不会修改堆栈指针（科斯坦和德瓦达斯，2016）建议，为了避免漏洞的可能性，预计每个飞地将其堆栈指针设置为完全包含 EPC 内存的区域。到飞地的参数输入被封送到缓冲区中，当转换完成后，飞地代码可以直接从不受信任的 DRAM 内存中复制数据。这不是本机 SGX 实现的一部分，而是由应用程序 SDK 提供的便利。

线程要么通过同步退出指令自愿退出飞地，要么通过在受影响逻辑核心上发生的硬件中断进行异步退出。以同步方式通过 EEXIT 指令退出，将导致逻辑处理器离开飞地模式。指令指针和堆栈指针在进入飞地之前会被恢复到它们之前的地址。SGX 不修改任何关于飞地出口的指示，因此作者有责任清除它们，以避免泄露秘密信息。在异步飞地退出(AEX)的情况下，页面故障等硬件中断会导致处理器退出飞地并跳转到内核以为故障提供服务。在此之前，SGX 将执行上下文保存到 EPC 内存中以保存，然后清除它，以便操作系统无法从飞地推断任何执行状态。完成中断处理程序后，SGX 将恢复执行上下文并恢复执行。

2.1 飞地页面缓存

飞地使用的内存在启动时从常规进程 DRAM 内存分离为所谓的处理器保留内存(PRM)。此连续的内存区域分为 4kb 页，统称为飞地页面缓存(EPC)。EPC 内存只能在飞地内或通过 SGX 指令集进行访问。在保护环 0（内核模式）运行的系统软件或在环 3（用户模式）运行的应用程序代码都不能直接访问其内存内容，任何读取或写入的尝试都被忽略。此外，硬件禁止 DMA 访问 PRM 内存，以防止恶意外围设备试图轻触系统总线。飞地的机密性由 Inters 内存加密引擎(Mee)保护，它在 L3 缓存后的系统总线上对 CPU 包边界上的内存进行加密和解密。

与虚拟内存类似，EPC 页面管理完全由操作系统处理。但是，EPC 模式不能直接访问内存，每个页面分配都通过 SGX 指令完成。该操作系统负责将页面分配给特定的飞地，并将页面驱逐给常规的 DRAM。当前一代的 SGX 硬件只支持 128MB 的最大 PRM 大小，但通过交换，对飞地的大小没有实际限制。交换出的页面的完整性可以通过始终检查一个也驻存在 PRM 中的辅助数据结构来保证，它被称为飞地页面缓存映射。此数据结构包含虚拟地址和物理 PRM 内存之间的正确映射，以及对每个页面的完整性检查。每个一页只能属于一个飞地，因此，禁止飞地之间共享记忆。然而，如果居住在同一进程的地址空间中，他们能够共享 DRAM 内存，并且飞地内存允许飞地内存读取和写入该进程中的不受信内存。页面驱逐指令还会为每个页面生成一个活力挑战，并将它们存储在特殊的 EPC 页面中，以供以后进行比较。这些预防措施可以防止恶意操作系统试图通过操纵地址翻译、显式操作页面或将旧页面服务回飞地（重播攻击）来颠覆飞地。

为了防止执行飞地的陈旧地址转换，处理器会对被驱逐的页面执行一个粗粒度的 TLB 射击。针对特定飞地的页面故障将导致内核为有关飞地内部运行的所有逻辑内核发出处理器间中断(IPI)。这将导致每个线程执行 AEX，并捕获到内核页面故障处理程序。此外，存储在 CR2 注册表中的故障虚拟地址的最低 12 位被清除，以便操作系统无法推断任何访问模式。为了摊销为每个页面驱逐中断在特定飞地内执行的所有核心的成本，SGX 实现支持一次批处理多达 16 页的驱逐。

2.2 飞地程序创建

SGX 支持单台机器上的多个在同一进程的地址空间内或在不同进程中相互不信任的飞地。飞地由系统软件代表应用程序创建，发出电子指令。这将导致 SGX 为 SGX 飞地控制结构(SECS)分配一个新的 EPC 页面，该页面存储每个飞地的元数据。SGX 指令使用它来识别飞地，并通过 EPCM 结构将飞地映射到物理 EPC 页面。在飞地准

备好执行代码之前，必须通过操作系统向每个页面的 SGX 实现发布特别精心制作的指令，将每个初始代码和数据段添加到飞地内存中。同样的指令也用于为飞地内的每个预期线程创建 TCS。此外，操作系统驱动程序还发布用于软件认证的飞地测量值的更新。关于 SGX 认证过程的描述，我们可以参考 SGX 开发者手册。当所有页面加载时，飞地被初始化，飞地从英特尔委托的特殊预设飞地收到发射令牌。在这一点上，这块飞地被认为是完全初始化的，不可能发生进一步的内存分配。SGX 版本 2 的修订规范包括通过动态分页支持对初始创建后扩展飞地的支持。但是，由于目前尚未发布支持这些规范的硬件。

当一个飞地被摧毁时，情况就会发生相反情况，因为操作系统将该飞地使用的每个页面标记为电子移动指令无效。在释放页面之前，SGX 要确保在拥有特定页面的飞地内没有执行任何逻辑处理器。最后，如果 EPCM 中提到该特定飞地的所有页面都被释放，则会释放 SECS。

3 实验结果

为了获得如何构建下一代基于云的 SaaS 系统以最好地利用现代处理器中的 SGX 特性的经验，我们在启用 SGX 的硬件上运行了一系列微基准测试。我们的实验设置包括一个戴尔光电工作站，带有一个 IntelCorei5-6500CPU@3.20GHz，4 个逻辑核和 2x8GB 的 DDR3DIMMDRAM。为了避免动态频率缩放造成的不准确，在我们所有的实验中都禁用了英特尔速度步进和状态。为了衡量该体系结构的峰值性能，我们还将硬件设置中的 PRM 大小修改为允许的最大值 128MB。我们使用 IntelSGX 的开源内核模块在 Ubuntu14.04 上运行了实验。[1]我们测试了 SGX 内核模块来记录操作成本。基于我们对系统的理解，我们推导了不同的基准测试平台的各种特性测试。所有实验的常见情况是，观察到更多的迭代不会产生较低的偏差。这可能归因于系统其他部分产生的噪声。这种噪音很微妙，但很重要，因为我们是在细粒度的时间间隔内测量的。

请注意，对 SGX 的当前迭代禁止在飞地内部使用 RDTSC 指令，因此在飞地内部没有可用的本地定时设施。后来的一个版本显示，SGX 版本 1 的更新规范确实支持飞地内的 RDTSC。提示表明，这可能可以通过对微码体系结构的更新来分发。然而，我们没有成功地获得这个更新。因此，在整个实验过程中进行的时间测量必须在被捕获之前离开这块飞地。因此，我们只能测量进出一块飞地之间所花费的总时间，如图 1 中所描述的事件序列。

[1]https://github.com/01org/linux-sgx-driver

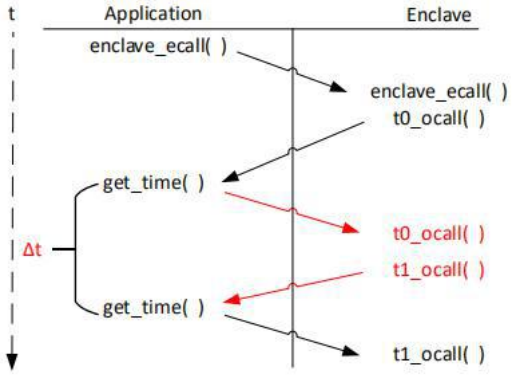

图 1：测量在飞地内花费的时间所涉及的事件顺序。

3.1 进出境费用

[2]在我们的第一个实验中，我们着眼于进出一块飞地的代价。了解这一成本很重要，因为它规定了启用 SGX 的 SaaS 服务如何将其功能分区在飞地和非飞地执行之间，以最小化 TCB 的大小。巨大的进入成本将需要减少进入呼叫的数量，从而增加驻留在飞地内的代码和数据的数量，增加所需的 TCB。极端的情况是一个完整的库操作系统，它包含了应用程序在飞地内所需要的几乎所有功能(Baumann 等人，2014)。《英特尔软件开发手册》指出，进入飞地的成本也是复制到飞地的数据大小的函数。因此，如果实验表明，进入该飞地的大量数据的成本大得惊人，只有需要保密的数据才能复制到该飞地。

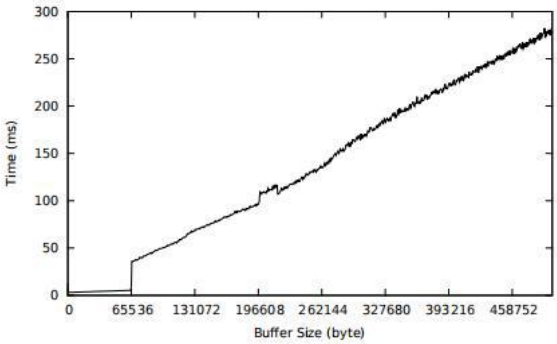

图 2：飞地转换成本作为缓冲区大小的函数。

图 2 显示了缓冲区大小增加的函数。如图所示，过渡到飞地的成本随缓冲区的大小呈线性增加。这个实验只在过渡到飞地时使用缓冲区作为参数。要能够在飞地内托管缓冲区，其堆大小必须足够大。没有缓冲区的观察到的基线成本是进入飞地的基本过渡成本。随着缓冲区大小的增加，这个成本很快就会变得无关紧要。这种行为是预期的，因为这个成本包括在转换时将缓冲区复

[2]https://software.intel.com/en-us/articles/intel-sdm

制到飞地内存中，从而调用 Mee 来获取写入飞地的内存。然而，令我们惊讶的是，我们观察到基线成本只增加到 64kb 以上。一种可能的解释是，对于小于 64kb 的缓冲区大小，页面可能已经存在于 EPC 内存中

对于较大的缓冲区，增加的成本也可以归因于以前被驱逐到 DRAM 的飞地内存造成的页面故障。这个问题将在下一个实验中得到进一步的探讨。

3.2 分页码

另一个可能的建筑权衡是一个逻辑假设，即 TCB 的增加将减少飞地的过渡，但需要更多的 PRM。如第 1 节所述，与常规 DRAM 相比，PRM 的资源非常有限，而且该系统共有 128MB。此外，当 PRM 资源变得稀缺时，任何飞地都会受到驱逐 EPC 页面的系统软件的约束。任何使用 SGX 的系统都应该考虑到在 PRM 和常规 DRAM 之间交换页面的成本。图 3 说明了内核和用户层飞地观察到的这一成本。

y 轴是以纳米秒为单位的离散成本，而 x 轴是实验所经过的时间。我们测试了操作系统内核驱动程序来测量将页面从 EPC 驱逐为由红点表示的 DRAM 所花费的时间，以及在页面故障处理程序中花费的总时间，由黑线显示。

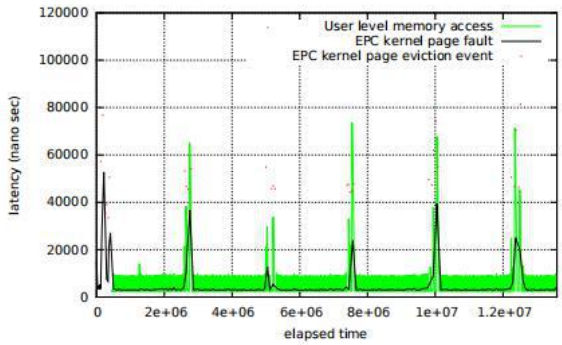

图 3：纳米秒内的寻呼开销作为所用时间的函数。

绿线表示用户级仪表，测量写入 EPC 内存中的特定地址所需的时间。与之前的实验类似，我们被禁止在飞地内进行定时测量。因此，用户级别测量包括飞地进出的基线成本，特别是每次都有 4 字节缓冲区转换。

为了导致飞地页面故障，我们将飞地堆的总大小设置为 256MB，这大于可用的 EPC 内存的总量。此外，要点击每个页面，我们可以沿着飞地内分配的内存空间按顺序对 4kb 页面大小内的每个地址进行写入操作。如第 2 节所述，飞地唯一能够分配内存的时间是在通过发出 EADD 调用 EINIT 指令之前。因此，在飞地执行开始之前，必须分配所有内存。在实验开始时，我们可以清楚地看到，当我们试图将 256MB 的飞地内存添加到潜在的 128MB 的物理 EPC 内存中时，页面故障会增加。

将在用户级和内核级发生的不同事件关联起来，我们观察到驱逐事件与在用户级写入时间的增加之间的强烈关系。系统可能增加这一成本的一个属性是，驱逐页面会导致在飞地内执行的任何逻辑处理器发生 AEX 事件，如第 2 节所述。

我们还观察到，内核驱动程序在根据执行过程中发生的页面故障的数量将 EPC 页面分配给飞地方面的操作方式非常保守。此外，如第 2 节所述，在捕获到页面故障处理程序之前，SGX 将清除虚拟页面故障地址的下 12 位。因此，驱动程序无法对飞地内的内存访问模式做出任何假设。此外，正如第 2 节所解释的那样，活动挑战数据也可能被删除 EPC 内存，导致从 DRAM 发生级联的页面负载。值得注意的是，我们的实验只使用了一个线程，而且所有发布 IPI 的页面驱逐程序都只中断了这个单一的线程。

很明显，高性能应用程序可能希望根据其需要调整分页支持。如果应用程序能够预测特定的访问模式，内核分页支持应该适应它。此外，通过优化 EPC 内存的详尽使用，在飞地内运行的应用程序可能会出现更少的页面故障。

此外，初始的设置将保留大量飞地的内存，这可能会消除一些飞地的寻页开销。这进一步减少了由 IPI 中断 trig-引起的开销

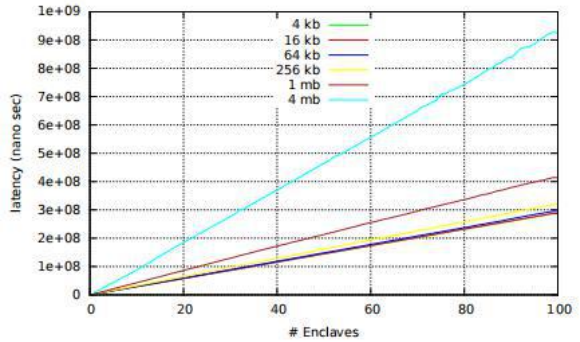

图 4：延迟作为针对不同大小的飞地同时创建的飞地数量的函数

从给定的飞地纠缠着 AEX。最初，创建大型飞地会触发内核的内存分配，而且应用程序开发人员可能有必要通过预览飞地来抵消这一初始成本。

3.3 飞地供应

现代分布式系统体系结构越来越依赖于模块化编程范式和可能具有不同信任域的多组件软件。这种分布式系统通常由几个第三方开源组件组成，包括受信任的和不受信任的。此外，分离这类系统的失败单元和信任单元通常是一个好主意。

SGX 支持创建多个相互不信任的飞地，可以用于这种模块化设计中。如第 2 节所述，SGX 编程模型允许飞地使用定义良好的接口与外部通信，该接口使架构的信任被划分为单独的飞地。图 4 说明了供应延迟作为同时

创建的飞地的额外成本，我们可以清楚地观察到，在创建飞地时的额外成本线性增加。通过这个实验的多次迭代，我们观察到增加飞地大小的增加成本。如前所述，当提供多个面积超过 256kb 的飞地时，这种增加的成本变得越来越显著。如第 2 节所述，飞地是通过为飞地内存中分配的每一页代码和数据发布特别精心制作的函数来创建的。值得一提的是，我们观察到在飞地创建过程中发生了大量的页面错误，并且可以合理地假设这也造成了成本。此外，图 2 所示的关于缓冲区大小小于 64kb 的进入成本的观察，进一步证实了对于小于 64kb 的飞地面积，供应成本几乎是相同的。

对于需要低延迟运行的应用软件，可能需要预调配飞地，以抵消这一延迟成本。然而，如果单个飞地足够大，这种方法可能会在在 EPC 内存中配置它们时带来额外的问题。

4 讨论内容

从我们在第三节的实验中，我们已经确定了 SGX 的几个重要的性能特性，在构建 SGX 启用的云服务时应该考虑这些特性：进出飞地的成本、数据复制的成本、提供新飞地的成本和内存使用的成本。

如第 2 节所述，进出程序在成本方面进行了类似数量的工作。正如我们的实验所表明的，过渡的最重要的成本因素是缓冲区大小的输入，作为通过进入或退出的过渡的参数。特别地，我们观察到当缓冲区大小大于 64kb 时，数据拷贝成本急剧上升。因此，我们的建议是：

建议 L 应用程序应将其功能组件进行分区，以最小化跨飞地边界复制的数据。

遵循建议 1 指南的一个可能的组件体系结构是将所有功能集中到一个飞地中，使其在很大程度上自给自足。采用这种方法的系统是 Haven(鲍 mann 等人，2014)，它通过库操作系统协同定位系统软件堆栈的更大一部分，减少了受信任代码和不受信任代码之间的接口。然而，这种方法的效率与我们在第 3.2 节中的观察直接矛盾，在该节中，我们测量了与飞地内存被输入到常规 DRAM 相关的开销。由于 EPC 是一种稀缺资源，系统软件积极页面飞地内存没有使用。然而，正如我们的实验表明，页面故障处理程序过于过度，未能充分利用 EPC 内存。由于安全考虑，内核没有给出每个飞地页面故障的确切错误地址，因此不会对内存访问模式做出任何假设。因此，我们建议：

. *建议 2 一块飞地的面积不应超过 64kb。*

. *建议 3 有关应用程序内存消耗和访问模式的事先知识应用于修改 SGX 内核模块，以减少内存页面驱逐。*

正如我们的实验所表明的那样，创造飞地是昂贵的和费时的。为了隐藏部分成本，底层操作系统可以在任何可以预测使用模式时预调配飞地。然而，一旦被使用，一块飞地可能会被秘密数据玷污。因此，将使用过的飞地回收到一个公共池中，可能会从一个过程泄露到另一个过程：使隔离保证无效。因此，我们建议：

. *建议 4 能够准确预测飞地实际使用情况的应用程序作者应该在一次性资源池中预先设置飞地，以保证隔离域之间不重复使用。*

创建飞地的成本还必须考虑到与每个飞地相关的元数据结构的增加的基线成本。设置飞地必须至少要考虑其 SECS、在飞地内执行的每个逻辑核心的一个 TCS 结构，以及执行 AEX 的每个线程的一个 SSA。（科斯坦和德瓦达斯，2016）解释说，为了简化实施，这些结构大多在 EPC 页面的开头分配，完全用于该实例。因此，不需要考虑一个具有 4 个逻辑核心的飞地，为它分配了 9 页(34kb)，不包括代码和数据段。应用程序应考虑单独飞地的增加内存成本以及可用 EPC 的相对数量。此外，为了抵消拥有多个飞地的成本，应用程序作者应该考虑连续规模的安全分离。某些安全模型可能满足基于角色的隔离，而不是要求所有用户单独明确隔离。因此，我们建议：

. *建议 5 应用程序的作者应仔细考虑其预期用途所需的隔离粒度，因为更细的粒度包括飞地创建的额外成本。*

在撰写本文的时候，唯一支持 SGX 的硬件是具有 SGX 版本 1 的天空湖代核心芯片。正如我们的实验所表明的那样，分页对性能有深远的影响，而一个自然的后续行动将是测量在 SGXV2 规范中提出的动态分页支持的性能特性。然而，如前所述，英特尔还没有发布任何有关 SGXV2 芯片到来的信息。即将到来的第 8 代卡比湖芯片不包括支持，因此最早的发布将在 2017 年第 4 季度作为炮湖的一部分。

SGX 支持通过使用签名的硬件度量建立信任来验证运行在不受信任平台上的软件。这些当事方可以在本地有两个不同的飞地在同一硬件上执行，也可以通过英特尔的认证服务进行远程执行。在未来，鉴于上述所展示的飞地过渡的巨大成本，检查飞地之间通信的安全通道的性能特征将是很有趣的。

5 相关工作

以前的一些工作量化了与基于 SGX 的复合体系结构相关的开销的各个方面。Haven(Baumann 等人，2014)通

过完全在 SGX 飞地中插入一个库操作系统来实现对未修改的遗留应用程序的屏蔽执行。这一努力导致了对 SGX 规范的体系结构更改，其中包括对动态分页的支持。Haven 概念实现的验证仅根据在系统顶部运行的遗留应用程序进行评估。此外，Haven 是建立在预发布的 SGX 上的，性能评估不能与实际应用程序直接媲美。阴影（Chen 等，2008）提供与 Haven 类似的功能，但不依赖专用的硬件支持。

斯康内（Arnautov 等人，2016）对 SGX 飞地内的安全集装箱提供支持。圆锥的设计是由关于飞地内部 TCB 大小的容器设计的实验驱动的，在最极端的时候，包括整个库操作系统，至少包括一个应用程序库的存根接口。对斯锥的评估，就像对 Haven 的评估一样，是基于在斯锥容器中运行的遗留应用程序。虽然（Arnotov 等人，2016）就 TCB 大小与内存使用和飞地转换成本得出的结论与（鲍曼等人，2014）相同，但他们没有量化这一成本。尽管如此，scone 还是为我们在第 3 节中概述的入口出口问题提供了一个解决方案，其中线程被固定在飞地内，而不会过渡到外部。相反，通信是通过飞地线程写入留在常规 DRAM 内存中的专用队列来实现的。然而，这种方法仍然很容易受到广告被 AEX 驱逐出飞地的影响，这是页面错误的一部分。

(Costan 和 Devadas，2016)基于现有技术、已发布的开发人员手册和专利，描述了 SGX 的架构。此外，他们对 SGX 进行了全面的安全分析，通过详细解释体系结构中可利用的漏洞，伪造了它的一些保证。这项工作大多与我们的努力是正交的，然而，我们对 SGX 的大部分知识都是基于这个主题的。这些之前的努力导致(Costan 等人，2016 年)实现了圣所，它实现了一个替代的硬件体系结构扩展，提供了许多与 SGX 相同的特性，但针对火箭 RISC-V 芯片体系结构。圣所通过模拟硬件评估其原型，根据不安全的基线评估其原型。（McKeen 等人，2016）对 SGX 规范引入了动态分页支持。我们无法使用这个原型硬件。

Ryoan(Hunt 等人，2016)试图通过解决介绍中概述的相同问题，实现一个分布式沙箱，以促进对第三方云服务上的秘密数据的不信任计算。Ryoan 提出了一种新的面向请求的数据模型，其中处理模块被激活一次，而不坚持对其的数据输入。此外，通过远程认证，Ryoan 能够验证沙箱实例的完整性并保护执行。通过将沙箱技术与 SGX 结合起来，Ryoan 能够创建一个屏蔽结构，支持应用程序和基础设施之间的相互不信任。同样，Ryoan 以真实世界的应用程序为基准，就像之前的其他工作一样，它并没有正确地量化归因于 SGX 原语的确切开销。此外，其大部分评估是在基于 QEMU 的 SGX 仿真器中进行的，该仿真器已经修改了基于真实硬件测量的延迟和 TLB 刷新，以更好地镜像真实的 SGX 性能。这些硬件测量值用于 EENTRY 和 EEXIT 指令，但并不确定移动参数数据进出飞地内存的成本。此外，Ryoan 推测了 SGXV2 分页支持的成本，尽管这是严格地基于模拟的测量结果和对物理成本的假设。

ARMTrustZone 是一种硬件安全架构，可集成到 ARMv7A、ARMv8-A 和 ARMv8-M 芯片上系统中(Ngabonziza 等人，2016；Shuja 等人，2016）。尽管底层的硬件设计、特性和接口与 SGX 有很大的不同，但两者本质上都提供了相同的硬件隔离执行域的关键概念，以及将已验证的软件堆栈引导到这些飞地的能力。但是，TrustZone 硬件只能区分两个执行域，并依赖于拥有一个基于软件的可信执行环境来进行任何进一步的改进。

6 研究结论

在线服务越来越依赖第三方云提供商来托管敏感数据。这种趋势引起了人们对数据所有者的安全和隐私的强烈担忧，因为不能完全信任云提供商来执行经常管理这类数据的限制性使用政策。IntelSGX 为商品硬件中的一般可信计算提供硬件支持。这些对 x86 指令集的扩展通过远程认证在不可信任的基础设施上提供的代码和数据段建立了信任，进一步保证了潜在恶意系统软件的机密性和完整性。

之前的努力通过严格的系统展示了 SGX 的能力，它能够在飞地内安全地托管大型遗留应用程序。然而，这些系统并没有量化与使用 SGX 相关的确切成本。本文评估了进出飞地的微观架构成本、数据复制成本、提供新飞地的成本和内存使用成本。由此，我们为希望使用 SGX 保护其云托管隐私敏感数据的应用程序作者提供了五个建议。

确认事项

这项工作得到了挪威研究理事会项目编号 231687/F20 的部分支持。我们要感谢匿名评论者的有用的见解和评论。

参考文献

阿纳蒂，盖伦，约翰逊，S.和斯卡拉塔，V.（2013）。基于 cpu 的认证和密封方面的创新技术。在第二届关于安全与隐私的硬件和架构支持的国际研讨会论文集中，第 13 卷。

阿尔诺托夫，特拉奇，格雷戈，格雷戈，F，可诺，马丁，A，普里比，林德，穆图库马兰，博士，奥基夫，医学博士，斯蒂尔威尔，医学博士，戈尔茨切，D，埃尔斯卡皮奇和费策尔，C.（2016）。Scone：保护 linux 容器。第 12 届 USENIX 操作系统设计与实施研讨会（OSD116），第 689-703 页，USENIX 协会。

鲍曼、皮纳多和亨特（2014）。使用 Haven 保护应用程序远离不受信任的云。在第 11 届 USENIX 操作系统设计与实现研讨会(OSDI‘14)上。USENIX 一高级计算系统协会。

陈，X.，加芬克尔，T.，刘易斯，E 苏伯罗门，P，沃尔德斯普格，博纳，D，博士，和港口，博士（2008）。阴影：基于虚拟化的保护方法。第 13 届编程语言和操作系统体系结构支持国际会议，ASPLOSXIII，第 2-13 页，美国纽约。ACM。

科斯坦五世案和德瓦达斯案（2016 年）。英特尔 sgx 解释说。在密码学中的电子打印存档。

科斯坦、列别捷夫和德瓦达斯（2016）。圣：最小的硬件扩展强大的软件隔离。在 USENIX 安全版中，第 16 卷，第 857874 页。

气鼓，约翰森，HD.D.，和约翰森，D.（2016）。将知情同意作为信息流政策实施于电子健康数据的安全分析：原则和实践。在中。IEEE 连接健康：应用、系统和工程技术会议：第一届医疗网络物理系统的安全、隐私和可信性国际研讨会，追逐‘16 年。我的。

亨特、朱、Z、徐、Y、彼得和威切尔、E.（2016）。Ryoan：用于秘密数据不可信任计算的分布式沙箱。在第 12 届 USENIX 操作系统设计与实施会议的论文集中，OSDI‘16 页，第 533-549 页，美国加州伯克利分校。USENIX 协会。

约翰森，H 州伯雷尔，范，雷内斯，施耐德，F 斯滕豪格，医学博士和约翰森，D.（2015）。使用元代码实施隐私政策。第 6 届亚太系统研讨会论文集，第 16 页。ACM。

麦肯，亚历山大罗维奇，我，阿纳蒂，我，卡斯皮约翰逊，莱斯利-赫德和罗萨斯，C.（2016）。Intel®软件保护扩展（英特尔®sgx）支持飞地内的动态内存管理。《安全与隐私硬件与体系结构支持 2016》，第 10 页。ACM。

奥斯本，J.。D.和冷却剂，华盛顿特区（2013）。可信平台模块演变。约翰霍普金斯大学美联社技术文摘，32（2）：536-543。

舒贾，J，甘尼，A.，比拉尔，K.，可汗，美国，马达尼，美国答，可汗，美国，和佐玛亚，纽约（2016）。移动设备虚拟化调查：分类和现状。ACM 计算调查(CSUR)，49（1）：1。

TCG 出版（2011 年）。TPM 主要是第 1 部分的设计原则。规范版本 1.2 版本 116，可信计算组。